![COPPE UFRJ logo - Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia]

REVISÃO EFETIVA DE PROGRAMAS LÓGICOS (BAYESIANOS) A PARTIR DE EXEMPLOS

Aline Marins Paes

Tese de Doutorado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Doutor em Engenharia de Sistemas e Computação

Orientadores: Gerson Zaverucha
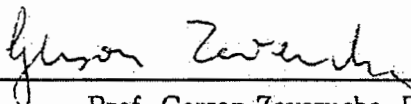   Vítor Manuel de Morais Santos Costa

Rio de Janeiro
Setembro de 2011

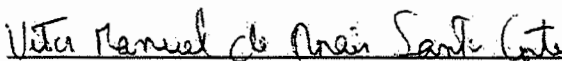# REVISÃO EFETIVA DE PROGRAMAS LÓGICOS (BAYESIANOS) A PARTIR DE EXEMPLOS

Aline Marins Paes

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DA ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.
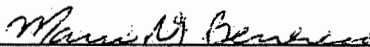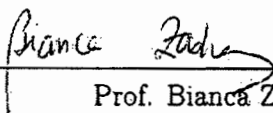
Examinada por:

_____
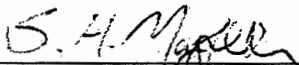Prof. Gerson Zaverucha, Ph.D.

_____
Prof. Vítor Manuel de Morais Santos Costa, Ph.D.

_____
Prof. Mário Roberto Folhadela Benevides, Ph.D.

_____
Prof. Bianca Zadrozny, Ph.D.

_____
Prof. Stephen Howard Muggleton, Ph.D.

RIO DE JANEIRO, RJ - BRASIL
SETEMBRO DE 2011

À minha querida mãe, o pilar da minha criação, e ao meu amado esposo, o pilar da minha evolução.

Agradecimentos

Agradeço primeiramente a Deus, pelo dom da vida. "Todas as coisas foram feitas por intermédio "Dele", e sem Ele nada do que foi feito se fez."(Jo 1:3).

Agradeço à minha família, particularmente:

Aos meus pais Antonio e Geiza por tudo que eles negaram e sacrificaram a si mesmos em prol de me oferecer a melhor educação dentro das suas possibilidades; por acreditarem em mim muito mais do que eu mesma poderia acreditar e me apoiarem e me darem suporte em todas as minhas decisões.

Ao meu esposo Ricardo por ser meu maior incentivador; por ouvir atentamente todas as idéias e problemas relacionados a tese e por me dar tantos conselhos altamente encorajadores; por fazer minha vida ser tão mais feliz e tão mais cheia de significado.

Aos meus irmãos Alessandra e Thiago e aos irmãos que eles me deram, Braz e Marcela, por todo carinho e incentivo.

Aos meus sobrinhos Susan e João Vítor, pela capacidade de encher minha vida de alegria.

A minha família de Niteroi, Jane, Ricado, Janete, Fernando, Arthur, "vó"Zélia e "vô"Ronald, por todo o carinho.

Agradeço aos professores que se dedicaram a me transmitir conhecimento de forma tão generosa, mas especialmente aqueles que aceitaram me orientar e guiar essa pesquisa, especificamente:

Agradeço ao meu orientador, professor Gerson Zaverucha por todo apoio, dedicação, encorajamento e exemplo de constante determinação durante o doutorado. Sua paixão com a pesquisa fez com que eu me decidisse desde o primeiro trimestre do mestrado sobre o caminho que pretendia trilhar dali em diante. Agradeço por ele sempre acreditar e incentivar esse trabalho. Agradeço também pelas oportunidades

profissionais que ele me proporcionou, ao contribuir para minhas participações em congressos nacionais e internacionais, ao me permitir co-orientar alunos de graduação e ao me incentivar a visitar o Imperial College no doutorado sanduíche.

Agradeço ao meu orientador, professor Vítor Santos Costa, por estar sempre pronto a me ajudar e elucidar dúvidas. Agradeço por desenvolver o melhor compilador Prolog de todos os tempos e por sempre resolver tão instantaneamente qualquer "bug" que surgisse em novas versões do YAP. Agradeço por converter a versão original do FORTE para YAP. Agradeço também por todas as dicas de escrita em inglês e sempre fazer magicamente com que meus textos ficasse tão elegantes. E ainda, por me receber de maneira tão confortante em sua casa, juntamente com sua esposa Inês Dutra, durante o período que passei em Portugal.

Ao meu orientador do doutorado sanduíche, professor Stephen Muggleton, por me dar a possibilidade de fazer parte do *Computational Bioinformatics Laboratory* do Imperial College London durante um ano. Sua generosidade, criatividade, paciência e profundo conhecimento científico transmitidos nos nossos encontros semanais contribuíram muito para essa pesquisa e para meu desenvolvimento como pesquisadora.

Agradeço aos meus amigos pelo incentivo, carinho e apoio durante o doutorado, especialmente:

A Kate Revoredo, por ser um grande exemplo de determinação, pela parceria, companheirismo e conselhos.

A Ana Luísa Duboc, pela parceria e encorajamento.

Aos amigos do laboratório de IA, Juliana Bernardes, Cristiano Pitangui, Carina Lopes, Elias Barenboim, Fábio Vieira, Glauber Cardoso, Aloísio Pina, pela troca de ideias e informações.

Aos amigos do *Computational Bioinformatics Laboratory*, José Santos, Ramin Ramezani, Pedro Torres, Niels Pahlavi, Jianzhong Chen, Alireza Tamaddoni-Nezhad, Hiroaki Watanabe, Robin Baumgarten e John Charnley, por me receberem tão bem no grupo e pela troca de informações.

Aos meus "co-alunos" de Iniciação Científica, Eric Couto, Rafael Pereira e Guilherme Niedu por me ajudarem a descobrir como é bom orientar trabalhos de

REVISÃO EFETIVA DE PROGRAMAS LÓGICOS (BAYESIANOS) A PARTIR DE EXEMPLOS

Aline Marins Paes

Setembro/2011

Orientadores: Gerson Zaverucha
                 Vítor Manuel de Morais Santos Costa

Programa: Engenharia de Sistemas e Computação

Revisão de Teorias a partir de exemplos visa melhorar teorias geradas automaticamente ou definidas por um especialista. Vários sistemas de revisão de teorias foram projetados com sucesso no passado, porém eles não tem sido mais vastamente utilizados. Argumentamos que a principal razão para esse fato é a ineficiência destes sistemas. Esta tese contribui na construção de sistemas de revisão de teorias praticáveis. Para tanto, utilizamos técnicas de Programação em Lógica Indutiva (ILP) e Busca Local Estocástica para reduzir o espaço de busca gerado por um sistema de revisão de teorias de pimeira-ordem e mostramos experimentalmente que ele se tornou tão eficiente quanto um sistema de ILP padrão porém gerando teorias mais acuradas. Apresentamos também uma aplicação com o jogo do xadrez que é resolvida com sucesso através de revisão de teorias, em constraste com o aprendizado, que falha em adquirir a teoria correta. Como ILP manipula bem domínios multirelacionais mas falha em representar incerteza, há um grande interesse em juntar aprendizado relacional com mecanismos de raciocínio probabilístico. Recentemente, contribuímos com um sistema de revisão de teorias probabilísticas de primeira ordem, chamado PFORTE. Apesar de apresentar resultados promissores em domínios artificiais, PFORTE enfrenta a complexidade de buscar e executar inferência probabilística em grandes espaços, fazendo com que o mesmo não seja passível de ser aplicado em domínios do mundo real. Assim, a segunda maior contribuição dessa tese é endereçar os pontos de gargalo do processo de revisão lógico probabilístico. Nós agregamos técnicas de ILP e modelos gráficos probabilísticos para reduzir o espaç de busca do processo de revisão e da inferência probabilística. O novo sistema de revisão foi aplicado com sucesso em domínios do mundo real.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.)

ON THE EFFECTIVE REVISION OF (BAYESIAN) LOGIC PROGRAMS FROM EXAMPLES

Aline Marins Paes

September/2011

Advisors: Gerson Zaverucha
                Vítor Manuel de Morais Santos Costa

Department: Systems Engineering and Computer Science

Theory Revision from Examples is the process of improving user-defined or automatically generated theories, guided by a set of examples. Despite several successful theory revision systems having been built in the past, they have not been widely used. We claim that the main reason is the inefficiency of these systems. This thesis contributes with the design of feasible theory revison systems. To do so, we introduce techniques from Inductive Logic Programming (ILP) and from Stochastic Local Search to reduce the size of each individual search space generated by a FOL theory revision system. We show through experiments that it is possible to have a revision system as efficient as a standard ILP system and still generate more accurate theories. Moreover, we present an application involving the game of Chess that is successfully solved by theory revision, in constrast with learning from scratch that fails in correctly achieving the required theory. As ILP handles well multi-relational domains but fails on representing uncertain data, there is a great recent interest in joining relational representations with probabilistic reasoning mechanisms. We have contributed with a probabilistic first-order theory revision system called PFORTE. However, despite promising results in artificial domains, PFORTE faces the complexity of searching and performing probabilistic inference over large search spaces, making it not feasible to be applied to real world domains. Thus, the second major contribution of this thesis is to address the bottlenecks of probabilistic logic revision process. We aggregate techniques from ILP and probabilistic graphical models to reduce the search space of the revision process and also of the probabilistic inference. The new probabilistic revision system was successfully applied in real world datasets.

Sumário

Lista de Símbolos

AIC *Akaike's Information Criteria*

BLP *Bayesian Logic Program*

CLP(BN) *Constraint Logic Programming with Bayes Nets*

CLL *Conditional Log-likelihood*

CPD *Conditional Probability Distribution*

DAG *Directed Acyclic Graph*

EM *Expectation-Maximization*

FN *False Negative*

FORTE *First-Order Revision of Theories from Examples*

FP *False Positive*

ICI *Independence of Causal Influence*

ICL *Independent Choice Logic*

IID *Independently and Identically Distributed*

ILS *Iterated Local Search*

ILP *Inductive Logic Programming*

LL *Log-Likelihood*

LPM *Logical Probabilistic Model*

MDIE *Mode Directed Inverse Entailment*

MDL *Minimum Description Length*

MLE *Maximum Likelihood Estimator*

MLN *Markov Logic Networks*

PCFG *Probabilistic Context-free Grammar*

PGM *Probabilistic Graphical Model*

PILP *Probabilistic Inductive Logic Programming*

PII *Probabilistic Iterative Improvement*

PLL *Probabilistic Logic Learning*

PLM *Probabilistic Logical Model*

PRM *Probabilistic Relational Models*

RII *Randomised Iterative Improvement*

RRR *Rapid Random Restarts*

SA *Simulated Annealing*

SLP *Stochastic Logic Program*

SLS *Stochastic Local Search*

SRL *Statistical Relational Learning*

TN *True Negative*

TP *True Positive*

wp *Walk Probability*

Capítulo 1

Introdução

Inteligência artificial se concentra em construir programas de computador que resolvem problemas que requerem inteligência quando resolvidos por um ser humano. Como uma entidade inteligente deve ser capaz de aprender, construir programas que podem aprender ocupa um papel central em inteligência artificial. Esta tarefa é estudada na área de aprendizado de máquina, cuja principal meta é construir programas de computador que podem melhorar seu comportamento automaticamente de acordo com sua experiência (MITCHELL, 1997). Algoritmos tradicionais de aprendizado de máquina aprendem a partir de exemplos independentes e homogêneos, descritos como pares de atributo-valor, em formato proposicional. Entretanto, em muitas aplicações do mundo real, os dados são multi-relacionados, altamente estruturados e amostrados a partir de relações complexas. Consequentemente, algoritmos proposicionais não são apropriados para aprender a partir de tais dados. Nesse caso, formalismos como lógica de primeira-ordem são mais adequados para representar tais dados do que o clássico formato proposicional.

Programação em Lógica Indutiva (ILP) (MUGGLETON, 1992), (MUGGLETON, DE RAEDT, 1994) é o processo de aprender automaticamente teorias lógicas de primeira-ordem a partir de um conjunto de exemplos e conhecimento preliminar fixo, ambos escritos como programas lógicos. ILP oferece várias vantagens. Com esse paradigma, é possível aprender a partir de uma linguagem expressiva, de fácil entendimento para os humanos. Além disso, o conhecimento preliminar é uma ferramenta útil para guiar o processo de aprendizado. Um grande número de algoritmos e sistemas tem sido desenvolvidos para aprender nesse con-

texto. Exemplos populares incluem FOIL (QUINLAN, 1990), Progol (MUGGLE-TON, 1995b), Claudien (DE RAEDT, 1997), Aleph (SRINIVASAN, 2001a), Tilde (BLOCKEEL, DE RAEDT, 1998), entre muitos outros. ILP tem sido aplicada com sucesso em uma grande quantidade de aplicações, envolvendo principalmente quimio e bio-informática (MUGGLETON, et al., 1992), (BRATKO, KING, 1994), (SRINI-VASAN, et al., 1997b), (SRINIVASAN, et al., 2006), (TAMADDONI-NEZHAD, et al., 2006), (SRINIVASAN, KING, 2008). A maioria dos sistemas de ILP aprendem uma cláusula de cada vez, empregando uma abordagem de cobertura: após aprender uma cláusula, exemplos positivos cobertos por tal cláusula são removidos do conjunto de exemplos, tal que somente exemplos positivos não cobertos permaneçam para serem explicados por novas cláusulas.

1.1 Revisão de Teorias Lógicas de Primeira-ordem a Partir de Exemplos

Sistemas de ILP consideram o conhecimento preliminar como correto. Entretanto, existem situações nas quais o conhecimento preliminar disponível está incompleto ou somente parcialmente correto. Uma teoria pode ter sido definida por um especialista do domínio, que baseia-se em suposições incorretas ou somente tem um entendimento parcial, embora útil, do domínio. Ou pode ser o caso que novos exemplos, que não podem ser explicados pela teoria corrente, se tornaram disponíveis. Ainda, pode ser o caso que uma teoria considerando um domínio específico foi aprendida ou definida por um especialista e agora seria útil transferi-la para um domínio relacionado ao anterior. Em todos esses casos, uma vez que a teoria inicial provavelmente contém informação importante, é desejável obter vantagens da mesma como um ponto de partida para o processo de aprendizado, e repara-la ou mesmo melhorá-la. Idealmente, essa abordagem deveria acelerar o processo de aprendizado e obter teorias mais acuradas.

Vários sistemas de *refinamento de teorias* tem sido propostos em direção a essa meta (SHAPIRO, 1981), (MUGGLETON, 1987), (WOGULIS, PAZZANI, 1993), (WROBEL, 1994), (ADÉ, et al., 1994), (WROBEL, 1996), (RICHARDS, MOONEY, 1995), (GARCEZ, ZAVERUCHA, 1999), (ESPOSITO, et al., 2000). Tais sistemas

assumem que a teoria inicial é aproximadamente correta. Então, somente alguns *pontos* (cláusulas e/ou literais) previnem a teoria de modelar corretamente a base de dados. Portanto, deveria ser mais efetivo buscar por tais pontos na teoria e *revisá-los*, do que usar um algoritmo que aprende uma nova teoria sem efetuar revisões.

Assim, sistemas de revisão de teoria operam buscando por *pontos de revisão*, que são cláusulas ou literais considerados como responsáveis por falhas na teoria, e então propondo*revisões* para esses pontos usando para tanto *operadores de revisão*. Dessa forma, sistemas de revisão de teorias refinam teorias inteiras ao invés de cláusulas individuais. Apesar das vantagens de refinar teorias inteiras (BRATKO, 1999), infelizmente o espaço de busca de hipóteses candidatas é usualmente muito grande.

Considere, por exemplo, o sistema FORTE (RICHARDS, MOONEY, 1995), que revisa automaticamente teorias lógicas de primeira-ordem a partir de um conjunto de exemplos positivos e negativos. Seu primeiro passo é buscar pelos pontos responsáveis pela classificação incorreta de exemplo. Após identificar todos os pontos de revisão na teoria corrente, FORTE propõe revisões para cada um desses pontos, usando um conjunto de operadores de revisão. Tais operadores incluem adicionar/excluir literais do corpo de uma cláusula e adicionar/excluir cláusulas da teoria. Finalmente, a partir do conjunto de revisões sugeridas, FORTE escolhe uma para ser implementada e reinicia o ciclo. Normalmente, cada passo do processo de revisão gera um grande espaço de busca, dependendo do número de pontos de revisão e da quantidade de possíveis revisões sugeridas para cada ponto.

Assim, apesar dos vários sistemas de revisão de teoria desenvolvidos no passado, eles não são mais vastamente usados. Conforme apontado em (DIETTERICH, et al., 2008), existem duas principais razões para isso: (1) a falta de aplicações com uma teoria inicial substancialmente codificada e (2) conforme discutido acima, o grande espaço de busca que sistemas de revisão de teoria exploram. O primeiro já não é mais um grande problema, uma vez que conhecimento preliminar em larga escala em áreas como biologia tem se tornado disponíveis (MUGGLETON, 2005). Portanto, existe uma necessidade crescente de sistemas de

revisão de teoria eficientes, de forma que as promessas de revisão de teorias possam ser cumpridas.

A presente tese contribui nessa direção, de obter um sistema de revisão de teorias relacionais efetivo, tanto no caso puramente lógico como no caso relacional probabilístico. Nós demonstramos que a busca pode ser substancialmente melhorada ao usar técnicas como limitar o espaço de busca de novos literais pela cláusula mais específica ou ao usar técnicas de busca local estocástica, com um desempenho comparável ou melhor do que sistemas que aprendem a partir de uma teoria inicial vazia. Também mostramos que técnicas de revisão de teoria podem ser aplicadas com sucesso em uma aplicação desafiante, onde um sistema que aprende do zero não consegue ter sucesso. A seguir, cada contribuição individual dessa tese é introduzida.

1.1.1 Contribuições em Revisão de Teorias de Primeira-ordem a Partir de Exemplos

Nesta tese nos focamos no sistema de revisão FORTE, uma vez que ele revisa teorias automaticamente a partir de um conjunto de exemplos positivos e negativos. FORTE executa uma busca iterativa *hill climbing* em três passos. Primeiro, ele busca pelos pontos na teoria corrente considerados como responsáveis pela classificação incorreta de algum exemplo. A seguir, ele busca por sugestões de modificações em cada ponto de revisão, considerando operadores de revisão adequados. Por último, ele escolhe qual revisão sugerida será implementada.

O tempo de execução depende fortemente do segundo passo. Ao buscar por possíveis modificações a serem implementadas na teoria, FORTE leva em consideração operadores que tentam generalizar ou especializar a teoria, conforme o ponto de revisão. Para especializar teorias, FORTE sugere simplesmente *apagar regras* ou *adicionar literais* no corpo de cláusulas existentes. Para generalizar teorias, FORTE ele pode *excluir literais* do corpo de cláusulas ou *adicionar novas regras* na teoria. [1]. Dependendo do número de pontos de revisão, FORTE pode levar uma grande quantidade de tempo para aplicar cada operador de revisão em cada ponto de revisão. Por outro lado, pode ser que o especialista do domínio já tenha alguma ideia ou alguma

---

[1]FORTE tem outros operadores que nós não consideramos nessa tese, a saber os operadores de absorção e identificação, baseados em resolução inversa

restrição de como a teoria poderia ser modificada. Por exemplo, ele/ela poderia querer uma nova teoria que preservasse todas as cláusulas originais e portanto *apagar regras* não seria permitido. Considerando essas questões, o sistema FORTE foi modificado nesta tese tal que seja possível restringir os operadores de revisão a serem aplicados na teoria. Primeiro, nós observamos que é possível que um operador mais simples (por exemplo, *apagar regras* em comparação com *adicionar antecedentes*) já alcançou a meta da revisão e portanto não seria necessário propor modificações mais complexas no ponto de revisão. Segundo, nós permitimos que o usuário defina de antemão o conjunto de operadores de revisão que poderão ser aplicados aos pontos de revisão. Assim, a primeira contribuição dessa tese é a redução da quantidade de possíveis operadores a serem aplicados aos pontos de revisão,.

Observamos também que FORTE gasta uma grande quantidade de tempo escolhendo literais a serem adicionados a cláusulas, a saber quando aplicando os operadores de adição de antecedentes e adição de regras. Isso é devido principalmente a abordagem *top-down* empregada no FORTE e que originou-se no sistema FOIL (QUINLAN, 1990). Nesta abordagem, literais são gerados a partir da base de conhecimento, sem levar em consideração o conjunto de exemplos. A base de dados é utilizada somente para pontuar as hipóteses geradas obtidas a partir da adição de cada literal candidato. Por outro lado, algoritmos padrão de ILP tais como Progol (MUGGLETON, 1995b) e Aleph (SRINIVASAN, 2001a) utilizam uma abordagem híbrida *top-down* e *bottom-up* para refinar cláusulas. Primeiro, eles criam a *cláusula mais específica*, chamada daqui em diante de*Bottom Clause*, a partir de um único exemplo positivo, seguindo a abordagem de *implicação inversa direcionada a modos (MDIE)* (MUGGLETON, 1995b). Próximo, essa Bottom Clause variabilizada define o espaço de busca por possíveis literais a serem adicionados na cláusula. Com isso, a quantidade de literais é grandemente reduzida, quando comparada com a quantidade de todos os possíveis literais gerados a partir da cláusula corrente. Assim, a segunda contribuição dessa tese é substituir a geração de literais oriunda do FOIL pelo uso da Bottom Clause. Conjuntos de literais subsumindo a Bottom Clause são pontuados e o melhor é escolhido para ser adicionado a cláusula.

Para mostrar o poder do sistema de revisão, projetamos uma aplicação desafi-

ante e que se aplica muito bem ao processo de revisão. O objetivo da aplicação é adquirir regras de variantes do jogo de xadrez, tendo como ponto de partida as regras do xadrez internacional. A motivação para o desenvolvimento de tal aplicação é que ao longo dos anos, o jogo do xadrez tem inspirado a criação de várias variantes, ou para serem mais desafiantes para o jogador, ou para produzir uma versão mais rápida e/ou mais fácil (PRITCHARD, 2007). Adicionalmente, existem várias variantes regionais do jogo do xadrez, como as versões japonesa e chinesa. Idealmente, se as regras do xadrez foram obtidas previamente, seria útil usá-las como ponto de partida para obter as regras de variantes. Entretanto, tais regras precisam ser não-trivialmente modificadas para representar os aspectos em particular de cada variante. Em um jogo como o xadrez, essa tarefa envolve manipular bordas de diferentes tamanhos, introduzir ou excluir novas peças e/ou novas regras e ainda redefinir o papel de peças específicas no jogo. Para tratar este problema do ponto de vista de revisão de teoria, foi necessário desenvolver abdução (FLACH, KAKAS, 2000a) e negação por falha (CLARK, 1978) no sistema YAVFORTE.

Apesar de ser possível melhorar o tempo do processo de revisão com as modificações propostas no penúltimo parágrafo, sem prejudicar a acurácia do sistema, o desempenho do sistema de revisão ainda deve ser reduzido, principalmente quando ele manipula teorias e bases de dados grandes. Ao buscar por pontos de revisão, o sistema primeiro seleciona os exemplos por uma grande quantidade de cláusulas e/ou existam vários exemplos classificados incorretamente, a busca pelos pontos de revisão pode sair muito cara. Além disso, se após selecionar os pontos de revisão existirem várias cláusulas a serem revisadas, várias modificações serão propostas para as mesmas. A situação ainda pode ficar pior se o conhecimento preliminar e a linguagem do domínio, definindo predicados e declarações de modo para os mesmos, é grande o suficiente para produzir uma enorme Bottom Clause. Em todas essas situações, as modificações descritas anteriormente podem não ser suficientes para conduzir a um tempo de revisão aceitável. Para tornar o sistema tratável nesse caso, propomos abandonar a completude do sistema de buscar e alterar todos os pontos de revisão com todos os operadores possíveis em favor de encontrar uma boa solução em um tempo aceitável. Para tanto, fazemos uso de técnicas de *busca local*

estocástica (HOOS, STÜTZLE, 2005). Componentes estocásticos são introduzidos nas buscas chave do processo de revisão, a saber: (1) na busca por pontos de revisão; (2) na busca pelos literais a serem sugeridos pelos operadores para entrarem ou saírem de cláusulas; (3) na busca da operação a ser escolhida para modificar a teoria. Os componentes estocásticos são incluídos como parte do sistema YAV-FORTE, de forma que os mesmos possam ser habilitados ou desabilitados para a execução do processo de revisão.

Em suma, a contribuição em lógica de primeira-ordem de um ponto de vista macro é a criação do sistema YAVFORTE, mais eficiente que o sistema FORTE ao reduzir os vários espaços de busca produzidos pelo processo de revisão. YAVFORTE inclui uma aplicação mais flexível dos operadores de revisão, o uso da Bottom Clause e de declarações de modos para limitar o espaço de busca de literais, abdução e negação por falha, e componentes estocásticos nos processos chave da revisão. O objetivo principal é tornar o processo de revisão tão eficiente quanto um sistema de ILP padrão, mas obtendo teorias finais mais acuradas.

1.2 Aprendizado Lógico Probabilístico

A aplicação com sucesso de sistemas clássicos de ILP a domínios relacionais é frequentemente limitada pela necessidade de representar incerteza e informação parcialmente observada. Embora técnicas tradicionais de aprendizado de máquina estatístico tenham a habilidade de manipular incerteza, eles não podem representar domínios relacionais, uma vez que eles são essencialmente proposicionais.

Assim, recentemente existe um grande interesse na integração de representação de informação relacional com mecanismos de raciocínio probabilístico. Construir linguagens e algoritmos para aprender hipóteses nessas linguagens é o principal assunto de uma nova área da Inteligência Artificial chamada de *Aprendizado Lógico Probabilístico* (PLL), também conhecida como Programação em Lógica Indutiva Probabilística (PILP) e fortemente relacionada com Aprendizado Estatístico Relacional (SRL). PLL lida com aprendizado de máquina em domínios relacionais onde pode existir informação perdida, somente parcialmente observada, com ruído ou incerta. Vários formalismos têm sido desenvolvidos nesta área nas últimas décadas,

incluindo SLP (*Stochastic Logic Programs*) (MUGGLETON, 1996), PRM (*Probabilistic Relational Models*) (KOLLER, PFEFFER, 1998), BLP (*Bayesian Logic Programs*) (KERSTING, DE RAEDT, 2001b), CLP(BN) (*Constraint Logic Programming with Bayes net*) (SANTOS COSTA, et al., 2003a), MLN (*Markov Logic Networks*)

(RICHARDSON, DOMINGOS, 2006), ProbLog (DE RAEDT, et al., 2007), entre outros. Na maioria de tais sistemas, o conhecimento é representado por cláusulas definidas anotadas com distribuições de probabilidade. Inferência pode ser executada ou usando um mecanismo lógico que leva em consideração as probabilidades ou através de um modelo gráfico probabilístico, construído a partir dos exemplos e da hipótese corrente. Este é, por exemplo, o caso dos BLPs, que generalizam ambos programas lógicos e redes Bayesianas. Inferência é executada sobre redes Bayesianas, construídas a partir de cada exemplo e cláusulas no BLP, usando a abordagem de Construção de Modelos Baseado no Conhecimento (NGO, HADDAWY, 1997).

A maioria dos algoritmos desenvolvidos em PLL assumem que ou as hipóteses são inicializadas vazias ou então elas são inicializadas com um conhecimento preliminar, mas as regras que definem esse conhecimento preliminar podem ser todas modificadas, considerando que todas elas estão no mesmo nível de corretude. Este último caso é a abordagem empregada pelo algoritmo de aprendizado de BLP. Existe menos trabalho envolvendo a tarefa de *reparar* ou *melhorar* uma hipótese lógica-probabilística inicial.

1.2.1 BFORTE: em Direção a um Sistema de Revisão Efetivo

Como a tarefa de aprendizado consome tempo e a hipótese pode conter informação valiosa, a mesma poderia ser considerada como ponto de partida e então ela seria refinada. Idealmente, isto resultaria em um tempo de aprendizado mais rápido e hipóteses mais acuradas. Motivados pelos benefícios trazidos pela técnica de revisão de teorias de primeira-ordem, nós propomos recentemente revisar automaticamente BLPs (REVOREDO, ZAVERUCHA, 2002), (PAES, et al., 2005b), (PAES, et al., 2005a), (PAES, et al., 2006a), o que produziu um sistema chamado de PFORTE.

Aprender ou revisar hipóteses probabilísticas de primeira-ordem introduz as-

pectos interessantes. Em PLL é conveniente ver exemplos como evidências para variáveis aleatórias. As distribuições de probabilidade darão uma estimativa se um exemplo deve ser associado a um valor específico, digamos, um valor verdadeiro ou falso. Seguindo uma abordagem discriminativa, uma teoria deveria ser revisada se ela falha em gerar a distribuição de probabilidade apropriada para um exemplo. Neste caso, dizemos que o exemplo está incorretamente classificado. A estratégia empregada pelo sistema PFORTE é como segue. Primeiro, ele aborda incompletudes na teoria, descobrindo pontos que falham em provar exemplos. PFORTE usa operadores de generalização para propor modificações para aqueles pontos, escolhendo o melhor para corrigir. Essa fase usa uma busca *hill climbing* e para quando não é mais possível encontrar uma revisão que melhore a cobertura de exemplos. Segundo, PFORTE usa a teoria generalizada como ponto de partida para compor o espaço de generalizações e especializações nos pontos da teoria que falharam em produzir a distribuição de probabilidade correta. Nesse caso, cláusulas participando da rede Bayesiana gerada a partir de um exemplo classificado incorretamente são marcadas como ponto de revisão. Após esse passo, espera-se que o sistema retorne um BLP mais acurado.

Embora nós tenhamos demonstrado que é possível obter BLPs mais acurados com PFORTE comparando com o aprendizado do zero, o tempo que esse sistema gasta para executar o processo completo de revisão é proibitivo. Nós identificamos pelo menos três gargalos responsáveis pelo alto tempo de execução do processo de revisão do PFORTE:

- Escolha dos pontos de revisão. O algoritmo de aprendizado de BLPs inicia a partir de uma teoria mais geral, porém exigindo que todas as variáveis que aparecem na cabeça da cláusula também apareçam no corpo, e então tenta modificar cada cláusula dessa teoria para melhorar a verossimilhança dos dados. PFORTE, por outro lado, propõe modificações apenas nas cláusulas usadas para construir redes Bayesianas de exemplos classificados incorretamente. Embora PFORTE reduza o número de cláusulas sujeitas a modificações comparado com o aprendizado de BLPs, PFORTE ainda seleciona várias cláusulas para serem revisadas, uma vez que a dependência existente entre os fatos do

exemplo faz com que várias cláusulas diferentes sejam parte da rede Bayesiana. Muitas delas não são relevantes para a classificação de um nó em particular.

- Tempo de inferência. Redes Bayesianas construídas a partir de exemplos relacionais devem capturar os relacionamentos entre diferentes objetos do domínio. Esse fato, adicionado a inerente complexidade da inferência em redes Bayesianas torna o cálculo das distribuições de probabilidade posteriores muito custoso. Adicionalmente, como assumimos uma abordagem discriminativa, toda possível modificação sugerida para o BLP corrente é pontuado usando uma função de avaliação que deve inferir a distribuição de probabilidade de cada variável de consulta. Por causa desses problemas, o tempo gasto em inferências acaba por dominar o processo de revisão.

- Geração de literais a serem adicionados na cláusula. Similar ao FORTE, PFORTE também considera adição de literais em cláusulas em dois operadores de revisão, a saber adição de antecedentes e adição de regras. PFORTE usa a abordagem top-down do FORTE para criar novos literais. Como na revisão de primeira-ordem, tal abordagem produz um grande número de novos literais a serem pontuados para entrarem ou não na cláusula sendo revisada.

Nós propomos várias contribuições com o objetivo de tornar o processo de revisão de BLPs praticável. Primeiro, observamos que o número de pontos de revisão pode ser reduzido, já que nem todas as variáveis aleatórias na rede Bayesiana de um exemplo classificado incorretamente está de fato influenciando a distribuição de probabilidade de um exemplo. O ideal seria então identificar as variáveis relevantes para a classificação incorreta, tal que somente as cláusulas relativas a tais variáveis sejam candidatas a serem modificadas. Com essa meta, empregamos o conceito de d-separação (GEIGER, et al., 1990) ao usar o algoritmo Bayes Ball (SHACHTER, 1998) para identificar variáveis relevantes a variável que representa o exemplo. Bayes Ball é um algoritmo de tempo linear que identifica o conjunto de nós relevantes para uma variável de consulta, dada evidência em algumas outras variáveis. Dessa forma, as variáveis aleatórias consideradas relevantes são selecionadas e as cláusulas que deram origem as mesmas na rede Bayesiana são marcadas como pontos de revisão.

Segundo, o tempo de inferência também pode ser reduzido se forem levados em consideração somente os nós indispensáveis para computar a distribuição de probabilidade da variável de consulta, ao invés de considerar a rede inteira. Com isso, o tempo de inferência pode ser bastante reduzido, uma vez que, normalmente, redes Bayesianas construídas a partir de exemplos relacionais contem muitos nós. Nós considerados como indispensáveis para uma consulta são coletados pelo algoritmo Bayes Ball. Além disso, o procedimento de construção de redes Bayesianas de BLPs coleta todas as provas de cada instância para construir a rede Bayesiana relativa a um exemplo. Várias cláusulas no conjunto de provas podem compartilhar as mesmas características: além de conter os mesmos predicados, diferentes variáveis aleatórias que se originaram das provas também podem possuir os mesmos valores de evidência ou a falta de evidência. Assim, nós propomos agrupar variáveis aleatórias referentes a mesma cláusula e que possuem os mesmos valores de evidência em apenas um nó da rede. Após separar apenas os nós indispensáveis e agrupar variáveis com características idênticas, pode acontecer que redes referentes a variáveis de consulta diferentes tenham um comportamento idêntico. Por comportamento idêntico, queremos dizer que os nós são originados das mesmas cláusulas Bayesianas, as evidências são as mesmas e as matrizes que representam o grafo qualitativo da rede Bayesiana também são idênticas. Nesse caso, essas redes são agrupadas em apenas uma, tal que a computação da distribuição de probabilidade das diferentes variáveis de consulta seja feita apenas uma vez.

Terceiro, e motivados pela grande redução no tempo de execução ao usar a Bottom Clause na revisão de teorias de primeira-ordem, propomos limitar o espaço de novos literais dos operadores de adição de antecedentes e de regras com a Bottom Clause. Nesse caso, a Bottom Clause é construída a partir de um exemplo classificado incorretamente, independente de sua classe.

Em resumo, de forma geral, a contribuição dessa tese na área de revisão (e aprendizado) de modelos lógicos probabilísticos, é a criação de um sistema chamado de *BFORTE*. O objetivo é o desenvolvimento de um sistema de revisão de BLPs que seja praticável e que consiga obter resultados melhores que o aprendizado/revisão puramente lógico.

1.3 Publicações

As seguintes publicações surgiram a partir do trabalho conduzido durante o desenvolvimento da tese:

- "*Using the Bottom Clause and Mode Declarations in FOL Theory Revision from Examples*", publicado no *Machine Learning Journal* (2009) (DUBOC, et al., 2009). Uma versão não expandida desse trabalho foi apresentada no *18th International Conference on Inductive Logic Programming* (ILP-2008) (DUBOC, et al., 2008). As principais ideias e algoritmos de tal artigo estão apresentadas como parte do capítulo 3 e apêndice B.

- "*Chess Revision: Acquiring the Rules of Chess through Theory Revision from Examples*", apresentado por mim no *19th International Conference on Inductive Logic Programming* (ILP-2009) (MUGGLETON, et al., 2009b). Uma versão preliminar do mesmo trabalho também foi apresentada por mim no *Workshop on General Game Playing / 21st International Joint Conference on Artificial Intelligence (IJCAI-09)* (MUGGLETON, et al., 2009a). Uma versão mais detalhada desses artigos é apresentada no capítulo 4 e apêndice C.

- "*ILP through Propositionalization and Stochastic k-term DNF Learning*" (PAES, et al., 2006b), que eu apresentei no *16th International Conference on Inductive Logic Programming* (ILP 2006) e que me motivou a iniciar os estudos em algoritmos de Busca Local Estocástica. Esse artigo emprega busca local estocástica para encontrar uma hipótese após proposicionalizar o conjunto de exemplos. As principais ideias do artigo estão na última seção do apêndice D.

- "*Revising First-order Logic Theories from Examples through Stochastic Local Search*" (PAES, et al., 2007b), que eu apresentei no *17th International Conference on Inductive Logic Programming* (ILP-2007). Uma versão preliminar em português desse artigo ganhou o prêmio de melhor artigo do VI Encontro Nacional de Inteligência Artificial (PAES, et al., 2007a). O apêndice E é uma

extensão significativa desses artigos, pois agora usamos a Bottom Clause para limitar o espaço de novos literais e incluímos três componentes estocásticos a mais em diferentes passos do processo de revisão.

- "*PFORTE: Revising Probabilistic FOL Theories*" (PAES, et al., 2006a), apresentado por mim no *2nd International Joint Conference (10th Ibero-American Conference on AI, 18th Brazilian AI Symposium)*, 2006. Esse artigo estende o sistema PFORTE introduzido em (PAES, et al., 2005b), ao incluir operadores de generalização no segundo passo do processo de revisão. O algoritmo juntamente com as principais ideias do sistema são revistos na última seção do apêndice F.

Durante a pesquisa para o desenvolvimento da presente tese, eu também fui co-autora dos seguintes artigos:

- "*Combining Predicate Invention and Revision of Probabilistic FOL theories*" (REVOREDO, et al., 2006), apresentado por mim no *16th International Conference on Inductive Logic Programming* (ILP-2006) e publicado como um artigo curto nos anais preliminares do congresso. Esse artigo introduz dois operadores de revisão baseados em invenção de predicados. Devido ao alto custo desses operadores, eles não foram considerados nos experimentos dessa tese. Uma versão estendida desse artigo foi publicada em português no VI Encontro Nacional de Inteligência Artificial (ENIA-2006).

- "*On the Relationship between PRISM and CLP(BN)*" (SANTOS COSTA, PAES, 2009), apresentado como poster no *International Workshop on Statistical Relational Learning* (SIM-2009).

- "Revisando Redes Bayesianas através da Introdução de Variáveis Não-observadas" (REVOREDO, et al., 2009), apresentado no VII Encontro Nacional de Inteligência Artificial(ENIA-2009).

1.4   Organização da Tese

A tese está organizada como segue.

O capítulo 2 revisa conceitos preliminares necessários ao entendimento da tese. Assim, nele inserimos uma introdução a Programação em Lógica Indutiva e Revisão de Teorias. O desenvolvimento dos conceitos relativos a esse assunto é feito no apêndice A, onde são apresentados conceitos básicos de ILP, incluindo o processo de construção da Bottom Clause. Conceitos chave de revisão de teorias e uma explicação detalhada do sistema FORTE também são apresentados nesse apêndice. Nesse mesmo capítulo, revisamos conceitos básicos de Busca Local Estocástica. Então, no apêndice D é feita uma revisão cuidadosa de algoritmos de busca local estocástica. Finalmente, a última seção desse capítulo introduz aprendizado lógico probabilístico e então remetemos ao apêndice F para explicações detalhadas dos principais conceitos envolvidos nesse tipo de aprendizado. Para tanto, revemos conceitos essenciais de redes Bayesianas, Programas Lógicos Bayesianos e nosso sistema de revisão PFORTE como publicado em (PAES, et al., 2005b; PAES, et al., 2006a).

O capítulo 3 introduz as contribuições da tese em revisão de teorias lógicas de primeira-ordem a partir de exemplos. Assim, temos a primeira seção introduzindo o sistema YAVFORTE, que inclui o uso da Bottom Clause e declarações de modo e redução da quantidade de operadores de revisão empregados no processo de revisão. O apêndice B traz o detalhamento dessa contribuição e introduz o sistema YAV-FORTE. Parte desse apêndice foi publicada em (DUBOC, et al., 2008; DUBOC, et al., 2009). A próxima seção traz uma introdução ao problema da revisão do xadrez tradicional para obter regras de variantes do xadrez. Então, no apêndice C apresentamos o *framework* desenvolvido para atingir a meta da revisão das regras do xadrez. Abdução e negação por falha são incluídas no sistema YAVFORTE para lidar melhor com a teoria do xadrez. Este trabalho está publicado em (MUGGLE-TON, et al., 2009b), (MUGGLETON, et al., 2009a), (MUGGLETON, et al., 2009c). Finalmente, introduzimos a última contribuição dessa tese no que diz respeito a revisão de teorias de primeira-ordem, a saber a inclusão de componentes estocásticos no processo de revisão. O apêndice E traz o detalhamento dos componentes estocásticos introduzidos em cada passo chave do processo de revisão: a definição dos pontos de revisão; dos literais a serem adicionados/removidos das cláusulas ao serem sugeridas modificações nos pontos de revisão e finalmente na busca pelo operador

a ser utilizado para implementar a revisão final. Parte deste trabalho foi publicado em (PAES, et al., 2007a; PAES, et al., 2007b).

O capítulo 4 introduz o sistema BFORTE. O apêndice G detalha cada modificação introduzida no processo de revisão para torná-lo praticável, a saber, o uso da Bottom Clause, a redução do espaço de inferência e a redução do espaço de cláusulas a serem modificadas.

Finalmente a tese é concluída no capítulo 5 e também apresentamos direções futuras de pesquisa baseadas no trabalho que desenvolvemos na presente tese.

A versão completa da tese em inglês em formato contínuo (apêndices referentes a capítulos nos seus lugares apropriados) encontra-se em `www.cos.ufrj.br/`
`~ampaes/thesis_ampc.pdf`. Sistemas desenvolvidos nessa tese e bases dados usadas para gerar experimentos com eles estarão disponíveis em `www.cos.ufrj.br/`
`~ampaes/theory_revision`.

Capítulo 2

Conceitos preliminares

Nesse capítulo introduzimos os conceitos básicos necessários ao entendimento da tese.

2.1 Programação em Lógica Indutiva e Revisão de Teorias

Aprendizado de máquina é uma sub-área da Inteligência Artificial que estuda sistemas que melhoram seu comportamento com experiência ao longo do tempo (MITCHELL, 1997). Algoritmos de aprendizado de máquina tradicionais são proposicionais, buscando por padrões a partir de dados de tamanho fixo e representados como pares de atributos-valor. Assim, tais algoritmos assumem que objetos do domínio são homogêneos e amostrados a partir de relações simples. Entretanto, dados do mundo real contem diferentes tipos de objetos e múltiplas entidades, dispostos tipicamente em várias tabelas relacionadas.

Programação em Lógica Indutiva (ILP) (MUGGLETON, 1992), (LAVRAC, DZEROSKI, 1994), (MUGGLETON, DE RAEDT, 1994), (NIENHUYS-CHENG, De Wolf, 1997), (DE RAEDT, 2008), também conhecida como *Mineração de Dados Multi-relacional* (DZEROSKI, LAVRAC, 2001), combina aprendizado de máquina e programação lógica para automaticamente induzir hipóteses escritas em lógica de primeira-ordem a partir de dados multi-relacionados. Sistemas de ILP tem sido experimentalmente testados em uma grande quantidade de aplicações importantes (KING, et al., 1995b), (SRINIVASAN, et al., 1997a), (MUGGLETON, 1999), (FANG, et al., 2001), (KING, et al., 2004). Teorias aprendidas em ILP representam

elegantemente e de forma expressiva situações complexas, envolvendo um número variável de entidades e de relacionamentos entre elas. Tais teorias são geralmente formadas com o objetivo de discriminar exemplos positivos de exemplos negativos, dado conhecimento preliminar. O conhecimento preliminar consiste de fatos lógicos e/ou regras de inferência.

Algoritmos de ILP assumem que o conhecimento preliminar é fixo e correto. Entretanto, existem situações como a chegada de novos exemplos ou a definição incorreta do conhecimento preliminar que fazem com que este último contenha regras incorretas. É então necessário modificá-las para que elas se tornem corretas. Essa é a tarefa de *Revisão de teorias*, que tem como meta consertas as regras responsáveis pela classificação incorreta de algum exemplo. No apêndice referente a esse capítulo, revisamos as principais técnicas usadas em ILP e os conceitos chave de revisão de teoria. Atenção especial é dada ao sistema FORTE (RICHARDS, MOONEY, 1995), uma vez que todos os sistemas desenvolvidos nessa tese são baseados nele. O leitor familiar com ILP e revisão de teorias pode desconsiderar o apêndice A, que contém esse capítulo de forma completa.

2.2   Busca Local Estocástica

Buscar em grandes espaços é um problema recorrente em Ciência da Computação. Para conseguir boas hipóteses enquanto ainda mantendo a busca praticável, pode-se tirar vantagem de algoritmos de busca local, que iniciam gerando hipóteses em algum ponto do espaço de busca e depois disso, s movem do ponto corrente para um ponto na vizinhança. Cada ponto tem um número relativamente pequeno de vizinhos e cada movimento é determinado por uma decisão baseada em conhecimento local (HOOS, STÜTZLE, 2005). Dessa forma, algoritmos de busca local abandonam a completude para ganhar eficiência.

É possível melhorar ainda mais a eficiência e também escapar de ótimos locais ao tomar decisões randomizadas quando gerando ou selecionando candidatos no espaço de busca de um problema, através de *Algoritmos de Busca Local Estocástica*(SLS). Uma aplicação de sucesso e que tem servido de motivação para outros trabalhos é na checagem de satisfatibilidade de fórmulas proposicionais, através

do algoritmos bem conhecidos GSAT (SELMAN, et al., 1992) e WalkSAT (SELMAN, et al., 1996). Um grande número de tarefas em áreas como planejamento, escalonamento e satisfação de restrições podem ser codificadas como um problema de satisfatibilidade. Observações empíricas mostram que SLS podem frequentemente melhorar a eficiência em tais tarefas (CHISHOLM, TADEPALLI, 2002; RÜCKERT, KRAMER, 2003).

Randomização também pode ser usada para melhorar outras estratégias de busca, como a busca *backtracking*. Por exemplo, técnicas de *Reinícios Aleatórios Rápidos* (RRR) introduzem um elemento estocástico em busca no estilo *backtrack* para reiniciar a busca a partir de outro ponto se nenhum progresso estiver sendo obtido (GOMES, et al., 1998; GOMES, et al., 2000). Assim, também existe um grande interesse em aplicar técnicas de SLS em mineração de dados, e mais especificamente em mineração de dados multi-relacional. Trabalhos prévios na área de fato mostram resultados promissores de técnicas estocásticas quando aprendendo teorias a partir do zero em sistemas de ILP (PAES, et al., 2006b; ŽELEZNÝ, et al., 2006). Para dois algoritmos bem diferentes, resultados mostraram melhoras substanciais em eficiência, com pouco ou nenhum custo em acurácia.

Motivados por esses trabalhos, contribuímos nessa tese com o desenvolvimento de métodos de busca local estocástica para revisar teorias lógicas de primeira-ordem. Então, para fundamentar nossa abordagem, apresentamos no apêndice D uma visão global de algoritmos de busca local estocástica e de métodos estocásticos empregados em ILP.

2.3 Aprendizado Lógico Probabilístico

Algoritmos estatísticos de aprendizado de máquina tradicionais lidam com incerteza mas assumem que os dados são independentes e identicamente distribuídos (i.i.d). Por outro lado, algoritmos relacionais de aprendizado tem a capacidade de representar objetos relacionados com múltiplos tipos, mas eles impõem severas limitações para representar e raciocinar na presença de incerteza. Entretanto, na maioria das aplicações do mundo real, dados são multi-relacionais, heterogêneos, incertos, parcialmente observados e com ruído. Exemplos incluem dados da web, dados

bibliográficos, análises de redes sociais, dados químicos e biológicos, mapeamento de robôs, linguagem natural, entre outros (LACHICHE, FLACH, 2002), (BATTLE, et al., 2004), (GETOOR, et al., 2004), (JAIMOVICH, et al., 2005), (DAVIS, et al., 2005a), (NEVILLE, et al., 2005), (WANG, DOMINGOS, 2008), (RAGHAVAN, et al., 2010). Portanto, para extrair toda informação útil de bases de dados desse tipo, é necessário usar técnicas que lidam com representações multi-relacionais e raciocínio probabilístico. *Aprendizado Lógico Probabilístico* (PLL), também conhecido como *Programação em Lógica Indutiva Probabilística* (PILP) (DE RAEDT, et al., 2008a) e *Aprendizado Estatístico Relacional*(SRL) (GETOOR, TASKAR, 2007) é uma área emergente da Inteligência Artificial, que está na interseção de raciocínio com incerteza, aprendizado de máquina e representação de conhecimento relacional (DE RAEDT, 2008). Como tal, PLL é capaz de lidar com aprendizado de máquina e mineração de dados em domínios complexos, onde a informação pode estar perdida, parcialmente observada e/ou com ruído.

Um grande número de sistemas de PLL tem sido propostos nos últimos anos, dando origem a vários formalismos para representar conhecimento lógico e probabilístico. Os formalismos podem ser divididos em várias classes (GETOOR, 2007), mas aqui optamos por colocá-los nos dois eixos mais relevantes (DE RAEDT, et al., 2008a): os *Modelos Probabilísticos Lógicos* são extensões de modelos probabilísticos que são capazes de lidar com objetos e relações ao incluir elementos lógicos ou relacionais. Tipicamente, eles constroem um modelo gráfico probabilístico direcionado ou não, usando a lógica como modelo. Dentro dessa categoria existe um grupo de formalismos baseados em modelos direcionados, tais como *Relational Bayesian Networks* (JAEGER, 1997), *Probabilistic Logic Programs* (NGO, HADDAWY, 1997; HADDAWAY, 1999), *Probabilistic Relational Models* (PRM) (KOLLER, PFEFFER, 1998), (KOLLER, 1999), (FRIEDMAN, et al., 1999), (GETOOR, et al., 2001), *Bayesian Logic Programs* (KERSTING, DE RAEDT, 2001d), (KERSTING, DE RAEDT, 2001b), (KERSTING, DE RAEDT, 2001a), (KERSTING, DE RAEDT, 2007), *Constraint Logic Programming for Probabilistic Knowledge* (SANTOS COSTA, et al., 2003a), *Hierarchical Bayesian Networks* (GYFTODIMOS, FLACH, 2004), *Logical Bayesian Networks* (FIERENS, et al., 2005), *Probabilistic Relational Lan-*

guage (GETOOR, GRANT, 2006), etc, e um grupo composto de sistemas baseados em modelos não direcionados, tais como *Relational Markov Networks* (TASKAR, et al., 2002), *Relational Dependency Networks* (NEVILLE, JENSEN, 2004) e *Markov Logic Networks* (SINGLA, DOMINGOS, 2005), (KOK, DOMINGOS, 2005), (RICHARDSON, DOMINGOS, 2006), (DOMINGOS, LOWD, 2009).

No outro eixo estão os *Modelos Lógicos Probabilísticos*, que são formalismos estendendo programas lógicos com probabilidades, mas permanecendo o mais próximo possível da programação lógica ao anotar cláusulas com rótulos de probabilidade. Nessa classe, a inferência lógica é modificada para lidar com os parâmetros de probabilidade. Formalismos nessa abordagem incluem *Probabilistic Horn Abduction* (POOLE, 1993) e sua extensão *Independent Choice Logic* (POOLE, 1997), *Stochastic Logic Programs* (MUGGLETON, 1996), (MUGGLETON, 2000), (MUGGLETON, 2002), PRISM (SATO, KAMEYA, 1997; SATO, KAMEYA, 2001), SAYU (DAVIS, et al., 2005a; DAVIS, et al., 2005b; DAVIS, et al., 2007), nFoil (LANDWEHR, et al., 2007), kFoil (LANDWEHR, et al., 2006), ProbLog (DE RAEDT, et al., 2007), (KIMMIG, et al., 2008), (KIMMIG, 2010), entre outros.

Na presente tese contribuímos com um algoritmo para revisar de forma prática um modelo que generaliza de forma elegante programas lógicos e redes Bayesianas, a saber, Programas Lógicos Bayesianos (BLP). No apêndice F, revisamos os conceitos necessários para entender BLPs e revisamos também nosso sistema de revisão PFORTE.

Capítulo 3

Contribuições em Revisão de Teorias Lógicas de Primeira-Ordem a Partir de Exemplos

Nesse capítulo introduzimos as contribuições da tese referentes a revisão de teorias de primeira-ordem a partir de exemplos. Cada seção introduz uma contribuição distinta, bem como um resumo dos resultados obtidos com as mesmas e conclusões a respeito dos resultados.

3.1 YAVFORTE: Um Sistema Praticável de Revisão de Teorias a Partir de Exemplos

Sistemas de revisão de teorias a partir de exemplos induzem hipóteses mais acuradas que sistemas de ILP que aprendem a partir do zero. Entretanto, obter tais teorias mais acuradas vem as custas de buscar em grandes espaços de hipóteses, principalmente porque hipóteses em revisão de teorias são teorias inteiras ao invés de cláusulas individuais, o que é conhecido como um problema difícil (WROBEL, 1996; BRATKO, 1999). Portanto, é essencial desenvolver sistemas de revisão de teorias eficientes, tal que as vantagens deles se tornem evidentes. A partir do sistema FORTE, neste capítulo contribuímos em direção a meta de identificar os pontos de gargalo do processo de revisão e desenvolver algoritmos baseados no estado da arte de algoritmos de ILP para superá-los.

O pior gargalo do processo de revisão do sistema FORTE está relacionado com

a geração de novos literais a serem pontuados, quando uma cláusula está sendo especializada nos operadores de adição de antecedentes e de adição de regras. FORTE segue a abordagem a abordagem *top-down* do sistema FOIL (QUINLAN, 1990), considerando todos os literais criados a partir da base de conhecimento para refinar cláusulas, o que leva a um espaço de busca gigantesco e termina por dominar o processo de revisão. Ao invés de usar uma abordagem *top-down* ao especializar cláusulas, sistemas de ILP como Progol (MUGGLETON, 1995a) e Aleph (SRINI-VASAN, 2001a) restringem o espaço de novos literais àqueles pertencendo a *Bottom Clause*. A Bottom Clause contem os literais relevantes a um exemplo positivo, coletados a partir da técnica de Resolução Inversa Direcionada a Modos (MDIE) (MUG-GLETON, 1995b). Essa abordagem *bottom-up* e *top-down* híbrida frequentemente gera muito menos literais, e também garante a cobertura de ao menos um exemplo positivo (aquele usado para gerar a Bottom Clause). Como parte da pesquisa desenvolvida nessa tese, em (DUBOC, 2008) e (DUBOC, et al., 2009) a Bottom Clause foi introduzida para limitar o espaço de busca de literais no sistema FORTE. No apêndice B referente a esse capítulo, descrevemos os algoritmos usados para usar a Bottom Clause nos operadores citados acima. Adicionalmente, mostramos como melhorar o processo desenvolvido em (DUBOC, et al., 2009) ao (1) fazer com que a construção da Bottom Clause comece a partir da cláusula base sendo revisada; e (2) usar a teoria corrente para provar literais, ao invés de considerar apenas o conhecimento preliminar. E ainda, nesssa tese usamos as declarações de modo para decidir se um literal pode ser excluído do corpo da cláusula.

FORTE tem seis diferentes operadores de revisão e usualmente todos eles são usados diversas vezes em uma única iteração para sugerir modificações nos pontos de revisão. Assim, o espaço formado pelos operadores de especialização e generalização pode se tornar grande o suficiente para que a busca seja muito cara. Entretanto, em muitos problemas, o especialista do domínio tem alguma ideia sobre os tipos de modificações que devem ser feitos na teoria para que a mesma seja consertada ou melhorada. Esse fato pode ser usado de forma vantajosa para reduzir o espaço de busca de operadores.

No apêndice B, que detalha esse capítulo, descrevemos uma séria de mod-

ificações implementadas no sistema FORTE para resolver as deficiências listadas acima. O sistema resultante é chamado de YAVFORTE (*Yet Another Version of FORTE*) e inclui *FORTE_MBC* (DUBOC, 2008; DUBOC, et al., 2009) para definir o espaço de novos literais. A seguir, apresentamos os resultados obtidos com o desenvolvimento do capítulo e conclusões a respeito dos mesmos.

3.1.1 Resultados Experimentais

Em (DUBOC, et al., 2009) demonstramos experimentalmente os benefícios de usar declarações de modos e a Bottom Clause ao revisar teorias lógicas de primeira-ordem. Resultados experimentais apresentaram uma diminuição de em média 50 vezes no tempo de execução, comparado com o sistema FORTE original, sem diminuir significativamente a acurácia. Além disso, nós mostramos naquele trabalho que o sistema de revisão fornece teorias mais precisas e menores, em comparação com um método padrão indutivo que também usa a Bottom Clause. Com o sistema YAVFORTE, gostaríamos de saber se é possível diminuir ainda mais o tempo de execução do processo de revisão, sem diminuir a acurácia. Com tal objetivo, apresentamos os resultados obtidos com a implementação atual, variando o conjunto de operadores usados para revisar a teoria, em comparação com a implementação do sistema FORTE contendo o algoritmo do FORTE_MBC e também com o sistema Aleph. Nós comparamos a média do tempo de execução, acurácia e tamanho das teorias, em número de cláusulas e literais.

Datasets

Nós consideramos as mesmas bases de dados utilizadas em (DUBOC, et al., 2009), nomeadamente o conjunto de dados do domínio do Alzheimer (KING, et al., 1995a), composto por quatro conjuntos de dados e o conjunto de dados DssTox (FANG, et al., 2001). O domínio do Alzheimer compara 37 análogos da Tacrine, um medicamento que combate a doença de Alzheimer, de acordo com quatro propriedades como descrito abaixo, onde cada propriedade origina um conjunto de dados diferente:

1. inibir a **amine** recaptação, composto de 343 exemplos positivos e 343 exemplos negativos

2. baixo **tóxic**idade, com 443 exemplos positivos e 443 exemplos negativos

3. inibição alta da **choline**sterase, composto de 663 exemplos positivos e 663 negativos e exemplos

4. boa reversão da deficiência de memória induzida por **scopolamine**, contendo 321 exemplos positivos e 321 exemplos negativos

O domínio do Alzheimer considera 33 diferentes predicados e 737 fatos no conhecimento preliminar.

O conjunto de dados DSSTox foi extraído do banco de dados da *EPA NC-TRER DSSTox*. Ele contém informações estruturais sobre um conjunto diverso de 232 estrogênios sintéticos naturais e ambientais e as classificações em relação à sua atividade de ligação para o receptor de estrógeno. O conjunto de dados é composto de 131 exemplos positivos e 101 exemplos negativos. Existem 25 diferentes predicados e 16.177 fatos no conhecimento preliminar.

Metodologia Experimental

Os conjuntos de dados foram divididos em 10 conjuntos disjuntos para usar a abordagem de validação cruzada estratificada com *10-folds*. Cada subconjunto mantém a taxa de distribuição original de exemplos positivos e negativos (KOHAVI, 1995). O teste de significância utilizado foi o teste-t pareado corrigido (NADEAU, BENGIO, 2003), com $p < 0,05$. Como afirmado em (NADEAU, BENGIO, 2003), o teste-t corrigido leva em conta a variabilidade devido à escolha do conjunto de treinamento e não apenas aos exemplos de teste, o que poderia levar a uma subestimação grosseira da variância do estimador de validação cruzada e à conclusão errada de que o novo algoritmo é significativamente melhor quando não é. Todos os experimentos foram executados em Yap Prolog (SANTOS COSTA, 2008).

As teorias iniciais foram obtidos a partir de Aleph sistema usando três configurações:

- A primeira configuração ajuste é executada no Aleph com seus parâmetros padrão, exceto para `minpos` [1], que foi definido como 2 para evitar que Aleph

---

[1] O parâmetro Minpos define um limite inferior no número de exemplos positivos a serem cober-

acrescentasse à teoria cláusulas instanciadas unitárias correspondentes a exemplos positivos. Essa configuração é identificada nas tabelas como **Teoria-def**.

- A segunda configuração, denominada bf Teoria-def+cl, executa o Aleph com seus parâmetros padrão, exceto para os parâmetros `minpos` e `clauselength`, para definir um limite superior sobre o número de literais em uma cláusula. Nós escolhemos este último parâmetro, porque a implementação do YAVFORTE também limita o número de literais em uma cláusula aceitável. Em todos os sistemas, o comprimento máximo é definido como 5 no domínio do Alzheimer e 10 no Dsstox, seguindo trabalhos anteriores (LANDWEHR, et al., 2007).

- A terceira configuração, **Theory-best**, executa o Aleph com parâmetros da literatura (HUYNH, MOONEY, 2008; LANDWEHR, et al., 2007). Assim, seguindo (HUYNH, MOONEY, 2008), Aleph foi definido para rodar considerando `minscore` como 0.6, a função de avaliação é a estimativa M (DZEROSKI, BRATKO, 1992), ruído é 300 para Alzheimers e 10 para DSStox e tamanho máximo da cláusula e minpos são definidos como no parágrafo anterior. O comando `induce_cover` foi invocado para obter a hipótese final, ao invés do "induce" padrão. A diferença é que exemplos positivos cobertos por uma cláusula não são removidos antes da semeadura de um novo exemplo ao usar induzir_cover. Observe que somente o tamanho máximo de cláusula é usado como parâmetro também no YAVFORTE e FORTE_MBC, uma vez que os sistemas de revisão não possuem os outros parâmetros.

A fim de identificar se há algum benefício em pré-definir o conjunto de operadores aplicáveis ao revisar teorias, YAVFORTE é executado com 5 diferentes conjuntos de operadores. Quatro configurações consideram uma combinação de um operador de especialização junto com um operador de generalização. A última configuração, identificada nas tabelas como $YAVFORTE$ considera todos os operadores.

- **YAV-del** considera exclusão de regras como operador de especialização e exclusão de antecedentes como operador de generalização.

---

tos por uma cláusula aceitável. Se a melhor cláusula cobre exemplos positivos abaixo deste número, então ela não é adicionada à teoria atual (SRINIVASAN, 2001a)

- **YAV-add** considera apenas a adição de antecedentes como operador de especialização e adição de regras como operador de generalização.

- **YAV-add-del** considera adição de antecedentes como operador de especialização e exclusão de antecedentes como operador de generalização.

- **YAV-del-add** considera exclusão de regras como operador de especialização e adição de antecedentes como operador de generalização.

Ambos YAVFORTE e FORTE_MBC executam o algoritmo *Relational Pathfinding*, seguido por *Hill climbing* na adição de antecedentes, quando aplicável.

As tabelas e as figuras com os resultados estão no apêndice B.

Considerações a respeito dos resultados   A partir dos resultados, podemos concluir o seguinte:

- YAVFORTE é sempre mais rápido do que FORTE_MBC e até mesmo produz teorias mais acuradas, com diferença significativa em quatro casos. Há dois casos em que YAVFORTE retorna teorias piores do que FORTE_MBC: Choline e DSSTox com os parâmetros do Aleph da literatura. No restante dos casos, ambos os sistemas produzem resultados com acurácias similares. A diferença de tempo se deve principalmente ao fato de que YAVFORTE pode parar de propor alterações sem o uso de todos os operadores, quando um operador mais simples já atinge o potencial do ponto de revisão. Além disso, há algum ganho no tempo ao incluir termos da cláusula base na Bottom Clause antes da mesma ser gerada da forma padrão. Lembre-se que FORTE_MBC unifica os termos da cláusula base e da Bottom Clause só depois que a Bottom Clause é construída. Sobre a diferença de acurácia, há dois fatores responsáveis por isso: (1) parar de propor revisões antes de tentar todos os operadores podem fazer com que a revisão deixe de lado alguma revisão que poderia ser melhor para o conjunto de teste. Este é o caso do conjunto de dados Choline; também, o operador de exclusão de antecedentes do YAVFORTE não tenta eliminar a prova de todos os exemplos negativos; ao invés disso, ele exclui antecedentes seguindo um valor de pontuação. Enquanto este operador mais

26

flexível é capaz de produzir resultados mais precisos nos quatro casos anteriormente mencionados, também é responsável pela pior revisão da teoria DSSTox. No entanto, na maioria dos casos, YAVFORTE é capaz de revisar as teorias tão bem quanto FORTE_MBC (sem diferença significativa) com um tempo de execução reduzido.

- Observe que os resultados do conjunto de dados Choline usando FORTE_-MBC são, na maioria das figuras destacadas do resto. FORTE _MBC tem mais dificuldade de revisar as teorias geradas a partir deste conjunto de dados do que nos outros conjuntos de dados por duas razões principais: esse é o conjunto de dados com o maior conjunto de exemplos e as teorias geradas a partir do Aleph para ele têm o maior tamanho.

- YAVFORTE considerando apenas exclusão de antecedentes e exclusão de regras como operadores de revisão atinge o tempo de revisão mais rápido e produz as menores teorias. O que é interessante sobre essa configuração é que, com raras exceções (Choline com os melhores parâmetros do Aleph e DssTox), os resultados mostram que usar apenas ambos operadores fazem com que o sistema seja capaz de proporcionar melhorias significativas sobre a teoria inicial. Na verdade, no Toxic com os parâmetros da literatura no Aleph, essa configuração é a única oferecendo o melhor da revisão: a teoria final é a menor e mais acurada e ainda foi revisada em menos tempo.

- Os resultados indicam que há benefícios de se considerar apenas um subconjunto dos operadores de revisão: na maioria dos casos obtêm-se teorias tão acuradas e em menos tempo quanto aquelas obtidas ao considerar todos os operadores. Assim, quando o perito da aplicação tem algumas dicas sobre o que esperar da teoria revisada, ele poderia usar esse conhecimento para reduzir o conjunto de operadores aplicável à revisão.

- É importante ressaltar que YAVFORTE considerando qualquer conjunto de operadores é mais rápido do Aleph em 11 dos 15 casos, com a maioria das configurações ainda retornando teorias mais acuradas. Com esse resultado,

podemos afirmar que obtivemos um sistema de revisão capaz de se comportar melhor do que um sistema indutivo, considerando-se tanto tempo de execução e acurácia.

3.1.2 Conclusões

Embora a pesquisa em revisão de teorias tenha ganhado muita atenção na década de 1990, nos últimos anos a comunidade de ILP praticamente deixou de lado a investigação nesta área, já que os benefícios de revisar teorias não podiam compensar o esforço despendido pelo grande tempo de execução desses sistemas. Este capítulo contribui para a mudança deste cenário, com base no sistema de revisão FORTE. Primeiro, em (DUBOC, et al., 2009) nós, abandonamos a busca *top-down* de literais com base no FOIL, a fim de utilizar a Bottom Clause, para reduzir o conjunto de literais levados em consideração ao refinar uma teoria, como sistemas padrão de ILP fazem.

Nesse capítulo, melhoramos a escalabilidade do sistema de revisão ao (1) fazendo o uso de operadores de revisão mais flexível, uma vez que é possível escolher quais os operadores vão ser considerados para revisar a teoria, (2) parar de propor revisões em um ponto de revisão, assim que um simples operador já atingir todo o potencial do ponto; (3) exigir que a cláusula refinada continue a obedecer a declaração de modo após um antecedente ser excluído. Além disso, também introduzimos uma modificação no operador de exclusão de antecedentes, uma vez que no FORTE original e FORTE_MBC um antecedente somente poderia ser excluído quando nenhum dos exemplos negativos não provados se tornassem provados. Foi necessário fazer esta exigência mais flexível porque há casos em que o conjunto de exemplos falso negativos só é reduzido se a cláusula é permitida também cobrir alguns exemplos que antes eram verdadeiros negativos. Finalmente, modificamos o procedimento de construção da Bottom Clause do FORTE_MBC de duas maneiras: (1) os termos da cláusula base instanciada são considerados como parte da Bottom Clause, de modo que eles podem ser usados para trazer termos adicionais para a Bottom Clause, e também para tornar desnecessário unificar a cláusula base com a Bottom Clause depois que esta última é construída, (2) a teoria corrente é levada

em conta para provar literais a serem incluídas na Bottom Clause. Chamamos o sistema, incluindo todos esses desenvolvimentos de YAVFORTE.

Resultados experimentais foram extraídos a partir de cinco conjuntos de dados relacionais de referência, ou seja, quatro conjuntos de dados do domínio do Alzheimer e o conjunto de dados DSSTox. Através deles, foi possível verificar que o processo de revisão pode ser de fato melhorado com as modificações propostas. YAVFORTE é mais rápido que FORTE_MBC sem diminuir a acurácia. Na verdade, houve casos em que a acurácia foi melhorada, considerando a teoria inicial. Também foi possível ver que ao usar um conjunto de operadores de revisão mais flexível, o tempo de execução é diminuído, enquanto a acurácia inicial ainda é melhorada, embora não tanto como quando o conjunto completo de operadores é considerado. Mais importante, nós mostramos que, quando o mesmo valor de parâmetro para limitar o tamanho de uma cláusula é usado para a revisão e para o aprendizado a partir do zero, a revisão executa mais rápido que o método indutivo. Dessa forma, alcançamos nosso objetivo de elaborar um sistema de revisão tão eficiente quanto um sistema indutivo padrão, ao menos quando conjuntos de dados de tamanho regular são usados.

Os conjuntos de dados utilizados neste capítulo não são considerados como robustos. Portanto, ainda precisamos verificar como se comporta o sistema de revisão quando os conjuntos de dados tem um grande número de exemplos e/ou conhecimento preliminar de grande porte. Além disso, como o sistema de revisão inicia o aprendizado a partir de uma teoria inicial, se esta tem um grande número de cláusulas problemáticas, o sistema provavelmente vai se comportar mal. Nesse caso, pode ser bem mais difícil de fazer a revisão do que aprender a partir do zero, uma vez que a revisão tenta propor modificações para cada cláusula defeituosa. Em virtude desses problemas, no capítulo 6 investigamos técnicas de busca local estocástica aplicada ao processo de revisão.

3.2 Revisão no Xadrez: Adquirindo as Regras de Variantes do Xadrez através de Revisão de Teorias de Primeira-ordem a Partir de Exemplos

No recente artigo (DIETTERICH, et al., 2008), os autores apontam que existiu um esforço considerável no desenvolvimento de sistemas de revisão de teorias no passado, mas que a falta de aplicações adequadas a essa tarefa fez com que tais sistemas não fossem amplamente implantados. Nesse capítulo pretendemos contribuir na direção de projetar uma aplicação na área de jogos que se encaixa perfeitamente em revisão de teorias.

Jogar é uma atividade humana fundamental e tem sido um tópico de muito interesse na comunidade de Inteligência Artificial desde o começo da área. Jogos frequentemente seguem rituais ou regras em domínios bem definidos, e portanto simplificam a tarefa de representá-los como programas de computador. Por outro lado, um bom desempenho em jogos requerem uma quantidade significante de raciocínio, fazendo com que essa área seja uma das melhores formas de testar inteligência similar a humana. A saber, bases de dados baseadas em jogos são testes de mesa comuns para sistemas de aprendizado de máquina (FÜRNKRANZ, 2007). Usualmente, sistemas de aprendizado de máquina são usados para aprender dois tipos diferentes de tarefas envolvendo jogos (FÜRNKRANZ, 1996). Uma primeira tarefa é aprender um modelo a ser usado para decidir se um movimento no jogo é permitido ou não. Ter tal modelo é fundamental para a segunda tarefa, que é aprender uma estratégia vencedora (BAIN, MUGGLETON, 1994; SADIKOV, BRATKO, 2006). Na presente tese, nós focamos na primeira tarefa. Com o objetivo de adquirir uma representação significativa para o conjunto de regras descrevendo um jogo, pode-se tirar vantagem da expressividade da lógica de primeira-ordem e sua habilidade de representar indivíduos, suas propriedades e os relacionamentos entre eles. Assim, usando métodos de ILP é possível - rudemente falando - induzir as regras do jogo escritas como um programa lógico, a partir de um conjunto de exemplos e conhecimento preliminar.

Um trabalho anterior demonstrou a viabilidade de usar ILP para induzir uma descrição baseada em regras do jogo do xadrez (GOODACRE, 1996). Entretanto,

para aprender efetivamente a teoria do xadrez, é necessário induzir não somente o conceito de legalidade de alto nível, mas também sub-conceitos para ajudar a classificar o movimento em legal ou ilegal. Considere, por exemplo, a Figura 4.1. No primeiro caso, um rei está em *cheque* e portanto seu conjunto de movimentos permitidos nesse momento são aqueles para sair do cheque. No segundo caso, uma peça está protegendo o rei (ela é um *pin*). Tal peça só pode ser movida para posições onde o rei continue protegido. Os conceitos de *cheque* e *pin* não são conceitos de alto nível, pois uma peça não pode decidir se o movimento pretendido é permitido apenas com base nos mesmos. Entretanto, eles são conceitos intermediários que também devem ser induzidos pelo sistema. Os autores de (GOODACRE, 1996) empregaram indução hierárquica para aprender um conceito de cada vez, iniciando do conceito de mais baixo nível (nenhum outro conceito depende dele) e incrementalmente adicionando as definições aprendidas para a teoria, tal que fosse possível aprender conceitos de nível intermediário e de mais alto nível.

(a)   (b)
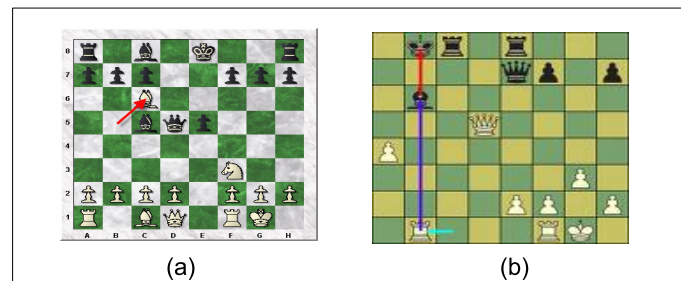
Figura 3.1: Exemplos de situações em que um sistema de ILP deve aprender definições de conceitos intermediários. A Figura mostra um tabuleiro de xadrez com um rei em cheque. A Figura (b) mostra uma peça atuando como um *pin*.

Por outro lado, jogos são ambientes dinâmicos, onde suas regras estão sempre sendo atualizadas para digamos, ser mais desafiante para o jogador, ou para produzir uma versão mais rápida e/ou mais fácil do jogo original. De fato, jogos populares frequentemente tem diferentes versões regionais, que podem ser consideradas como *variantes* ou mesmo *novas versões* do jogo. Considere, por exemplo, o jogo do xadrez, sem dúvida o jogo de tabuleiro mais jogado no mundo. Existem diversas variantes do xadrez, onde uma variante é definida como qualquer jogo que é derivado,

relacionado ou inspirado pelo xadrez, tal que a captura do rei inimigo seja o objetivo principal (PRITCHARD, 2007). Por exemplo, considere o jogo Shogi (HOOPER, WHYLD, 1992), que é a versão japonesa mais popular do xadrez. Embora ambos os jogos tenham regras similares e a mesma meta, eles também diferem em aspectos importantes. Por exemplo, no Shogi uma peça capturada pode trocar de lado e retornar ao tabuleiro [2], o que não é permitido no xadrez internacional. A Figura 4.2 mostra tabuleiros de várias variantes do xadrez. Idealmente, se as regras de uma variante do jogo foram obtidas anteriormente, gostaríamos de tirar vantagem delas como um ponto de partida para obter as regras de uma outra variante. Entretanto, tais regras precisam ser modificadas para representar os aspectos particulares da variante. Em um jogo como o xadrez essa é uma tarefa complexa, que pode requerer manipular tabuleiros de diferentes tamanhos, introduzir ou excluir novas regras de captura ou promoção e pode requerer uma redefinição das peças no jogo.

Nós abordamos esse problema como uma instância de revisão de teorias a partir de exemplos (WROBEL, 1996). Nessa tarefa, revisão de teorias se mostrar bastante relacionada com *Aprendizado por transferência* (THRUN, 1995; CARU-ANA, 1997), uma vez que as regras do xadrez internacional (a teoria inicial passada para o sistema de revisão) foram previamente aprendidas usando ILP. Aprendizado por transferência se concentra em reter e aplicar conhecimento aprendido em uma ou mais tarefas relacionadas, para eficientemente desenvolver uma hipótese efetiva para uma tarefa completamente nova.

Mostraremos que é possível aprender regras para diferentes variantes do xadrez. Para tanto, o sistema YAVFORTE foi modificado com **(i)** uma nova estratégia para simplificar a teoria inicial ao remover regras que não devem ser transferidas entre as variantes; **(ii)** suporte a *abdução* e **(iii)** suporte a *negação por falha*. Experimentos em variantes reais do xadrez mostram que nossa técnica pode transferir entre variantes com tabuleiros maiores ou menores, adquirir regras não usuais e adquirir diferentes peças. O apêndice C detalha nossas contribuições nesse tema e a próxima seção mostra resumidamente os resultados obtidos nessa aplicação.

---

[2]É sugerido que esta regra inovativa foi inspirada pela prática de mercenários do século XVI que trocavam sua lealdade quando capturados (PRITCHARD, 2007).
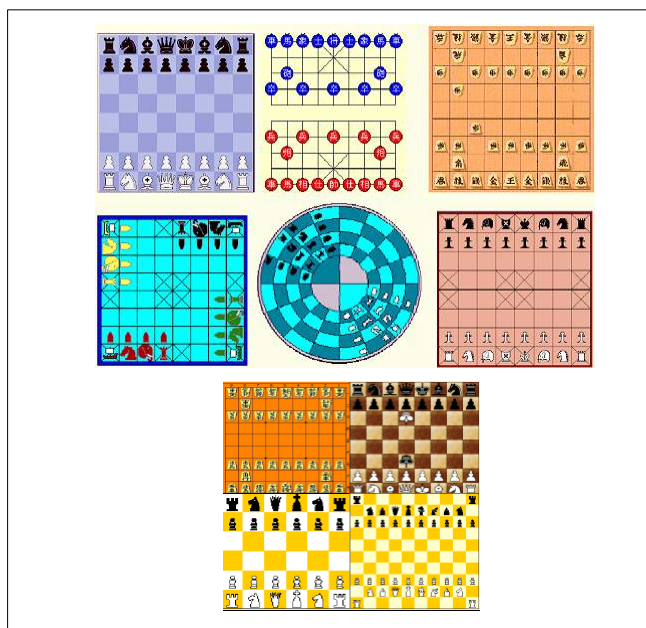
Figura 3.2: Tabuleiros de variantes do xadrez. Na primeira linha, está o tabuleiro do xadrez internacional; ao seu lado está o Xiangqi (xadrez chinês), seguido pelo Shogi (xadrez japonês). Na próxima linha aparece uma das primeiras versões do jogo do xadrez, chamado de Chaturanga (versão Hindu), seguida pelo xadrez em circulo e uma versão mais moderna do Chaturanga, já bem similar ao xadrez internacional. No último grupo aparecem o Shogi, xadrez anti-rei, Los Alamos e Grand xadrez, nessa ordem.

3.2.1 Resultados Experimentais

Metodologia Experimental As variantes do Xadrez usadas nessa tese estão descritas a seguir (PRITCHARD, 2007).

- Minixadrez Gardner. É o menor jogo de xadrez (5X5) em que todas as peças e movimentos originais ainda são usados.

- Xadred em captura livre. Difere do xadrez Internacional ao permitir que cada lado capture, além das peças inimigas, também as peças amigas, exceto pelo rei.

- Xadres Neunerschach chess. É jogado em um tabuleiro 9X9, com duas peças extras, mas sem a rainha. A primeira peça extra chama-se *Marshall* e se move igual a rainha do xadrez internacional. A segunda peça extra se move como a

33

rainha, mas somente duas casas e seu nome é *Hausfrau*.

Os resultados obtidos com as três variantes acima são exibidos no apêndice C.

3.2.2 Conclusões

Nesse capítulo contribuímos com um *framework* para revisar as regras do xadrez internacional com a meta de obter automaticamente as regras de variantes do xadrez, a partir de exemplos. O *framework* é composto de um sistema de revisão de teorias, as regras do xadrez internacional (expressas como a teoria inicial que é permitida a ser modificada), a teoria fundamental do domínio (assumida como correta) e um gerador de movimentos para a obtenção de exemplos. Nós descrevemos as modificações realizadas no processo de revisão para melhor abordar a revisão do problema de xadrez, incluindo (1) a introdução de um passo inicial para eliminar regras responsáveis por exemplos negativos classificados incorretamente, (2) o uso de abdução em três momentos diferentes do processo de revisão e (3) o uso de negação por falha. Os resultados experimentais incluíram 3 variantes do xadrez, que vão desde uma versão especializada do xadrez (minichess) para uma versão mais geral do xadrez (incluindo um tabuleiro maior e peças novas). A revisão foi capaz de retornar teorias finais corretamente descrevendo a maioria das regras das variantes. Os casos omissos foram devido à falta de exemplos de eventos raros durante um jogo, como a promoção. Além disso, a teoria final se beneficiaria de um procedimento de poda para remover as cláusulas que se tornaram inúteis após a revisão.

Estamos conscientes de que os conjuntos de dados foram gerados de uma forma bastante arbitrária, já que o número de exemplos e profundidade dos jogos foram escolhidos de acordo com a necessidade de gerar situações necessárias para a revisão. A melhor configuração experimental deve incluir vários conjuntos de dados de pelo menos tamanhos variados. Observe, contudo, que o nosso principal objetivo foi demonstrar a capacidade do sistema de revisão adquirir as regras de variantes do jogo, usando as regras do jogo internacional como ponto de partida. Além disso, gostaríamos de mostrar que era necessário mudar o sistema base de revisão para conseguir isso. Dessa forma, novas questões foram introduzidas no sistema de revisão que também pode ser úteis para outros domínios, além do xadrez.

Assim, como trabalho futuro, pretendemos experimentar mais o *framework*, considerando conjuntos de dados de tamanhos diferentes. Gostaríamos de aplicar o modelo a outras variantes mais complexas, ou seja, variantes regionais, como Shogi. Shogi e diversas outras variantes do xadrez exigem o conhecimento de um conceito inteiramente novo. A variante do xadrez conhecida como *Einstein*, por exemplo, tem o conceito de rebaixamento, onde a cada vez que uma peça se move sem capturar outra peça, ela é rebaixada a uma peça de menor valor. Tal conceito não existe no xadrez internacional e portanto, seria necessário usar invenção de predicados (MUGGLETON, BUNTINE, 1988) para obter as regras corretamente.

3.3 Revisando Teorias Lógicas de Primeira-ordem com Busca Local Estocástica

Geralmente, um sistema de revisão de Teorias de Primeira-ordem executa seu processo em três passos. Primeiro, ele busca por pontos na teoria responsáveis pela classificação incorreta de algum exemplo. Segundo, ele sugere modificações em cada ponto de revisão, incluindo adição e exclusão de literais em cláusulas e regras na teoria. Finalmente, ele busca a partir dos possíveis operadores de revisão aquele que será responsável por de fato implementar uma modificação na teoria corrente. Em cada uma dessas buscas, um sistema como YAVFORTE segue uma estratégia enumerativa, produzindo grandes espaços de busca que podem se tornar intratáveis, de acordo com os seguintes fatores:

1. O número de exemplos classificados incorretamente, uma vez que o sistema de revisão percorre as possíveis provas de cada exemplo procurando por pontos de falha.;

2. O tamanho da teoria inicial, uma vez que toda cláusula na teoria pode ser um potencial ponto de revisão;

3. A quantidade de cláusulas responsáveis pela classificação incorreta de algum exemplo, uma vez que o sistema de revisão propõe modificações para cada um deles;

4. O tamanho da base de conhecimento e do conhecimento preliminar, uma vez que novos literais devem ser gerados para serem adicionados nos corpos de cláusulas, como sugestões de modificações a serem implementadas na teoria.

Adicionalmente, sistemas de revisão de teoria lidam com hipóteses que são teorias inteiras ao invés de buscar por cláusulas individualmente a cada iteração. Buscar por hipóteses compostas de teorias inteiras é conhecido como um problema bem difícil (BRATKO, 1999). Como resultado, sistemas de revisão de teorias tradicionais devem buscar em espaços extremamente grandes, podendo se tornar bastante ineficientes ou até mesmo intratáveis.

Os últimos anos tem mostrado que métodos de busca local estocástica, originalmente projetados para resolver problemas proposicionais combinatórios difíceis (SELMAN, et al., 1992), (SELMAN, et al., 1996), (RÜCKERT, KRAMER, 2003), também podem executar bem em uma variedade de aplicações. Além disso, combinar busca local estocástica com programação em lógica indutiva mostrou que melhoras substanciais em eficiência podem ser alcançadas, com pouco ou nenhum custo em acurácia (SRINIVASAN, 2000), (PAES, et al., 2006b), (ŽELEZNÝ, et al., 2006), (MUGGLETON, TAMADDONI-NEZHAD, 2008). Tais resultados motivaram a contribuição deste capítulo. E além de tudo, nós levamos Trefheten's Maxim No. 30 em alta consideração, que declara que se o espaço de busca é enorme, a única forma razoável de explorá-lo é aleatoriamente (TREFETHEN, 1998). Assim, nossa meta é alcançar um balanço entre eficiência e eficácia, ao diminui o impacto negativo dos fatores numerados acima no tempo de execução do processo de revisão. Para tanto, sacrificamos completude para favorecer a descoberta de boas soluções em um tempo razoável. Componentes estocásticos são incluídos nas buscas chave do processo de revisão, a saber:

1. Busca por pontos de revisão: Uma decisão aleatória pode retornar um subconjunto dos pontos de revisão ao invés de retornar todos eles.

2. Busca por literais: como as propostas de modificações são dominadas pela adição e exclusão de literais de cláusulas, pode ser benéfico randomizar literais gerados (PAES, et al., 2007a; PAES, et al., 2007b).

3. Busca pela revisão a ser implementada: ao invés de propor todas as revisões, podemos enumerar os operadores de revisão e escolher um aleatoriamente para modificar a teoria (PAES, et al., 2007b).

Experimentos preliminares com o sistema de revisão FORTE mostraram resultados promissores ao incluir busca estocástica nas duas últimas buscas acima (PAES, et al., 2007a; PAES, et al., 2007b). O apêndice E revisita aquele trabalho e mostra como ele pode ser melhorado ao se projetar um número de busca estocástica em cada passo do processo de revisão do sistema YAVFORTE. Observe que em (PAES, et al., 2007a; PAES, et al., 2007b) componentes estocásticos foram introduzidos no sistema FORTE, que não usava a Bottom Clause para limitar o espaço de novos literais, ao contrário do presente trabalho.

3.3.1 Resultados Experimentais

Neste capítulo, gostaríamos de investigar se é possível reduzir o tempo de execução do processo de revisão, mesmo quando o espaço de busca é maior do que o habitual, pelo uso de técnicas de busca local estocástica. Assim, a grande questão que gostaríamos de responder é se o tempo de execução do processo de revisão guiada por algoritmos SLS, pode ser mais rápido do que a revisão tradicional, mas sem prejudicar a acurácia. A questão secundária é se a revisão consegue ser comparável ao aprendizado a partir do zero em termos de tempo de execução, mas ainda alcançando acurácias melhores. Para tanto, foram selecionados três conjuntos de dados que o sistema YAVFORTE teve um tempo difícil para revisar. Os conjuntos de dados são os seguintes.

Conjuntos de dados

- Pyrimidines: esse é um problema de relacionamento entre a atividade e a estrutura quantitativa (QSAR), envolvendo a inibição da *E. Coli Dihydrofolate Reductase* por *pyrimidines*, que são antibióticos agindo para inibir a *Dihydrolate Reductase*, uma enzima no caminho de formação do DNA (KING, et al., 1992; HIRST, et al., 1994a). O conjunto de dados usado nesse trabalho é composto de 2361 exemplos positivos e 2361 exemplos negativos.

- Proteins: Envolve a tarefa de predição da estrutura secundária de uma proteína. A tarefa é aprender regras para identificar se um posição na proteína é um (MUGGLETON, et al., 1992). O conjunto de dados possui 1070 exemplos positivos e 970 exemplos negativos.

- Yeast_sensitivity (SPELLMAN, et al., 1998; KADUPITIGE, et al., 2009): Esse é um conjunto de dados sobre o problema da interação do gene da*yeast Saccharomyces cerevisiae.*O conjunto é composto de 430 exemplos positivos e 680 exemplos negativos e o conhecimento preliminar é composto de cerca de 170.000 fatos.

A metodologia experimental é a mesma empregada no capítulo 3. Para detalhes adicionais, consulte o apêndice E.

Antes de experimentar os algoritmos estocásticos com parâmetros específicos, nós fizemos experimentos variando o valor dos diferentes parâmetros. Assim, variamos o número máximo de pontos de revisão ao randomizar os pontos de revisão, variamos as probabilidades dos caminhos aleatórios e variamos o número máximo de iterações do algoritmo, quando assim era requerido. O apêndice E detalha cada um desses experimentos.

Comparando Acurácia e Tempo de Execução dos Algoritmos Estocásticos com Aleph e YAVFORTE sem os Componentes Estocásticos

Nessa seção, comparamos o desempenho dos algoritmos de busca local estocástica com o estado-da-arte de um sistema de ILP e de revisão de teorias. Além de rodar cada algoritmo estocástico individualmente, também os combinamos como segue.

1. Busca estocástica pelos pontos de revisão e busca estocástica por literais.

2. Busca estocástica pelos pontos de revisão e busca estocástica *Hill climbing* por revisões.

3. Busca estocástica *Hill climbing* por revisões com busca estocástica por literais.

4. Busca estocástica pelos pontos de revisão com busca estocástica por literais e busca estocástica *Hill climbing* por revisões.

A partir dos resultados obtidos, seguem abaixo as conclusões mais relevantes.

- Todos os algoritmos de revisão nos três conjuntos de dados atingem acurácias significativamente melhores do que o sistema Aleph.

- Com exceção da busca gulosa por revisões, no conjunto de dados Pyrimidines, não há diferença estatística significativa entre a acurácia retornado pelo sistema YAVFORTE e os algoritmos estocásticos.

- O melhor resultado de acurácia no conjunto de dados é conseguido através da busca estocástica por literais combinada com a busca *Hill climbing* estocástica por revisões, embora não seja significativamente melhor que os outros.

- No conjunto de dados Proteins, os melhores resultados são obtidos pela busca estocástica por literais e busca estocástica por literais combinado com busca estocástica *Hill Climbing* por revisões.

- No conjunto de dados Proteins, a busca estocástica por pontos de revisão e a busca estocástica gulosa por revisões atinge acurácias significativamente piores, em comparação com YAVFORTE e os melhores resultados dos algoritmos SLS.

- Ainda considerando o Proteins, não há diferença significativa entre YAVFORTE e o resto dos algoritmos SLS.

- No conjunto de dados Yeast Sensitivity, a busca estocástica por literais e a combinação dos três componentes estocásticos alcançam acurácias significativamente melhores do que YAVFORTE.

A partir desses resultados, podemos ver que os algoritmos SLS ou fornecem acurácias melhores ou equivalentes em comparação com o sistema base YAVFORTE. Além disso, eles são sempre muito melhores do que o sistema Aleph.

A partir dos resultados comparando tempo de execução, os seguintes aspectos são observados.

- Conjunto de dados Pyrimidines.

- O tempo de revisão de YAVFORTE é significativamente mais lento do que todos os outros algoritmos no conjunto de dados Pyrimidines.

- O único algoritmo significativamente mais lento do que Aleph é SLS com escapadas estocásticas para buscar revisões. Isso é provavelmente devido ao elevado número de iterações que o algoritmo precisa para obter acurácias desejáveis.

- O algoritmo *Hill Climbing* estocástico por revisões combinado com a busca estocástica por literais não apresentou diferença em relação ao Aleph. Neste caso, a combinação destes algoritmos SLS leva mais tempo para convergir.

- Todos os outros casos estocásticos são significativamente mais rápidos do que o Aleph.

- A maior velocidade de um algoritmo SLS, em comparação com YAV-FORTE tem um fator de 17, enquanto a menor velocidade é de 4.

- No conjunto de dados Proteins todos os algoritmos SLS são significativamente mais rápidos do que Aleph e YAVFORTE, exceto para a busca estocástica por literais, que é mais rápido que YAVFORTE mas não é significativamente diferente do Aleph. A maior velocidade de um algoritmo de SLS, em comparação com YAVFORTE tem um fator de 16, enquanto a menor velocidade é $2X$ melhor.

- No Yeast Sensitivity, o sistema Aleph é significativamente mais rápido que todos os algoritmos de revisão. Acreditamos que isto é devido ao enorme conhecimento preliminar que esse conjunto de dados tem, o que faz com que os testes de cobertura sejam mais lentos na revisão. Por outro lado, todos os algoritmos SLS são significativamente mais rápidos do que YAVFORTE. A maior velocidade de um algoritmo SLS, em comparação com YAVFORTE tem um fator de 26, enquanto a menor velocidade é $2X$ melhor.

A partir dos resultados, podemos concluir que com o uso de algoritmos de busca local estocástica individualmente e, especialmente, combinando-os, é possível reduzir

significativamente o tempo de revisão e também ser mais rápido ou competitivo com o aprendizado a partir do zero. Além disso, a acurácia alcançada por estratégias SLS são sempre melhores do que o aprendizado a partir do zero e competitivo com a abordagem tradicional de revisão.

3.3.2 Conclusões

Nesse capítulo, projetamos uma série de algoritmos de busca local estocástica para explorar os espaços-chave do processo de revisão de forma mais eficiente. Os algoritmos de busca local estocástica abandonam a completude para favorecer a descoberta de boas soluções em um tempo razoável. A maioria deles são baseados em caminhadas aleatórias, de modo que a escolha de seguir um movimento guloso ou estocástico é feita de acordo com um parâmetro de probabilidade. Algoritmos estocásticos foram implementados no sistema YAVFORTE (veja o capítulo 3 e (DUBOC, et al., 2009)), em cada busca chave do processo de revisão.

Primeiro, um algoritmo SLS foi construído para evitar a coleta de todos os pontos de revisão, de todos os exemplos classificados incorretamente. Com uma probabilidade p, exemplos classificados erroneamente são randomizados e um número pré-definido de pontos de revisão é gerado. A pesquisa é alternada com movimentos gulosos, pois sem a probabilidade p todos os pontos de revisão encontrados na teoria de cada instância classificada incorretamente são coletados. No entanto, através de resultados experimentais, descobrimos que em vários casos não há necessidade de empregar movimentos gulosos, uma vez que definir o parâmetro de probabilidade em 100% já fornece bons resultados: o tempo de revisão é bastante reduzido e a acurácia não é estatisticamente diferente em comparação com o sistema base de revisão. O desempenho do componente estocástico na busca por pontos de revisão é mais influenciado pelo parâmetro que define o número de pontos de revisão que deve ser retornado.

Em segundo lugar, os componentes SLS foram incluídos na busca por literais a serem adicionados ou removidos de uma cláusula. A busca estocástica foi incluída em ambos os algoritmos *Hill Climbing* e *Relational Pathfinding* para especializar cláusulas. No primeiro caso, quando o movimento é estocástico, literais da Bottom

Clause são randomizados e o primeiro literal descoberto que melhorar a pontuação é adicionado à cláusula. No algoritmo *Relational Pathfinding*, caminhos criados a partir da Bottom Clause e de um exemplo positivo são randomizados. Percebemos a partir da avaliação empírica que, embora a busca estocástica por literais seja capaz de reduzir o tempo de execução em relação a abordagem *Hill Climbing* gulosa do YAVFORTE, se for executado sem os outros componentes estocásticos, não é tão eficaz como os outros algoritmos SLS. As razões para isso são principalmente devidas à Bottom Clause cláusula: ou ela é pequena e o componente estocástico não faz uma muita diferença, ou ela é grande, mas nesse caso o uso de movimentos estocásticos leva mais iterações para convergir do que a abordagem original. Novas abordagens podem ser investigadas para melhorar ainda mais este componente estocástico, como pré-processamento de literais usando redes Bayesianas ou algoritmos genéticos (OLIPHANT, SHAVLIK, 2008; MUGGLETON, TAMADDONI-NEZHAD, 2008; PITANGUI, ZAVERUCHA, 2011).

Em terceiro lugar, quatro diferentes componentes estocásticos foram incluídos para decidir qual revisão vai ser implementada. Três deles são baseados em caminhadas aleatórias e um deles executa o algoritmo *Simulated Annealing*. Nos resultados, foi possível ver que o algoritmo estocástico guloso ao busca por revisões, que em ambos movimentos estocásticos e gulosos permite uma revisão com pontuação piore do que a corrente ser implementada, não desempenha bem. Como uma melhoria na pontuação não é sempre necessária, a acurácia se deteriora ao longo das iterações. A abordagem de escapadas estocástica, que aceita movimentos ruins, mas somente se ele não degradar tanto a pontuação, tem melhor desempenho do que a abordagem gulosa, com boa acurácia e tempo de execução reduzido em vários casos. No entanto, a estratégia mais simples, que com uma certa probabilidade randomiza revisões e implementa a primeira a melhorar a pontuação, obtem os melhores resultados globais.

As três estratégias acima foram comparadas individualmente e combinando as melhores estratégias. A grande maioria dos casos mostrou que as abordagens estocástica atingem acurácias melhores do que o aprendizado a partir do zero do sistema Aleph, com tempo de execução mais rápido ou competitivos. Além disso, a

acurácia na maioria dos casos são significativamente equivalentes as do obtidas com YAVFORTE, mas o tempo de execução é sempre muito mais rápido. No melhor dos casos, um algoritmo SLS é de $25X$ mais rápido que o sistema YAVFORTE, com acurácia equivalente.

Capítulo 4

BFORTE: Resolvendo os Pontos de Gargalo no Processo de Revisão de Programas Lógicos Bayesianos

Inferência em Programas Lógicos Bayesianos é executada sobre redes construídas a partir de todas as provas do conjunto de exemplos. O algoritmo de aprendizado de BLPs inicia a partir de um programa lógico maximamente geral e que satisfaça a parte lógica dos exemplos. A seguir, refinamentos são propostos para cada cláusula, através de adição e exclusão de literais. Dessa forma, o algoritmo de aprendizado de BLPs requer buscar em um espaço grande de cláusulas candidatas, por um lado, e a construção de todas as provas para construir as redes Bayesianas por outro lado. Para cada hipótese candidata, deve-se executar inferência Bayesiana para computar sua pontuação na presença de dados não observados.

O sistema PFORTE foi desenvolvido para obter BLPs acurados ao revisar um BLP existente e modificá-lo apenas nas regras usadas em redes Bayesianas de exemplos classificados incorretamente. Entretanto, apesar de resultados promissores em domínios artificiais, PFORTE enfrenta gargalos similares aos do aprendizado de BLPs e de outros algoritmos de aprendizado lógico estatístico, uma vez que PFORTE também usa espaços de busca relacionados a programas lógicos e também executa inferência em redes Bayesianas construídas a partir de cada exemplo relacional.

O objetivo do presente capítulo é abordar os pontos de gargalo do sistema PFORTE. Primeiro, o espaço de possíveis refinamentos pode ser reduzido ao limitar os novos literais a serem adicionados à cláusulas para aqueles presentes na Bottom

Clause (MUGGLETON, 1995b; DUBOC, et al., 2009). Segundo, mostramos que ao coletar pontos de revisão através do algoritmo Bayes Ball (SHACHTER, 1998) fundamentalmente reduz o número de cláusulas marcadas como candidatas a serem revisadas pelos operadores de revisão, assim como o tipo de operador que pode ser aplicado. Terceiro, pontos de revisão lógicos e probabilísticos não são mais vistos separadamente, evitando o custo de tratá-los em dois passos distintos do processo de revisão. Quarto, observamos que redes construídas a partir de exemplos lógico probabilísticos podem ter muitos literais em comum, devido a entidades compartilhadas, resultando em redes conjuntas muito grandes. Assim, aplicamos métodos para (1) agrupar cláusulas instanciadas a partir de um exemplo com características idênticas, (2) separar nós indispensáveis para uma consulta específica e executar inferência apenas com eles e (3) sobrepor nós indispensáveis encontrados em para instâncias diferentes. Tais métodos seguem as ideias em desenvolvimentos recentes de inferência levantada (POOLE, 2003), (SALVO BRAZ, et al., 2005), (MEERT, et al., 2010). Ao focar nos pontos de modificação realmente relevantes e capacitar o processo de revisão a explorar todas as suas potencialidades, esperamos obter um sistema de revisão de fato efetivo e praticável, conforme apontado por (DIETTERICH, et al., 2008) como um desenvolvimento necessário na área de SRL. O novo sistema desenvolvido é chamado de BFORTE, se referindo a **B**LP, **B**ayes **B**all and **B**ottom clause.

O apêndice G detalha o desenvolvimento do sistema BFORTE.

4.1   Resultados Experimentais

Nessa seção, experimentamos o sistema de revisão BFORTE, comparando cada contribuição desenvolvida neste capítulo com uma versão do sistema sem a contribuição. Como não há implementação atual do algoritmo de aprendizado da estrutura de BLPs, usamos nosso próprio sistema para fazer as comparações devidas. Optamos por mostrar cada contribuição em separado, de modo que seja possível conhecer os benefícios trazidos por elas.

Bases de dados Conjuntos de dados usados nas comunidades de ILP e SRL foram considerados. Eles são brevemente descritos a seguir.

1. *UW-CSE* é um conjunto de dados muito utilizado na comunidade de SRL (SINGLA, DOMINGOS, 2005; RICHARDSON, DOMINGOS, 2006; KOK, DOMINGOS, 2007). É composto por informações sobre o Departamento de Ciência da Computação e Engenharia da Universidade de Washington. Existem 5 exemplos, onde cada um deles contém instâncias que representam uma relação de orientação para uma linha de pesquisa diferente no departamento. Há 113 instâncias da classe positiva e 2711 instâncias da classe negativa, e 2673 fatos.

2. *Metabolism* é uma base de dados fornecida pelo *KDDCup*2001 (CHENG, et al., 2002b). Os dados consistem em cerca de 6910 fatos e 115 instâncias positivas e 115 instâncias negativas. Como parte dos fatos representa a interação entre genes, as instâncias são reunidas em um mega exemplo contendo todas as instâncias positivas e negativas.

3. *Carcinogenesis* é um domínio bem conhecido para a previsão de relação estrutura-atividade (SAR) sobre a atividade em bioensaios de roedores (SRINIVASAN, et al., 1997a). Há 162 instâncias da classe positiva e 136 instâncias da classe negativa e 24342 fatos elas. Esse conjunto de dados é totalmente separável, já que cada exemplo é composto de apenas uma instância.

Metodologia Experimental Os conjuntos de dados foram separados em 5 conjuntos disjuntos para usar a abordagem de validação cruzada estratificada 5-folds. Cada conjunto mantém a taxa de distribuição original de exemplos positivos e negativos (KOHAVI, 1995). Para evitar *overfitting* durante o processo de revisão, semelhante a (BAIÃO, et al., 2003), aplicou-se 5 vezes a abordagem de validação cruzada estratificada para dividir os dados de entradaem conjuntos disjuntos de treinamento e teste e, dentro do conjunto de treinamento, aplicamos validação cruzada estratificada 2-fold para dividir os dados de treinamento em conjuntos disjuntos de treinamento e validação. O algoritmo de revisão monitora o erro com relação ao conjunto

de validação após cada revisão, mantendo sempre uma cópia da teoria com a melhor pontuação no conjunto de validação, e a melhor teoria salva é aplicada ao conjunto de teste. O teste de significância utilizado foi o teste t corrigido (NADEAU, BENGIO, 2003), com $p < 0,05$.

As teorias iniciais para *Carcinogenesis* e *Metabolism* foram obtidas a partir do sistema Aleph usando seus parâmetros padrão, exceto para o tamanho máximo da cláusula, que é definido como 5, o ruído, definido como 30, e minpos, definido como 2. Como UW-CSE é uma base de dados altamente desbalanceada, usamos a estimativa M como função de avaliação, e o ruído definido como 1000.

Todos os experimentos foram executados em Yap Prolog (SANTOS COSTA, 2008) e Matlab. Para lidar com redes Bayesianas, o toolbox *Bayes net* (MURPHY, 2001) foi devidamente modificado para enfrentar as particularidades de BLPs. Regras de combinação representadas em dois níveis, onde o primeiro nível (instanciações diferentes da mesma cláusula) usa *mdia* como regra e o segundo nível (cláusulas diferentes com a mesma cabeça) usa *mdia ponderada* como função de combinação. Para aprender os parâmetros, implementamos o algoritmo descida do gradiente discriminativo descrito em (NATARAJAN, et al., 2008), que minimiza o erro médio quadrático. No UW-CSE, usamos o erro médio quadrático ponderado, com o peso inversamente proporcional à distribuição das classes, de modo que os parâmetros não favoreçam totalmente a classe negativa por causa de sua quantidade de exemplos. Para realizar inferências, adaptamos o algoritmo de eliminação de variáveis. Nós impomos um número mínimo de exemplos classificados incorretamente como 2 para uma cláusula ser considerada como ponto de revisão, a fim de evitar *outliers*. O limiar para indicar se uma instância foi classificada corretamente é definido como $0,5$.

4.1.1 Comparando BFORTE com ILP e Revisão de Teorias de Primeira-ordem

Em primeiro lugar, gostaríamos de saber se BFORTE consegue resultados com melhor pontuação do que sistemas padrão de aprendizado em primeira-ordem. Nós comparamos BFORTE depois de aprender os parâmetros iniciais e BFORTE de-

pois de revisar a estrutura, com os sistemas Aleph e FORTE. As mesmas teorias aprendidas pelo Aleph são fornecidas para ambos os sistemas de revisão.

Resultados podem ser vistos no apêndice G. A partir deles, foi possível verificar que BFORTE, ao revisar a estrutura inicial, pode melhorar as probabilidades dos exemplos. Além disso, o valor final de pontuação é melhor do que os casos de primeira-ordem, mostrando que é possível obter hipóteses mais acuradas quando a incerteza é levada em consideração.

4.1.2 Melhorando o Tempo de Revisão com a *Bottom Clause*

Agora, gostaríamos de verificar se a introdução da *Bottom Clause* para limitar o espaço de literais pode diminuir o tempo de execução sem prejudicar a pontuação. Para tanto, rodamos BFORTE utilizando a *Bottom Clause* e BFORTE usando o algoritmo FOIL para gerar literais. Resultados do tempo de execução e pontuação são exibidos no apêndice G. Como esperado, UW-CSE executa significativamente mais rápido quando a *Bottom Clause* é o espaço de busca de literais. Surpreendentemente, o resultado também é melhor ao usar a *Bottom Clause*, embora a diferença não seja significativa.

Infelizmente, não fomos capazes de coletar resultados do FOIL nos outros dois conjuntos de dados, uma vez que o sistema computacional fica sem memória depois de algum tempo rodando. O motivo, além do espaço de busca muito maior gerado para a abordagem FOIL, é que diferentemente de UW-CSE, *Metabolism* e *Carcinogenesis* tem vários predicados com constantes definidas em declarações de modo. A abordagem top-down do FOIL não pode gerar literais com constantes, e então uma variável é colocada no lugar de uma constante possível. O problema que ao colocar uma variável, o número de instanciações do literal pode ficar muito grande, gerando uma enorme quantidade de caminhos diferentes para a prova do literal.

4.1.3 Seleção dos Pontos de Revisão

Neste experimento, vamos nos concentrar em saber se a abordagem do BFORTE para selecionar pontos de revisão pode diminuir o tempo do processo de revisão.

Devido às limitações do PFORTE e aprendizado de BLPs, principalmente em relação ao espaço de busca de literais (seção anterior), nós comparamos o BFORTE ao BFORTE simulando os algoritmos no que se refere à seleção de pontos de revisão. Comparamos BFORTE com as configurações a seguir.

- BFORTE: esse é o sistema implementado com todas as contribuições apresentadas neste capítulo: a Bottom Clause é usada para limitar o espaço de busca de literais, os pontos de revisão são selecionados utilizando Bayes Ball e procedimento de inferência é otimizado.

- PFORTE-like: esse caso considera a Bottom Clause para limitar o espaço de busca de literais e o procedimento de inferência otimizado. No entanto, a seleção dos pontos de revisão segue o sistema PFORTE, o qual considera todas as cláusulas que aparecem na rede de uma instância classificada incorretamente como pontos de revisão.

- BLP-like: esse caso considera a Bottom Clause para limitar o espaço de busca de literais e o procedimento de inferência otimizado. No entanto, as hipóteses candidatas a serem refinadas incluem todas as cláusulas Bayesianas no BLP corrente, seguindo o algoritmo de aprendizado de BLPs.

As tabelas com os resultados estão no apêndice G. O tempo de execução do sistema BFORTE é significativamente melhor do que ambos PFORTE-like e BLP-like. PFORTE executa pior do que BLP, devido ao tamanho da rede, que em SRL costuma ser bem grande. Nessa situação, o caso mais provável é que todas as cláusulas sejam marcadas como ponto de revisão, mas PFORTE ainda gasta tempo para verificar instâncias classificadas incorretamente, enquanto BLP não precisa desse passo. Apesar da pontuação final do BFORTE ser menor do que dos outros sistemas, a diferença não é significativa.

Infelizmente, não foi possível obter resultados com PFORTE ou BLP para o *Metabolism*, pois a teoria para esse domínio tem cerca de 18 cláusulas e todas elas eram marcadas para serem modificadas por todos os operadores nos demais sistemas, o que causou problemas de memória para esses sistemas.

Considerando o conjunto de dados do Carcinogenesis, BFORTE e PFORTE-like se comportaram da mesma maneira, pois esse conjunto de dados é separável, o que faz com que as redes sejam criadas apenas considerando as instâncias individuais.

4.1.4 Tempo de Inferência

Por fim, mostramos a redução no tempo de execução da inferência devido ao uso do Bayes Ball e do agrupamento de cláusulas e redes. Note que todos os casos discutidos anteriormente levam em conta a inferência com Bayes Ball, ao aprender ou revisar teorias. Ao invés de executar o processo de revisão geral, optamos por executar somente o procedimento que calcula a pontuação discriminativa para cada conjunto de treinamento, considerando as teorias iniciais. Desta forma, somos capazes de ver a *melhora de tempo de execução na inferência* sozinha, sem levar em conta as demais particularidades do processo de revisão.

Os resultados para cada *fold* podem ser vistos no apêndice G. Vamos focar primeiro no UW-CSE. O tempo de execução é reduzido por um fator de 170. A maior redução se deve ao Bayes Ball, embora o agrupamento de nós semelhantes também contribua. Por outro lado, a detecção de redes semelhantes não ajuda muito na redução do tempo de execução.

O tempo de execução da inferência no Carcinogenesis não é diferente para nenhum dos sistemas. Na verdade, na maioria dos casos, o tempo de execução é um pouco pior, devido ao custo de execução do Bayes Ball. No entanto, para um conjunto de treinamento em particular, ao executar a inferência sem agrupar nós semelhantes, uma das cláusulas Bayesianas produz uma grande quantidade de cláusulas instanciadas para fazer parte da rede final.

4.2   Conclusões

bordamos neste capítulo os gargalos da revisão de Programas Lógicos Bayesianos. Mostramos através de resultados experimentos que é possível obter um sistema de revisão viável e que fornece modelos mais precisos do que os sistemas de revisão de teorias lógicas de primeira-ordem.

Primeiro, nos focamos na redução do espaço de busca de novos literais. O

sistema de revisão anterior, PFORTE, gerava literais usando a abordagem do FOIL: todos os literais criados a partir da linguagem e que tinham ao menos uma variável em comum com a cláusula corrente eram considerados para serem adicionados à cláusula. O espaço de busca de literais foi reduzido ao o limitarmos a *Bottom Clause* de uma instância classificada incorretamente, e não houve prejuízo na acurácia. Como uma única instância classificada incorretamente pode não carregar informação suficiente para gerar bons literais, pretendemos no futuro investigar o uso de várias instâncias para criar a *Bottom Clause*.

Em seguida, nos focamos em reduzir o espaço de busca de cláusulas a serem refinadas. O algoritmo de aprendizado de BLPs começa a partir de um conjunto inicial de cláusulas Bayesianas e propõe modificações para cada uma delas. PFORTE considera todas as cláusulas utilizadas para construir a rede Bayesiana de um exemplo onde havia uma instância classificada incorretamente. Ambos os sistemas geram um grande espaço de busca que pode até mesmo tornar-se intratável. Nós mostramos que é possível reduzir esse espaço de busca, marcando como pontos de revisão somente as cláusulas que influenciam a distribuição de probabilidade calculada para instâncias classificadas incorretamente. O algoritmo Bayes Ball é usado para identificar essas cláusulas. Experimentos sugerem que o espaço de busca é de fato reduzido, propondo melhorias apenas para as cláusulas pertinentes à instância classificada incorretamente. No futuro, gostaríamos de investigar com mais detalhes se é possível reduzir o número de operadores de revisão de acordo com o ponto de revisão, como já começamos a analisar neste capítulo.

Por fim, tentamos reduzir o grande espaço de busca de inferência, uma vez que PLL gera redes Bayesianas para exemplos relacionais, que possuem muitas variáveis aleatórias relacionadas. Para tanto, desenvolvemos um procedimento para considerar apenas os nós identificados como indispensáveis pelo algoritmo Bayes Ball, para o cálculo da distribuição de probabilidade de uma instância. Adicionalmente, reduzimos o tamanho da rede, identificando cláusulas instanciadas, cuja única diferença são os termos substituindo variáveis. Argumentamos que não é necessário representá-los como nós separados na rede, o que faria o procedimento de inferência repetir vários cálculos de distribuições de probabilidade. Além disso, identificamos que depois

de selecionar os nós indispensáveis, podem surgir "sub-redes", com exatamente a mesma estrutura e evidências. Evitamos calcular as probabilidades para todas essas redes, ao considerar apenas uma representante do grupo. Experimentos mostraram que o tempo de inferência é de fato muito reduzido pelos algoritmos propostos. Esse é apenas um pequeno passo na direção de trabalhos recentes sobre inferência levantada (POOLE, 2003; SALVO BRAZ, et al., 2005; SINGLA, DOMINGOS, 2008; MILCH, et al., 2008; KOK, DOMINGOS, 2009; NATH, DOMINGOS, 2010; MEERT, et al., 2010).

Para detalhes adicionais e trabalhos futuros nessa direção, consulte o apêndice G.

Capítulo 5

Conclusões Gerais e Trabalhos Futuros

O presente capítulo sumariza as realizações desta tese e apresenta direções para pesquisa futura no tema.

5.1   Sumário

A meta dessa tese foi contribuir com sistemas de revisão de teorias efetivos. Sem dúvida, a principal razão para o abandono da pesquisa em revisão de teorias na última década é o custo de buscar por revisões a serem implementadas em espaços de busca muito grandes. Com essa tese, mostramos que é possível ter um sistema de revisão de teoria que executa em um tempo praticável e ainda alcança teorias mais acuradas do que o aprendizado do zero. Para tanto as seguintes contribuições foram feitas:

No capítulo 3, contribuímos com um sistema de revisão nomeado de YAV-FORTE, que foi construído a partir do sistema FORTE (RICHARDS, MOONEY, 1995). YAVFORTE oferece ao usuário a escolha de selecionar operadores de revisão, tal que o número de propostas de modificações para uma teoria seja menor do que se todos os operadores fossem aplicados. Foi mostrado experimentalmente que existem vários casos em que um conjunto menor de operadores de revisão consegue alcançar as mesmas acurácias do que se todos os operadores estivessem em uso, com a vantagem do tempo ser significativamente reduzido. Adicionalmente, operadores de revisão são empregados a partir do mais simples para o mais complexo, tal que quando um operador mais simples já for capaz de alcançar o potencial máximo do ponto de

revisão, aplicar um operador mais complexo se torna desnecessário. Outra importante contribuição implementada como um componente do sistema YAVFORTE é a construção da Bottom Clause (MUGGLETON, 1995b) e o uso de declarações de modo para limitar o espaço de busca de novos literais a serem considerados para especializar uma cláusula (DUBOC, et al., 2009). O uso da Bottom Clause reduz de forma bastante significativa o tempo total do processo de revisão, se comparado com o sistema FORTE que gera literais seguindo o sistema FOIL (QUINLAN, 1990).

No capítulo 4 uma aplicação desafiante baseada no jogo do xadrez foi projetada, com o objetivo de mostrar o poder da revisão de teoria principalmente ao lidar com predicados inter-dependentes. Foi desenvolvido um *framework* que inclui um gerador de exemplos para variantes do xadrez, a teoria inicial descrevendo as regras internacionais do xadrez, modificada a partir de um trabalho anterior (GOODACRE, 1996) e o sistema de revisão YAVFORTE com três importantes modificações: a inclusão de um passo inicial de compressão da teoria, tal que a mesma se tornasse mais simples para que todos os operadores fossem aplicados; o uso de estratégias adicionais de abdução na busca por pontos de revisão e na aplicação de operadores de revisão e suporte a negação por falha. Mostramos que após a revisão, é possível obter teorias descrevendo variantes do xadrez, enquanto que ao aprender do zero, considerando a teoria do xadrez como conhecimento preliminar apenas, o mesmo não é possível. Dessa forma, também mostramos que problemas de transferência de aprendizado podem ser manipulados através de revisão de teorias.

Finalmente, para que a revisão também se tornasse praticável em problemas envolvendo teorias com muitas cláusulas e em bases de dados não triviais, com uma quantidade considerável de exemplos e fatos, incluímos vários componentes estocásticos no sistema YAVFORTE. No capítulo 6 explicamos detalhadamente os algoritmos e espaços de busca de cada componente estocástico, que possuem em comum o fato de encontrarem boas soluções ao invés de ótimas em um tempo razoável.Ao randomizar pontos de revisão, operadores de revisão e literais a serem sugeridos a saírem ou entrarem em cláusulas, conseguimos alcançar um tempo de aprendizado de em média $25X$ mais rápido e com acurácias equivalentes ao sistema YAVFORTE. Além disso, acurácias melhores e tempo de execução competitivos com um sistema

indutivo padrão de ILP também foram alcançados.

No capítulo 8 contribuímos com um sistema praticável de revisão de Programas Lógicos Bayesianos (BLP). Nós abordamos os pontos de gargalo do nosso sistema de revisão anterior, chamado PFORTE, que se mostrou impraticável de ser aplicado com bases de dados reais. Primeiro, seguindo nosso algoritmo FORTE_-MBC (DUBOC, et al., 2009), incluímos a Bottom Clause para limitar o espaço de novos literais no processo de revisão de BLPs, em ambos algoritmos *hill climbing* e *relational pathfinding* (RICHARDS, MOONEY, 1992). A seguir, abordamos a seleção de pontos de revisão, ao usar o algoritmo Bayes Ball (SHACHTER, 1998) para encontrar os nós que de fato influenciam uma instância prevista incorretamente. Operadores de revisão tem como meta ou mudar a influência corrente ou trazer nova influência através de modificações nas cláusulas referentes aqueles nós, tal que instâncias classificadas incorretamente possam se tornar corretas. Por último, atacamos o espaço de inferência do sistema de revisão. Primeiro, apenas os nós considerados como indispensáveis são levados em consideração ao computar a distribuição de probabilidade de um nó. O conjunto de nós indispensáveis também é identificado pelo algoritmo Bayes Ball. Segundo, cláusulas instanciadas com características idênticas são sobrepostas na rede de um exemplo. Finalmente, conjuntos de nós indispensáveis idênticos de exemplos diferentes são sobrepostos, tal que a distribuição de probabilidade seja computada apenas uma vez para todos eles. Mostramos que com essas otimizações o tempo de revisão é significativamente reduzido, comparado com o sistema PFORTE e também com o algoritmo de aprendizado de BLPs.

5.2 Trabalhos futuros: Revisão de Teorias de Primeira Ordem

O tempo de execução da revisão de teorias lógicas de primeira-ordem pode ser ainda mais reduzido se usarmos técnicas desenvolvidas em ILP para redução de complexidade da inferência. Por exemplo, estratégias que transformam consultas, desenvolvidas em (SANTOS COSTA, et al., 2003b) podem reduzir o tempo de inferência ao tornar as cláusulas mais eficientes de serem avaliadas. Além disso, existe uma grande necessidade de pontuar hipóteses de forma mais eficiente, especialmente na

presença de um grande conjunto de exemplos. Técnicas desenvolvidas para tornar o teste de *subsumption* mais eficiente (KUZELKA, ZELEZNÝ, 2008b), (KUZELKA, ZELEZNÝ, 2008a), (SANTOS, MUGGLETON, 2011) e técnicas de aprendizado incremental (LOPES, ZAVERUCHA, 2009) podem ajudar nessa direção.

O tempo de execução dos algoritmos estocásticos podem variar bastante, dependendo das escolhas aleatórias feitas ao longo do processo de revisão. Para evitar ficar preso por muito tempo em uma busca improdutiva podem ser usadas estratégias que *reiniciam* a busca a partir de diferentes pontos, na tentativa de retornar em um ponto mais promissor. A estratégia desenvolvida em (ŽELEZNÝ, et al., 2006) pode ser aplicada no processo de revisão, em que depois de uma certa quantidade de hipóteses serem avaliadas sem sucesso, a busca escolhe outro ponto para reiniciar. Adicionalmente, busca estocástica também pode ser usada para estimar a cobertura de cláusulas (SEBAG, ROUVEIROL, 1997).

Ainda existe a necessidade do desenvolvimento e descoberta de aplicações desafiantes e que se encaixem bem em revisão de teoria, assim como fizemos com a aplicação do xadrez. Áreas como aprendizado de gramáticas, processamento de linguagem natural e biologia contem informação codificada o suficiente para fornecerem bons exemplos de problemas a serem manipulados por revisão de teoria.

5.3 Trabalhos futuros: Revisão de Modelos lógico-probabilísticos

5.3.1 Revisão de Programas Lógicos Bayesianos

Inferência executada em redes Bayesianas podem dominar o tempo de execução do processo de revisão, uma vez que esse é o procedimento executado mais vezes no sistema. Mostramos nessa tese que o tempo de inferência pode ser reduzido ao evitarmos computações repetidas e ao reduzirmos o espaço de inferência. Para melhorar ainda mais o tempo de inferência, torna-se necessário explorar técnicas mais sofisticadas de *inferência levantada* (POOLE, 2003), (KOK, DOMINGOS, 2009), (KERSTING, et al., 2010). Além disso, similar ao caso de primeira-ordem, busca local estocástica pode ser muito útil para explorar os espaços de busca dos diversos componentes da revisão de BLPs de forma mais eficiente.

Na presente tese, abordamos a revisão usando um ponto de vista *discriminativo*: instâncias em exemplos recebem uma distribuição de probabilidade e, caso a probabilidade da classe da instância não seja satisfatória, cláusulas Bayesianas influenciando o cálculo daquela distribuição são revisadas. Uma questão que surge é se a revisão também pode ser útil em uma abordagem *generativa*. Nesse caso, a necessidade de revisão poderia ser indicada por, digamos, uma pontuação generativa baixa, como por exemplo, baixa verossimilhança. Dessa forma, exemplos não seriam mais separados em variáveis de consulta e evidências, mas todas elas seriam consideradas como de mesma importância, similar ao que é feito na configuração de *aprendizado a partir de interpretações*.

5.3.2   Revisão de Programas Lógicos Estocásticos

Programas Lógicos Bayesianos codificam distribuições de probabilidade sobre mundos possíveis, ao associar distribuições de probabilidade para possíveis valores de átomos. Programas Lógicos Estocásticos (MUGGLETON, 2002), por outro lado, seguem uma semântica distribucional baseada na frequência do domínio (HALPERN, 1989), ao associar distribuições de probabilidade à átomos básicos através de árvores de prova. Um SLP é composto por um conjunto de cláusulas estocásticas no formato $p : c$, onde c é uma cláusula definida e p é um rótulo de probabilidade. A soma dos rótulos de probabilidade de todas as cláusulas com o mesmo predicado na cabeça deve ser um. SLPs combinam programas lógicos definidos com Gramáticas Probabilísticas Livres de Contexto.

Nós projetamos um sistema chamado de SCULPTOR para revisar Programas Lógicos Estocásticos. Exemplos no SCULPTOR possuem um rótulo de probabilidade, da mesma forma que (CHEN, et al., 2008) e este rótulo é usado para computar o erro médio quadrático (RMSE). Exemplos com alto RMSE são escolhidos para indicar as cláusulas que precisam ser revisadas. Operadores de revisão incluem revisão dos rótulos de probabilidade, exclusão e adição de regras. Novas cláusulas são criadas usando *unfolding* (SATO, 1992) em cláusulas existentes e em cláusulas pertencentes a uma teoria top (MUGGLETON, et al., 2008).

Um primeiro protótipo do sistema SCULPTOR está implementado, requerendo

ainda ser testado e avaliado experimentalmente. Gostaríamos de aplicar o sistema SCULPTOR a desafiante tarefa de revisar estratégias de jogos.

Para finalizar, as principais realizações dessa tese foram (1) tornar a Revisão de Teorias a Partir de Exemplos tão eficiente quanto Programação em Lógica Indutiva, porém mais acurada; (2) aplicar a revisão de teorias em uma aplicação de xadrez onde ILP padrão falha; (3) atualizar nosso sistema de revisão de Programas Lógicos Bayesianos para que o mesmo fosse aplicado efetivamente em problemas do mundo real. Acredito que a revisão de teorias lógicas e probabilísticas de primeira-ordem tem potencial para manipular aplicações do mundo real mais complexas do que as técnicas mais simples de aprendizado de máquina. Com essa tese, conseguimos mostrar que isso pode ser alcançado em um tempo praticável.

Referências Bibliográficas

ADÉ, H., MALFAIT, B., RAEDT, L. D., 1994, "RUTH: an ILP Theory Revision System", In:*8th International Symposium on Methodologies for Intelligent Systems ISMIS-94*, volume 869 of *LNCS*, pp. 336–345 Springer.

ALLEN, T., GREINER, R., 2000, "Model selection criteria for learning belief nets: An empirical comparison", In:*Proceedings of the 17th International Conference on Machine Learning ICML-00*, pp. 1047–1054.

ALPHONSE, É., ROUVEIROL, C., 2000, "Lazy Propositionalisation for Relational Learning", In:*Proceedings of the 14th European Conference on Artificial Intelligence ECAI-00*, pp. 256–260.

ANGLUIN, D., FRAZIER, M., PITT, L., 1992, "Learning Conjunctions of Horn Clauses", *Machine Learning*, v. 9, pp. 147–164.

BÄCK, T., 1996, *Evolutionary Algorithms in Theory and Practice*, Oxford University, New York.

BAFFES, P., 1994, *Automatic Student Modeling and Bug Library Construction using Theory Refinement*, Ph.D. thesis, University of Texas, Austin, TX.

BAFFES, P., MOONEY, R., 1993, "Symbolic revision of theories with M-of-N rules", In:*Proceedings of the 13th International Joint Conference on Artificial Intelligence IJCAI-93*, pp. 1135–1140, Chambery, France.

BAIÃO, F., MATTOSO, M., SHAVLIK, J., ZAVERUCHA, G., 2003, "Applying Theory Revision to the Design of Distributed Databases", In:*Proceedings of*

the 13th International Conference on Inductive Logic Programming ILP 2003, volume 2835 of *LNAI*, pp. 57–74 Springer.

BAIN, M., 2004, "Predicate Invention and the Revision of First-Order Concept Lattice", In:*2nd International Conference on Formal Concept Analysis ICFCA 2004*, volume 2961 of *LNAI*, pp. 329–336 Springer.

BAIN, M., MUGGLETON, S., 1994, "Learning Optimal Chess Strategies", *Machine Intelligence*, v. 13, pp. 291–309.

BATTLE, A., SEGAL, E., KOLLER, D., 2004, "Probabilistic discovery of overlapping cellular processes and their regulation", In:*RECOMB*, pp. 167–176.

BINDER, J., KOLLER, D., RUSSELL, S. J., KANAZAWA, K., 1997, "Adaptive probabilistic networks with hidden variables", *Machine Learning*, v. 29, n. 2-3, pp. 213–244.

BLOCKEEL, H., DE RAEDT, L., 1998, "Top-Down Induction of First-Order Logical Decision Trees", *Artificial Intelligence*, v. 101, n. 1-2, pp. 285–297.

BOOTH, J. G., HOBERT, J. P., 1999, "Maximizing generalized linear mixed model likelihoods with an automated Monte Carlo EM algorithm", *Journal of the Royal Statistical Society, Series B*, v. 61, pp. 265–285.

BRATKO, I., 1999, "Refining Complete Hypotheses in ILP", In:*Proceedings of the 9th Inductive Logic Programming ILP-99*, volume 1634 of *LNAI*, pp. 44–55 Springer.

BRATKO, I., KING, R. D., 1994, "Applications of Inductive Logic Programming", *SIGART Bulletin*, v. 5, n. 1, pp. 43–49.

BUNESCU, R. C., MOONEY, R. J., 2004, "Collective Information Extraction with Relational Markov Networks", In:*ACL*, pp. 438–445.

BUNTINE, W., 1991, "Theory Refinement on Bayesian Networks", In:*Proceedings of the Seventeenth Annual Conference on Uncertainty in Artificial Intelligence UAI-91*, pp. 52–60, San Mateo, CA.

BURG, D. B., JUST, T., 1987, *IUS Chess Federation Official Rules of Chess*, McKay, New York, USA.

CAFFO, B. S., JANK, W. S., JONES, G. L., 2005, "Ascent-Based Monte Carlo EM", *Journal of the Royal Statistical Society, Series B*, v. 67, pp. 235–252.

CAI, D. M., GOKHALE, M., THEILER, J., 2007, "Comparison of feature selection and classification algorithms in identifying malicious executables", *Computational Statistics and Data Analysis*, v. 51, n. 6.

CARUANA, R., 1997, "Multitask Learning", *Machine Learning*, v. 28, n. 1, pp. 41–75.

CELEUX, G., CHAUVEAU, D., DIEBOLT, J., , "On Stochastic Versions of the EM Algorithm", Technical Report RR-2514.

CELEUX, G., DIEBOLT, J., 1985, "The SEM algorithm: a probabilistic teacher algorithm derived from the EM algorithm for the mixture problem", *Comput. Statist. Quater.*, v. 2, pp. 73–82.

CERNY, V., 1985, "A Thermodynamical Approach to the Traveling Salesman Problem", *Journal of Optimization Theory and Applications*, v. 45, n. 1, pp. 41–51.

CHAN, D., 1988, "Constructive Negation Based on the Completed Database", In:*Proc. of the 5th Int. Conf. and Symposium on Logic Programming*, pp. 111–125 The Mit Press.

CHEN, J., MUGGLETON, S., SANTOS, J. C. A., 2008, "Learning probabilistic logic models from probabilistic examples", *Machine Learning*, v. 73, n. 1, pp. 55–85.

CHENG, J., GREINER, R., KELLY, J., BELL, D. A., LIU, W., 2002a, "Learning Bayesian Networks from Data: An Information-Theory Based Approach", *Artificial Intelligence*, v. 137, n. 1-2, pp. 43–90.

CHENG, J., HATZIS, C., HYASHI, H., KROGEL, M.-A., MORISHITA, S., PAGE, D., , SESE, L., 2002b, "KDD Cup 2001 report", *SIGKDD Explorations*, v. 3, n. 2, pp. 47–64.

CHICKERING, D. M., 2002, "Optimal Structure Identification With Greedy Search", *Journal of Machine Learning Research*, v. 3, pp. 507–554.

CHICKERING, M., HECKERMAN, D., 1997, "Efficient approximations for the marginal likelihood of Bayesian Networks with hidden variables", *Machine Learning*, v. 29, pp. 181–212.

CHISHOLM, M., TADEPALLI, P., 2002, "Learning decision rules by randomized iterative local search", In:*Proceedings of the 19th International Conference on Machine Learning ICML-02*, pp. 75–82.

CLARK, K., 1978, "Negation as Failure Rule", In:Gallaire, H., Minker, J., , editors, *Logic and Data Bases*, pp. 293–322 Plenum Press.

COHEN, W., 1992, "Compiling prior knowledge into an explicit bias", In:*Proceedings of the Nineteenth International Conference on Machine Learning ICML-92*, pp. 102–110, Aberdeen, Scotland.

COHEN, W. W., 1995, "Fast Effective Rule Induction", In:*Proceedings of 12th International Conference on Machine Learning ICML-95*, pp. 115–123 Morgan Kaufmann.

COLLET, P., RENNARD, J.-P., 2007, "Stochastic Optimization Algorithms", *CoRR*, v. abs/0704.3780.

COOPER, G. F., 1990, "The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks", *Artificial Intelligence*, v. 42, n. 2-3, pp. 393–405.

COWELL, R. G., DAWID, A. P., LAURITZEN, S. L., SPIEGELHALTER, D. J., 1999, *Probabilistic Networks and Expert Systems*, Springer, New York.

CUSSENS, J., 2001, "Parameter Estimation in Stochastic Logic Programs", *Machine Learning*, v. 44, n. 3, pp. 245–271.

DAGUM, P., LUBY, M., 1993, "Approximating Probabilistic Inference in Bayesian Belief Networks is NP-Hard", *Artificial Intelligence*, v. 60, n. 1, pp. 141–153.

DARWICHE, A., 2010, "Bayesian networks", *Communications of the ACM*, v. 53, n. 12, pp. 80–90.

DAVID HECKERMAN, C. M., KOLLER, D., 2004, "Probabilistic Models for Relational Data", Technical Report MSR-TR-2004-30, Microsoft Research.

DAVIS, J., BURNSIDE, E. S., DUTRA, I. D. C., PAGE, D., RAMAKRISHNAN, R., SANTOS COSTA, V., , SHAVLIK, J. W., 2005a, "View Learning for Statistical Relational Learning: With an Application to Mammography", In:*Proceedings of the 19th International Joint Conference on Artificial Intelligence IJCAI-05*, pp. 677–683.

DAVIS, J., BURNSIDE, E. S., DUTRA, I. D. C., PAGE, D., SANTOS COSTA, V., 2005b, "An Integrated Approach to Learning Bayesian Networks of Rules", In:*Proceedings of the 16th European Conference on Machine Learning ECML-05*, volume 3720 of *LNAI*, pp. 84–95 Springer.

DAVIS, J., GOADRICH, M., 2006, "The relationship between Precision-Recall and ROC curves", In:*Proceedings of the 23rd International Conference on Machine Learning ICML 2006*, pp. 233–240.

DAVIS, J., ONG, I. M., STRUYF, J., BURNSIDE, E. S., PAGE, D., SANTOS COSTA, V., 2007, "Change of Representation for Statistical Relational Learning", In:*Proceedings of the 20th International Joint Conference on Artificial Intelligence IJCAI-07*, pp. 2719–2726.

CAMPOS, L. M. D., 2006, "A Scoring Function for Learning Bayesian Networks based on Mutual Information and Conditional Independence Tests", *Journal of Machine Learning Research*, v. 7, pp. 2149–2187.

DE RAEDT, L., 1996, *Advances in Inductive Logic Programming*, IOS Press.

DE RAEDT, L., 1997, "Logical Settings for Concept-Learning", *Artificial Intelligence*, v. 95, n. 1, pp. 187–201.

DE RAEDT, L., 2008, *Logical and Relational Learning*, Springer, Berlin, German.

DE RAEDT, L., BRUYNOOGHE, M., 1993, "A theory of clausal discovery", In:*Proceedings of the 13th International Joint Conference on Artificial Intelligence IJCAI-93*, pp. 1058–1063.

DE RAEDT, L., DEHASPE, L., 1997, "Clausal Discovery", *Machine Learning*, v. 2, n. 3, pp. 99–146.

DE RAEDT, L., DŽEROSKI, S., 1994, "First-Order jk-Clausal Theories are PAC-Learnable", *Artificial Intelligence*, v. 70, n. 1-2, pp. 375–392.

DE RAEDT, L., Frasconi, P., Kersting, K., Muggleton, S., , editors2008a, *Probabilistic Inductive Logic Programming – Theory and Applications,*, volume 4911 of *LNAI* Springer.

DE RAEDT, L., KERSTING, K., 2004, "Probabilistic Inductive Logic Programming", In:*Proceedings of the 15th International Conference on Algorithmic Learning Theory ALT-2004, Invited paper in S. Ben-David, J. Case and A. Maruoka, editors*, pp. 19–36.

DE RAEDT, L., KERSTING, K., KIMMIG, A., REVOREDO, K., TOIVONEN, H., 2008b, "Compressing Probabilistic Prolog Programs", *Machine Learning*, v. 70, n. 2-3, pp. 151–168.

DE RAEDT, L., KERSTING, K., TORGE, S., 2005, "Towards Learning Stochastic Logic Programs from Proof-Banks", In:*Proceedings of the 20th National Conference on Artificial Intelligence AAAI-05*, pp. 752–757.

DE RAEDT, L., KIMMIG, A., TOIVONEN, H., 2007, "ProbLog: A Probabilistic Prolog and Its Application in Link Discovery", In:*Proceedings of the 20th International Joint Conference on Artificial Intelligence IJCAI-2007*, pp. 2462–2467.

DEGROOT, M. H., 1989, *Probability and Statistics*, Addison-Wesley.

DEMPSTER, A. P., LAIRD, N. M., RUBIN, D. B., 1977, "Maximum likelihood from incomplete Data via the EM algorithm", *Journal of the Royal Statistical Society*, v. 39, n. 1, pp. 1–38.

DIETTERICH, T., 1998, "Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms", *Neural Computation*, v. 10, pp. 1895–1924.

DIETTERICH, T., DOMINGOS, P., GETOOR, L., MUGGLETON, S., TADE-PALLI, P., 2008, "Structured Machine Learning: The Next Ten Years", *Machine Learning*, v. 73, pp. 3–23.

DÍEZ, F. J., GALÁN, S. F., 2002, "Efficient computation for the noisy max", *International Journal of Intelligent Systems*, v. 18, pp. 165–177.

DIMOPOULOS, Y., KAKAS, A., 1995, "Learning Non-Monotonic Logic Programs: Learning Exceptions", In:*Proceedings of the 8th European Conference on Machine Learning ECML-95*, volume 912 of *LNAI*, pp. 122–138 Springer.

DOMINGOS, P., LOWD, D., 2009, *Markov Logic: An Interface Layer for Artificial Intelligence,*, Synthesis Lectures on Artificial Intelligence and Machine Learning Morgan & Claypool.

DOMINGOS, P., PAZZANI, M. J., 1997, "On the Optimality of the Simple Bayesian Classifier under Zero-One Loss", *Machine Learning*, v. 29, n. 2-3, pp. 103–130.

DRABENT, W., 1995, "What is Failure? An Approach to Constructive Negation", *Acta Inf.*, v. 32, n. 1, pp. 27–29.

DRABENT, W., 1996, "Completeness of SLDNF-Resolution for Nonfloundering Queries", *Journal of Logic Programming*, v. 27, n. 2, pp. 89–106.

DUBOC, A. L., 2008, *Tornando a Revisão de Teorias de Primeira-ordem mas eficiente através da utilização da cláusula mais específica*, Master's thesis, COPPE - UFRJ, Rio de Janeiro, RJ.

DUBOC, A. L., PAES, A., ZAVERUCHA, G., 2008, "Using the Bottom Clause and Modes Declarations on FOL Theory Revision from Examples", In:*Proceedings of the 18th International Conference on ILP ILP-08*, volume 5194 of *LNAI*, pp. 91–106 Springer.

DUBOC, A. L., PAES, A., ZAVERUCHA, G., 2009, "Using the bottom clause and mode declarations in FOL theory revision from examples", *Machine Learning*, v. 76, n. 1, pp. 73–107.

DZEROSKI, S., BRATKO, I., 1992, "Handling Noise in Inductive Logic Programming", In:*Proceedings of the 2nd International Workshop on Inductive Logic Programming*.

Dzeroski, S., Lavrac, N., , editors2001, *Relational Data Mining*, Springer-Verlag.

ELIDAN, G., FRIEDMAN, N., 2005, "Learning Hidden Variable Networks: The Information Bottleneck Approach", *Journal of Machine Learning Research*, v. 6, pp. 81–127.

ELIDAN, G., LOTNER, N., FRIEDMAN, N., KOLLER, D., 2000, "Discovering Hidden Variables: A Structure-Based Approach", In:*Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems NIPS 2000*, pp. 479–485 MIT Press.

ESPOSITO, F., SEMERARO, G., FANIZZI, N., FERILLI, S., 2000, "Multistrategy Theory Revision: Induction and Abduction in INTHELEX", v. 38, pp. 133–156.

FANG, H., TONG, W., SHI, L. M., BLAIR, R., PERKINS, R., BRANHAM, W., HASS, B. S., XIE, Q., DIAL, S. L., MOLAND, C. L., , SHEEHAN, D. M., 2001, "Structure-Activity Relationships for a Large Diverse Set of Natural, Synthetic, and Environmental Estrogens", *Chemical Research in Toxicology*, v. 3, n. 14, pp. 280–294.

FELLER, W., 1970, *An Introduction to Probability Theory and Its Applications*, third ed.,, volume 1 John Wiley & Sons.

FENSEL, D., ZICKWOLFF, M., WIESE, M., 1995, "Are substitutions the better examples?", In:*Proceedings of the 15th International Conference on Inductive Logic Programming ILP-95*, pp. 120–127.

FEO, T., RESENDE, M., 1995, "Greedy Randomised Adaptive Search Procedures", *Journal of Global Optimization*, , n. 6, pp. 109–133.

FIDE, "FIDE Laws of Chess".

FIERENS, D., BLOCKEEL, H., BRUYNOOGHE, M., RAMON, J., 2005, "Logical Bayesian Networks and Their Relation to Other Probabilistic Logical Models", In:*15th International Conference on Inductive Logic Programming*, pp. 121–135.

FLACH, P., 1994, *Simply Logical - Intelligent Reasoning by Example*, John Wiley & Sons.

FLACH, P., KAKAS, A., 2000a, *Abduction and Induction: Essays on their Relation and Integration*, Kluwer Academic Publishers.

FLACH, P., KAKAS, A., 2000b, "Abductive and inductive reasoning: Background and issues", In:Flach, P. A., Kakas, A. C., , editors, *Abductive and Inductive Reasoning* Kluwer Academic Publishers.

FOGEL, L., ZAVERUCHA, G., 1998, "Normal Programs and Multiple Predicate Learning", In:*Proc. of the 8th Int. Conf. on ILP*, volume 1446 of *LNAI*, pp. 175–184 Springer.

FRANK, E., WITTEN, I. H., 1998, "Generating Accurate Rule Sets Without Global Optimization", In:*Proceedings of 15th International Conference on Machine Learning ICML-98*, pp. 144–151 Morgan Kaufmann.

FRAZIER, M., PITT, L., 1993, "Learning From Entailment: An Application to Propositional Horn Sentences", In:*Proceedings of the Tenth International Conference on Machine Learning ICML-93*, pp. 120–127.

FRIEDMAN, N., 1998, "The Bayesian Structural EM Algorithm", In:*Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence UAI-98*, pp. 129–138 Morgan Kaufmann.

FRIEDMAN, N., GEIGER, D., GOLDSZMIDT, M., 1997, "Bayesian network classifiers", *Machine Learning*, v. 29, pp. 131–163.

FRIEDMAN, N., GETOOR, L., KOLLER, D., PFEFFER, A., 1999, "Learning Probabilistic Relational Models", In:*Proceedings of the 16th International Joint Conference on Artificial Intelligence IJCAI-99*, pp. 1300–1309.

FÜRNKRANZ, J., 1996, "Machine Learning in Computer Chess: The Next Generation", *Int. Computer Chess Association Journal*, v. 19, n. 3, pp. 147–161.

FÜRNKRANZ, J., 2007, "Recent Advances in Machine Learning and Game Playing", *OGAI-Journal*, v. 26, n. 2, pp. 147–161.

GARCEZ, A., ZAVERUCHA, G., 1999, "The Connectionist Inductive Learning and Logic Programming System", *Applied Intelligence*, v. 11, pp. 59–77.

GEIGER, D., VERMA, T., PEARL, J., 1989, "d-Separation: From Theorems to Algorithms", In:*Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence UAI-89*, pp. 139–148 North-Holland.

GEIGER, D., VERMA, T., PEARL, J., 1990, "Identifying independence in Bayesian networks", *Networks*, v. 20, pp. 507–534.

GEMAN, S., GEMAN, D., 1984, "Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. 6, n. 6, pp. 721–741.

GETOOR, L., 2007, "An introduction to Statistical Relational Learning", In:*Tutorial Notes of the 18th European Conference on Machine Learning ECML and the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases PKDD.*

GETOOR, L., FRIEDMAN, N., KOLLER, D., TASKAR, B., 2001, "Learning Probabilistic Models of Relational Structure", In:*Proceedings of the 8th International Conference on Machine Learning ICML-2001*, pp. 170–177.

GETOOR, L., GRANT, J., 2006, "PRL: A probabilistic relational language", *Machine Learning*, v. 62, n. 1-2, pp. 7–31.

GETOOR, L., RHEE, J. T., KOLLER, D., SMALL, P., 2004, "Understanding Tuberculosis Epidemiology Using Structured Statistical Models", *Artificial Intelligence in Medicine*, v. 30, n. 3, pp. 233–256.

Getoor, L., Taskar, B., , editors2007, *An Introduction to Statistical Relational Learning*, MIT Press.

GILKS, W., RICHARDSON, S., SPIEGELHALTER, D., 1995, *Markov Chain Monte Carlo in Practice: Interdisciplinary Statistics*, Chapman & Hall/CRC.

GLOVER, F., 1989, "Tabu Search - Part I", *INFORMS Journal on Computing*, v. 1, n. 3, pp. 190–206.

GLOVER, F., 1990, "Tabu Search - Part II", *INFORMS Journal on Computing*, v. 2, n. 1, pp. 4–32.

GOADRICH, M., OLIPHANT, L., SHAVLIK, J. W., 2004, "Learning Ensembles of First-Order Clauses for Recall-Precision Curves: A Case Study in Biomedical Information Extraction", In:*Proceedings of the 14th International Conference on ILP ILP-2004*, volume 3194 of *LNAI*, pp. 98–115 Springer.

GOLDBERG, D. E., 1989, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley.

GOMES, C. P., SELMAN, B., CRATO, N., KAUTZ, H., 2000, "Heavy-Tailed Phenomena in Satisfiability and Constraint Satisfaction Problems", *Journal of Automated Reasoning*, v. 24, n. 1-2, pp. 67–100.

GOMES, C. P., SELMAN, B., MCALOON, K., TRETKOFF, C., 1998, "Randomization in Backtrack Search: Exploiting Heavy-Tailed Profiles for Solving Hard Scheduling Problems.", In:*AIPS*, pp. 208–213.

GOODACRE, J., 1996, "Master Thesis, Inductive Learning of Chess Rules Using Progol. Programming Research Group, Oxford University".

GREINER, R., ZHOU, W., 2002, "Structural extension to logistic regression: Discriminative parameter learning of belief net classifiers", In:*Proceedings of the 18th Annual National Conference on Artificial Intelligence AAAI-02*, pp. 167–173.

GROSSMAN, D., DOMINGOS, P., 2004, "Learning Bayesian Network Classifiers by Maximizing Conditional Likelihood", In:*Proceedings of the 21st International Conference on Machine Learning ICML-04*, pp. 361–368.

GU, J., 1992, "Efficient Local Search for Very Large-scale Satisfiability Problems", *SIGART Bulletin*, v. 3, n. 1, pp. 8–12.

GYFTODIMOS, E., FLACH, P. A., 2004, "Hierarchical Bayesian Networks: An Approach to Classification and Learning for Structured Data", In:*Methods and Applications of Artificial Intelligence, 3rd Helenic Conference on AI SETN 2004*, volume 3025 of *LNAI*, pp. 291–300 Springer.

HADDAWAY, P., 1999, "An Overview of Some Recent Developments on Bayesian Problem Solving Techniques", *AI Magazine - Special issue on Uncertainty in AI*, v. 20, n. 2, pp. 11–29.

HALL, M. A., 2000, "Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning", In:*Proceedings of the 17th International Conference on Machine Learning ICML 2000*, pp. 359–366.

HALL, M. A., SMITH, L. A., 1999, "Feature Selection for Machine Learning: Comparing a Correlation-Based Filter Approach to the Wrapper", In:*Proceedings of the 12th International Florida Artificial Intelligence Research Society Conference*, pp. 235–239.

HALPERN, J. Y., 1989, "An analysis of first-order logics of probability", *Artificial Intelligence*, v. 46, pp. 311–350.

HECKERMAN, D., 1996, "A tutorial on learning with Bayesian networks", Technical Report MSR-TR-95-06, Microsoft Research, http://research.microsoft.com/ heckerman/.

HECKERMAN, D., BREESE, J., 1994, "Causal Independence for Probability Assessment and Inference using Bayesian Networks", Technical Report MSR-TR-94-08, Microsoft Research, http://research.microsoft.com/ heckerman/.

HILDEN, J., 1984, "Statistical Diagnosis based on Conditional Independence does not Require it", *Compu. Biol. Med.*, v. 14, n. 4, pp. 429–435.

HIRST, J. D., KING, R. D., STERNBERG, M. J. E., 1994a, "Quantitative structure-activity relationships by neural networks and inductive logic programming. I. The inhibition of dihydrofolate reductase by pyrimidines", *Journal of Computer-Aided Molecular Design*, v. 8, n. 4, pp. 405–420.

HIRST, J. D., KING, R. D., STERNBERG, M. J. E., 1994b, "Quantitative structure-activity relationships by neural networks and inductive logic programming. II. The inhibition of dihydrofolate reductase by triazines", *Journal of Computer-Aided Molecular Design*, v. 8, n. 4, pp. 421–432.

HOOPER, D., WHYLD, K., 1992, *The Oxford Companion to Chess*, Oxford University Press.

HOOS, H. H., STÜTZLE, T., 2005, *Stochastic Local Search: Foundations and Applications*, 1 ed., Elsevier, California, USA.

HUTTER, F., HOOS, H. H., STÜTZLE, T., 2005, "Efficient Stochastic Local Search for MPE Solving", In:*Proceedings of the 19th International Joint Conference on Artificial Intelligence IJCAI-05*, pp. 169–174.

HUYNH, T. N., MOONEY, R. J., 2008, "Discriminative structure and parameter learning for Markov logic networks", In:*Proceedings of the 25th International*

Conference on Machine Learning (ICML 2008), volume 307 of *ACM International Conference Proceeding Series*, pp. 416–423 ACM.

INOUE, K., 2001, "Induction, Abduction, and Consequence-Finding", In:*Proceedings of the 11th International Conference on Inductive Logic Programming*, volume 2157 of *LNCS*, pp. 65–79 Springer.

JAEGER, M., 1997, "Relational Bayesian networks", In:*Proceedings of the 13th Annual Conference on Uncertainty in Artificial Inteliigence UAI-97*, pp. 266–273 Morgan Kaufmann.

JAIMOVICH, A., ELIDAN, G., MARGALIT, H., FRIEDMAN, N., 2005, "Towards an Integrated Protein-Protein Interaction Network", In:*RECOMB*, pp. 14–30.

JANK, W., 2006, "The EM Algorithm, Its Randomized Implementation and Global Optimization: Some Challenges and Opportunities for Operations Research", In:Francis B. Alt, M. C. F., Golden, B. L., , editors, *Perspectives in Operations Research*, pp. 367–392 Springer.

JENSEN, D., NEVILLE, J., 2002, "Linkage and autocorrelation cause feature selection bias in relational learning", In:*Proceedings of the Nineteenth International Conference on Machine Learning ICML-02, Morgan Kaufmann*, pp. 259–266.

JENSEN, D., NEVILLE, J., BRIANGALLAGHER, 2004, "Why collective inference improves relational classification", In:*Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '04, pp. 593–598 ACM.

JENSEN, F. V., 1996, *An introduction to Bayesian networks*, Springer, New York.

JENSEN, F. V., 2001, *Bayesian Networks and Decision Graphs*, Springer-Verlag.

JOHN, G. H., KOHAVI, R., PFLEGER, K., 1994, "Irrelevant Features and the Subset Selection Problem", In:*Proceedings of the 11th International Conference on Machine Learning ICML-1994*, pp. 121–129.

JOSHI, S., RAMAKRISHNAN, G., SRINIVASAN, A., 2008, "Feature Construction Using Theory-Guided Sampling and Randomised Search", In:*Proceedings of the 18th International Conference on ILP*, pp. 140–157.

JUŽELKA, O., ŽELEZNÝ, F., 2007, "A Restart Strategy for Fast Subsumption Check and Coverage Estimation", In:*Proceedings of the 6th Workshop on Multi-Relational Data Mining MRDM 2007*.

KADUPITIGE, S. R., JULIA, K. C. L., SELLMEIER, SIVIENG, J., CATCH-POOLE, D. R., BAIN, M., , GAETA, B. A., 2009, "MINER: exploratory analysis of gene interaction networks by machine learning from expression data", *BMC Genomics*, v. 10, n. Suppl 3, pp. S17.

KAUTZ, H., SELMAN, B., JIANG, Y., 1996, "A general stochastic approach to solving problems with hard and soft constraints".

KEARNS, M. J., MANSOUR, Y., 1998, "Exact Inference of Hidden Structure from Sample Data in noisy-OR Networks", In:*Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence UAI-98*, pp. 304–310 Morgan Kaufmann.

KEARNS, M. J., VAZIRANI, U. V., 1994, *An Introduction to Computational Learning Theory*, Cambridge, Massachusetts.

KERSTING, K., 2006, *An Inductive Logic Programming Approach to Statistical Relational Learning„* volume 148 of *Frontiers in Artificial Intelligence and its Applications series Dissertations* IOS Press.

KERSTING, K., DE RAEDT, L., 2001a, "Adaptive Bayesian Logic Programs", In:*Proceedings of the 11th Conference on Inductive Logic Programming ILP-01*, volume 2157 of *LNAI*, pp. 104–117 Springer.

KERSTING, K., DE RAEDT, L., 2001b, "Bayesian Logic Programs", *CoRR*, v. cs.AI/0111058.

KERSTING, K., DE RAEDT, L., 2001c, "Bayesian Logic Programs", Technical Report 151, University of Freiburg, Institute for Computer Science, http://www.informatik.uni-freiburg.de/ kersting/publications.html.

KERSTING, K., DE RAEDT, L., 2001d, "Towards Combining Inductive Logic Programming with Bayesian Networks", In:*Proceedings of the 11th Conference on Inductive Logic Programming ILP-01*, volume 2157 of *LNAI*, pp. 118–131 Springer.

KERSTING, K., DE RAEDT, L., 2002, "Basic Principles of Learning Bayesian Logic Programs", Technical Report 174, University of Freiburg, Institute for Computer Science, http://www.informatik.uni-freiburg.de/ kersting/publications.html.

KERSTING, K., DE RAEDT, L., 2007, "Bayesian Logic Programming: Theory and Tool", In:Getoor, L., Taskar, B., , editors, *An Introduction to Statistical Relational Learning*, pp. 291–322.

KERSTING, K., DE RAEDT, L., RAIKO, T., 2006, "Logical Hidden Markov Models", *Journal of Artificial Intelligence Research JAIR*, v. 25, pp. 425–456.

KERSTING, K., MASSAOUDI, Y. E., HADIJI, F., AHMADI, B., 2010, "Informed Lifting for Message-Passing", In:*Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)* AAAI Press.

KERSTING, K., RAIKO, T., KRAMER, S., DE RAEDT, L., 2003, "Towards Discovering Structural Signatures of Protein Folds Based on Logical Hidden Markov Models", In:*Pacific Symposium on Biocomputing*, pp. 192–203.

KIMMIG, A., 2010, *A Probabilistic Prolog and its Applications*, Ph.D. thesis, Katholieke Universiteit Leuven, Belgium.

KIMMIG, A., SANTOS COSTA, V., ROCHA, R., DEMOEN, B., RAEDT, L. D., 2008, "On the Efficient Execution of ProbLog Programs", In:*24th International Conference on Logic Programming ICLP-08*, volume 5366 of *LNCS*, pp. 175–189 Springer.

KING, R. D., MUGGLETON, S., STERNBERG, M. J. E., 1992, "Drug design by machine learning: The use of inductive logic programming to model the

structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase", *Proc. of the National Academy of Sciences*, v. 89, n. 23, pp. 11322–11326.

KING, R. D., STERNBERG, M. J. E., SRINIVASAN, A., 1995a, "Relating Chemical Activity to Structure: An Examination of ILP Successes.", *New Generation Computing*, v. 13, n. 3-4, pp. 411–433.

KING, R. D., STERNBERG, M. J. E., SRINIVASAN, A., 1995b, "Relating Chemical Activity to Structure: An Examination of ILP Successes", *New Generation Computing*, v. 13, n. 3-4, pp. 411–433.

KING, R. D., WHELAN, K., JONES, F., REISER, P., BRYANT, C., MUGGLETON, S. H., KELL, D., , , OLIVER, S., 2004, "Functional Genomic Hypothesis Generation and Experimentation by a Robot Scientist", *Nature*, v. 427, pp. 247–252.

KIRKPATRICK, S., GELATT, C. D., VECCHI, M. P., 1983, "Optimization by simulated annealing", *Science*, v. 220, pp. 671–680.

KOHAVI, R., 1995, "A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection", In:*Proceedings of the 14th International Joint Conference on Artificial IntelligenceIJCAI-95*, pp. 1137–1145.

KOK, S., DOMINGOS, P., 2005, "Learning the structure of Markov logic networks", In:*Proceedings of the 22nd International Conference on Machine Learning ICML 2005*, volume 119 of *ACM International Conference Proceeding Series*, pp. 441–448 ACM.

KOK, S., DOMINGOS, P., 2007, "Statistical predicate invention", In:*Proceedings of the 24th International Conference on Machine Learning ICML-07*, pp. 433–440 ACM Press.

KOK, S., DOMINGOS, P., 2009, "Learning Markov logic network structure via hypergraph lifting", In:*Proceedings of the 26th Annual International Conference on*

Machine Learning, ICML 2009, volume 382 of *ACM International Conference Proceeding Series*, pp. 64 ACM.

KOLLER, D., 1999, "Probabilistic Relational Models", In:*Proceedings of the 9th International Conference on Inductive Logic Programming ILP-99*, pp. 3–13.

KOLLER, D., FRIEDMAN, N., 2009, *Probabilistc Graphical Models: Principles and Techniques*, MIT Press.

KOLLER, D., PFEFFER, A., 1997, "Learning Probabilities for Noisy First-Order Rules", In:*Proceedings of the 15th International Joint Conference on Artficial Intelligence IJCAI-97*, pp. 1316–1323.

KOLLER, D., PFEFFER, A., 1998, "Probabilistic Frame-Based Systems", In:*Proceedings of the 15th National Conference on Artificial Intelligence AAAI-98*, pp. 580–587.

KOLLER, D., SAHAMI, M., 1996, "Toward Optimal Feature Selection", In:*Proceedings of the 13th International Conference on Machine Learning ICML-96*, pp. 284–292.

KOPPEL, M., FELDMAN, R., SEGRE, A. M., 1994, "Bias-Driven Revision of Logical Domain Theories", *Journal of Artificial Intelligence Research JAIR*, v. 1, pp. 159–208.

KOVACIC, M., LAVRAC, N., GROBELNIK, M., ZUPANIC, D., MLADENIC, D., 1992, "Stochastic Search in Inductive Logic Programming", In:*Proceedings of the European Conference on Artificial Intelligence ECAI-92*, pp. 444–445.

KRAMER, S., 1995, "Predicate Invention: A Comprehensive View", Technical Report FAI-TR-95-32, Austrian Research Institute for Artificial Intelligence.

KROGEL, M.-A., RAWLES, S., ŽELEZNÝ, F., FLACH, P. A., LAVRAC, N., WROBEL, S., 2003, "Comparative Evaluation of Approaches to Propositionalization", In:*Proceedings of the 13th ILP*, volume 2835 of *LNAI*, pp. 197–214 Springer.

KUZELKA, O., ZELEZNÝ, F., 2008a, "Fast estimation of first-order clause coverage through randomization and maximum likelihood", In:*Proceedings of the 25th International Conference on Machine Learning (ICML-2008)*, volume 307 of *ACM International Conference Proceeding Series*, pp. 504–511 ACM.

KUZELKA, O., ZELEZNÝ, F., 2008b, "A Restarted Strategy for Efficient Subsumption Testing", *Fundam. Inform.*, v. 89, n. 1, pp. 95–109.

KUZELKA, O., ZELEZNÝ, F., 2011, "Seeing the World through Homomorphism: An Experimental Study on Reducibility of Examples", In:*Revised Papers of the 20th International Conference on Inductive Logic Programming (ILP-2010)*, volume 6489 of *LNAI*, pp. 138–145 Springer.

LAARHOVEN, P. J. V., ARTS, E. H. L., 1987, *Simulated annealing: Theory and applications*, Reidel Publishers, Amsterdam.

LACHICHE, N., FLACH, P. A., 2002, "1BC2: A True First-Order Bayesian Classifier", In:*Proceedings of the 12th International Conference on Inductive Logic Programming ILP-02*, volume 2583 of *LNAI*, pp. 133–148 Springer.

LAM, W., BACCHUS, F., 1994, "Learning Bayesian belief networks: an approach based on the MDL principle", *Computational Intelligence*, v. 10, n. 4, pp. 269–293.

LANDWEHR, N., KERSTING, K., DE RAEDT, L., 2007, "nFOIL: Integrating Naïve Bayes and FOIL", v. 8, pp. 481–507.

LANDWEHR, N., PASSERINI, A., RAEDT, L. D., FRASCONI, P., 2006, "kFOIL: Learning Simple Relational Kernels", In:*Proceedings of the 21st National Conference on Artificial Intelligence AAAI-2006*.

LANGLEY, P., 1995, *Elements of Machine Learning*, Morgan Kaufman.

LANGLEY, P., IBA, W., THOMPSON, K., 1992, "An Analysis of Bayesian Classifiers", In:*Proceedings of the 8th Annual National Conference on Artificial Intelligence AAAI-92*, pp. 223–228.

LAURITZEN, S. L., 1995, "The EM algorithm for graphical association models with missing data", *Computational Statistics and Data Analysis*, v. 19, pp. 191–201.

LAVRAC, N., DZEROSKI, S., 1994, *Inductive Logic Programming: Techniques and Applications*, Ellis Horwood, New York.

LAVRAC, N., DZEROSKI, S., 2001, *Relational Data Mining*, Springer, New York.

LEVY, D. N. L., NEWBORN, M., 1990, *How Computers Play Chess*,.

LI, Z., D'AMBROSIO, B., 1994, "Efficient Inference in Bayes Networks as a Combinatorial Optimization Problem", *International Journal of Approximate Reasoning*, v. 11, n. 1, pp. 55–81.

LLOYD, J., 1987, *Foundations of Logic Programming*, second ed., Springer.

LOPES, C., ZAVERUCHA, G., 2009, "HTILDE: scaling up relational decision trees for very large databases", In:*Proceedings of the 2009 ACM Symposium on Applied Computing (SAC)*, pp. 1475–1479 ACM.

LOURENĆO, H. R., O.MARTIN, STÜTZLE, T., 2002, "Iterated Local Search", In:Glover, F., Kochenberger, G., , editors, *Handbook f Metaheuristics*, pp. 321–353 Kluwer Academic Publishers.

MARRIOTT, K., SØNDERGAARD, H., DART, P., 1990, "A characterization of non-floundering logic programs", In:*Proceedings of the 1990 North American conference on Logic programming*, pp. 661–680 MIT Press.

MCCULLOCH, C. E., 1994, "Maximum likelihood variance components estimation for binary data", *Journal of the American Statistical Association*, v. 89, pp. 330–335.

MCLACHLAN, G. J., KRISHNAN, T., 1997, *The EM algorithm and Extensions*, 1 ed., Wiley Interscience, New York.

MEERT, W., TAGHIPOUR, N., BLOCKEEL, H., 2010, "First-Order Bayes-Ball", In:*Proceedings of the 22nd ECML/PKDD 2*, LNAI 6322, pp. 369–384 Springer.

MICHALSKI, R. S., LARSON, J., 1977, "Inductive Inference of VL decision rules", In:*Workshop in pattern-Directed Inference Systems, SIGART Newsletter*, volume 63, pp. 38–44 ACM.

MIHALKOVA, L., HUYNH, T., MOONEY, R. J., 2007, "Mapping and Revising Markov Logic Networks for Transfer Learning", In:*Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, pp. 608–614.

MIHALKOVA, L., MOONEY, R. J., 2007, "Bottom-up learning of Markov logic network structure", In:*Proceedings of the 24h International Conference on Machine Learning (ICML-07)*, volume 227 of *ACM International Conference Proceeding Series*, pp. 625–632 ACM.

MILCH, B., MARTHI, B., RUSSELL, S. J., SONTAG, D., ONG, D. L., KOLOBOV, A., 2005, "BLOG: Probabilistic Models with Unknown Objects", In:*Proceedings of the 19th International Joint Conference on Artificial Intelligence IJCAI-2005*, pp. 1352–1359.

MILCH, B., ZETTLEMOYER, L. S., KERSTING, K., HAIMES, M., KAELBLING, L. P., 2008, "Lifted Probabilistic Inference with Counting Formulas", In:*Proceedings of the 23rd AAAI Conference on Artificial Intelligence, AAAI 2008*, pp. 1062–1068 AAAI Press.

MITCHELL, M., 1996, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, USA.

MITCHELL, T., 1997, *Machine Learning*, McGraw-Hill, New York.

MOONEY, R., 1997, "Integrating Abdcution and Induction in Machine Learning", In:*Workshop on Abduction and Induction in AI, IJCAI-97*, pp. 37–42.

MOONEY, R., 2000, "Integrating Abduction and Induction in Machine Learning", In:Flach, P. A., Kakas, A. C., , editors, *Abduction and Induction*, pp. 181–191 Kluwer Academic Publishers.

MOYLE, S., 2003, "Using Theory Completion to Learn a Robot Navigation Control Program", In:*Proceedings of 12th International Conference on ILP*, volume 2583 of *LNAI*, pp. 182–197 Springer.

MUGGLETON, S., 1987, "Duce, An Oracle-based Approach to Constructive Induction", In:*Proceedings of the 10th International Joint Conference on Artificial IntelligenceIJCAI-87*, pp. 287–292.

MUGGLETON, S., 1991, "Inductive logic programming", *New Generation Computing*, v. 13, n. 4, pp. 245–286.

MUGGLETON, S., 1992, *Inductive logic programming*, Academic Press, New York.

MUGGLETON, S., 1994, "Predicate Invention and Utilisation", *Journal of Experimental and Theoretical Artficial Intelligence*, v. 6, n. 1, pp. 127–130.

MUGGLETON, S., 1995a, "Inverse Entailment and Progol", *New Generation Computing*, v. 13, pp. 245–286.

MUGGLETON, S., 1995b, "Inverse Entailment and Progol", *New Generation Computing*, v. 13, n. 3&4, pp. 245–286.

MUGGLETON, S., 1996, "Stochastic logic programs", In:RAEDT, L. D., , editor, *Advances in Inductive Logic Programming*, pp. 254–264 IOS Press.

MUGGLETON, S., 1999, "Scientific Knowledge Discovery Using Inductive Logic Programming", *Communications of the ACM*, v. 42, n. 11, pp. 42–46.

MUGGLETON, S., 2000, "Learning Stochastic Logic Programs", *Electron. Trans. Artif. Intell.*, v. 4, n. B, pp. 141–153.

MUGGLETON, S., 2002, "Learning structure and parameters of stochastic logic programs", In:*Proceedings of the 12th International Conference on Inductive Logic Programming ILP-02*, volume 2583 of *LNAI*, pp. 198–206 Springer.

MUGGLETON, S., 2005, "Machine Learning for Systems Biology", In:*Proceedings of the 15th International Conference on Inductive Logic Programming*, volume 3625 of *Lecture Notes in Computer Science*, pp. 416–423 Springer.

MUGGLETON, S., BRYANT, C. H., 2000, "Theory Completion Using Inverse Entailment", In:*Proceedings of the 10th International Conference on ILP*, volume 1866 of *LNAI*, pp. 130–146 Springer.

MUGGLETON, S., BUNTINE, W. L., 1988, "Machine Invention of First Order Predicates by Inverting Resolution", In:*Procedings of the Fifth International Conference on Machine Learning*, pp. 339–352.

MUGGLETON, S., DE RAEDT, L., 1994, "Inductive Logic Programming: Theory and Methods", *Journal of Logic Programming*, v. 19, n. 20, pp. 629–679.

MUGGLETON, S., FENG, C., 1990, "Efficient induction of logic programs", In:*Proceedings of the 1st Conference on Algorithmic Learning Theory COLT-90*, pp. 368–381.

MUGGLETON, S., PAES, A., SANTOS COSTA, V., ZAVERUCHA, G., 2009a, "Chess Revision: Acquiring the Rules of Chess Variants through FOL Theory Revision from Examples", In:*Proceedings of the IJCAI-09 Workshop on General Game Playing General Intelligence in Game-Playing Agents GIGA'09*, pp. 61–71.

MUGGLETON, S., PAES, A., SANTOS COSTA, V., ZAVERUCHA, G., 2009b, "Chess Revision: Acquiring the Rules of Chess Variants through FOL Theory Revision from Examples", In:*Proceedings of the 19th International Conference on ILP ILP-2009*, volume 5989 of *LNAI*, pp. 123–130 Springer.

MUGGLETON, S., PAES, A., SANTOS COSTA, V., ZAVERUCHA, G., 2009c, "Chess Revision: Acquiring the Rules of Chess Variants through FOL Theory Revision from Examples", In:*Workshop on Transfer Leaning - NIPS 2009*.

MUGGLETON, S., PAHLAVI, N., 2007, "Stochastic Logic Program: A Tutorial", In:Getoor, L., Taskar, B., , editors, *An Introduction to Statistical Relational Learning*, pp. 323–338.

MUGGLETON, S., SANTOS, J. C. A., TAMADDONI-NEZHAD, A., 2008,

"TopLog: ILP Using a Logic Program Declarative Bias", In:*Proceedings of the 24th International Conference on Logic Programming (ICLP-08)*, pp. 687–692.

MUGGLETON, S., SANTOS, J. C. A., TAMADDONI-NEZHAD, A., 2010, "Pro-Golem: A System Based on Relative Minimal Generalisation", In:*Proceedings of the 1th International Conference on Inductive Logic Programming (ILP-09)*, pp. 131–148.

MUGGLETON, S., TAMADDONI-NEZHAD, A., 2008, "QG/GA: a stochastic search for Progol", *Machine Learning*, v. 70, n. 2-3, pp. 121–133.

MUGGLETON, S. H., KING, R. D., STERNBERG, M. J. E., 1992, "Protein secondary structure prediction using logic-based machine learning", *Protein Engineering*, v. 5, n. 7, pp. 647–657.

Murphy1997, "Bayes Net Toolbox for Matlab. U.C. Berkeley.".

MURPHY, K., 2001, "The Bayes Net Toolbox for Matlab", *Computing Science and Statistics*, v. 33.

MURPHY, K. P., WEISS, Y., JORDAN, M. I., 1999, "Loopy Belief Propagation for Approximate Inference: An Empirical Study", In:*Proceedings of the 15th Conference on Uncertainty in Artificial Intelligence UAI-99*, pp. 467–475 Morgan Kaufmann.

NADEAU, C., BENGIO, Y., 2003, "Inference for the Generalization Error", *Machine Learning*, v. 53, n. 3, pp. 239–281.

NATARAJAN, S., TADEPALLI, P., DIETTERICH, T. G., FERN, A., 2008, "Learning First-Order Probabilistic Models with Combining Rules", *Annals of Mathematical Artificial Intelligence*, v. 54, n. 1-3, pp. 223–256.

NATH, A., DOMINGOS, P., 2010, "Efficient Lifting for Online Probabilistic Inference", In:*Proceedings of the 24th AAAI Conference on Artificial Intelligence, AAAI 2010* AAAI Press.

NEUMANN, J., SCHNöRR, C., STEIDL, G., , "Combined SVM-Based Feature Selection and Classification", *Machine Learning*.

NEVILLE, J., JENSEN, D., 2004, "Dependency Networks for Relational Data", In:*Proceedings of the 4th IEEE International Conference on Data Mining ICDM-2004*, pp. 170–177.

NEVILLE, J., SIMSEK, Ö., JENSEN, D., KOMOROSKE, J., PALMER, K., GOLDBERG, H. G., 2005, "Using Relational Knowledge Discovery to Prevent Securities Fraud", In:*Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD-05*, pp. 449–458 ACM.

NGO, L., HADDAWY, P., 1997, "Answering Queries from Context-Sensitive Probabilistic Knowledge Bases", *Theoretical Computer Science*, v. 171, n. 1-2, pp. 147–177.

NIELSEN, S., 2000, "The stochastic EM algorithm: Estimation and asymptotic results", *Journal of the Bernoulli Society for Mathematical Statistics and Probability*, v. 6, n. 3, pp. 457–489.

NIENHUYS-CHENG, SHAN-HWEI, De Wolf, R., 1997, *Foundations of Inductive Logic Programming*, Springer-Verlag.

NIENHUYS-CHENG, S.-H., De Wolf, R., 1997, *Foundations of Inductive Logic Programming*, Springer, Berlin.

NILSSON, U., MALUSZYNKSKI, J., 2000, *Logic Programming and PROLOG*, 2nd ed., John Wiley and Sons.

OLIPHANT, L., SHAVLIK, J. W., 2008, "Using Bayesian Networks to Direct Stochastic Search in Inductive Logic Programming", In:*Revised Selected Papers of the 17th International Conference on Inductive Logic Programming (ILP-07)*, volume 4894 of *LNAI*, pp. 191–199 Springer.

ONG, I. M., DUTRA, I. C., PAGE, D., SANTOS COSTA, V., 2005, "Mode Directed Path Finding", In:*Proceedings of the 16th ECML*, volume 3720, pp. 673–681.

OURSTON, D., MOONEY, R. J., 1994, "Theory refinement combining analytical and empirical methods", *Artificial Intelligence*, , n. 66, pp. 311–344.

PAES, A., REVOREDO, K., ZAVERUCHA, G., COSTA, V. S., 2005a, "Further Experimental Results of Probabilistic First-order Revision of Theories from Examples", In:*Workshop on Multi-Relational Data Mining* SIGKDD.

PAES, A., REVOREDO, K., ZAVERUCHA, G., SANTOS COSTA, V., 2005b, "Probabilistic First-Order Theory Revision from Examples", In:*Proceedings of the 15th International Conference on Inductive Logic Programming ILP-05*, volume 3625 of *LNAI*, pp. 295–311 Springer.

PAES, A., REVOREDO, K., ZAVERUCHA, G., SANTOS COSTA, V., 2006a, "PFORTE: Revising Probabilistic FOL Theories", In:*Proceedings of the 18th Brazilian AI Symposium SBIA-06*, volume 4140 of *LNAI*, pp. 441–450 Springer.

PAES, A., ZAVERUCHA, G., SANTOS COSTA, V., 2007a, "Revisando Teorias Lógicas de Primeira-ordem a partir de Exemplos usando Busca Local Estocástica", In:*Anais do VI Encontro Nacional de Inteligência Artificial ENIA-07. Prêmio de melhor artigo do congresso. XXVII Congresso Brasileiro de Computação SBC 2007* SBC.

PAES, A., ZAVERUCHA, G., SANTOS COSTA, V., 2007b, "Revising First-order Logic Theories from Examples through Stochastic Local Search", In:*Proceedings of the 17th International Conference on ILP*, volume 4894 of *LNAI*, pp. 200–210 Springer.

PAES, A., ŽELEZNÝ, F., ZAVERUCHA, G., PAGE, D., SRINIVASAN, A., 2006b, "ILP through Propositionalization and Stochastic k-term DNF Learning", In:*Proceedings of the Revised Papers of 16th International Conference on ILP*, volume 4455 of *LNAI*, pp. 379–393 Springer.

PAES, A. M., 2005, *PFORTE - Revisão de Teorias Probabilísticas de Primeira-ordem através de Exemplos*, Master's thesis, COPPE - UFRJ, Rio de Janeiro, RJ.

PAGE, D., SRINIVASAN, A., 2003, "ILP: A Short Look Back and a Longer Look Forwad", *Journal of Machine Learning Research*, v. 4, pp. 415–430.

PEARL, J., 1988, *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, San Francisco, CA.

PEARL, J., 1991, *Reasoning in Intelligent Systems: Networks of Plausible Inference*, second ed., Morgan Kaufmann, San Francisco, CA.

PEARL, J., 2009, *Causality: Models, Reasoning and Inference*, 2nd ed., Cambridge University Press.

PELLET, J.-P., ELISSEEFF, A., 2008, "Using Markov Blankets for Causal Structure Learning", *Journal of Machine Learning Research*, v. 9, pp. 1295–1342.

PEOT, M. A., SHACHTER, R. D., 1991, "Fusion and Propagation with Multiple Observations in Belief Networks", *Artificial Intelligence*, v. 48, n. 3, pp. 299–318.

PINA, A. C., ZAVERUCHA, G., 2004, "An Algorithmic Presentation of Accuracy Comparison of Classification Learning Algorithms", Technical report, Universidade Federal do Rio de Janeiro, COPPE/PESC, Rio de Janeiro, Brasil.

PITANGUI, C. G., ZAVERUCHA, G., 2011, "Inductive Logic Programming through Estimation of Distribution Algorithm", In:*Proceedings of the IEEE Congress on Evolutionary Computation (CEC-2011)*, pp. 54–61 IEEE.

PLOTKIN, G. D., 1971, *Automatic Methods of Inductive Inference*, Ph.D. thesis.

POOLE, D., 1993, "Probabilistic Horn abduction and Bayesian networks", *Artificial Intelligence*, v. 64, n. 1, pp. 81–129.

POOLE, D., 1997, "The Independent Choice Logic for Modelling Multiple Agents Under Uncertainty", *Artificial Intelligence*, v. 94, n. 1-2, pp. 7–56.

POOLE, D., 2003, "First-order probabilistic inference", In:*Proc. of the 18th International Joint Conference on Artificial Intelligence IJCAI-03*, pp. 985–991 Morgan Kaufmann.

POOLE, D., 2008, "The Independent Choice Logic and Beyond", In:DE RAEDT, L., Frasconi, P., Kersting, K., Muggleton, S., , editors, *Probabilistic Inductive Logic Programming – Theory and Applications*, volume 4911 of *LNAI*, pp. 00–00 Springer.

PRITCHARD, D. B., 2007, *The Classified Encyclopedia of Chess Variants*, John Beasley.

PROVOST, F. J., FAWCETT, T., KOHAVI, R., 1998, "The Case against Accuracy Estimation for Comparing Induction Algorithms", In:*Proceedings of the 15th International Conference on Machine Learning ICML-98*, pp. 445–453 Morgan Kaufmann.

QUINLAN, J., 1990, "Learning logical definitions from relations", *Machine Learning*, v. 5, pp. 239–266.

RAGHAVAN, S., KOVASHKA, A., MOONEY, R., 2010, "Authorship Attribution Using Probabilistic Context-Free Grammars", In:*Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics ACL-2010*, pp. 38–42 The Association for Computer Linguistics.

RAMACHANDRAN, S., MOONEY, R., 1998, "Theory Refinement of Bayesian Networks with Hidden Variables", In:*Proceedings of the 15th International Conference on Machine Learning ICML-98*, pp. 454–462.

RAY, O., 2009, "Nonmonotonic abductive inductive learning", *Journal of Applied Logic*, v. 7, n. 3, pp. 329–340.

RAY, O., BRODA, K., RUSSO, A., 2004, "A Hybrid Abductive Inductive Proof Procedure", *Logic Journal of the IGPL*, v. 12, n. 5, pp. 371–397.

REVOREDO, K., 2009, *PFORTE_PI: Revisão de Teorias Probabilísticas Relacionais com Invenção de Predicados*, Ph.D. thesis.

REVOREDO, K., PAES, A., ZAVERUCHA, G., SANTOS COSTA, V., 2006, "Combining Predicate Invention and Revision of Probabilistic FOL theories", In:*Short*

paper proceedings of 16th International Conference on Inductive Logic Programming ILP-07, pp. 176–178.

REVOREDO, K., PAES, A., ZAVERUCHA, G., SANTOS COSTA, V., 2007, "Combinando Invenção de predicados e revisão de teorias de primeira-ordem probabilísticas", In:*Anais do VI Encontro Nacional de Inteligência Artificial. XXVII Congresso Brasileiro de Computação SBC 2007* SBC.

REVOREDO, K., PAES, A., ZAVERUCHA, G., SANTOS COSTA, V., 2009, "Revisando Redes Bayesianas através da Introdução de Variáveis não-observadas", In:*Anais do VII Encontro Nacional de Inteligência Artificial ENIA-09. XXIX Congresso Brasileiro de Computação SBC 2009* SBC.

REVOREDO, K., ZAVERUCHA, G., 2002, "Revision of First-Order Bayesian Classifiers", In:*Proceedings of the 12th International Conference on ILP ILP-2002*, pp. 223–237.

RICHARDS, B. L., MOONEY, R. J., 1992, "Learning relations by pathfinding", In:*Proceedings of the 10th Annual National Conference on Artificial Intelligence AAAI-92*, pp. 50–55.

RICHARDS, B. L., MOONEY, R. J., 1995, "Automated Refinement of First-Order Horn-Clause Domain Theories", *Machine Learning*, v. 19, n. 2, pp. 95–131.

RICHARDSON, M., DOMINGOS, P., 2006, "Markov logic networks", *Machine Learning*, v. 62, n. 1-2, pp. 107–136.

ROSS, S. M., 1988, *A First Course in Probability*, third ed., Macmillan.

RÜCKERT, U., KRAMER, S., 2003, "Stochastic Local Search in k-Term DNF Learning", In:*Proceedings of the 20th International Conference on Machine Learning ICML-03*, pp. 648–655.

RÜCKERT, U., KRAMER, S., 2004, "Towards Tight Bounds for Rule Learning", In:*Proceedings of the 21st International Conference on Machine Learning ICML-04*, volume 69 ACM.

RUSSELL, S., NORVIG, P., 2010, *Artificial intelligence: A modern approach, 3rd edition*, Englewook Cliffs, NJ: Prentice-Hall.

SADIKOV, A., BRATKO, I., 2006, "Learning Long-Term Chess Strategies from Databases", *Machine Learning*, v. 63, n. 3, pp. 329–340.

SALVO BRAZ, R., AMIR, E., ROTH, D., 2005, "Lifted First-Order Probabilistic Inference", In:*Proceedings of the 19th International Joint Conference on Artificial Intelligence IJCAI-05*, pp. 1319–1325 Professional Book Center.

SALVO BRAZ, R., AMIR, E., ROTH, D., 2006, "MPE and Partial Inversion in Lifted Probabilistic Variable Elimination", In:*Proceedings of the 21st National Conference on Artificial Intelligence AAAI-06* AAAI Press.

SANG, T., BEAME, P., KAUTZ, H. A., 2007, "A Dynamic Approach for MPE and Weighted MAX-SAT", In:*Proceedings of the 20th International Joint Conference on Artificial Intelligence IJCAI-07*, pp. 173–179.

SANGHAI, S., DOMINGOS, P., WELD, D., 2003, "Dynamic probabilistic relational models".

SANTOS, J., MUGGLETON, S., 2011, "When Does It Pay Off to Use Sophisticated Entailment Engines in ILP?", In:*Revised Papers of the 20th International Conference on Inductive Logic Programming (ILP-2010)*, volume 6489 of *LNAI*, pp. 214–221 Springer.

SANTOS COSTA, V., 2008, "The Life of a Logic Programming System", In:*Proceedings of the 24th International Conference on Logic Programming ICLP 2008*, volume 5366 of *LNCS*, pp. 1–6 Springer.

SANTOS COSTA, V., PAES, A., 2009, "On the Relationship between PRISM and CLP(BN)", In:*Proceedings of the International Workshop on Statistical Relational Learning (SRL-2009)*.

SANTOS COSTA, V., PAGE, D., QAZI, M., , CUSSENS, J., 2003a, "CLPBN: Constraint Logic Programming for Probabilistic Knowledge", In:*Proceedings of*

the 19th Annual Conference on Uncertainty in Artificial Intelligence UAI-03, pp. 517–524.

SANTOS COSTA, V., SRINIVASAN, A., CAMACHO, R., BLOCKEEL, H., DE-MOEN, B., JANSSENS, G., STRUYF, J., VANDECASTEELE, H., , LAER, W. V., 2003b, "Query Transformations for Improving the Efficiency of ILP Systems", *Journal of Machine Learning Research*, v. 4, pp. 465–491.

SATO, T., 1992, "Equivalence-Preserving First Order Unfold/fold Transformation Systems", *Theoretical Computer Science*, v. 105, pp. 57–84.

SATO, T., KAMEYA, Y., 1997, "PRISM: A Language for Symbolic-Statistical Modeling", In:*Proceedings of the 15th International Joint Conference on Artificial Intelligence IJCAI-97*, pp. 1330–1339.

SATO, T., KAMEYA, Y., 2001, "Parameter Learning of Logic Programs for Symbolic-Statistical Modeling", *Journal of Artificial Intelligence Research JAIR*, v. 15, pp. 391–454.

SATO, T., KAMEYA, Y., ZHOU, N.-F., 2005, "Generative Modeling with Failure in PRISM", In:*Proceedings of the 19th International Joint Conference on Artificial Intelligence IJCAI-05*, pp. 847–852.

SCHWARZ, G., 1978, "Estimating the dimension of a model", *Annals of Statistics*, v. 6, pp. 461–464.

SEBAG, M., ROUVEIROL, C., 1997, "Tractable Induction and Classification in First Order Logic Via Stochastic Matching", In:*IJCAI (2)*, pp. 888–893.

SEBAG, M., ROUVEIROL, C., 2000, "Any-time Relational Reasoning: Resource-bounded Induction and Deduction Through Stochastic Matching", *Machine Learning*, v. 38, n. 1-2, pp. 41–62.

SELMAN, B., KAUTZ, H., COHEN, B., 1994, "Noise Strategies for Improving Local Search", In:*Proceedings of the 12th National Conference on Artificial Intelligence*, pp. 337–343 AAAI Press/The MIT Press.

SELMAN, B., KAUTZ, H., MCALLESTER, D., 1997, "Ten challenges in Propositional Reasoning and Search", In:*Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pp. 50–54 Morgan Kaufmann Publishers.

SELMAN, B., KAUTZ, H. A., 1993, "Domain-Independent Extensions to GSAT: Solving Large Structured Satisfiability Problems", In:*Proceedings of the 13th International Joint Conference on Artificial Intelligence IJCAI-93*, pp. 290–295.

SELMAN, B., KAUTZ, H. A., COHEN, B., 1996, "Local Search Strategies for Satisfiability Testing", *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge, October 11-13, 1993. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, v. 26, pp. 521–532.

SELMAN, B., LEVESQUE, H., MITCHELL, D., 1992, "A New Method for Solving Hard Satisfiability Problems", In:*Proceedings of the 10th Annual National Conference on Artificial Intelligence AAAI-92*, pp. 440–446.

SERRURIER, M., PRADE, H., 2008, "Improving Inductive Logic Programming by Using Simulated Annealing", *Information Sciences*, v. 178, n. 6, pp. 1423–1441.

SHACHTER, R. D., 1998, "Bayes-Ball: The Rational Pastime for Determining Irrelevance and Requisite Information in Belief Networks and Influence Diagrams", In:*Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, pp. 480–487 Morgan Kaufmann.

SHAPIRO, E. Y., 1981, "The Model Inference System", In:*Proceedings of the 7th International Joint Conference on Artificial Intelligence IJCAI '81*, pp. 1064 William Kaufmann.

SHAPIRO, E. Y., 1983, *Algorithm Program Debugging*, MIT Press.

SINGLA, P., DOMINGOS, P., 2005, "Discriminative Training of Markov Logic Networks", In:*Proceedings of the 21st National Conference on Artificial Intelligence AAAI-05*, pp. 868–873.

SINGLA, P., DOMINGOS, P., 2006, "Memory-Efficient Inference in Relational Domains", In:*Proceedings of the 21st National Conference on Artificial Intelligence AAAI-06*.

SINGLA, P., DOMINGOS, P., 2008, "Lifted First-Order Belief Propagation", In:*Proceedings of the 23rd AAAI Conference on Artificial Intelligence, AAAI 2008*, pp. 1094–1099 AAAI Press.

SPECIA, L., SRINIVASAN, A., JOSHI, S., RAMAKRISHNAN, G., NUNES, M. D. G. V., 2009, "An investigation into feature construction to assist word sense disambiguation", *Machine Learning*, v. 76, n. 1, pp. 109–136.

SPELLMAN, P., SHERLOCK, G., MQ, M. Z., VR, V. I., K, K. A., MB, M. E., BROWN, P., BOTSTEIN, D., , FUTCHER, B., 1998, "Comprehensive identification of cell cycle-regulated genes of the yeast Saccharomyces cerevisiae by microarray hybridization", *Molecular Biology of the cell*, v. 9, n. 12, pp. 3273–3297.

SPIRTES, P., GLYMOUR, C., SCHEINES, R., 2001, *Causation, Prediction, and Search*, 2nd ed., MIT Press.

SRINIVASAN, A., 1999, "A study of Two Sampling Methods for Analysing Large Datasets with ILP", *Data Mining and Knowledge Discovery*, v. 3, n. 1, pp. 95–123.

SRINIVASAN, A., 2000, "A Study of Two Probabilistic Methods for Searching Large Spaces with ILP", Technical Report PRG-TR-16-00, Oxford University Computing Laboratory, Oxford.

SRINIVASAN, A., 2001a, *The Aleph manual*, , `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html`.

SRINIVASAN, A., 2001b, "Extracting Context-Sensitive Models in Inductive Logic Programming", *Machine Learning*, v. 3, n. 44, pp. 301–324.

SRINIVASAN, A., KING, R. D., 2008, "Incremental Identification of Qualitative Models of Biological Systems using Inductive Logic Programming", *Journal of Machine Learning Research*, v. 9, pp. 1475–1533.

SRINIVASAN, A., KING, R. D., MUGGLETON, S., STERNBERG, M. J. E., 1997a, "Carcinogenesis Predictions Using ILP", In:*Proceedings of the 7th Int. Workshop on ILP*, volume 1297, pp. 273–287 Springer-Verlag.

SRINIVASAN, A., KING, R. D., MUGGLETON, S., STERNBERG, M. J. E., 1997b, "Carcinogenesis Predictions Using ILP", In:*Proceedings of the 7th International Conference on Inductive Logic Programming (ILP-97)*, volume 1297 of *Lecture Notes in Computer Science*, pp. 273–287 Springer.

SRINIVASAN, A., MUGGLETON, S., STERNBERG, M. J. E., KING, R. D., 1996, "Theories for Mutagenicity: A Study in First-Order and Feature-Based Induction", *Artificial Intelligence*, v. 85, n. 1-2, pp. 277–299.

SRINIVASAN, A., PAGE, D., CAMACHO, R., KING, R. D., 2006, "Quantitative pharmacophore models with inductive logic programming", *Machine Learning*, v. 64, n. 1-3, pp. 65–90.

STAHL, I., 1993, "Predicate Invention in ILP - an Overview", In:*Proceedings of the 4th European Conference on Machine Learning ECML 94*, volume 667 of *LNCS*, pp. 313–322 Springer.

STERLING, L., SHAPIRO, E., 1986, *The Art of Prolog: Advanced programming Techniques*, The MIT Press.

STONE, M., 1977, "An asymptotic equivalence of choice of model by cross-validation and Akaike's criterion", *Journal of the Royal Statistical Society series B*, v. 39, pp. 44–47.

TAMADDONI-NEZHAD, A., CHALEIL, R., KAKAS, A. C., MUGGLETON, S., 2006, "Application of abductive ILP to learning metabolic network inhibition from temporal data", *Machine Learning*, v. 64, n. 1-3, pp. 209–230.

TAMADDONI-NEZHAD, A., MUGGLETON, S., 2000, "Searching the Subsumption Lattice by a Genetic Algorithm", In:*Proceedings of the 10th International Conference on ILP ILP-00*, pp. 243–252.

TANG, L. R., MOONEY, R. J., MELVILLE, P., 2003, "Scaling Up ILP to Large Examples: Results on Link Discovery for Counter-Terrorism", In:*Proceedings of the KDD-2003 Workshop on Multi-Relational Data Mining MRDM-2003*, pp. 107–121.

TASKAR, B., ABBEEL, P., KOLLER, D., 2002, "Discriminative Probabilistic Models for Relational Data", In:*Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence UAI-2002*, pp. 485–492.

THRUN, S., 1995, "Is Learning The n-th Thing any Easier than Learning The First?", In:*Advances in Neural Information Processing Systems 8, NIPS*, pp. 640–646 MIT Press.

TIAN, J., 2000, "A Branch-and-Bound Algorithm for MDL Learning Bayesian Networks", In:*Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, UAI '00, pp. 580–588 Morgan Kaufmann Publishers Inc.

TIAN, J., PEARL, J., 2002, "On the Testable Implications of Causal Models with Hidden Variables", In:*Proceedings of the 18th Conference in Uncertainty in Artificial Intelligence UAI-02*, pp. 519–527 Morgan Kaufmann.

TITOV, I., HENDERSON, J., 2007, "Incremental Bayesian networks for structure prediction", In:*Proceedings of the 24th International Conference on Machine Learning (ICML 2007)*, volume 227 of *ACM International Conference Proceeding Series*, pp. 887–894 ACM.

TOWELL, G., SHAVLIK, J., 1994, "Knowledge-Based Artificial Neural Networks", *Artificial Intelligence*, v. 70, n. 1–2, pp. 119–165.

TOWELL, G. G., SHAVLIK, J. W., 1993, "Extracting Refined Rules from Knowledge-Based Neural Networks", *Machine Learning*, v. 13, pp. 71–101.

TREFETHEN, N., 1998, *Maxims about numerical mathematics, computers, science, and life,* , SIAM News.

TSAMARDINOS, I., BROWN, L. E., ALIFERIS, C. F., 2006, "The max-min hill-climbing Bayesian network structure learning algorithm", *Machine Learning*, v. 65, n. 1, pp. 31–78.

VALIANT, L. G., 1984, "A Theory of the Learnable", *Communications of the ACM*, v. 27, n. 11, pp. 1134–1142.

RIJSBERGEN, C. J. V., 1979, *Information Retireval*, Butterworths.

VENNEKENS, J., VERBAETEN, S., BRUYNOOGHE, M., 2004, "Logic Programs with Annotated Disjunctions", In:*Proceedings of the 20th International Conference on Logic Programming ICLP-04*, pp. 431–445.

VERMA, T., PEARL, J., 1990, "Equivalence and synthesis of causal models", In:*Proceedings of the Sixth Annual Conference on Uncertainty in Artificial Intelligence*, pp. 255–270 Elsevier.

WANG, J., DOMINGOS, P., 2008, "Hybrid Markov Logic Networks", In:*Proceedings of the 23rd AAAI Conference on Artificial Intelligence AAAI 2008*, pp. 1106–1111 AAAI Press.

WEI, G. C. G., TANNER, M. A., 1990, "A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms", *Journal of the American Statistical Association*, v. 85, pp. 699–704.

WEISS, Y., 2000, "Correctness of Local Probability Propagation in Graphical Models with Loops", *Neural Computation*, v. 12, n. 1, pp. 1–41.

WESTON, J., MUKHERJEE, S., CHAPELLE, O., PONTIL, M., POGGIO, T., VAPNIK, V., 2000, "Feature Selection for SVMs", In:*Papers from Neural Information Processing Systems NIPS-2000*, pp. 668–674.

WOGULIS, J., 1994, *An Approach to Repairing and Evaluationg First-Order Theories Containing Multiple Concepts and Negation*, Ph.D. thesis, University of California, Irvine, CA.

WOGULIS, J., PAZZANI, M., 1993, "A methodology for evaluationg theory revision systems: results with Audrey II", In:*Proceedings of the 13th International Joint Conference on Artificial Intelligence IJCAI-93*, pp. 1128–1134.

WROBEL, S., 1993, "On the Proper Definition of Minimality in Specialization and Theory Revision", In:*Machine Learning: ECML-93, European Conference on Machine Learning*, volume 667 of *LNCS*, pp. 65–82 Springer.

WROBEL, S., 1994, "Concept Formation During Interactive Theory Revision", v. 14, pp. 169–191.

WROBEL, S., 1996, "First-order Theory Refinement", In:RAEDT, L. D., , editor, *Advances in Inductive Logic Programming*, pp. 14–33 IOS Press.

YAMAMOTO, A., 1997, "Which Hypotheses Can Be Found with Inverse Entailment?", In:*Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *LNCS*, pp. 296–308 Springer.

YEDIDIA, J. S., FREEMAN, W. T., WEISS, Y., 2000, "Generalized Belief Propagation", In:*Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems NIPS 2000*, pp. 689–695.

ŽELEZNÝ, F., LAVRAC, N., 2006, "Propositionalization-Based Relational Subgroup Discovery with RSD", *Machine Learning*, v. 62, n. 1-2, pp. 33–63.

ŽELEZNÝ, F., SRINIVASAN, A., PAGE, D., 2002, "Lattice-Search Runtime Distributions May Be Heavy-Tailed", In:*Proceedings of the Twelth International Conference on Inductive Logic Programming ILP-02*, volume 2583 of *LNAI*, pp. 341–358 Springer.

ŽELEZNÝ, F., SRINIVASAN, A., PAGE, D., 2004, "A Monte Carlo Study of Randomised Restarted Search in ILP", In:*Proceedings of the 14th International Con-*

ference on Inductive Logic Prgramming ILP-04, volume 3194 of *LNAI*, pp. 341–358 Springer.

ŽELEZNÝ, F., SRINIVASAN, A., PAGE, D., 2006, "Randomised restarted search in ILP", *Machine Learning*, v. 64, n. 1-3, pp. 183–208.

ZELLE, J. M., MOONEY, R. J., KONVISSER, J. B., 1994, "Combining Top-down and Bottom-up Techniques in Inductive Logic Programming", In:*Proceedings of the E11th International Conference on Machine Learning ICML-94*, pp. 343–351.

ZHANG, N. L., POOLE, D., 1996, "Exploiting Causal Independence in Bayesian Network Inference", *Journal of Artificial Intelligence Research*, v. 5, pp. 301–328.

Índice Remissivo

Appendix A: Inductive Logic Programming and Theory Revision

Machine learning is a sub area of artificial intelligence which studies systems that improve their behavior over time with experience (MITCHELL, 1997). Standard machine learning algorithms are propositional, searching for patterns from data of fixed size and represented as attribute-value pairs. In this way, they assume the objects of the domain are homogeneous and sampled from a simple relation. However, real data usually contain many different types of objects and multiple entities, disposed typically in several related tables. *Inductive Logic Programming (ILP)* (MUGGLETON, 1992; LAVRAC, DZEROSKI, 1994; MUGGLETON, DE RAEDT, 1994; NIENHUYS-CHENG, De Wolf, 1997; DE RAEDT, 2008), also known as Multi-Relational Data Mining (DZEROSKI, LAVRAC, 2001), combines machine learning and logic programming to automatically induce sets of first-order clauses (*theory*) from multi-relational data. ILP systems have been experimentally tested on a number of important applications (KING, et al., 1995b; SRINIVASAN, et al., 1997a; MUGGLETON, 1999; FANG, et al., 2001; KING, et al., 2004). Theories learned by ILP elegantly and expressively represent complex situations, even involving a variable number of entities and relationships among them. Such theories are formed with the goal of discriminating between positive and negative examples, given background knowledge. The background knowledge consists of a list of logical facts and/or a set of inference rules.

ILP algorithms assume the background knowledge is fixed and correct. However, it may be the case that background knowledge also contains incorrect rules and therefore it would be necessary to *modify* them so that they become correct. This is the task of *Theory revision* which has as goal to fix the rules responsible for the

misclassification of some example. In this chapter, we start by reviewing the main techniques used in ILP community in section 1. Next, we discuss theory revision in section 5.3.2 with special attention to the system used in this thesis, the FORTE system (RICHARDS, MOONEY, 1995). The reader familiar with ILP and Theory Revision may skip this chapter.

Inductive Logic Programming

Inductive Logic Programming algorithms have as goal to induce from a set of training examples a logic program describing a relational domain, using concepts defined in the *background knowledge*. The returned logic program will be used to classify new examples into positive or negative, using techniques such as resolution. A brief description of some logic programming terms can be found in Table 1 and more details can be explored in (STERLING, SHAPIRO, 1986; LLOYD, 1987; FLACH, 1994; NILSSON, MALUSZYNKSKI, 2000). The learning problem in ILP is usually defined as follows.

Tabela 1: Some standard Logic Programming terms and their definitions

Term	Definition
constants	Symbols for denoting individuals. Following Prolog convention, we represent constants in lower case.
variables	Symbols referring to an unspecified individual. Following Prolog convention, we represent variables in upper case.
predicate	Symbols for denoting relations, such as mother, loves, etc.
term	They are constants, variables or functions in predicates.
atom	Formulas in the form $p(t_1, ..., t_n)$, where p is a predicate and $t_1, ..., t_n$ are terms.
ground atom	An atom which contains no variable
clause	A formula $\forall(L_1 \vee ... \vee L_n)$, where each L_i is a an atom (positive literal) or the negation of an atom (negative literal)
definite clause	A clause with exactly one positive literal, in the form $A_0 \leftarrow A_1, ..., A_n$, or equivalently $A_0 \vee \neg A_1 \vee ... \vee \neg A_n$, where $n \geq 0$. A_0 is the *head* of the clause, whereas $A_1, ..., A_n$ is the *body* of the clause.
fact	A definite clause where n = 0.
Horn clause	A clause with at most one positive literal.
Herbrand universe	The set of all ground terms constructed from functors and constants in a domain
Herbrand base	The set of all ground atoms over a domain

Definition .1 *Given:*

- *A set E of examples, divided into positive E^+ and negative E^- examples and*

- *Background knowledge BK,*

both expressed as first-order Horn clauses.

    Learn:

- *A hypothesis H composed of first-order definite clauses such that $BK \wedge H \vDash E^+$ (H is complete) and $BK \wedge H \nvDash E^-$ (H is consistent), i.e., H is correct.*

Figure 1 brings a schema of the learning task in ILP.
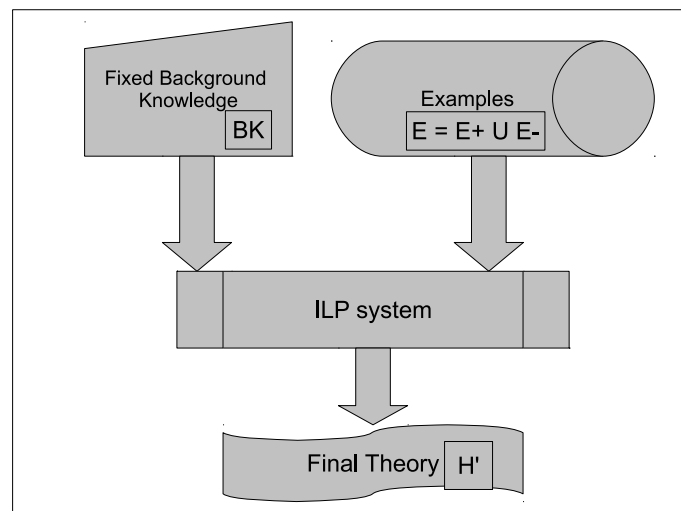
Figura 1: Schema of the learning task in ILP, where BK is background knowledge, E is the set of positive (E^+) and negative (E^-) examples and H' is the theory learned by the ILP system.

Often it is not possible to find a correct hypothesis and then the criteria $BK \wedge H \vDash E^+$ and $B \wedge H \nvDash E^-$ are relaxed.

Learning Settings in ILP

The standard learning setting previously defined is known as *learning from entailment* (FRAZIER, PITT, 1993). ILP algorithms may also follow the *Learning from interpretations* approach (VALIANT, 1984; ANGLUIN, et al., 1992; DE RAEDT, DŽEROSKI, 1994), where the examples are Herbrand interpretations and the goal is to induce a true hypothesis H in the minimal Herbrand model of $BK \wedge E$. In this setting, it is assumed the examples are completely specified. Otherwise, the learning is done from partial interpretations (FENSEL, et al., 1995) and H must be true in the Herbrand model created by extending $BK \wedge E$. The generalization of learning from partial interpretations is called *learning from satisfiability* (DE RAEDT, 1997). In this last case, it is required that $H \wedge BK \wedge E \nvDash \square$.

From the point of view of examples representation, there are two learning settings (DE RAEDT, 1997):

1. Extensional ILP. An example $e \in E$ is a ground atom. In this case, the whole BK is shared by all the examples, i.e., the conditions are the same for each example. This setting is reduced to learning from entailment.

2. Intentional ILP. An example $e \in E$ is a definite ground clause. Each example may have different information associated to them, i.e., the BK may be divided into a set of definite clauses shared by all examples and sets of definite clauses restricted to each example. This setting is reduced to learning from satisfiability.

Usually, ILP systems such as (MUGGLETON, 1995a), Aleph (SRINIVASAN, 2001a), FOIL (QUINLAN, 1990), TopLog (MUGGLETON, et al., 2008) learn a single target predicate, are extensional and learn from entailment. ILP algorithms may also be designed to learn a set of target concepts, possibly related to each other. CLAUDIEN (DE RAEDT, 1997) learns multiple predicates from interpretations. TILDE (BLOCKEEL, DE RAEDT, 1998) has as goal to learn relational decision trees from complete interpretations. Hyper (BRATKO, 1999) learns multiple related predicates from entailment. We refer the reader to (LAVRAC, DZEROSKI, 1994; DZEROSKI, LAVRAC, 2001; DE RAEDT, 2008) for more details on ILP systems.

Ordering the Hypothesis Search Space

In order to generalize and/or specialize hypothesis in ILP it is used the framework of θ-subsumption. A clause c_1 *θ-subsumes* a clause c_2 if and only if $\exists$ a variable substitution θ such that $c_1\theta \subseteq c_2$. Due to θ-subsumption, ILP algorithms may traverse the search space from bottom to up (generalizing the hypothesis), from top to down (specializing the hypothesis) or even combining both specialization and generalization strategies. Top-down ILP systems such as (SHAPIRO, 1983), (QUINLAN, 1990) and (DE RAEDT, DEHASPE, 1997) search the hypothesis space by considering, at each iteration, all valid refinements to a current set of candidate hypotheses. The hypotheses are then evaluated considering their coverage of positive and negative examples, and also measures such as the information gain or the length of the clauses. This strategy considers the set of examples only to evaluate the candidate hypotheses but no to create new hypotheses.

On the other hand, algorithms following a bottom-up strategy use the examples to propose hypotheses. They start with the most specific hypothesis and proceed to generalize it until no further generalizations are possible, without covering some negative examples. Usually, they are based on (1) the least general generalization relative to the background knowledge (RLGG) (PLOTKIN, 1971), such as it is done in Golem system (MUGGLETON, FENG, 1990) and its descendant Pro-Golem (MUGGLETON, et al., 2010) or (2) inverse resolution (MUGGLETON, 1987; MUGGLETON, BUNTINE, 1988; MUGGLETON, DE RAEDT, 1994). Nowadays, it is common to somehow combine the bottom-up and top-down strategies, in order to exploit the strengths of both techniques while avoiding their weaknesses. This is the case of systems such as Progol (MUGGLETON, 1995a), Aleph (SRINI-VASAN, 2001a), CHILLIN (ZELLE, et al., 1994), BETH (TANG, et al., 2003) and TopLog (MUGGLETON, et al., 2008), among others.

Mode Directed Inverse Entailment and the Bottom Clause

The bottom clause (MUGGLETON, 1995a) $\perp (e)$ with regard to a clause e and background theory BK is the most specific clause within the hypothesis space that

covers the example e, i.e.,

$$BK \cup \perp (e) \models e \tag{1}$$

Any single clause hypothesis covering the example e with regard to BK must be more general than $\perp (e)$. Any clause that is not more general than $\perp (e)$ cannot cover e and can be safely disregarded. Thus, the bottom clause bounds the search for a clause covering the example e, as it captures all relevant information to e and BK. A top-level algorithm of the bottom clause construction process defined in (DE RAEDT, 2008) is reproduced here as Algorithm 1.

Algorithm 1 Top-level Algorithm of the Bottom Clause Construction Process (DE RAEDT, 2008)

1: Find a skolemization substitution θ for e with regard to BK
2: Compute the least Herbrand model M of $BK \cup \neg body(e)\theta$
3: Deskolemize the clause $head(e\theta) \leftarrow M$
4: **return** the result of step 3

In order to understand how the bottom clause is constructed, consider the example e as

$$nice(X) \leftarrow dog(X).$$

and the BK as

$$animal(X) \leftarrow pet(X).$$

$$pet(X) \leftarrow dog(X).$$

First of all, notice that $BK \cup \perp (e) \models e$ is equivalent to $BK \cup \neg e \models \neg \perp (e)$. The first step is to replace all variables in $\neg e$ by distinct constants not appearing in the clause (skolemization). The result is one false ground fact coming from the head of $\neg e\theta$, since it is a definite clause and a set of positive ground facts coming from the body of $\neg e\theta$. In the example above, considering the skolemization substitution as $\theta = \{X \leftarrow skol\}$, we have

$$\neg e\theta = \{\neg nice(skol), dog(skol)\}$$

The next step is to find the set of all ground facts entailed by $BK \cup \neg e$, i.e., the ground literals which are true in all models of $BK \cup \neg e$. This is achieved by com-

puting the least Herbrand model of BK and $\neg body(c\theta)$. In the example, we have

$$\neg \perp (e)\theta = \neg\ e\theta \cup \{pet(skol), animal(skol)\}$$

Finally, each skolemization constant is replaced by a different variable in $\neg \perp (e)\theta$ and the result is negated to obtain $\perp (e)$. In the example,

$$\perp (e) = nice(X) \leftarrow dog(X), pet(X), animal(X)$$

In general, $\perp$ could have an infinite cardinality. Thus, *Mode Directed Inverse Entailment* (MUGGLETON, 1995a) systems such as Aleph and Progol, consider a set of user-defined mode declarations together with other settings to constrain the search for a good hypothesis. A mode declaration (MUGGLETON, 1995a) has either the form $modeh(recall, atom)$ or $modeb(recall, atom)$, where recall is an integer greater than 1 or $'*'$ and atom is a ground atom. *Modeh* declarations indicate predicates appearing in the head of clauses and *modeb*, predicates in the body of clauses. *Recall* is the maximum number of different instantiations of *atom* allowed to appear in a clause (where '*' means an indefinite number of times). Terms in the atom are either normal or place-marker. A normal term is either a constant or a function symbol followed by a bracketed tuple of terms. A place-marker is either +type, -type or #type, where type is a constant defining the type of term. The meaning of +, - and # is as follows.

- Input (+) - an input variable of type T in a body literal B_i appears as an output variable of type T in a body literal that appears before B_i, or appears as an input variable of type T in the head of the clause.

- Output($-$) - an output variable of type T in the head of the clause must appear as an output variable of type T in any literal of the body of the clause.

- Constant(#) - an argument denoted by $\#T$ must be ground with terms of type T.

The Algorithm 2 illustrates in more details the construction of the bottom clause in Progol (MUGGLETON, 1995a) system, considering modes declaration.

The list $InTerms$ keeps the terms responsible for instantiating the input terms in the head of the clause and the terms instantiating output terms in the body of the clause. The function $hash$ associates a different variable to each term. The depth of a variable v in a definite clause C is 0 if v is in the head of C and $(max_{u \in U_v} d(u)) + 1$ otherwise, where U_v are the variables in the body of C containing v.

Algorithm 2 Bottom clause construction Algorithm (MUGGLETON, 1995a)

Input: Background knowledge BK, a positive example e, where $\neg e$ is a clause normal form logic program $\neg a, b_1, ..., b_n$
Output: The bottom clause BC

1:   $InTerms \leftarrow \emptyset$, $\perp \leftarrow \emptyset$
2:   $i \leftarrow 0$, corresponding to the variables depth
3:   $BK \leftarrow BK \cup e$
4:   find the first modeh h such that h subsumes e with substitution θ
5:   **for each** v/t in θ **do**
6:       **if** v is a $\sharp$ type **then**
7:          replace v in h by t
8:       **if** v is a $+$ or $-$ type **then**
9:          replace v in h by v_k, where v_k is a variable such that $k = hash(t)$
10:      **if** v is a $+$ type **then**
11:         $InTerms \leftarrow InTerms \cup t$
12: $\perp \leftarrow \perp \cup h$
13: **for each** modeb declaration b **do**
14:      **for** all possible substitution θ of arguments corresponding to $+$ type by terms in the set $InTerms$ **do**
15:         **repeat**
16:            **if** b succeeds with substitution θ' **then**
17:               **for each** v/t in θ and θ' **do**
18:                  **if** v corresponds to $\sharp$ type **then**
19:                     replace v in b by t
20:                  **else**
21:                     replace v in b by v_k, where $k = hash(t)$
22:                  **if** v corresponds to $-$ type **then**
23:                     $InTerms \leftarrow InTerms \cup t$
24:               $\perp \leftarrow \perp \cup \overline{b}$
25:         **until** reaches recall times
26: $i \leftarrow i + 1$
27: Go to line 13 if the maximum depth of variables is not reached
28: **return** $\perp$.

Suppose, for example, the modes declaration below, expressing a fatherhood relationship, where the first argument is the recall number

$modeh(1, father(+person, +person))$

$modeb(10, parent\_of(+person, +person))$

$modeb(10, parent\_of(-person, +person))$

and the background knowledge

$parent\_of(jack, anne)$

$parent\_of(juliet, anne)$

$parent\_of(jack, james)$

$parent\_of(juliet, james)$

and the example $father(jack, anne)$. The most specific clause is

$$\perp = father(jack, anne) \leftarrow parent\_of(jack, anne), parent\_of(juliet, anne)$$

where the first literal on the body was obtained by $modeb(10, parent\_of(+person, +person))$ and the second literal by $modeb(10, parent\_of(-person, +person))$.

The bottom clause with the constants replaced by variables is

$$\perp = father(A, B) \leftarrow parent\_of(A, B), parent\_of(C, B)$$

The space complexity of the bottom clause is the cardinality of it and is bounded by $r(|M|j + j-)^{ij^+}$, where $|M|$ is the cardinality of M (the set of modes declarations), j^+ is the number of $+$ type occurrences in each modeb in M plus the number of $\smile$ type occurrences in each modeh, j^- is the number of $\smile$ type occurrences in each modeb in M plus the number of $+$ type occurrences in each modeh, r is the recall of each mode $m \in M$, and i is the maximum variable depth (MUGGLETON, 1995a). For more details on formal definitions of the Bottom Clause and Inverse Entailment we refer the reader to (MUGGLETON, 1995a).

First-order Logic Theory Revision from Examples

ILP algorithms learn first-order clauses given a set of examples and a static and assumed as correct background knowledge. On the other hand, *theory revision from examples* (WROBEL, 1996) has as goal to improve a previously obtained knowledge. To do so, theory revision assumes the provided BK may also contains incorrect rules, which should be modified to better reflect the set of examples. Revision in certain clauses of the BK can be avoided by letting a part of the preliminary knowledge be defined as correct and invariant. Thus, in theory revision the BK is divided in two parts: A set of rules assumed as correct and therefore not modifiable, called here as *Fundamental Domain Theory (FDT)* (RICHARDS, MOONEY, 1995); and the remaining rules which may be incorrect and are subject to modifications, called the *Initial Theory*. The goal of a theory revision process is to identify points in the initial theory which prevent it from correctly classifying positive or negative examples, and propose modifications to such points, so that the revised theory together with the FDT is as close to a correct theory as possible. The task of theory revision from examples is defined as follows (WROBEL, 1996).

Definition .2 *Given:*

- *A background knowledge BK divided into*

  - *A modifiable set of clauses which might be incorrect (H′) and*

  - *An invariant and assumed as correct set of clauses (FDT) and*

- *A set of positive E^+ and negative examples E^- composing the set of examples E*

  both written as logic programs;
  Find:

- *A revised theory H consisting of definite first-order clauses such that $FDT \wedge H \models E^+$ (H is complete) and $FDT \wedge H' \nvDash E^-$ (H is consistent), i.e., H is correct and obeys a minimality criteria*

The minimality criteria may be seen as the attempt of obtaining a revised theory as syntactically and semantically close as possible from the original BK. Usually it is not possible to find a correct theory and such a criteria is relaxed to find a theory as close to be correct as possible.

The schema of theory revision is shown in Figure 2. Note that ILP could be seen as a subset of theory revision, where H' is empty and therefore $BK = FDT$.

Theory Revision is particularly powerful and challenging because it must deal with the issues arising from revising multiple clauses (theory) and even multiple predicates (multiple target concepts). Additionally, as the initial theory is a good starting point and the revision process takes advantage of it, the theories returned by revision systems are usually more accurate than theories learned from standard ILP systems using the same dataset. Several papers such as (SHAPIRO, 1981), (WOGULIS, PAZZANI, 1993), (RICHARDS, MOONEY, 1995), (BUNTINE, 1991), (TOWELL, SHAVLIK, 1994), (ADÉ, et al., 1994), (WROBEL, 1996), (RAMACHANDRAN, MOONEY, 1998), (GARCEZ, ZAVERUCHA, 1999), (ESPOSITO, et al., 2000) show that propositional and first-order theory revision systems are capable of learning more accurate theories than purely inductive systems and using less examples.

Revision Points

Usually, the first step in a revision system is to identify the misclassified examples. A positive example not covered by the theory (a *false negative*) indicates the theory is too specific and therefore needs to become more general. In the opposite case, a negative example covered by the theory indicates it is too general and therefore it needs to become more specific. In the former case we need to *generalize* the theory and in the latter case we need to *specialize* the theory.

As the theory may be composed of several target concepts, described by several clauses, it is necessary to find out which clauses and/or literals are responsible for the misclassification of the examples. Also, many clauses can be responsible for proving negative examples as many clauses could be generalized so that the misclassified positive examples become covered. In theory revision, the clauses and literals
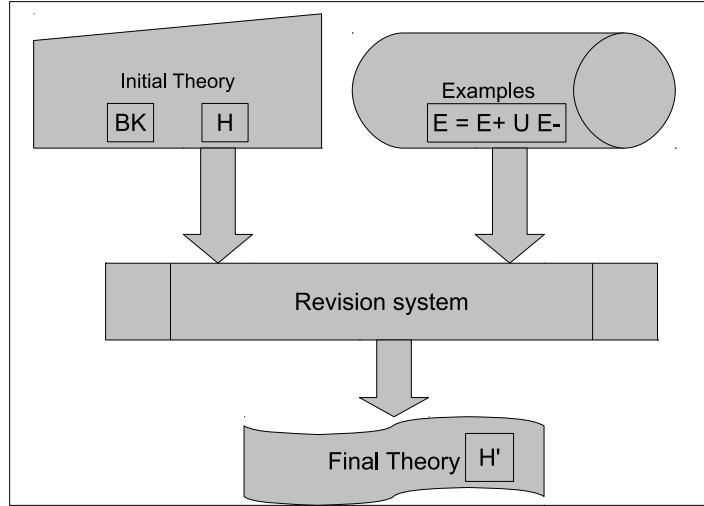
Figura 2: Schema of Theory Revision from Examples, where FDT is the fixed preliminary knowledge and H' is the modifiable preliminary knowledge, E is the set of positive (E^+) and negative (E^-) examples. H is the theory returned by the revision system

.

considered as responsible for misclassifying examples are called *revision points*. It is expected that modifications performed on such points improve the quality of the theory. Depending on the type of the misclassified example being considered we can define two types of revision points:

- *Generalization revision points* - Those are the points in the theory where proofs of positive examples fail.

- *Specialization revision points* - clauses used in successful proof paths of negative examples.

Revision points were known as *culprit clauses* in MIS system(SHAPIRO, 1981), which are clauses covering negative examples, found through the SLD tree, and clauses not covering positive examples. In this last case, these clauses are leading to missing clauses, which can be suggested by posing queries to an oracle and using examples it already knows (DE RAEDT, 2008).

The specification of the revision point determines the type of revision operator that will be applied to make the theory consistent with the dataset. One may consider two types of operators: *generalization* operators, applied on generaliza-

tion revision points and *specialization* operators, applied on specialization revision points (WROBEL, 1996).

Revision Operators

Theory revision relies on operators that propose modifications at each revision point aiming to transform a theory into another one. Any operator used in first-order machine learning can be used in a theory revision system. In this work we use some operators previously defined in (RICHARDS, MOONEY, 1995). Below, we briefly describe them. Next section we show in details how some of theses operators work.

The operators for specialization are:

- *Delete-rule* - this commonly used operator removes a clause that was used to prove a negative example.

- *Add-antecedent* - this operator adds antecedents to an inconsistent clause.

Other approaches exist. Indeed, there are several specialization operators based on the idea of inventing new concepts (predicate invention (STAHL, 1993; KRAMER, 1995)) while revising theories (WROBEL, 1994; BAIN, 2004).

Different from specialization operators, which only modify existing clauses, generalization operators may create entirely new clauses. As the goal is to cover unprovable positive examples, any ILP operator which accepts positive examples as input can be used. Next we cite usual generalization operators:

- *Delete-antecedent* - this operator removes failed antecedents from clauses that could be used to prove positive examples.

- *Add-rule* - this operator generates new clauses, either from failed existing clauses (deleting antecedents followed by addition of antecedents) or from scratch (starting only from the generalized head of the example).

Other generalization operators exist. One can use *abduction*: first, look for a clause that might satisfy the example but has missing premises, and then add the missing premises. For more details on revision operators we refer the reader to (RICHARDS, MOONEY, 1995) and (WROBEL, 1996).

FORTE

In this work we follow the First Order Revision of Theories from Examples (FORTE) system (RICHARDS, MOONEY, 1995), which automatically revises function-free first-order Horn clauses. There are several first-order theory revision systems described in the literature, including MIS (SHAPIRO, 1981), RUTH (ADÉ, et al., 1994), MOBAL (WROBEL, 1993; WROBEL, 1994), INTHELEX (ESPOSITO, et al., 2000), among others.

FORTE performs a hill-climbing search through a space of both specialization and generalization operators in an attempt to find a minimal revision to a theory that makes it consistent with the set of training examples. In order to find the revision points FORTE follows a bottom-up strategy, as the training examples are used to find out the clauses/literals presenting some problem. The key ideas of the system are:

1. Identify all the revision points in the current theory using the misclassified training examples.

2. Generate a set of proposed modifications for each revision point using the revision operators. It starts from the revision point with the highest *potential*, defined as the number of misclassified examples that could be turned into correctly classified from a revision in that point. FORTE stops to propose revisions when the potential of the next revision point is less than the score of the best revision to date. Conceptually, each operator develops its revision using the entire training set. However, in practice, this is usually unnecessary and thus FORTE considers only the examples whose provability can be affected after by the revision.

3. Score each revision through the actual increase in theory accuracy it achieves, calculated as the difference between the misclassified examples which turned into correctly classified and the correctly classified examples which turned into misclassified because of the revision.

4. Retain the revision with the highest score.

5. Implement the revision in case the overall score is really improved.

The top-level algorithm is exhibited in 3. The algorithm finishes when it cannot find any revision capable of improving the score.

Algorithm 3 FORTE Algorithm (Richards and Mooney, 1995)

 1: **repeat**
 2:      generate revision points;
 3:      sort revision points by potential (high to low);
 4:      **for each** revision point **do**
 5:         generate revisions;
 6:         update best revision found;
 7:      **until** potential of next revision point is less than the score of the best revision to date
 8:      **if** best revision improves the theory **then**
 9:         implement best revision;
10: **until** no revision improves the theory;

Next we detail the learning setting and representation of the examples in FORTE, followed by the procedures for finding revision points and the algorithms employed by each revision operator.

Examples Representation and Learning Setting

Most ILP algorithms employ the extensional approach to represent examples and background knowledge. In this case, each example is a ground fact and the background knowledge is equally shared by all the examples. FORTE differs from them representing the examples intentionally as they are written as sets of clauses in the format

Ground Instances of Target Predicates ← *Conjunction of facts from the context.*

The head of the above clause is a set of positive and/or negative ground facts, with the same predicate as the concepts intended to be learned. The conjunction of facts from the context is a set of definite clauses confined to the example, i.e., the BK restricted only to that example. There is also a set of background clauses FDT common to all the examples. The FDT and the BK restricted to each example compose the background knowledge of the domain. In this way, each example can be considered as a partial interpretation (DE RAEDT, 1997).

113

Although the dataset is composed of partial interpretations, the learning requirement is that the positive instances of each example are logically entailed by the current hypothesis H' together with the background knowledge, i.e., $\forall \mathbf{e} \forall e_+ \in e, H' \wedge FDT \wedge B_e \vDash e_+$ and the negative instances of each example are not logically entailed by the hypothesis and background knowledge, i.e., $\forall \mathbf{e} \forall e_- \in e, H \wedge FDT \wedge B_e \nvDash e_-$, where e_+ represents the positive instances, e_- represents the negative instances and B_e is the background knowledge confined to each example e.

It should be noted from section 5.3.2 that most ILP systems learn a single target predicate only. FORTE, on the contrary, is able to learn multiple predicates simultaneously. Such a task is facilitated because the search space is composed of whole theories instead of individual clauses. When learning individual clauses there is a great chance that the final returned theory is strongly composed of locally optimal and unnecessarily long clauses. On the other hand, when learning whole theories the final hypothesis tends to be globally optimal and smaller. However, learning whole theories is known to be much more expensive than learning individual clauses.

Finding revision points in FORTE

FORTE identifies revision points by annotating proofs of incorrectly provable negative instances or by annotating attempted proofs of incorrectly unprovable positive instances. When the goal is to find the specialization revision points, all the provable instances are considered, since they are the ones whose provability may be affected by a specialization in the theory: any of these instances might become unprovable because of the specialization. These instances are either *True Positive* (TP) - correctly classified positive instances - or *False Positive* (FP) - misclassified negative instances. The algorithm for collecting specialization revision points is shown in Algorithm 4.

First, FORTE annotates each clause participating in the successful proof of the instances. The positive instances are annotated separately from the negative instances in the clauses. In case the clause has no annotation of false positive instances, it is discarded. The remaining clauses compose the set of specialization

114

Algorithm 4 FORTE Algorithm for collecting specialization revision points (RICHARDS, MOONEY, 1995)

Input: The current theory H' and the FDT and EP, the set of provable instances composed of the TP, the set of correctly provable positive instances; FP the set of incorrectly provable negative instances

Output: RPS, a set of clauses marked as specialization revision points, each one annotated with TPC, true positive instances relative to clause C, FPC, false positive instances relative to clause C, and PC the potential of the revision point

1: **for** each provable instance $e \in EP$ **do**
2:      $Ce \leftarrow$ clauses participating in the proof of the instance e using H' and FDT
3:      **for** each clause $C \in Ce$ **do**
4:          **if** $e \in FP$ **then**
5:              $FPC \leftarrow FPC \cup e$;
6:              $PC \leftarrow PC + 1$;
7:          **else**
8:              $TPC \leftarrow TPC \cup e$;
9:      $RPS \leftarrow RPS \cup Ce$;
10: **for** each clause $C \in RPS$ **do**
11:      **if** $PC = 0$ **then**
12:          delete C from RPS;

---

revision points. The true positive and false positive instances relative to the clauses are used to calculate the score of revision proposed to those points.

In case there are misclassified positive instances, the goal is to find generalization revision points. In this case, all the unprovable instances are considered, since they are the ones whose provability may be affected by a generalization in the theory: any of these instances might become provable because of the generalization. These instances are either *True Negative* (TN) - correctly classified negative instances - or *False Negative* (FN) - misclassified positive instances. In order to identify generalization revision points, it is necessary to make annotations from failed proofs of positive instances. Three types of points in the failed proof path are collected:

1. the literal in a clause responsible for the failure proof,

2. the clause whose body contains such literal and

3. the literals which might have contributed to the failure by assigning incorrect values to variables (it is a contribution point).

Thus, each time a backtrack occurs, the failed antecedent is noted and marked as a failure point. Next, the literals binding values to variables in failure points are collected recursively. Finally, the clauses with the failure antecedents and with the contribution antecedents are also marked as failure points. This process is also followed to identify which failure points are responsible for not proving negative instances, since they might become provable after a revision in such points. The list of TN and FN instances are used to calculate the potential of the revision point, and, after proposing some revision, to calculate the score of the revision in such a point. The procedure is exhibited as Algorithm 5.

Proposing revisions in FORTE

As stated before, the four basic revision operators add rules, delete rules, add antecedents to clauses and delete antecedents from clauses. FORTE revision operators are ultimately composed of these four basic operations, aggregated to techniques for escaping local maxima, such as deleting/adding several antecedents at once from/to a clause. Additionally, the system uses two operators based on inverse resolution (MUGGLETON, 1992), namely the absorption and identification operators. The operators are described in terms of the changes they make to the theory. However, recall that each one is *proposing* a revision and not really *implementing* it on the theory. This is only done after proposing all possible revisions and choosing the one with the highest score. In order to calculate the score, FORTE employs a simple evaluation function: the number of incorrect instances which become correct less the number of correct instances which become incorrect because of the revision. Next, we review the way FORTE revision operators work. Note that here we only describe the cases for non-recursive clauses. For more details, including how FORTE deals with recursive clauses, we refer the reader to the original paper (RICHARDS, MOONEY, 1995).

Specialization operators

Delete rule The clause marked as a specialization revision point is deleted from the theory. If there is only such a clause explaining a concept, it is replaced by

---

Algorithm 5 FORTE Algorithm for collecting generalization revision points (RICHARDS, MOONEY, 1995)

Input: The current theory H' and the FDT; EU, the set of unprovable instances composed of the TN, the set of correctly unprovable negative instances and FN the set of incorrectly unprovable positive instances

Output: RPG, a set of clauses marked as generalization revision points, each one annotated with TNC, true negative instances relative to clause C, FNC, false negative instances relative to clause C, and PC the potential of the revision point

1: **for** each unprovable instance $e \in EU$ **do**
2:      Try to prove instance e using H' and FDT
3:      **for** each time that a literal fails **do**
4:          collect the failed literal lfe;
5:          collect the literals LCe responsible for binding variables in lfe, recursively
6:          collect the clauses Ce where lfe failed and where LCe appeared
7:          **if** $e \in TN$ **then**
8:              $TN\_lfe \leftarrow TN\_lfe \cup e$
9:              **for** each literal $lce \in LCe$ **do**
10:                 $TN\_lce \leftarrow TN\_lce \cup e$
11:             **for** each clause $ce \in Ce$ **do**
12:                 $TN\_ce \leftarrow TN\_ce \cup e$
13:         **else**
14:             $FN\_lfe \leftarrow FN\_lfe \cup e$
15:             $P\_lfe \leftarrow P\_lfe + 1$
16:             **for** each literal $lce \in LCe$ **do**
17:                 $FN\_lce \leftarrow FN\_lce \cup e$
18:                 $P\_lce \leftarrow P\_lce + 1$
19:             **for** each clause $ce \in Ce$ **do**
20:                 $FN\_ce \leftarrow FN\_ce \cup e$
21:                 $P\_ce \leftarrow P\_ce + 1$
22:         $RPG \leftarrow RPG \cup lfe \cup LCe \cup Ce$
23: **for** each point $P \in RPG$ **do**
24:     **if** $PG = 0$ **then**
25:         delete P from RPG;

---

concept :- fail. This simple process is stated as Algorithm 6.

Add Antecedents   The operation of adding antecedents to clauses works through two nested processes: the innermost case adds antecedents to the input clause in an attempt to make as many as possible negative instances become unprovable. This specialized clause is included in the revision. In case the specialized clause makes previously provable positive become unprovable, the outer case restarts the special-

---

Algorithm 6 FORTE Delete Rule Revision Operator Algorithm

Input: The current theory H' and the FDT, a clause C marked as specialization revision point together with TPC, the set of correctly provable positive instances using clause C and FPC, the set of incorrectly provable negative instances using clause C

Output: A proposed revision Rev together with score Sc

 1: $Rev \leftarrow H' - C$;
 2: **if** C is the only clause explaining *concept* **then**
 3:     $Rev \leftarrow Rev \cup concept : -fail$;
 4: $Sc \leftarrow$ calculate_score(Rev, FDT, FPC, TPC)

---

ization from the original input clause, looking for alternative specializations which retain the proof of positive instances while still making the negative instances unprovable. The top-level process of this revision operator is exhibited as Algorithm 7.

Algorithm 7 FORTE Top Level Add Antecedents Revision Operator Algorithm

---

Input: The current theory H' and the FDT, a clause C marked as specialization revision point together with TPC, the set of correctly provable positive instances using clause C and FPC, the set of incorrectly provable negative instances using clause C

Output: A proposed revision Rev, containing one or more clauses specialized from C, together with score Sc

 1: $H' \leftarrow H - C$
 2: $Rev \leftarrow H'$;
 3: **repeat**
 4:     $C' \leftarrow$ addAntecedents(C, H', FDT, TPC, FPC);
 5:     **if** C' is different from C **then**
 6:         $Rev \leftarrow Rev \cup C'$;
 7:         $FNC \leftarrow$ instances in TPC which become unprovable by $Rev \cup FDT$
 8:         $TPC \leftarrow FNC$
 9: **until** $FNC = \emptyset$ or it is not possible to create a specialized version of C ($C' ==$ C)
10: $Sc \leftarrow$ calculate_score(Rev, FDT, FPC, TPC)

There are two algorithms for adding antecedents to a clause, which may be executed in replacement to line 4 of Algorithm 7:

1. Hill Climbing (Algorithm 8) - This algorithm follows FOIL (QUINLAN, 1990), adding one antecedent at a time. It works as follows. First, all possible antecedents are created and scored using a slightly modified version of FOIL scoring function, displayed in formula 2. There, *Old_score* is the score of the

118

clause without the literal being evaluated, $\#TPCA$ is the number of positive instances proved by the clause with the literal added to it and $\#FPCA$ is the number of negative instances proved by the clause with the literal. The difference concerns the fact that FOIL score counts the number of proofs of instances, whereas FORTE counts the number of provable instances, ignoring the fact that one instance may be provable in several different ways. Next, the antecedent with the best score is selected. If the best score is better than the current clause score, the antecedent is added to the clause. This process continues until either there are no further antecedents to be added to the clause or no antecedent can improve the current score. This approach is susceptible to local maxima.

$$foil\_based\_score = \#TPCA*(Old\_score - \log(\#TPCA/(\#TPCA + \#FPCA)))$$

(2)

Algorithm 8 Hill climbing add antecedents Algorithm

Input: The current theory H' and the FDT, a clause C, TPC, the set of correctly provable positive instances using clause C and FPC, the set of incorrectly provable negative instances using clause C

Output: A (specialized) clause C'

1: **repeat**
2:   $antes \leftarrow$ generateAntecedents(C);
3:   $Ante \leftarrow$ best antecedent from $antes$, scored with FPC and TPC;
4:   **if** $score\_(C + \cup ante) > score\_(C)$ **then**
5:     $C \leftarrow C \cup ante$;
6:   $FPC \leftarrow FPC -$ instances in FPC not proved by C;
7: **until** $FPC = \emptyset$ or it is not possible to improve the score of the current clause
8: **return** C

---

2. Relational Pathfinding (Algorithm 9 - This approach adds a sequence of antecedents to a clause at once in attempt to skip local maxima, as, sometimes, none of the antecedents put individually in the clause improves its performance.

The Relational Pathfinding algorithm is based on the assumption that generally in relational domains there is a path with a fixed set of relations connecting a set of terms, and such path satisfies the target concept. Its goal is to find

such paths given a relational domain, since important concepts are represented by a small set of fixed paths between terms defining a positive instance. In order to find the paths, the relational domain is represented as a graph where the nodes are the terms and the edges are the relations among them. Thus, a *relational path* is defined as the set of edges (relations) which connect nodes (terms) of the graph. To better visualize such an approach, consider, for instance, the graph in Figure 3, which represents part of the family domain. Horizontal lines denote marriage relationships, and the remaining lines denote parental relationships:
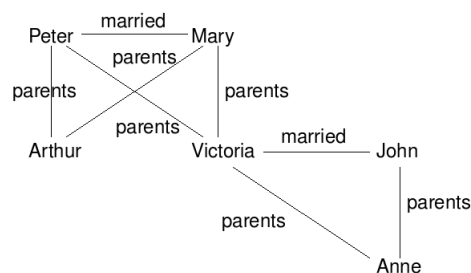
Figura 3: An instance of a relational graph representing part of the family domain (RICHARDS, MOONEY, 1995)

Now, suppose the goal is to learn the target concept *grandfather*, given an empty initial rule and the positive instance *grandfather(peter,anne)*. The relational path between the terms *peter* and *anne* is composed of the relation *parents* connecting *peter* to *victoria*, and also of the relation *parents* connecting *victoria* to *anne*. From these relations, the path *parents(peter,victoria)*, *parents(victoria,anne)* is formed, which can be used to define the target concept $grandfather(A, B) : -parents(A, C), parents(C, B)$.

From the point of view of theory revision, this algorithm can be used whenever a clause needs to be specialized and it does not have relational paths connecting its variables. In this case, a positive instance proved by the clause is chosen to instantiate it, and, from it, relational paths to the terms without a relationship in the clause are searched.

If the found relations introduce new terms appearing only once, FORTE tries

to complete the clause by adding relations that hold between these singletons terms and other terms in the clause; these new relations are not allowed to eliminate any of the currently provable positive instances. If FORTE is unable to use all of the new singletons, the relational path is rejected.

Algorithm 9 Top level Algorithm for Adding Antecedents to a Clause Using Relational Pathfinding Approach

---

Input: The current theory H' and the FDT, a clause C, TPC, the set of correctly provable positive instances using clause C and FPC, the set of incorrectly provable negative instances using clause C
Output: A (specialized) clause C'
1: $ex \leftarrow$ a positive instance from TPC, covered only because of C (and not because of others clauses with the same head);
2: find all clauses created from C and from paths generated through the terms in head of ex;
3: $C' \leftarrow$ the clause retaining the most instances in TPC as provable, or, in case of a tie, the shortest clause
4: $FNC \leftarrow$ negative instances still provable
5: **if** $FNC \neq \emptyset$ **then**
6:     $C' \leftarrow$ Hill climbing add antecedents algorithm(C', TPC, FPC)
7: **return** C'

Antecedents Generation   Following FOIL approach, all the predicates in the knowledge base are considered for creating literals to be added to the clause. A literal is created from a predicate by instantiating its arguments by variables, while respecting the following constraints:

1. At least one variable of the new literal must be in the clause being revised;

2. The arguments of the literals must obey the types defined in the knowledge base.

The larger the number of new variables in the clause is, the more literals are created. Actually, the space complexity grows exponentially in the number of new variables since the complexity of enumerating all possible combinations of variables is exponential in the arity of the predicate.

The algorithms for generating literals used by Hill climbing approach and Relational Pathfinding can be seen in Algorithm 10 and Algorithm 11, respectively.

Algorithm 10 Hill Climbing Antecedents Generation Algorithm

Input: A clause C
Output: A set of literals *antes*
1: **for** each literal *lit* in the knowledge base **do**
2:     $varsC \leftarrow$ variables in C with their types;
3:     $argsL \leftarrow$ terms in *lit* with their types;
4:     **for** each combination *comb* of variables $\in varsC$ in the arity of *lit* **do**
5:         **if** *comb* is compatible with $argsL$ considering the types in $argsL$ **then**
6:             create a new antecedent *ante* by replacing the terms of *lit* with the variables in *comb*;
7:             $antes \leftarrow antes \cup ante$;
8:     $n \leftarrow$ arity of *lit* - 1;
9:     $i \leftarrow 1$;
10:    **while** $i \leq n$ **do**
11:        create a new variable v
12:        $varsN \leftarrow varsC \cup v$
13:        **for** each combination *comb* of variables $\in varsN$ in the arity of *lit*, including at least one variable $\in varsC$ **do**
14:            **if** *comb* is compatible with $argsL$ considering the types in $argsL$ **then**
15:                create a new antecedent *ante* by replacing the terms of *lit* with the variables in *comb*;
16:                $antes \leftarrow antes \cup ante$;
17:        $i \leftarrow i + 1$;

As already mentioned, the Relational Pathfinding algorithm starts from a clause grounded from a positive instance covered by the clause. The terms in the ground clause will be the nodes in the graph, connected by the relations defined in the body of the clause. The algorithm constructs the graph iteratively, starting from these initial nodes and expanding them until finding the relational paths. The *end values* are the terms (nodes) created when a node is expanded.

In practice, Relational pathfinding and Hill climbing algorithms might be executed competitively and then the clause chosen in the inner loop is the one with the highest FOIL score. Also, if it is desired, only one of these two approaches may be executed.

Generalization operators

Algorithm 11 Relational Pathfinding Antecedent Generation Algorithm (RICHARDS, MOONEY, 1995)

Input: A clause C
Output: A set of sequence of literals *paths*
1: $CI \leftarrow C$ instantiated with a randomly chosen positive instance;
2: find isolated sub-graphs among the terms in CI;
3: **for** each sub-graph **do**
4:     terms become initial end-values;
5: **repeat**
6:     **for** each sub-graph **do**
7:         expand paths by one relation in all possible ways;
8:         remove paths with previously seen end-values;
9: **until** intersection found or resource bound exceeded
10: **if** one or more intersections found **then**
11:     **for** each intersection **do**
12:         $C' \leftarrow C$ with path-relations added;
13:         **if** C' contains new singleton variables **then**
14:             add relations using the singleton variables;
15:             **if** all singletons cannot be used **then**
16:                 discard C';
17:         replace terms with variables;
18:         *paths* $\leftarrow$ *paths* $\cup\, C'$;

Delete antecedents The algorithm followed by the delete antecedents operator is shown in Algorithm 12. The antecedents are deleted from a clause marked as a generalization revision point using a hill climbing approach or a method that removes a set of antecedents simultaneously, in case the hill climbing has no success. The generalized clause is added to the proposed revision and the process restarts, in an attempt to make remaining false negative instances become provable, until it is not possible to create useful generalized versions of the original clause. The two methods used to remove antecedents are described below.

1. The first method deletes one antecedent from the clause at each time, following a hill-climbing approach. FORTE chooses the antecedent whose removal makes the largest number of unprovable positive instances become provable, requiring that no unprovable negative instance becomes provable. This process is iterated until there are no antecedents in the clause whose deletion is going to make misclassified positive instances become provable. The process is exhibited as Algorithm 13.

2. Multiple antecedents are removed from the clause at once, in order to overcome the vulnerability of the former approach to get stuck in local maxima. First, the antecedents whose individual removal does not allow true negative instances become provable are collected. Then, combinations of such antecedents are produced, looking for the combination that makes the largest number of positive instances become provable without making negative instances become provable. The process continues deleting antecedents in this way, trying to prove as many positive instances as possible. This algorithm is computationally expensive and it is only executed when the hill climbing approach cannot propose any modification in the current theory. It is shown as Algorithm 14.

Algorithm 12 Top-level Delete Antecedents Revision Operator Algorithm

---

Input: The current theory H' and the FDT, a clause C marked as generalization revision point together with TNC, the set of correctly unprovable negative instances because clause C and FNC, the set of incorrectly unprovable positive instances possibly because of clause C

Output: A proposed revision Rev, containing one or more generalized versions of clause C, together with its score Sc

1:    $Rev \leftarrow H' - C$
2:    $stop \leftarrow false$
3:    **repeat**
4:       $C' \leftarrow$ hillClimbingDeleteAntecedents(C, TNC, FNC, Rev, FDT);
5:       **if** $C' = C$ **then**
6:         $C' \leftarrow$ delMultipleAntecedents(C, TNC, FNC, Rev, FDT);
7:         **if** $C' = C$ **then**
8:           $stop = true$
9:       **if** $C' \neq C$ **then**
10:        $Rev \leftarrow Rev \cup C'$
11:        $FNC \leftarrow FNC-$ instances in FNC which become provable by Rev
12:        **if** $FNC = \emptyset$ **then**
13:          $stop \leftarrow true$
14: **until** $stop == true$

Add rules FORTE implements two operators for adding rules to the current theory. It may create rules from an existing one or it may create a completely new rule from scratch. In the first case, the operator makes a copy of the clause marked as

---

Algorithm 13 Hill Climbing Delete Antecedents Algorithm

Input: A clause C, TNC, a set of unprovable negative instances, FNC, a set of
    unprovable positive instances, a theory H'', the static BK FDT

Output: A (generalized) clause C

1:   $stop \leftarrow false$
2: **repeat**
3:      **for** each antecedent $ante \in C$ **do**
4:          $CTemp \leftarrow C - ante$
5:          $score\_CTemp \leftarrow$ number of instances in FNC which are proved by
         $H'' \cup CTemp \cup FDT$
6:          $score2\_CTemp \leftarrow$ number of instances in TNC which are proved by
         $H'' \cup CTemp \cup FDT$
7:          **if** $score\_CTemp <= 0$ or $score2\_CTemp > 0$ **then**
8:             discards $CTemp$
9:          $antes \leftarrow (CTemp, score\_CTemp)$
10:     **if** $antes\ neqemptyset$ **then**
11:         $C \leftarrow CTemp \in antes$ with the highest score $score\_CTemp$
12:     **else**
13:         $stop \leftarrow true$
14: **until** $stop = true$
15: **return** C

Algorithm 14 Delete Multiple Antecedents Algorithm

Input: A clause C, TNC, a set of unprovable negative instances, FNC, a set of
    unprovable positive instances, a theory H'', the static BK FDT

Output: A (generalized) clause C'

1:   $antes \leftarrow$ all antecedents in C whose deletion does not change TNC
2: **repeat**
3:      $ante \leftarrow$ an antecedent in $antes$
4:      $CTemp \leftarrow C - ante$
5:      **if** no negative instance in TNC become provable by $H'' \cup CTemp \cup FDT$
     then
6:          $C \leftarrow C - ante$;
7: **until** there are no antecedents left to try
8: **if** one or more positive instances in FNC become provable by $H'' \cup CTemp \cup$
  FDT **then**
9:      **return**   C;

---

a generalization revision point and tries to modify such a copy of the clause in two steps. First, antecedents are deleted from the clause while they make false negative instances become provable. Antecedents are deleted even though true negative instances become provable. Then, the next step of the add rule operator is to add antecedents to the created clause in an attempt to make such negative instances be

unprovable again. This operation is performed by the add antecedents operators as previously explained. The process is exhibited as Algorithm 15. The second add rule operator starts by creating the head of the rule from a predicate marked as a generalization revision point. The next step is to compose the body of the clause, using the add antecedents operator.

Algorithm 15 Top-level Add rule Revision Operator Algorithm

Input: The current theory H' and the FDT, a clause C marked as a generalization revision point together with TNC, the set of correctly unprovable negative instances because clause C and FNC, the set of incorrectly unprovable positive instances possibly because of clause C

Output: A proposed revision Rev, containing one or more generalized versions of clause C, together with its score Sc

1: $H'' \leftarrow H' \cup C$
2: C' clause C after deleting antecedents which make instances from FNC become provable
3: $H'' \leftarrow H' \cup C'$
4: $TPC \leftarrow$ instances in FNC which become provable by $H'' \cup FDT \cup C'$
5: $FPC \leftarrow$ instances in TNC which become provable by $H'' \cup FDT \cup C'$
6: $Rev' \leftarrow$ addAntecedents(H'', FDT, C', TPC, FPC);

Appendix B: YAVFORTE: A revised version of FORTE, including Mode Directed Inverse Entailment and the Bottom Clause

Introduction

Theory revision systems usually induce more accurate theories than ILP techniques learning from scratch. However, such more accurate theories come at the expense of searching in a large search space, mainly because theory revision refines whole theories instead of individual clauses and this is known to be a hard problem (WROBEL, 1996; BRATKO, 1999). Therefore, it is essential to develop efficient theory revision systems so that the advantages of them become feasible. Focusing on FORTE theory revision system, in this chapter we contribute towards this goal by identifying a number of the bottlenecks of the revision process and developing algorithms based on state-of-the-art ILP systems to overcome it. The worst bottleneck is related to generation of literals to be included in the body of clauses, which is done inside the add antecedents and add rules operators. FORTE followed the FOIL top-down approach (QUINLAN, 1990), considering all the literals of the knowledge base to create antecedents to clauses, which leads to a huge search space, dominating the cost of the revision process. Instead of following a pure top-down approach when specializing clauses, ILP algorithms such as Progol (MUGGLETON, 1995a) and Aleph (SRINIVASAN, 2001a) restrict the search for literals to those belonging to the *Bottom Clause*. The bottom clause contains the literals relevant to a positive

example, collected from a Mode Directed Inverse Entailment (MDIE) search in the BK. This hybrid bottom-up and top-down approach often generates much fewer literals, and they are also guaranteed to cover at least one positive example (the one used to generate the bottom clause) and for this reason it is a good option to reduce the search space for literals in the revision process. As part of this work, in (DUBOC, 2008) and (DUBOC, et al., 2009) the Bottom Clause approach is introduced as the search space of literals to speed up FORTE. We describe that process here and also how to further take advantage of the bottom clause by (1) allowing its generation to start from a base clause and (2) using the current theory to create literals, in addition to the BK. Moreover, we use mode directed search when deleting antecedents from a clause so that after the deletion the clause is still valid according to the mode declarations.

Besides following FOIL's pure top-down approach, FORTE has six different revision operators and may try to propose revisions suggested by all of them in a single step, depending on the revision points found. This may lead to a large search space of operators. In many tasks, the expert of the domain has some idea about the kind of modifications the theory needs to represent. This could be used as an advantage for reducing the search space of operators. Another weakness of FORTE is that the delete antecedent operator does not allow any incorrectly provable negative example to become unprovable, which may cause overfitting of the clause or not generate a revision by this operator because of such a hard requirement.

In this chapter we describe a number of modifications performed on the FORTE system to handle the shortcomings listed above. We call the resulting system as YAVFORTE (Yet Another Version of FORTE), which includes $FORTE\_MBC$ (DUBOC, 2008; DUBOC, et al., 2009) to create the search space of new literals. The chapter starts by depicting the modified top level revision process in section 5.3.2. Next, modifications implemented on the revision operators are devised in section 5.3.2, concerning mainly the use of Bottom clause and Mode Directed Search. Experimental results are shown in section 5.3.2, followed by conclusions about this work in section 5.3.2.

Restricting the Search Space of Revision Operators

The original FORTE system proposes modification in the theory through six different revision operators, four designed to generalize the theory and two designed to specialize the theory. Moreover, it tries to apply all these possible operators, even when one of them has already achieved its maximum potential[1] or a maximum score. Suppose for example that we have a clause proving 10 positive examples and 20 negative examples. Now, suppose an extreme case where these 10 positive examples are also covered by another clause. Therefore, the simplest way to rectify the theory would be deleting that clause. However, besides FORTE proposing the deletion of the clause, it would also try to add antecedents to the body of the clause, so that the negative examples become unprovable, which is clearly a waste of time. Thus, in order to make the revision proposals more flexible and efficient, we modified the revision process by including two amendments:

1. The user is able to stipulate which operators the system must apply in order to propose modifications in the theory. In case the user does not specify any operator, the system tries to apply all of them, according to the revision points. Thus, the revision process is able to only specialize/generalize the theory, as well as to apply a subset of specialization and/or generalization operators when proposing modifications.

2. Instead of applying all possible revision operators on a revision point, the system establishes an order of simplicity to apply the operators. Using this order, in case a simpler operator has already achieved the maximum potential or a maximum score, the system stops to propose modifications. Put in other words, the system only applies a more complex operator when no simpler operator simpler than it was able to attain the maximum potential or score. The order imposed to specialization operators is first to apply the delete rule and then the add antecedents operator, clearly because the number of operations performed to evaluate the delete rule is much less than when proposing addi-

---

[1]Remember from the previous chapter that the potential is the number of examples indicating the necessity of revision in one point.

tion of antecedents. The order imposed to generalization revision operators is delete antecedents, absorption, identification, add rule. Delete antecedents has as search space only the literals belonging to the initial clause. Identification and absorption defines their search space in terms of the literals presented in the theory. Add rule first uses delete antecedents and then add antecedents operators. This last operator may have a large number of literals to evaluate, depending on the background knowledge and mode definitions, therefore we assume it as the more complex generalization revision operator.

Algorithm 16 presents the modified revision process, built upon Algorithm 3 of chapter 1.

Algorithm 16 YAVFORTE Top-Level Algorithm

Input: An initial theory T, background knowledge FDT, a set of examples E, list of applicable operators Rev
Output: A revised theory T'

1: **if** $Rev = \emptyset$ **then**
2:     $Rev \leftarrow$ all revision operators
3: $GenRev \leftarrow$ ordered list of generalization operators in Rev, starting from the simplest one
4: $SpecRev \leftarrow$ ordered list of specialization operators in Rev, starting from the simplest one
5: **repeat**
6:     generate revision points;
7:     sort revision points by potential (high to low);
8:     **for each** revision point RP **do**
9:         **if** RP is a specialization revision point **then**
10:             **for** each revision operator $RO \in SpecRev$ **do**
11:                 apply RO in RP
12:                 compute $score_{RO}$
13:         **else**
14:             **for** each revision operator $RO \in GenRev$ **do**
15:                 apply RO in RP
16:                 compute $score_{RO}$
17:         update best revision found;
18:         **until** $score_{RO} =$ maximum score or RO achieved maximum potential of RP
19:     **if** best revision improves the theory **then**
20:         implement best revision;
21: **until** no revision improves the theory;

Improvements Performed on the Revision Operators

This section starts by reviewing the theory behind bottom clause and MDIE, since we use them to restrict the search space of the add antecedents and the delete antecedents operators. Next, the operator developed in (DUBOC, et al., 2009) to add antecedents is reviewed but also including improvements performed on it in this work. Then, we describe modifications implemented on delete antecedents operator.

Using the Bottom Clause as the Search Space of Antecedents when Revising a FOL theory

FORTE, following FOIL, generates literals to be added to a clause obeying two conditions: (1) the variables of the literals must follow their types defined in the knowledge base and (2) they must have at least one variable in common with the current clause. While this makes the generation of antecedents simple and fast, it also leads to a large search space composed of all the possible literals of the knowledge base. Such a large search space turns the complexity of the add antecedents operation in a clause very high, contributing to the bottleneck of the revision process. Aiming to reduce such cost, we implemented the following modifications to FORTE system:

1. The variabilized bottom clause generated by Algorithm 2 became the search space of literals, which reduces the search space and also impose the following constraints:

   - Limits the maximum number of different instantiations of a literal (the recall number);

   - Limits the number of new variables in a clause;

   - Guarantees that at least one positive example is covered (the one which generates the bottom clause).

2. The mode declarations are used to further constrains the antecedents, which means they only may be added to the clause if their terms respect the mode defined in the knowledge base.

3. Determination definitions of the form

$$determination(HeadPredicate/Arity, BodyPredicate/Arity)$$

state which predicates can be called in the clauses defining *HeadPredicate*.

The modified version of FORTE considering mode declarations and the bottom clause is called *FORTE_MBC*. Recall that when specializing a clause, the goal of the operation is to make false positive examples become unprovable while still covering true positive examples. Thus, the bottom clause is created immediately before the search for antecedents begins, by saturating a positive example covered by the clause being specialized (called base clause). The bottom clause is going to be composed of the literals relevant to at least such a positive example and it is guaranteed to be a super-set of the base clause. The created bottom clause becomes the search space for antecedents, which improves the efficiency of the addition antecedents operation since it usually has many fewer literals than the previous space of the whole knowledge base. It is important to emphasize that the constraints of FOIL continue to be met here, as the arguments of the literals in the bottom clause must obey their types and there is a linking variable between the literal being added in the clause and the literals of the current clause.

Algorithm 17 shows the process of constructing the bottom clause in YAV-FORTE. It differs from Algorithm 2 when the base clause has a non-empty body, since in this case it is necessary to take into account the terms of the current clause. Note that in (DUBOC, et al., 2009) the variables of the bottom clause were unified with the variables of the base clause *only after* the Bottom Clause had been constructed. Besides this being an expensive process involving lots of backtracks, sometimes it was not possible to find a correct unification. We noticed two of such situations: cases where the example contains two or more equal terms in the head but the base clause has two different variables in their place (1) and when the base clause has no constant in the head but the first `modeh` has a constant, or vice-verse (2). In these cases the Bottom Clause would follow the unification according to the example, differing from the base clause and making more difficult to find a substitution that would match the bottom clause and the base clause. Thus, the first

132

step in Algorithm 17 is to find the ground literals of the base clause considering the example but maintaining the substitution of the variables in the base clause. Then, the terms of the base clause are put in the list of terms to be used by the procedure and the bottom clause is initialized with the literals of the base clause. The rest of the process is the same as the original algorithm, except that we do not allow the inclusion of a literal being already in the bottom clause. Note that if the clause is being constructed from scratch, it is not necessary to keep an association with the base clause passed as input argument. In this case, we completely follow the original algorithm.

Another difference from the algorithm developed in (DUBOC, et al., 2009) is the input: there, the bottom clause is constructed considering only the BK but here we also allow the current theory to be used to produce literals. This is essential in case we have intermediate clauses in the current theory.

Next, we show how the Bottom Clause is used as the space of new literals.

Using the Bottom Clause in Hill Climbing Add Antecedents Algorithm

The Hill Climbing add antecedents algorithm modified from Algorithm 2.6, to take into account the bottom clause as search space, is detailed in Algorithm 18. Both algorithms differ in three aspects:

1. Algorithm 18 has as first step the construction of the bottom clause in line 2, as it is used as search space for antecedents. In order to obtain the Bottom Clause from a positive instance correctly proved by the base clause it is used Algorithm 17.

2. Such a bottom clause becomes the input for antecedents generation in line 4. This procedure is soon going to be explained in details.

3. As the bottom clause created in the beginning is not modified during the execution of this algorithm, it is necessary to remove the antecedent added to the clause from the bottom clause, in line 11. From this last difference, it follows that the algorithm also stops when there is no more literal left in the bottom clause.

Algorithm 17 Bottom clause Construction Algorithm in FORTE-MBC

Input: The current theory H' and the FDT, a clause C, Ex, an instance
Output: The bottom clause BC

1: **if** C has an empty body **then**
2:      $\bot \leftarrow$ Algorithm 2(H', FDT, Ex);
3: **else**
4:      $InTerms \leftarrow \emptyset$, $\bot \leftarrow \emptyset$
5:      $C_{ground} \leftarrow$ instantiation of the clause C using H', FDT, and Ex, with substitution θ maintaining the variables of C
6:      **for** each v/t in θ **do**
7:          $InTerms \leftarrow InTerms \cup t$
8:      $\bot \leftarrow \bot \cup C$
9:      $i \leftarrow 0$, corresponding to the variables depth
10:      $BK \leftarrow FDT \cup Ex$
11:      **for each** modeb declaration b **do**
12:          **for** all possible substitution θ of arguments corresponding to + type by terms in the set $InTerms$ **do**
13:              **repeat**
14:                  **if** b succeeds with substitution θ' **then**
15:                      **for each** v/t in θ and θ' **do**
16:                          **if** v corresponds to $\sharp$ type **then**
17:                              replace v in b by t
18:                          **else**
19:                              replace v in b by v_k, where $k = hash(t)$
20:                          **if** v corresponds to $-$ type **then**
21:                              $InTerms \leftarrow InTerms \cup t$
22:                    **if** $b \notin C$ **then**
23:                      $\bot \leftarrow \bot \cup \bar{b}$
24:              **until** reaches recall times
25:      $i \leftarrow i + 1$
26:      Go to line 14 if the maximum depth of variables is not reached
27: **return** $\bot$.

---

4. There is a parameter specifying the maximum size a clause is allowed to have.

Using the Bottom Clause in Relational Pathfinding Add Antecedents Algorithm

The Relational Pathfinding algorithm adding more than one antecedent at once in a clause and considering as search space the bottom clause is exhibited in Algorithm 19.

There are only two major differences between Algorithm 19 and Algorithm 2.7: the creation of the bottom clause from a positive example covered by the

---

Algorithm 18 Hill Climbing Add Antecedents Algorithm Using the Bottom Clause

---

Input: The current theory H' and the FDT, a clause C, TPC, the set of correctly provable positive instances using clause C and FPC, the set of incorrectly provable negative instances using clause C, CL, maximum size of a clause

Output: A (specialized) clause C'

1: $Ex \leftarrow$ an instance from TPC;
2: $BC \leftarrow$ createBottomClause(Ex, H', FDT, C); %use Algorithm 17
3: **repeat**
4:      $antes \leftarrow$ getAntecedentsfromBC(C, BC);
5:      $Ante \leftarrow$ best antecedent from $antes$, scored with FPC and TPC;
6:      **if** $score\_(C \cup ante) > score\_(C)$ **then**
7:          $C \leftarrow C \cup ante$;
8:      remove $Ante$ from BC
9:      $FPC \leftarrow FPC-$instances in FPC not proved by C;
10: **until** $FPC = \emptyset$ or there are no more antecedents in BC or it is not possible to improve the score of the current clause or $|C| = CL$
11: **return** C

---

base clause happens before the searching for relational paths (1); consequently, the bottom clause is used as search space for paths, together with the same positive example used to generate the bottom clause and the base clause (2).

Using the Bottom Clause as Search Space for Antecedents Generation

The original FORTE dynamically generates antecedents at each iteration of the add antecedents procedure, since it is necessary to collect the variables of the current clause to create new literals. FORTE_MBC, on the contrary, generates literals statically, at the beginning of the process by creating the bottom clause. However, not every literal in the whole bottom clause can be added to a current clause in a specific moment. Suppose, for example, the clause $head(A, B)$. whose body is empty is being specialized. The `modeh` definition to this predicate is $modeh(1, head(+ta, +ta))$, indicating that it can be used in the head of a clause and it is allowed only one instantiation of it ($recall = 1$) since we deal with definite clauses, and their arguments are both of input, whose types are ta. Consider the BK and a positive instance producing the bottom clause

$$head(A, B) : -body_1(A, C), body_2(B, C), body_3(C, A).$$

Algorithm 19 Top-level Relational Pathfinding Add Antecedents Algorithm Using the Bottom Clause

Input: The current theory H' and the FDT, a clause C, TPC, the set of correctly provable positive instances using clause C and FPC, the set of incorrectly provable negative instances using clause C

Output: A (specialized) clause C'

1: $Ex \leftarrow$ an example from TPC, covered only because of C (and not because of others clauses with the same head);
2: $BC \leftarrow$ createBottomClause(Ex, H', FDT, C); %use Algorithm 17;
3: find all clauses created from C and from paths generated through the terms in head of ex, considering BC as search space;
4: $C' \leftarrow$ the clause retaining the most instances in TPC as provable, or, in case of a tie, the shortest clause
5: $FNC \leftarrow$ negative instances still provable
6: **if** $FNC \neq \emptyset$ **then**
7:     $C' \leftarrow$ Hill climbing add antecedents algorithm(C', TPC, FPC)
8: **return** C'

based on the following mode declaration:

$$\{modeb(*, body_1(+ta, -ta)), modeb(*, body_2(+ta, -ta)), modeb(*, body_3(+ta, -ta)\}$$

which indicates that the clause can have infinite different instantiations of the predicates $body_i$ ($recall = *$) and the arguments of these predicates have both type ta, where the first one is an input term and the second one is an output term. In case every literal in the bottom clause is a candidate to specialize the base clause, the literal $body_3(C, A)$ could be added in the current clause. However, notice the variable C is in the place of an input term and therefore such a variable should have appeared before in the current clause, which is not the case. To consider only the FOIL constraint of having a connection variable between the clause and the literal is not enough, since in this case $body_3(C, A)$ would be valid because of the second variable. Note that such a literal is correctly placed inside the bottom clause because the variable C appeared before in $body_2(B, C)$. Because of that ILP systems such as Progol and Aleph take advantage of the order of literals in the bottom clause when choosing an antecedent to be added to a clause. We follow a different approach for collecting the eligible literals to be added to a clause, from a bottom clause. A literal in the bottom clause is a candidate to be included in a current clause if and

only if their input variables have already appeared before in another literal of the current clause. Thus, Line 4 of Algorithm 18 is composed of two steps: (1) collect the literals of the current bottom clause and (2) validate such candidate antecedents to verify if each one of them is actually allowed to be part of the clause, according to their input variables.

In regard to the Relational Pathfinding Algorithm we follow a slightly different approach. First of all, it is important to notice that the generation of antecedents to this algorithm is the same as the one performed in Algorithm 9, with only one obvious difference: now the literals are searched in the bottom clause generated at the beginning of the process. Thus, when looking for paths the literals considered to be in a path are the ones from the bottom clause, but still taking into account the end values of the relational paths. However, we do not validate literals concerning modes immediately before they are considered to be in a path, since more than one antecedent will be added at once. In this way, it may be the case the whole path is valid according to the modes but the isolated literal would not be. Thus, the whole path is validated according to mode declarations: if the relational path does not obey the modes it is discarded just before it is evaluated. Line 3 of Algorithm 19 returns the clauses created from C and from paths, but ensuring such clauses are valid according to mode declarations.

Remarks about the Complexity of Antecedents Addition

The revision process of FORTE has an exponential complexity in the size of the input theory and in the arity of the theory predicates (RICHARDS, MOONEY, 1995). When adding antecedents to a rule, the number of permutations of arguments to a predicate is an exponential function of the predicate's arity. Thus, the space complexity of possible literals grows exponentially on the number of new variables, since the complexity of enumerating all possible combinations of variables is exponential according to the arity of the predicate. On the other hand, the space complexity of the bottom clause is the cardinality of it and is bounded by $r(|M|j + j-)^{ij^+}$, where $|M|$ is the cardinality of M (the set of mode declarations), j^+ is the number of $+$ type occurrences in each modeb in M, plus the number of $-$ type occurrences in

each modeh, j^- is the number of $-$ type occurrences in each modeb in M plus the number of $+$ type occurrences in each modeh, r is the recall of each mode $m \in M$, and i is the maximum variable depth (MUGGLETON, 1995a). On the same way the use of the bottom clause brings the advantage of reducing the search space of Progol when compared to a top-down approach as FOIL, the use of the bottom clause also reduces the search space of antecedents of FORTE, which originally generated antecedents based on FOIL. However, it is important to point out that this advantage is only guaranteed if the i variable is small enough. In the extreme case that i and the background knowledge are both large, the bottom clause has a very large number of literals and hence the search space is as large as or even larger than the search space of FORTE. Considering exactly this problem, (TANG, et al., 2003) proposed the BETH system, which makes use of a hybrid top-down and bottom-up approach when constructing the bottom clause, aiming to reduce the cardinality of the bottom clause and consequently the search space of literals.

Modifying the Delete Antecedent Operator to use Modes Language and to Allow Noise

Using Mode Directed Search when Deleting Literals from a Clause

Besides using the theory of MDIE for generating the bottom clause and take advantage of it when adding literals to be added to a clause, we would also need to yield a theory that follows the modes language. This is not only a requirement to make the final theory valid according to the modes, but is also essential to the specialization procedure since, as it was said before, the first step is to include the terms in the base clause as terms to be used in the bottom clause. Such terms must correctly follow the modes, otherwise it will not exist a match for them in the bottom clause construction procedure.

To begin with, we assume the initial theory follows the modes established in the language bias. To continue following modes when revising the theory, it is necessary to validate each single proposed modification according to them. When specializing clauses, this is already done, since any literal to be added to the body of a clause comes from the bottom clause and immediately before to be indeed included,

it is checked if it obeys any mode. It remains to be checked whether a proposed modification does not make the theory invalid according to the modes when it is generalized. Thus, a mode check is included after deleting a literal from the body of the clause being generalized. It works as follows. Immediately before evaluating a literal to be removed from a clause, it is checked if the resulting clause follows one of the modes assertions. The inspection will prevent a literal to be removed in case it falls in one of two cases: (1) the literal to be removed is the only one with an output variable of the head of the clause; or (2) the literal to be removed is the only one with a variable that is input to other literal. Both cases would make the clause not follow the modes language and therefore the procedure does not allow this to happen.

Allowing noise in Delete Antecedent Operator

In addition to using modes definitions to validate a deletion of a literal from a clause, we also remodel the delete antecedents operator so that it becomes more flexible in the sense of allowing *noise*, i.e., true negative instances become provable. Rather than specifying a maximum number of negative examples to be considered as noise, as it is done in Aleph and Progol systems for example, the number of negative examples that a clause may cover is decided by the score of the clause. Thus, at each iteration the antecedent improving the score at most is selected. In case after deleting such an antecedent from the body of a clause the score is improved, yet a true negative instance becomes false positive, the procedure nevertheless proceeds to further deletions. The deletion of antecedents stops when it is not possible to improve the score, following a classical greedy hill climbing approach. This modus operandi is used in deletion antecedents operator and to delete antecedents in order to create a new rule from an existing one (add rules operator).

Experimental Results

We have already experimentally demonstrated in (DUBOC, et al., 2009) the benefits of using mode declarations and the bottom clause when revising FOL theories. Experimental results have presented a speed-up of 50 times compared to the original

---
Algorithm 20 Remodeled Greedy Hill Climbing Delete Antecedents Algorithm
---
Input: A clause C, TNC, a set of unprovable negative instances, FNC, a set of
  unprovable positive instances, a theory H'', the static BK FDT
Output: A (generalized) clause C
  1: $score_C \leftarrow$ compute current score
  2: **repeat**
  3:   **for** each antecedent $ante \in C$ **do**
  4:     $CTemp \leftarrow C - ante$
  5:     $score_{CTemp} \leftarrow$ compute score for $CTemp$
  6:     $antes \leftarrow (CTemp, score_{CTemp})$
  7:   $C \leftarrow CTemp \in antes$ with the highest score $score_{CTemp}$
  8:   **if** $score_{CTemp}$ is better than $score_C$ **then**
  9:     $C \leftarrow CTemp$
  10:    $score_C \leftarrow score_{CTemp}$
  11: **until** it not possible to improve score
  12: **return** C

FORTE system, without significantly decreasing accuracies. Additionally, we have
shown there that the revision system provides more accurate and smaller theories,
compared to a standard inductive method also based on the bottom clause. In this
chapter, we would like to know if it is possible to further decrease the runtime of
the revision process without decreasing the accuracy. With this goal, we present
the results obtained from the current implementation, varying the set of operators
used to revise the theory, compared to the implementation of FORTE containing
FORTE_MBC algorithm and also to Aleph system. We compare the average run-
time, accuracy and size of the theories, in number of clauses and literals.

Datasets   We consider the same datasets used in (DUBOC, et al., 2009), namely
the Alzheimer (KING, et al., 1995a) domain, composed of four datasets and the
DssTox dataset (FANG, et al., 2001). Alzheimer domain compares 37 analogues
of Tacrine, a drug combating Alzheimer's disease, according to four properties as
described below, where each property originates a different dataset:

1. inhibit **amine** re-uptake, composed of 343 positive examples and 343 negative
   examples

2. low **toxic**ity, with 443 positive examples and 443 negative examples

3. high acetyl **choline**sterase inhibition, composed of 663 positive examples and 663 negative examples and

4. good reversal of **scopolamine**-induced memory deficiency, containing 321 positive examples and 321 negative examples

Alzheimer domain considers 33 different predicates and 737 facts in background knowledge. DssTox dataset was extracted from the EPA's DSSTox NCTRER Database. It contains structural information about a diverse set of 232 natural, synthetic and environmental estrogens and classifications with regard to their binding activity for the estrogen receptor. The dataset is composed of 131 positive examples and 101 negative examples. There are 25 different predicates and 16177 facts in backgrund knowledge.

Experimental Methodology The datasets were split up into 10 disjoint folds sets to use a K-fold stratified cross validation approach. Each fold keeps the rate of original distribution of positive and negative examples (KOHAVI, 1995). The significance test used was corrected paired t-test (NADEAU, BENGIO, 2003), with $p < 0.05$. As stated by (NADEAU, BENGIO, 2003), corrected t-test takes into account the variability due to the choice of training set and not only that due to the test examples, which could lead to gross underestimation of the variance of the cross validation estimator and to the wrong conclusion that the new algorithm is significantly better when it is not. In this work, we are assuming that both modes and types definitions are correct and therefore cannot be modified. All the experiments were run on Yap Prolog (SANTOS COSTA, 2008).

The initial theories were obtained from Aleph system using three settings:

- The first setting runs Aleph with its default parameters, except for `minpos`[2], which was set to 2 to prevent Aleph from adding to the theory ground unit clauses corresponding to positive examples. It is identified in the Tables as **Theory-def**.

---

[2]Minpos parameter set a lower bound on the number of positive examples to be covered by an acceptable clause. If the best clause covers positive examples below this number, then it is not added to the current theory ()

141

- The second setting named as **Theory-def+cl**, runs Aleph with its default parameters, except for minpos and `clauselength` parameters, to set an upper bound on the number of literals in a clause. We choose this last parameter because YAVFORTE implementation also limits the number of literals in an acceptable clause. To all systems, clause length is defined as 5 in Alzheimer domain and 10 in Dsstox, following previous work (LANDWEHR, et al., 2007).

- The third setting, **Theory-best** runs Aleph with "literature" parameters according to previous work (HUYNH, MOONEY, 2008; LANDWEHR, et al., 2007). Thus, following (HUYNH, MOONEY, 2008), Aleph was set to consider `minscore` as 0.6, evaluation function is M-estimate (DZEROSKI, BRATKO, 1992), noise is 300 for Alzheimers and 10 for DSStox and clause length and minpos are defined as above. Additionally, `induce_cover` command was invoked instead of the standard "induce". The difference is that positive examples covered by a clause are not removed prior to seeding on a new example when using induce_cover. Note that only clause length parameter is used in YAVFORTE, as the others parameters are not implemented there.

To generate such theories, the whole dataset was considered but using a 10-fold cross validation procedure. Thus, a different theory was generated for each fold and each one of these theories was revised considering its respective fold (the same fold is used to generate and revise the theories).

In order to identify if there are any benefit on pre-defining the set of applicable operators when revising the theories, YAVFORTE is run with 5 different sets of operators. Four settings consider a combination of one specialization operator together with one generalization operator. The last setting, identified in tables as $YAVFORTE$ considers all operators.

- **YAV-del** considers delete rule as specialization operator and delete antecedents only as generalization operator.

- **YAV-add** considers only addition of antecedents as specialization operator and addition of rules as generalization operator.

142

- **YAV-add-del** considers additions of antecedents as specialization operator and delete antecedents as generalization operator.

- **YAV-del-add** considers delete rules as specialization operator and add antecedents as generalization operator.

Both YAVFORTE and FORTE_MBC run add antecedents algorithm considering Relational Pathfinding algorithm, followed by Hill climbing in Alzheimers domains. As DSSTox top-level predicate is unary, Relational Pathfinding is not applicable. Because of that, only Hill climbing algorithm is taken into account. Both systems considers clause length parameter as 5 for Alzheimers and clause length as 10 for DssTox.

Tabela 2: Runtime in seconds, predictive accuracy and size in number of literals and clauses of final theories for Alzheimer amine dataset

System	Theory-def			Theory-def+cl			Theory-best		
	Runtime (s)	Acc (%)	#Lits, #Clauses	Runtime (s)	Acc (%)	#Lits, #Clauses	Runtime (s)	Acc (%)	#Lits, #Clauses
Aleph	7.53	62.67	20.8 **5.5**	48.66	65.62	36.9 **8.5**	43.19	73.64	69.7 **7.16**
FORTE_MBC	21.96	71.38	34.6 8.7	26.29	75.09	35.5 8.6	69.09	76.87	52.5 12
YAV-del	**3.91** ◇●⋆	70.13 ◇⋆	**18.4** **5.5**	**6.01** ◇●⋆	75.09 ◇	**34.7** **8.5**	**12.89** ◇●⋆	76.87 ◇	**51.9** 11.8
YAV-add	7.29 ●⋆	70.48 ◇⋆	46.7 10.8	8.61 ◇●	74.94 ◇	53.8 11.9	15.31 ◇●⋆	76.55 ◇	79 17.2
YAV-add-del	9.14 ◇●⋆	71.92 ◇⋆	42.2 9.8	7.13 ◇●⋆	75.54 ◇	39.1 9.2	15.93 ◇●⋆	76.55 ◇	74.9 16.4
YAV-del-add	7.18 ●⋆	72.34 ◇⋆	43.2 10.1	8.93 ◇●	75.24 ◇	52.8 11.7	14.14 ◇●⋆	**77.57** ◇	55.5 12.4
YAV-FORTE	12.76 ◇●	**74.23** ◇●	34.6 8.7	11.15 ◇●	**75.68** ◇	38 9	29.12 ◇●	77.27 ◇	58.9 12.8

144

Tabela 3: Runtime in seconds, predictive accuracy and size in number of literals and clauses of final theories for Alzheimer toxic dataset

System	Theory-def			Theory-def+cl			Theory-best		
	Runtime (s)	Acc (%)	#Lits, #Clauses	Runtime (s)	Acc (%)	#Lits, #Clauses	Runtime (s)	Acc (%)	#Lits, #Clauses
Aleph	17.29	62.48	13 / **4.3**	53.57	63.85	32.2 / **8.1**	25.98	77.75	40.1 / 10.4
FORTE_MBC	22.54	**71.32** ◇	25.4 / 7.8	37.67	71.19	**27.9** / **8.1**	22.14	79	33.5 / 8.9
YAV-del	**2.93** ◇●★	68.48 ◇	**11.6** / 4.5	**8.79** ◇●★	71.19 ◇	28.1 / **8.1**	**4.38** ◇●★	**79.12** ◇	**32.4** / **8.7**
YAV-add	4.21 ◇●	63.62 ●★	22.3 / 6.3	16.57 ◇●	75.71 ◇●★	62.1 / 14.3	5.92 ◇●★	77.53 ●	51.5 / 11.8
YAV-add-del	4.64 ◇●	68.94 ◇	14.8 / 5.1	10.99 ◇●★	71.64	30.9 / 8.6	6.13 ◇●★	77.76 ●	51.4 / 11.8
YAV-del-add	4.69 ◇●	64.30 ◇●★	20.9 / 6	22.39 ◇●★	**75.93** ◇	53.6 / 12.6	4.42 ◇●★	78.89 ◇	32.5 / **8.7**
YAV-FORTE	6.10 ◇●	69.05 ◇	14.8 / 5.1	16.73 ◇●	71.64 ◇	30.9 / 8.6	11.51 ◇●	78.67	44.9 / 10.6

145

Tabela 4: Runtime in seconds, predictive accuracy and size in number of literals and clauses of final theories for Alzheimer choline dataset

System	Theory-def			Theory-def+cl			Theory-best		
	Runtime (s)	Acc (%)	#Lits, #Clauses	Runtime (s)	Acc (%)	#Lits, #Clauses	Runtime (s)	Acc (%)	#Lits, #Clauses
Aleph	39.23	56.02	30.7 **8.5**	100.63	56.62	39.33 **9.4**	147.94	64.47	74.9 17.1
FORTE_ MBC	100.73	61.01	34.6 9.8	79.21	64.93	40 10.3	228.18	**67.6**	**15.9** **8.9**
YAV-del	**19.83** ◇●★	65.21 ◇●	**27.7** 8.9	**24.13** ◇●★	64.48	**38.7** 10.1	**38.33** ◇●★	65.09 ●	66.2 15.6
YAV-add	37.57 ●★	64.32 ◇●	66.1 6.3	36.23 ◇●★	65.16 ◇	67.6 14.3	46.85 ◇●★	64.70 ●	91.1 19.8
YAV-add-del	46.05 ●★	62.98	60.2 15.2	39.07 ◇●★	**67.09**	51.9 12.1	59.22 ◇●★	64.94 ●	82.4 18.2
YAV-del-add	39.58 ●★	**66.32** ◇●★	63.7 15.2	38.63 ◇●★	65.39 ◇	65.6 14.4	51.42 ◇●★	64.93 ●	75 16.7
YAV-FORTE	64.36 ◇●	64.02 ◇●	51.7 13.5	70.57 ◇	64.93 ◇	40 10.3	98.76 ◇●	64.86 ●	77.8 16.7

Tabela 5: Runtime in seconds, predictive accuracy and size in number of literals and clauses of final theories for Alzheimer Scopolamine dataset

System	Theory-def				Theory-def+cl				Theory-best			
	Runtime (s)	Acc (%)	#Lits, #Clauses		Runtime (s)	Acc (%)	#Lits, #Clauses		Runtime (s)	Acc (%)	#Lits, #Clauses	
Aleph	15.96	51.71	24.9 **6.5**		41.40	51.41	31.3 **7.5**		57.24	58.42	45.3 10.5	
FORTE_MBC	47.43	61.53	37.8 9.7		42.56	**66.53**	37 9.2		48.04	**64.51**	45.1 10.9	
YAV-del	**4.97** ◇●★	60.74 ◇★	**22.8** **6.5**		**6.45** ◇●★	64.79 ◇	**28.3** **7.5**		**7.27** ◇●★	64.20 ◇	**40.9** **10**	
YAV-add	3.54 ◇●★	51.87 ●★	28.2 7.4		14.67 ◇●	62.02 ◇●★	57.5 13		15.02 ◇●★	62.17 ◇●★	66.9 14.8	
YAV-add-del	5.30 ◇●★	60.59 ◇★	23.70 6.7		7.70 ◇●★	63.87 ◇●★	29.8 7.8		8.9 ◇●★	64.35 ◇	45.8 11	
YAV-del-add	3.48 ◇●★	51.87 ●★	28.2 7.4		14.8 ◇●	62.02 ◇●★	57 12.9		14.06 ◇●★	61.70 ◇●★	62.6 13.9	
YAV-FORTE	13.93 ◇●	**62.47** ◇	37.5 9.6		15.68 ◇●	**66.53** ◇	37 10.3		48.04 ◇	**64.51** ◇	45.1 11.3	

Tabela 6: Runtime in seconds, predictive accuracy and size in number of literals and clauses of final theories for DssTox dataset

System	Theory-def			Theory-def+cl			Theory-best		
	Runtime (s)	Acc (%)	#Lits, #Clauses	Runtime (s)	Acc (%)	#Lits, #Clauses	Runtime (s)	Acc (%)	#Lits, #Clauses
Aleph	25.55	51.36	**10.2** **2.9**	50.07	51.36	**11.2** **2.9**	39.96	55.39	**9** **2.3**
FORTE_ MBC	4.26	71.37	31.4 6	4.27	71.37	31.4 4.2	3.12	**77.43**	20.8 4.2
YAV-del	0.25 ◇•⋆	49.08 •⋆	10.2 2.9	0.24 ◇•⋆	49.08 •⋆	10.2 2.9	0.15 •⋆	54.73 •⋆	8.4 2.3
YAV-add	2.47 ◇•	78.33 ◇•	40.5 7.4	2.46 ◇•	78.33 ◇•	40.5 7.4	1.5 ◇•	71.34 ◇•	26.2 5
YAV-add- del	2.39 ◇•	75.29 ◇•⋆	35.9 6.4	2.38 ◇•	75.29 ◇•⋆	35.9 6.4	1.66 ◇•	72.64 ◇•	27.6 5
YAV-del- add	**2.06** ◇•	**79.20** ◇•	33.9 6.6	**2.06** ◇•	**79.20** ◇•	33.9 6.6	**0.57** ◇•⋆	61.34 ◇•⋆	14.7 3.3
YAV- FORTE	2.56 ◇•	78.33 ◇•	38.9 7.2	2.61 ◇•	78.33 ◇•	38.9 7.2	1.72 ◇•	71.34 ◇•	24 4.7

Results and remarks about them Tables 2, 3, 4, 5 and 6 bring the results for Amine, Toxic, Choline, Scopolamine and DssTox datasets, respectively. The best values for each column are in bold, but we require that, in case the best runtime or theory size result is of one of the revision settings, the accuracy of the initial theory is still improved. The symbol $\diamondsuit$ identifies the cases where the runtime and accuracy of the new systems have significant difference compared to Aleph. The symbol $\bullet$ is used for a similar reason, comparing YAVFORTE with its difference settings to FORTE_MBC. Finally, $\star$ identifies the cases where there is significant difference between YAVFORTE disregarding some revision operator and YAVFORTE using all the operators. From the results we make remarks as follows.

- YAVFORTE is always faster than FORTE_MBC and even produces more accurate theories with significant difference in four cases. There are two cases where YAVFORTE returns worse theories than FORTE_MBC: Choline and DSSTox with the literature Aleph parameters. In the rest of the cases both systems produces similar accuracy results. The difference in time is mainly due to the fact that YAVFFORTE may stop to propose revisions without using all operators, when a simpler operator already reaches the potential of the revision point. Additionally, there is some gain in time when terms of the base clause are included in the set of terms of the bottom clause before generating literals. Remember that FORTE_MBC is used to unify the terms of the base clause and the bottom clause only after the bottom clause has been constructed. Concerning the difference in accuracy, there are two factors responsible for that: (1) to stop proposing revisions before trying all operators may cause the revision let aside some revision that could be better to the test set. This is the case with Choline dataset; also, delete antecedents operator in YAVFORTE does not try to eliminate the proof of all negative examples; instead it deletes antecedents following a score value. While this more flexible operator is able of producing more accurate results in the four cases previously mentioned, it is also responsible for worse revision of the DSSTox theory. However, in most cases, YAVFORTE is able to revise theories as well as FORTE_MBC (without significant difference) in reduced runtime.

149

- Note that the results for Choline dataset using FORTE_MBC are in most of the figures detached from the rest. FORTE_MBC has a harder time to revise the theories generated from this dataset than in the others datasets for two main reasons: this is the dataset with the largest set of examples and the theories generated from Aleph for it have the highest size.

- YAVFORTE considering only delete antecedents and delete rules as revision operators achieves the fastest revision system and produces the smallest theories according to both number of literals and number of clauses. What is interesting about this setting is that with rare exceptions (Choline with best parameters of Aleph and DssTox), the results show that those both operators alone are able to provide significant improvements over the initial theory. Actually, in Toxic with literature Aleph parameters this setting is the one providing the best revision: the final theory is the smallest and more accurate and it even has been revised in less time.

- The results indicate that there are benefits of considering only a subset of the revision operators: most cases produce theories as accurate as when considering all operators and in less time. Thus, when the expert of the application has some insight on what to expect from the revised theory, he could use this knowledge to reduce the set of applicable revision operators.

- It is important to emphasize that YAVFORTE considering any set of operators is faster than Aleph in 11 of 15 cases, with most of the settings still returning more accurate theories. With this result, we can claim that a revision system is capable to behave better than an inductive system, considering both runtime and accuracy.

Figures 4, 5 and 6 exhibit scatter plots for all systems and theories, considering theories learned with Aleph default parameters, Aleph default plus clause length changed and Aleph best parameters, respectively. Symbols in the bottom right corner indicate the best results. Note that when considering default parameters Aleph concentration is in bottom left corner: runtime is small, at the cost of bad accurate theories. When changing clause length parameter, the situation slightly
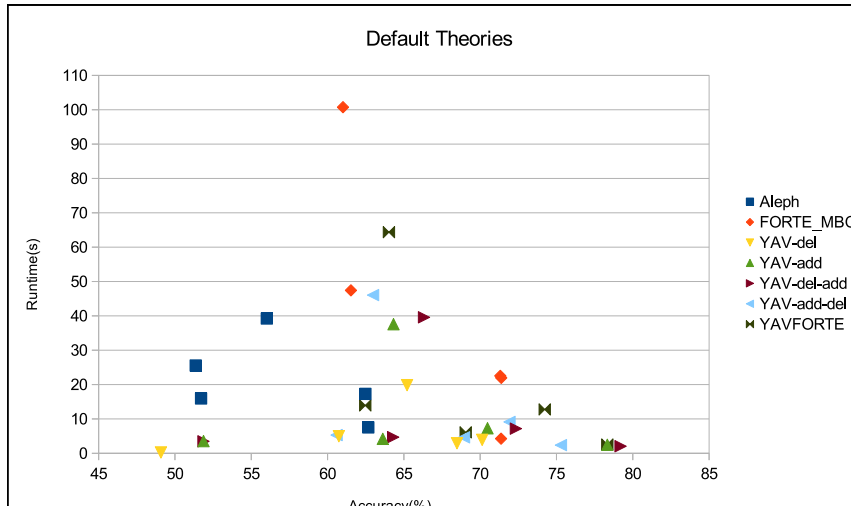
150

Figura 4: Scatter plot for all datasets and systems settings considering theories learned by Aleph with its default parameters
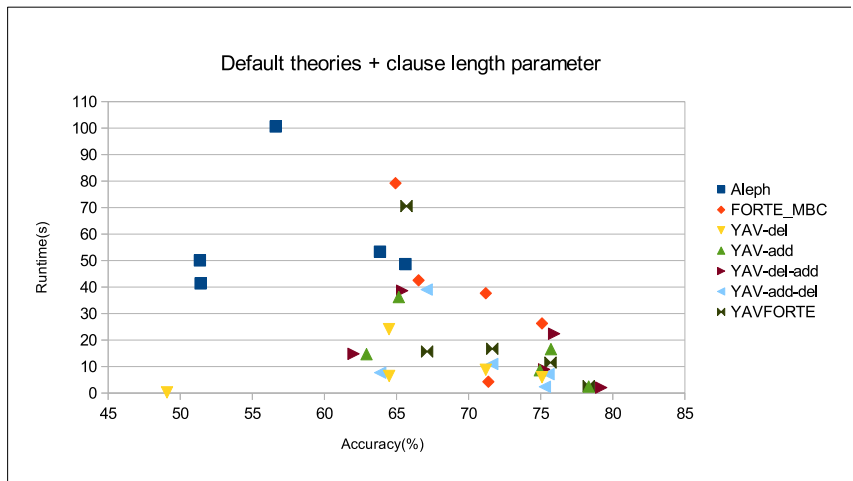

Figura 5: Scatter plot for all datasets and systems settings considering theories learned by Aleph with its default parameters, except for clause length

changes for accuracy, but runtime increases, as expected. Better parameters improve accuracy, at the cost of increasing runtime. Most of the cases FORTE_MBC is in higher y-axis than the others systems, although it is more present in the left side, indicating its accuracies results are still elevated, compared to Aleph. YAVFORTE settings, on the other hand concentrates in the bottom (right) corner, indicating in the overall they are capable of producing the best accurate theories, in less time.
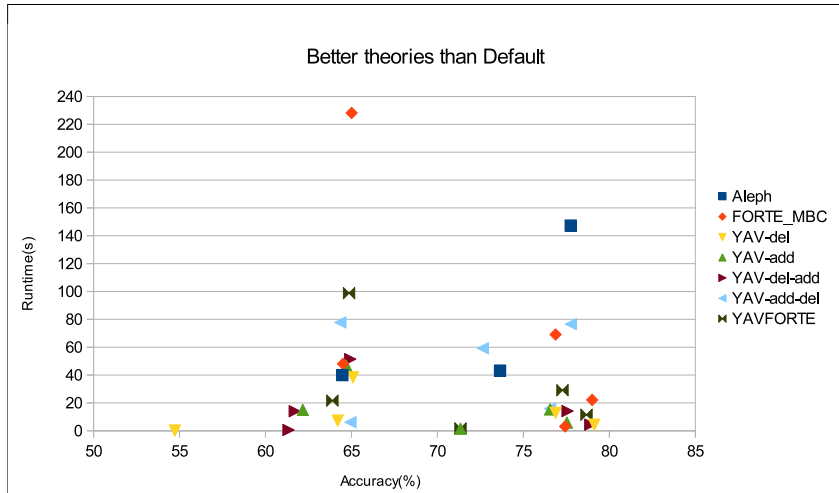
Figura 6: Scatter plot for all datasets and systems settings considering theories learned by Aleph with better parameters settings than just default

Conclusions

Although work in theory revision gained great attention in the 1990s, in recent years ILP community had practically set aside research in this area, since the benefits of revising theories could not outweigh the large runtime effort expended by those systems. This chapter contributes towards changing this scenario, based on the FORTE revision system. First, in (DUBOC, et al., 2009) we have abandoned the top-down search of literals based on FOIL in order to use the Bottom clause, to reduce the set of literals taken into account when refining a theory, as standard ILP systems do. In this chapter we further improve the scalability of the revision system, by (1) making the use of revision operators more flexible, once it is possible to choose which operators are going to be considered to revise the theory; (2) stopping to propose revisions in one revision point as soon as a simpler operator already achieves the full potential of the point; (3) requiring the clause continues to obey mode declaration after an antecedent is deleted. Moreover, we also introduced one modification in the delete antecedents operator, once in the original FORTE and FORTE_MBC an antecedent could be deleted only when none of the true negative instances become provable. It was necessary to make this hard requirement more

flexible because there are cases when the set of false negative instances is only reduced if the clause is allowed to also prove some negative instances. Finally, we have modified the bottom clause construction procedure of FORTE_MBC in two ways: (1) the terms of the ground base clause are considered to be part of the bottom clause, so that they can be used to bring further terms to the bottom clause, and also to make unnecessary to unify the base clause with the bottom clause after this last one is constructed; (2) the current theory is taken into account to prove literals to be included in the Bottom clause. We named the system including all this issues as YAVFORTE.

Experimental results were extracted from five relational benchmark datasets, namely four datasets from Alzheimer domain and the DSSTox dataset. Through them, it was possible to verify that the revision process can be indeed improved with those modifications. YAVFORTE is faster than FORTE_MBC without decreasing the accuracy. In fact, there were cases where the accuracy was further improved, considering the initial theory. It was also possible to see that when we turn the set of revision operators more flexible, the runtime diminishes while the initial accuracy is still improved, though not that much as when the complete set of operators are considered. More important, we could show that when the same parameter for limiting the size of a clause is used for revising and for learning from scratch, the revision performs faster than the inductive method. In this way, we achieve our goal of devising a revision system as feasible as a standard inductive system, at least when datasets of regular size are used.

Datasets used in this chapter are not considered as robust ones. Therefore, we still need to verify how the revision system behaves when the datasets have a large number of examples and/or large background knowledge. Also, as the revision system starts from an initial theory, if this one has a large number of faulty clauses, the system is probably going to behave badly. In this case, it is going to be more difficult to do the revision than learning from scratch, since the revision must propose modifications to each faulty clause. Because of those issues, in chapter 6 we investigate stochastic local search techniques applied to the revision process.

Appendix C: Chess Revision: Acquiring the Rules of Variants of Chess through First-order Theory Revision from Examples

Motivation

In the recent paper (DIETTERICH, et al., 2008), the authors point out that there was considerable effort in the development of theory revision systems in the past, but the lack of applications suited to that task made those systemsnot be widely deployed. In this chapter we intend to contribute in this direction by designing an application in the area of games that fits perfectly to theory revision.

Game playing is a fundamental human activity, and has been a major topic of interest in AI communities since the very beginning of the area. Games quite often follow well defined rituals or rules on well defined domains, hence simplifying the task of representing them as computer programs. On the other hand, good performance in games often requires a significant amount of reasoning, making this area one of the best ways of testing human-like intelligence. Namely, datasets based on games are common testbeds for machine learning systems (FÜRNKRANZ, 2007). Usually, machine learning systems may be required to perform two different kinds of tasks (FÜRNKRANZ, 1996). A first task is to learn a model that can be used to decide whether a move in a game is legal, or not. Having such a model is fundamental for the second task, where one wants to learn a winning strategy (BAIN, MUGGLETON, 1994; SADIKOV, BRATKO, 2006). We focus on the the first task

in this work. In order to acquire a meaningful representation of the set of rules describing a game, one can take advantage of the expressiveness of first-order logic and hence its ability to represent individuals, their properties and the relationship between them. Thus, using ILP methods it is possible - roughly speaking - to induce the game's rules written as a logic program, from a set of positive and negative examples and background knowledge.

Previous work has demonstrated the feasibility of using ILP to acquire a rule-based description of the rules of chess (GOODACRE, 1996). However, to effectively learn chess theory, it is necessary to induce not only the top-legal concept of how a piece should legally move, but also subconcepts to help on that task. Consider, for example, Figure 7. In the first case, a king is in check and therefore its only legal moves are those which get it out of check. In the second case, a piece is playing the role of protector to the king (it is a pin). Before moving such a piece one should realize if the king continues to be protected, if the piece leaves its position. Those are intermediate concepts that must also be induced by the revision system. The authors of (GOODACRE, 1996) employed hierarchical induction to learn a concept at each time, starting from the lowest level concept and incrementally adding the learned definitons to the final theory, until it reaches the ultimate goal: learning the concept of legal moves.
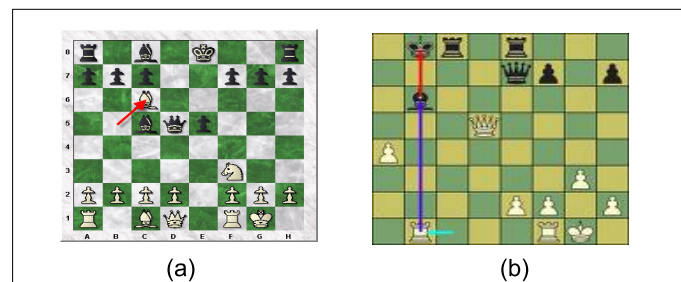

Figura 7: Visualization of situations when an ILP system should learn definitions for subconcepts. Figure (a) shows a board of chess with a checked king. Figure (b) shows a piece working as a pin.

On the other hand, game playing is a dynamic environment where games are always being updated, say, to be more challenging to the player, or to produce an

easier and faster variant of the original game. In fact, popular games often have different regional versions, that may be considered *variants* or even a *new version* of the game. Consider, for example, the game of Chess, arguably, the most widely played board game in the world. It also has been a major game testbed for research on artificial intelligence and it has offered several challenges to the area. There are numerous chess variants, where we define *chess variant* as any game that is derived from, related to or inspired by chess, such that the capture of the enemy king is the primary objective (PRITCHARD, 2007). For instance, the *Shogi* game (HOOPER, WHYLD, 1992), is the most popular Japanese version of Chess. Although both games have similar rules and goal, they also have essential differences. For example, in Shogi a captured piece may change sides and return to the board [3], which is not allowed in Western Chess. Figure 8 shows boards of several chess variants. Ideally, if the rules of a variant of a game have been obtained, we would like to take advantage of them as a starting point to obtain the rules of a variant. However, such rules need to be modified in order to represent the particular aspects of the variant. In a game such as chess this is a complex task that may require addressing different board sizes, introducing or deleting new promotion and capture rules, and may require redefining the role of specific pieces in the game.

In this work, we address this problem as an instance of *Theory Revision from Examples* (WROBEL, 1996). In this case, theory revision is closely related to *Transfer Learning* (THRUN, 1995; CARUANA, 1997), since the rules of international chess (the initial theory for the theory revision system) have been learned previously using ILP. Arguably, transfer learning is concerned about retaining and applying the knowledge learned in one or more tasks, to efficiently develop an effective hypothesis for a completely new task while theory revision deals with very related problems. For example, transfer learning may carry out a mapping between two different predicates but theory revision systems are not designed to perform such a task. However, after mapping one predicate to another it is usually necessary to change the definition of the predicate, which is in fact a task of theory revision. Thus, theory revision may be seen as an important part of transfer learning systems.

---

[3]It is suggested that this innovative drop rule was inspired by the practice of 16th century mercenaries who switched loyalties when captured (PRITCHARD, 2007).
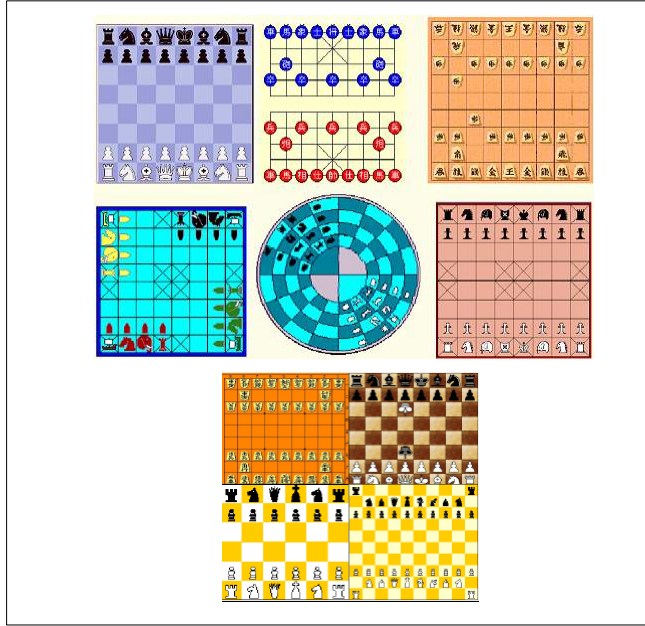
Figura 8: Boards of variants of chess. In the first row it is the international Chess, on its side it is the Xiangqi (Chinesse chess), followed by Shogi (Japanese Chess). Next row it one of the first chess games, the Chaturanga in the Hindu version, followed by the Chess in the Round game and a more modern version of Chaturanga, quite close to the International Chess. In the last group it is Shogi, Anti King Chess, Los Alamos and Grand chess, in this order.

We show that we can learn rules between different variants of the game of chess. Starting from YAVFORTE revision system explained in a previous chapter, we contribute with **(i)** a new strategy designed to *simplify* the initial theory by removing facts that will *not* be transferred between variants; **(ii)** support for *abduction*; and **(iii)** support for *negation as failure*. Experimental evaluation on real variants of chess shows that our technique can transfer between variants with smaller and larger boards, acquire unusual rules, and acquire different pieces. This chapter is organized as follows. First, we describe the components besides the revision system composing the framework for revising the rules of chess in section 5.3.2. Next, the modifications performed on YAVFORTE to support abduction and negation so that the problem is best addressed are described in section 5.3.2. Finally, we show the effectiveness of our approach through experimental results on chess revision in section 5.3.2 and conclude the work in section 5.3.2. A reduced version of this chap-

ter has been published in (MUGGLETON, et al., 2009b) and (MUGGLETON, et al., 2009a).

Revision Framework for Revising Rules of Chess to Turn Them in the Rules of Variants

The framework designed in this work for revising rules of chess so that laws of a variant of this game can be found is composed of five components, three of them are input components, one is the transformer component and the last one is the resulting component. The input components are (1) the initial theory, containing rules of international chess learned previously from examples and/or defined by an expert of the domain, (2) a set of examples reflecting the laws of the game's variant and responsible for pointing out where the initial theory differs from the domain of the variant and (3) a set of fundamental concepts supporting the provability of the initial theory. The transformer component is the theory revision system, which is responsible for modifying the initial theory according to the examples so that it reflects the rules of the variant of the game. Finally, the last component is the resulting revised theory, which ideally is a logic program capable of deciding whether a move in the game is legal or not, according to its governing rules. Figure 9 displays the components of the framework and how they cooperatively work to achieve the goal.

The Format of Chess Examples

As most ILP frameworks, theory revision refines an initial theory from a set of positive and negative examples. The Chess domain addressed in this work has as positive instances the legal moves allowed by the rules of the game. Consequently, the negative instances are the illegal moves, i.e., the moves not obeying the rules of the game. In order to represent the moves executed during a game, the dataset is composed of a set of simulated games up to a specified number of rounds. The moves are within a game aiming to represent castling and en-passant, which require the history of the games, and promotion, which require update of the board. In case of castling, history is necessary to check whether the king and rook has already moved
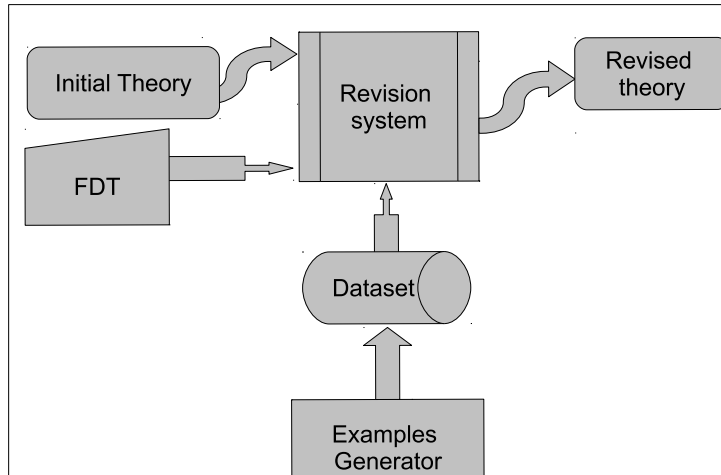
Figura 9: Components of the chess revision framework: Initial theory is going to be revised by the Revision system, using FDT and Dataset. Dataset is created from the Examples generator component.

in previous rounds. To allow an en-passant it is necessary to check the immediately preceding move. Promotion requires to replace the promoted pawn with another piece in subsequent moves.

We take advantage of FORTE examples representation discussed in chapter 1, where an example obeys the format

$Ground\ Instances\ of\ Target\ Predicate \leftarrow Conjunction\ of\ facts\ from\ the\ context$

Thus, the ground instances of the target predicate are instances of legal (positive instances) and illegal (negative instances) moves and the facts from the context are the positions of the pieces related to each round (the board of the game). Each simulated game has its separate set of legal and illegal moves and set of positions of the piece during the game, in the format exhibited in Table 7.

The terms of the target predicate are the round of the move, with 1 as the first round, 2 the subsequent round, and so on, and the current and next *status* of the piece. The status is the name of the piece, its color and position, composed by File and Rank. For example, $move(9, pawn, white, c, 7, rook, white, c, 8)$ states that in round 9 a *white pawn* moved from $c, 7$ to $c, 8$ and is promoted to a *rook*, i.e, its next status is $rook, white, c, 8$. Similarly, the $move(5, bishop, black, c, 8, bishop, black, e, 5)$ states that in round 5 the *black bishop* moved from $c, 8$ to $e, 5$. The facts from the

159

Tabela 7: Format of one example in Chess dataset

Target Predicate:
Positives:
move(Round,Piece,Colour,File,Rank,NextPiece,
NextColour,NextFile,NextRank),...
Negatives:
move(Round,Piece,Colour,File,Rank,NextPiece,
NextColour,NextFile,NextRank),...
Context:
board(Round,Piece,Colour,File,Rank),...
out_board(Round,Piece,Colour,-1,-1),...
out_board(Round,Piece,Colour,0,0),...

context represent the position of the pieces on the board at each round of the game, through the predicate $board/5$ and the pieces removed of the game by capturing or promotion moves, through the predicate $out\_board/5$. In out_board/4 predicate, the two last terms are $-1, -1$ in case of a capture and $0, 0$ in case of a promotion.

The board setting is updated according to the legal move(s) performed on the previous round. Suppose, for example, the white bishop move above. There is a fact $board(5, bishop, black, c, 8)$ in that example and after the move, another fact is generated to represent the new position of the piece, namely $board(6, bishop, black, e, 5)$. Suppose there were a fact $board(5, pawn, white, e, 5)$ in the context of this same game. Thus, when the bishop has moved it captured a white pawn, which "produces" another fact $out\_board(6, pawn, white, -1, -1)$ in the new setting of the board. The board facts for the first round are composed of the initial position of the pieces for the game and therefore they are the same for each example in the dataset. Figure 10 shows the initial board for the international chess and $board/5$ predicates representing it.

A move generator procedure is responsible for creating the dataset of simulated games, since we could not find any saved games for variants of chess. Basically, it starts from the initial board setting and successively chooses a piece from the current board, generating legal moves for it. After that it chooses at random one of the legal moves to be the positive example of the round. Then, it generates il illegal moves from pieces chosen at random from the current board to compose the
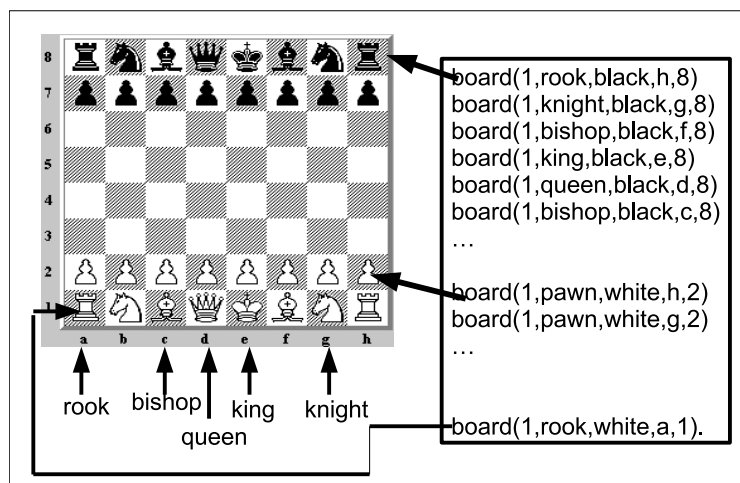
Figura 10: The initial setting of the board in international chess and atomic facts corresponding to it. Figure of the board is taken from $http : //www.mark - weeks.com/aboutcom/bloa0000.htm$

set of negative examples of the round. Note that only one legal move is chosen because naturally each round of the game has one legal move. However, several possible illegal moves are selected, since they are not in fact performed. Moreover, the board must be updated based on the legal move, which does not happen from illegal moves, allowing us to have as many negative examples for each round as desired. The process continues up to the specified maximum number of rounds, respecting for each round the piece with the right to play. It may be also necessary to represent negative examples created from a non-existing board. This is the case of negative examples representing non-existing pieces or non-existing positions in the game. A variant of chess which does not use all the usual pieces or it uses a smaller board would require that. In this situation, a random board is created, specifically to represent such a negative example. Algorithm 21 shows the steps necessary to generate the set of examples. Table 8 exhibits part of one example in the dataset, extracted from a chess variant known as Gardner mini-chess (5X5 board).

161

example(
  % *positive examples*
    [move(p1,pawn,white,a,2,pawn,white,a,3),
    move(p2,pawn,black,c,4,pawn,black,c,3),
   % *Note this is precisely the black pawn that moved from c,4 to c,3*
    %*in round 2*
    move(p4,pawn,black,c,3,pawn,black,b,2),...,
    move(p23,knight,white,c,3,knight,white,a,4),
   % *A pawn is promoted to a knight*
    move(p24,pawn,black,2,black,knight,c,1),
    move(p25,rook,white,d,1,rook,white,d,2)],
  % *negative examples*
   % *promotion in an existing rank (therefore, from a non-existing board)*
    [move(r94,pawn,bishop,white,b,7,bishop,white,b,8),
    move(p1,pawn,white,c,2,pawn,white,b,2),...,
    move(w84,queen,black,f,2,queen,black,b,2),...,
    move(p10,pawn,black,d,4,pawn,black,d,1),....,
    move(p25,king,white,e,1,king,white,f,1)],
% *facts from the context: board setting for each round*
    board(p26,king,white,a,1),board(p26,knight,black,c,1),...,
    board(p25,knight,black,d,5),board(p25,rook,black,e,5),...,
   % *A white pawn has been promoted to a knight*
   % *The fact such a pawn does not more belong to the board is*
   % *represented by out_board predicate*
    out_board(p24,pawn,white,0,0),
    board(p24,knight,white,a,4),...,
    board(p23,pawn,white,a,3),board(p23,knight,white,c,3),...,
    board(p23,rook,black,e,5),...,
   % *The black queen has been captured*
    out_board(p25,queen,black,-1,-1),...,
    board(p5,rook,white,e,1),board(p5,queen,white,a,2),
   % *Board updated after fourth round: a black pawn has moved to b,2*
    board(p5,pawn,black,b,2),...,board(p5,pawn,black,e,4),...,
    board(p5,knight,black,d,5),board(p4,king,white,a,1),...,
   % *Board updated after third round: the white queen has moved to a,2*
    board(p4,queen,white,a,2),board(p4,pawn,white,b,2),...,
    board(p3,king,white,a,1),board(p3,queen,white,b,1),...,
   %*Board updated after second round: a black pawn has moved to c,3*
    board(p3,pawn,black,c,3),...,board(p2,pawn,white,e,2),...,
   %*Board updated after first round: a white pawn has moved to a,3*
    board(p2,pawn,white,a,3),board(p2,pawn,black,a,4),...,
    board(p1,bishop,white,c,1),board(p1,rook,white,e,1),
    board(p1,pawn,white,a,2),board(p1,pawn,white,b,2),...,
    board(p1,knight,black,d,5),board(p1,rook,black,e,5),
   % *Non-existing boards are uptaded after a negative move*
    board(r95,bishop,white,b,8),...,board(w9,rook,black,b,1)

Algorithm 21 Chess examples generator Algorithm

Input: The initial board setting CB, int d, representing the depth of each example/game, int il, number of negative examples for each round, number of examples n

Output: A dataset for a chess variant
1:   $nex \leftarrow 1$
2:   **while** $nex <= n$ **do**
3:      $round \leftarrow 1$
4:      $current\_board \leftarrow CB$
5:      $color \leftarrow$ white
6:      **while** $r <= d$ **do**
7:         $piece \leftarrow$ choose a piece with color $color$ from $current\_board$
8:         $legal\_moves \leftarrow$ generate legal moves for $piece \in CB$
9:         $pos\_move\_round \leftarrow$ a move chosen at random from $legal\_moves$
10:        $neg\_moves\_round \leftarrow il$ illegal moves generated at random from CB and/or a non-existing board
11:        $current\_board \leftarrow$ update $current\_board$ according to pos_move_round
12:        $color \leftarrow$ black
13:        $r++$
14:      $n++$

The Background Knowledge

In the chess revision problem, the initial theory describes the rules of the standard game of chess, which will be modified to state the rules of the variants, using appropriate examples. This theory is inspired on the one learned in (GOODACRE, 1996) where hierarchical structured induction was employed in Progol system to learn clauses from the lowest level predicate (which does not have any other target predicate in the body) to the top-level predicate (the one which is not in the body of any other predicate). The resulting theory was approved by Professor Donald Michie, who was a world authority in computer chess research. To accomplish that, a set of examples were generated at each time as the target predicate of the respective level. Thus, different from her, the examples were not represented within a game, but isolated from each other.

The major differences between the theory exploited in the present work and the one learned in (GOODACRE, 1996) concern the clauses describing special moves namely, castling, en-passant and promotion, since the authors of that work opted

by not representing those moves. Additionally, we try as much as possible to avoid negated literals with output variables to increase efficiency, as explained above, and because of that some rules were re-learned to obey this requirement.

Initial Theory

* The initial theory, which must be revised to turn into the theory of the variant game rules, describes the rules for achieving a legal move following the rules of chess (BURG, JUST, 1987). From a query about the legality of the move, the first step of the logic program is to inspect whether the opponent king is in check. If so, it is illegal to move the piece and the clause does not succeed. Otherwise, it proceeds trying to find a valid next position for the piece. In more detail:

1. Either the piece is a king, and:

    (a) It must be in a valid position (existing file and rank), and,

    (b) it must not go *next* to the opponent king, and,

    (c) it must not go into check, that is to a position threatened by an opponent piece and,

    (d) it cannot go to a position where there is already a piece of the same color, and then,

    (e) it can perform *castling* with a rook, if the castling conditions hold; or,

    (f) it moves one square in any direction to an *existing* position in the board.

2. Or the piece is *not* a king, and:

    (a) It must be in a valid position (existing file and rank), and,

    (b) it cannot move if its king is in double check (in this situation the only piece allowed to move is the king), and,

    (c) if it is protecting the king from a check (absolute pin) it can only move to a position where it continues protecting the king, or,

    (d) it can move to stop a check if the king is in simple check, either by capturing the threatening piece or by blocking the path between the threatening piece and the king, as long as it also respects conditions (g) and (h), or,

164

(e) it may perform an *attack*, only in case the intended position is already occupied by an opponent piece, by moving according to the direction it is allowed to, except for the pawn, or,

(f) if the piece is a pawn, and ((d) or (e)), it can perform an en-passant move; otherwise it captures one square diagonal, or,

(g) if the intended position is not occupied, the piece moves without capturing other piece;

(h) except for a knight, it must not pass over any other piece on the board while going to its next position;

(i) finally, it must move according to its basic move (bishops diagonally, rooks orthogonally, etc), to a valid position in the board (existing file and rank).

(j) If the piece is a pawn reaching the last rank it is promoted. The pawn can move 2 squares vertical forward only if this is its first move in the game.

Note that if the rules of the variant of the chess differ from the standard chess in any of the conditions above, the revision process must identify this through the examples and change those rules, so that they reflect the rules of the variant.

The pieces, files, ranks and color are represented in the initial theory as ground facts, where each fact is related to a piece, a file, a rank or a color allowed in the game of chess, as exhibited in Table 9.

Tabela 9: Ground facts representing pieces and colors in the game and file and ranks of the board, considering the interantional Chess

Pieces	Colors	Files	Ranks
piece(rook).		file(a). file(e).	rank(1). rank(5).
piece(knight).		file(b). file(f).	rank(2). rank(6).
piece(bishop). piece(pawn).	color(white).	file(c). file(g).	rank(3). rank(7).
piece(queen). piece(king).	color(black).	file(d). file(h).	rank(4). rank(8).

In order to avoid output variables in negated literals, some clauses were modified to provide the same definition but using only input variables. Suppose, for example, the clause below explaining the concept of *check*, where the terms related to

the piece checking the opponent king are unified with $(Piece, Colour_1, File_1, Rank_1)$.

$check(Round, Piece, Colour_1, File_1, Rank_1, \texttt{king}, Colour_2, File_2, Rank_2) \leftarrow$

$\quad nequal(Colour_1, Colour_2),$

$\quad \%load\ the\ information\ about\ a\ piece\ on\ the\ board$

$\quad board(Round, Piece, Colour_1, File_1, Rank_1),$

$\quad attack(Round, Piece, Colour_1, File_1, Rank_1, \texttt{king}, Colour_2, File_2, Rank_2).$

The clause above holds true if there is an opponent piece attacking the king; in this case $(Piece, Colour_1, File_1, Rank_1)$ states the opponent. If one simply needs to know whether the king is in check or not, the information regarding the attacking piece can be only local to the clause, i.e., it is not necessary to return them as answer substitution. Thus, a simplified predicate for deciding if the king is in check has a clause wherein all variables may be considered input:

$check(Round, King, Colour_2, File_2, Rank_2) \leftarrow$

$\quad check(BoardID, Piece, Colour, File_1, Rank_1, \texttt{king}, Colour_2, File_2, Rank_2).$

In this case, the predicate $check/5$ has as input only the information about the king which might be in check, i.e., $Colour_2, File_2, Rank_2$ refers to the color and position of the possibly checked king.

Now, we can have $\texttt{not}(check(Round, King, Colour_2, File_2, Rank_2))$ in the body of a clause where all the terms are input variables, such as $load$, if we have a subgoal $board(Round, King, Colour_2, File_2, Rank_2)$ first. The same was done with other concepts in the theory appearing as negated literals in the body of some clause. Although this optimization was not required by the revision itself, it was an important useful step to achieve efficient construction of the bottom-clause.

The clauses representing special moves such as castling, en-passant and promotion were written by an expert and inserted by hand into the initial theory in appropriate clauses. Defining some part of the knowledge with the help of an expert in the domain is usual in theory revision area.

Fundamental Domain Theory

Besides the initial theory, there is previous knowledge about the domain which is assumed as correct and therefore does not need to be revised. Fundamental Domain Theory contains definitions valid to the standard chess and also to the chess variants. During the revision process, this set of clauses is not modified and some of the clauses in the theory being revised need to use them so that they can be satisfied. After the revision process, the clauses in the FDT, together with the revised theory, will compose the theory of the chess variant. FDT is mainly responsible for keeping the definitions of fundamental concepts such as differences between positions (files or ranks), relations of equality, non-equality, greater than or less than between files and ranks, directions, among others. Below there is an example of a clause belonging to FDT, which defined the concept of *northdirection*.

%(b,2 is north from b,3)

$direction(File, Rank1, File, Rank2, north) \leftarrow less\_than Rank1, Rank2).$

where $less\_than$ concept is defined as

$less\_than(A, B) \leftarrow integer(A), integer(B), !, A < B.\% for\ ranks$

$less\_than(A, B) \leftarrow name(A), name(B), A < B.\% for\ files$

The initial theory has 109 clauses with 42 intermediate predicates and the FDT 42 clauses. The work in (GOODACRE, 1996) had 61 clauses in the BK and it learned 61 clauses. The difference in the total number of clauses is from the clauses that were added to avoid output variables in negated literals, as explained in the previous section, and from clauses representing special moves.

Modifying YAVFORTE to Acquire Rules of Chess Variants

Unfortunately, the revision algorithm described in chapter 5.3.2 cannot tackle the problem of revising between variants of chess. Analysis showed that chess generates

a very large search space that could not be addressed well with uninformed search. Moreover, we must consider changes in the domain (such as differences in board size), that are not addressed well through the standard revision operators. We therefore propose a number of modifications to the current version of the system, described as follows.

Starting the Revision Process by Deleting Rules

In an attempt to decrease the complexity of the theory and consequently of the whole revision process, we introduce a first step of deletion of rules. This process is performed as a hill-climbing iterative procedure, where at each iteration the clauses used in proofs of negative examples are selected, each one is deleted in turn using the delete rule operator and the resulting theory is scored. The modified theory with best score is selected for the next step. The process finishes when no deletion is able to improve the score.

A similar algorithm was proposed to revise ProbLog programs (DE RAEDT, et al., 2008b), where it has shown to be quite effective at finding minimal explanations. In our case, the goal would be to find a "common denominator" between the two different games. We found this procedure both reduces theory size and noise, namely when the target theory is a specialized version of the initial theory. On the other hand, it could implicate that the final revision is not the "minimally revised" one, since the proof of negative examples could be avoided through addition of few antecedents to the rules. However, the benefits on the decrease of runtime outweighs the possibility of returning a non-minimal revision.

After this step, the algorithm is executed as usual. The procedure for deleting rules is exhibited as Algorithm 22 and it is executed before the *line* 1 of the Algorithm 16. Note that the operator for deleting rules might be normally used again during the rest of the revision process (since it is one of the revision operators).

Using abduction during the revision process

Abduction is concerned with finding explanations for observed facts, viewed as missing premises in an argument, from available knowledge deriving those facts (FLACH,

168

Algorithm 22 First Step for Deleting Rules

1: **repeat**
2:     generate specialization revision points;
3:     **for each** specialization revision point (a clause) **do**
4:         delete clause;
5:         update best revision found;
6:     **if** best deletion improves the current score **then**
7:         delete clause with best score;
8: **until** no deletion improves the theory;

KAKAS, 2000a). Usually, theory revision systems, including FORTE, use abduction when searching for generalization revision points, to locate faults in a theory and suggest repairs to it. Using abduction, a revision system determines a set of assumptions (atomic ground or existentially quantified formulae) that would allow the positive example to be proved. Consider, for example, the theory below, taken from (MOONEY, 2000)

$$p(X) : -r(X), q(X).$$
$$q(X) : -s(X), t(X).$$

and the positive instance $p(a)$, with BK $r(a), s(a), v(a)$, unprovable by the current theory. The algorithm discovering revision points would find out by abduction that the instance is unprovable because the literal $t(a)$ fails. One of its suggestions to fix the theory would be to add a new definition for $t(X)$

We further benefit from abduction in three distinct situations of the revision process.

Intermediate Predicates Abduction Intermediate predicates are those ones appearing in the head of clauses and also in the body of others clauses, but there is neither example nor facts in the dataset corresponding to them. Suppose, for example the extracted piece of the chess theory in Table 10. The clause we show in the table handles the case of a king moving by calling the clause defining how a king should legally move. Note that there are two clauses for doing this, one of them dealing with a king in check and another one to handle a move of a king not in check.

169

Predicates *kingmove*/7, *kingmove_in_check*/7 and *kingmove_no_check*/7 are intermediate predicates, as they appear in the body of clauses and also in the head of others clauses. The instances in the dataset are of predicate *move*/4 only.

Tabela 10: An extracted piece of the chess theory, to exemplify the need of abduction when learning intermediate concepts

```
move(BoardID, king, Color, File1, Rank1, king, Color, File2, Rank2):-
       file(File1), rank(Rank1),
       file(File2), rank(Rank2),
       color(Color1), nequal(Co lour,Color1),
       equal(Piecek, king), % no constants in negated predicates
       % there is only one king of Color1 in the board
       board(BoardID, Piecek,Color1, File3, Rank3),
       % the king is not in check
       not(any_check(BoardID, Piecek, Color1, File3, Rank3)),
       % clause defining the legal move of a king
       kingmove(BoardID, king, Co lour, File1, Rank1, File2, Rank2).

kingmove(BoardID, king, Co lour, File1, Rank1, File2, Rank2):-
       kingmove_in_check(BoardID, king, Co lour, File1, Rank1, File2, Rank2).

kingmove(BoardID, king, Co lour, File1, Rank1, File2, Rank2):-
       kingmove_no_check(BoardID, king, Co lour, File1, Rank1, File2, Rank2).
```

It may be the case that the clause(s) defining intermediate predicates are wrongly defined. The error propagates to clauses containing literals with such predicate in their bodies, and so on. In this situation, such a literal is marked as a revision point, since it is in the path of the failing/successful proof of some positive/negative example. Suppose, for example there is no definition or a wrong definition for a king moving when it is in check, as in the example of Table 10.

In the first case, when the predicate is contributing to a positive example not to be proved, besides trying to modify clauses from where the intermediate predicate is called, the revision is going to propose the following modifications concerning the predicate itself:

- Delete antecedents from the body of the clause defining an intermediate predicate, if such a clause exists;

- Add rules with the intermediate predicate in the head of the clause.

In the second case, an intermediate predicate is in the path of a successful proof of a negative example. Possible proposed modifications trying to solve this problem are going to be:

- Delete the rule with the intermediate predicate in the head.

- Add antecedents to the body of such a clause.

The second modification of each item concerns refining the clause by adding literals to its body. As we discussed in the previous chapter, this search space is composed of literals in the Bottom Clause, generated from a positive example covered by the clause, whose example predicate is the same as the one in the head of the clause. The problem is we have no examples for such intermediate predicates. In the chess theory, for example, the instances correspond to move predicates, but there is no example in the dataset for $kingmove$, $kingmove\_in\_check$, or $kingmove\_no\_check$.

Therefore, we need to tackle non-observation predicate learning (MUGGLETON, BRYANT, 2000), where the concept being learned differs from that observed in the examples. We introduce *intermediate predicate abduction* in order to "fabricate" the required example. From a positive instance belonging to the answer set (relevant examples) of the intermediate clause, i.e, the proof of the instance includes the clause, we obtain an "intermediate instance" using the current theory and FDT. The procedure takes an example from the answer set together with the intermediate predicate and instantiates such a predicate to its first call encountered when attempting to prove the goal. The proof starts from the example and finds an instantiation for the specified intermediate predicate. After constructing the intermediate example, the bottom clause construction procedure is ready to run, followed by the refinement of the clause. The whole procedure can be visualized in Algorithm 23.

In Table 10, assume the predicate $kingmove\_in\_check/7$ is marked as a revision point and $move(p12, king, black, e, 9, king, black, b, 2)$ is one of the relevant examples. The intermediate instance $kingmove\_in\_check(p12, king, black, e, 9, b, 2)$ would be generated by the procedure, by first calling clause corresponding to $move/7$

predicate, next calling clause with *kingmove*/7 in its head and finally instantiating the intermediate instance.

FORTE also uses a similar procedure to generate extensional definitions for lower level recursive predicates, when revising theories with recursive clauses. In (MUGGLETON, BRYANT, 2000) the procedure we employ here to abduct the intermediate instance is used to obtain "contra-positive examples" to the intermediate predicates and then construct a Bottom Clause for them.

Note that we can only reach an intermediate predicate from higher level clauses. In case there already exists a correct clause defining such a predicate, there is no need to further refine it and such a literal is a candidate to be included in the body of higher level clauses (clauses where the intermediate predicate is in the second term of a determination definition). However, if there is no clause defining this predicate, and also it does not belong to the body of a higher level clause, currently the revision process has no means of reaching this predicate and consequently propose the creation of a clause for it. This happens because, in order to make some modification in the provability of the example, the revision would have to be able to include the intermediate predicate in the body of some clause taking part in the proof of the example, and at the same time to create a clause for the intermediate predicate. Nowadays, the revision system is not capable of proposing modifications to more than one revision point at the same time. On the other hand, if there is a definition of the intermediate predicate, even if it is not correct for all required examples, but it solves the problems of some of them, the revision would be able to include such a predicate in the body of a clause. In a future iteration, the predicate should be marked as revision point and have its definition modified, so that the missing examples before become correctly proved. To sum up, although we do not escape from the limitations of inverse entailment (YAMAMOTO, 1997) with this procedure, it is enough to attend this class of requirement when revising a chess theory.

To finish, it is worth mention there is great interest on learning concepts from non-observed predicates in ILP community, by handling two strategies of logical reasoning: induction and abduction (FLACH, KAKAS, 2000b; INOUE, 2001; MOYLE,

2003; RAY, et al., 2004).

Algorithm 23 Algorithm for Refining a Clause whose Head Corresponds to an Intermediate Predicate

---

Input: A theory T and FDT BK, a positive instance pi, a clause $C = l : -Body_0$, where l corresponds to an intermediate predicate, and $Body_0$ is a (possible empty) set of literals

Output: A possibly refined clause $C' = l : -Body_0 + Body_1$

1: $ipi \leftarrow$ Algorithm 24
2: $BC \leftarrow$ Bottom clause created from ipi, T and BK
3: $C' \leftarrow C$ refined with literals in BC
4: **return** C'

Algorithm 24 Algorithm for fabricating an intermediate instance

---

Input: A theory T and FDT BK, a positive instance pi, a clause $C = l : -Body_0$, where l corresponds to an intermediate predicate, and $Body_0$ is a (possible empty) set of literals

Output: An intermediate instance ipi

1: **repeat**
2:     choose a clause to start the proof of pi
3:     **while** $l \notin$ the derivation path or the proof does not finish, with a failure or a success **do**
4:         continue to build the proof tree
5:     **if** l is the atom in the leaf of the proof path **then**
6:         try to prove $l : -Body_0$, accordingly unified
7:         **if** it is possible to prove $l : -Body_0$ **then**
8:             $ipi \leftarrow$ ground atom coming from the proof of l
9: **until** ipi is found

Abducting Atomic Facts to be Part of the Theory The simpler way of employing abduction in theory revision is to include in the theory assumptions that would allow positive examples become provable. Thus, the need for introducing an abducible predicate is identified during the search for generalization revision points. Suppose, for example, the extracted piece of the chess theory in Table 10. Assume the goal is to modify the theory so that it is possible to represent a game of chess played in a larger board, say a 9X9 board. The positive instance $move(p12, [king], black, e, 8, e, 9)$ will fail, since it would not be possible to prove $rank(9)$.

We let any failing predicate to be a candidate to be abducted, once it obeys three additional requirements: (1) the predicate must have a `modeh` definition in

173

modes declaration, so that it can become a clause whose body is empty, (2) there must have at least one modeh definition with only constants as its terms, and (3) the predicate must be defined as abducible. To abduct $rank(9)$, as necessarily demonstrated in the example above, a $modeh(1, rank(\#rank))$ would be required to be in modes definitions.

There is a maximum number of abducible predicates allowed in the theory, in order to prevent it to have several predicates maybe proving only one positive example. Note that this abduction is part of the add new rule operator, which acts on predicates, by creating an explanation for it. Because of that, an abducted predicate is only proposed as a revision if there is no way of creating a clause with such a predicate in its head. Thus, what really happens is: if the add new rule operator is unable of creating a clause defining the predicate, it is checked if it is abducible and if the maximum number of abducible predicates has not yet been reached. If the answer is affirmative, the operator looks for the modeh definition mentioned above and from it and an instance indicating the necessity of abduction, the atomic fact is created. As it is a proposed revision, the fact is only abducted if it brings an improvement to the current theory. Note that it is necessary to call Algorithm 24 so that the final atomic fact is found out. The algorithm performing this task is exhibited as Algorithm 25.

Note that if the abducible predicate becomes a faulting point in the theory, it may eventually be generalized/specialized in the next iterations.

Algorithm 25 Atomic Facts Abduction Algorithm

Input: Modes definitions M, abducible predicate $pred/N$, maximum number of abducible max_abd, Theory T, FDT BK, a positive instance pi
Output: An atomic fact $pred(t1, ..., tn)$, proposed to be included in T
 1: **if** number of abducible predicates so far is less than max_abd **then**
 2:       $pred(t1, ..., tn) \leftarrow$ Algorithm 24
 3:       **if** $pred(t1, ..., tn)$ matches a modeh definition, where the terms are constants only **then**
 4:             **return** $pred(t1, ..., tn)$

Look-ahead abduction   The last abduction approach addressed in this thesis is to use a strategy called here as *look-ahead abduction*, acting under the search for revi-

sion points. It works as follows. When searching for generalization revision points, it is assumed that faulty abducible predicates are true and the search continues, looking for further revision points, possibly depending on the abducible predicate. In this way, it is possible to fix a clause with at least two problematic literals, since the first one failing is considered as proved.

Suppose, for example, we have the positive instance $a(1, 2)$, the negative instance $a(1, 3)$, the clause $a(X, Y) : -b(X, Z), c(X, Y, Z)$ and BK $d(1, 2, 4), d(1, 2, 5)$. The positive instance $a(1, 2)$ is unprovable and then generalization revision points must be found. When doing that, it is noticed the literal $b(1, Z)$ cannot be proved. Thus, originally, the search for revision points finishes, marking that clause and the literals $b(X, Z)$ and $a(X, Y)$ as generalization revision points. Notice that the revision points search procedure has no means to verify whether $c(1, 2, Z)$ could or not be proved, since the literal before it has already failed.

Although there are two revision points (the single clause and predicate $b/2$) it may be the case that no revision is able to fix the misclassification of the positive instance, without also making the negative instance provable. Considering the most common delete antecedent and add rule operators, note what the revision operators could propose:

1. Deleting either antecedent $b(X, Z)$ or $c(X, Y, Z)$: this does not make the positive instance become provable.

2. Creating the most general rule $a(X, Y)$ would make the negative instance provable. Only the most general rule could be created to this case.

3. Creating a rule capable of proving $b(1, \_)$ does not solve the problem of the non-proof for $c(1, 2, \_)$ and therefore does not bring any benefits to the theory.

However, if we assume $b/2$ as an abducible predicate during the search for revision points, $c(X, Y, Z)$ is also marked as a generalization revision point and a rule could be created to explain $c(1, 2, \_)$, say $c(X, Y, Z) : -d(X, Y, Z)$. Later, either $b(X, Y)$ would have to be removed from the clause or a definition would be created for it (possibly including the single assumption $b(1, 2)$).

For a more concrete example regarding the chess revision problem, consider the case of a game that has a different piece, say a *counselor*, that moves exactly like the bishop, but it cannot move backwards, only forward. Consider a simplified version of the move concept, in Table 11. The revision system cannot define the basic move for such a new piece, as it is required that the piece itself is known by the theory. However, if one abducts the literal *piece(counselor)* during the search for revision points, the system would be able to mark *basic_move* as revision point, and then propose a definition for it. The abducted literal *piece(counselor)* is going to be part of the proposed revision, besides the possibly created definition for basic_-move(BoardID, counselor, ...).

Tabela 11: An extracted piece of the chess theory, to exemplify the need of abducting predicates when searching for revision points

```
move(BoardID, Piece, Co lour, File1, Rank1, Piece, Co lour, File2, Rank2):-
      file(File1), rank(Rank1),
      file(File2), rank(Rank2),
      color(Co lour), piece(Piece),
      % the piece is on the board
      board(BoardID, Piece, Co lour, File1, Rank1),
      % any others necessary verifications
      ...
      basic_move(BoardID, Piece, Co lour, File1, Rank1, File2, Rank2).


      % how a bishop moves
basic_move(BoardID, bishop, Co lour, File1, Rank1, File2, Rank2) :-
      % abs_fdiff(File1, File2, Diff), abs_rdiff(Rank1, Rank2, Diff).
```

Thus, to outline the problem of more than one faulty literal in a clause, we take advantage of abduction and consider abducible predicates (up to a maximum number of predicates) in the theory under revision. To do so, if a failing abducible literal is found during the search for generalization revision points (points failing on proving positive examples), it is assumed as correct and kept around in the revision point structure as a pending abducible predicate. Note that in this moment the literal must be an atomic fact. The search for revision points proceeds, by

possibly including other abducible predicates as pending facts, until the proof tree reaches either a leaf or a failing literal not considered as abducible. In the first situation, the proposed revision is to include the abducible literals in the theory, as they were new rules, similar to what is done in the previous subsection. In the second case, the system marks the failing literal as a generalization revision point with pending abducible literals. When proposing modifications to such a point, the abducible predicates are considered as part of the theory. Eventually, in case the revision is chosen to be implemented, besides the generalization performed in the revision point, the system must also include the abducible predicates, otherwise the generalized revision point would continue failing.

Note that if the abducted predicate prevents some positive example of being proved or helps a negative example to be proved, the next iterations are going to mark them as generalization/specialization revision points. The process of obtaining abductive explanations and possibly generalizing them later is performed in (MOYLE, 2003), but to learn clauses (completing definitions of background predicates) and not to revise them.

Using negated literals in the theory

YAVFORTE and FORTE_MBC are neither able to introduce negated literals in the body of the clause nor to revise negated literals. Negation is essential to elegantly model the chess problem, since we need to represent concepts such as *the king is* **not** *in check* or *a piece is* **not** *landing on another piece*, among others.

In order to add negated literals in body of clauses it is required that the Bottom Clause construction procedure is able to elicit them. Therefore, it is necessary to explicitly specify `modeb` and `determination` declarations for them. In the first case, the `modeb` declaration must be in the format:

$$\texttt{modeb}(RecallNumber, \texttt{not}(literal(Modes)))$$

stating that $\texttt{not}(literal(...))$ can appear in the body of some clause. Determination declaration must be in the format

$$\texttt{determination}(HeadPredicate/Arity, \texttt{not}(BodyPredicate/Arity))$$

stating that not(*BodyPredicate/Arity*) is allowed to be in the body of the clause whose head has *HeadPredicate/Arity*. In contrast to Aleph, we explicitly define the arity of the negated literal in determination definitions.

Inclusion of negation into logic programs is traditionally considered as a hard task since the incorporation of full logic negation tends to super-exponential time complexity of the prover. One of the most successful and widely alternative to full negation is the procedural approach of Negation as Failure (NAF) (CLARK, 1978). One of the major drawbacks of NAF is that it cannot produce answer substitutions (new bindings) to negated query variables, requiring the negated literal is ground in order to be proved. If one wants more than a simple "yes" or "no" as answer, it is better not to apply negation as failure, since it may result in floundering of the goal (MARRIOTT, et al., 1990; DRABENT, 1996). As a result, to use pure NAF inside the bottom clause construction procedure it is necessary to ensure there are no free variables in any not(*Goal*) that might be called. To do so, the mode type of all variables of negated literals must be of input so that it is guaranteed when not(*Goal*) is called, all terms are ground, since they should had been instantiated by previous literals.

The initial theory learned for the game of chess used in this work makes use of negated literals with output variables, since in some cases it is necessary to check out the negation of a concept. For example, it could be necessary to verify whether a king is not in check through the clause defining the check concept itself. Normally, such a clause should provide as output the piece responsible for putting the king in check. To address this problem the work presented in (GOODACRE, 1996) made a special provision: a redefinition of not was introduced to the general proof which, after succeeding on the fail of the literal, it skolemises the variables so that they cannot be used again. We add the further requirement in the bottom clause construction procedure to guarantee the output variables of such literals are singleton (they appear only once in the clause) and therefore they must not be used as input variables of others literals. Although we have re-learned some concepts so that output variables in negated literals become dispensable (subsection 5.3.2), we let the implementations above in the system, to attend a situation where this

178

re-learn process would not be possible. Constant modes are not allowed in both works, since this would involve generating literals with all possible values, leading to a very inefficient process.

In case it is essential to bind variables in a negated query, the alternative most used to overcome the drawbacks of NAF is to employ constructive negation techniques (CHAN, 1988; DRABENT, 1995). It extends the NAF to handle non ground negative subgoals so that it is possible to construct new bindings for query variables. It works as follows: after running the positive version of the negated literal in the same way NAF does, the solution of the possibly non ground goal is collected as a disjunction and then this disjunction is negated to get a formula equivalent to the negative subgoal. The chess problem addressed here has not showed necessary to use constructive negation, since the initial theory does not need to bind variables from negated literals to answer some query. However, it would be nice to implement such an approach to the general case of negation in the revision process. We leave this question to future work.

In addition to inserting negated literals in the body of clauses, it may be the case that a negated literal is the culprit for the misclassification of examples. Remember from the laws of chess that a king cannot move next to the other king, in such a way that this last king can reach the former after a basic move. In the theory we tackle here, this is verified through a negated literal, as exemplified in line 6 of Table 12.

Tabela 12: An extracted piece of the chess theory, to exemplify the need of marking a negated literal as a revision point

```
kingmove1(BoardID, king, Color, File1, Rank1, File2, Rank2):-
        not(any_land_on(BoardID, king, Color, File1, Rank1, Color, File2, Rank2)),
        color(Color), nequal_color(Color,Color2),
        board(BoardID, king, Colour2, Filek, Rankk),
        basic_move(BoardID, king, Color, File1, Rank1, File2, Rank2),
        not(king_next_king(BoardID, king, Color2, Filek, Rankk, king, Color, File2, Rank2)).

king_next_king(BoardID, king, Color1, File1, Rank1, king, Color2, File2, Rank2):-
        basic_move(BoardID, king, Color1, File1, Rank1, File2, Rank2).
```

Now, suppose a variant of chess that is more restricted to obey this rule: a king cannot move to a position that is reachable by the other king after *two* basic moves. In this case, a positive example would be unprovable, since king_next_king/9 would succed, making its negation to fail. To fix this problem, one could modify the second clause in the table, so that it becomes more specific. One possible modification is to add a second basic_move/7 predicate in the body of the rule. Note that the clause is going to be specialized, even though we want to solve the problem of a positive example. Because the revision has been indicated by a negated literal, it is necessary to reverse the proposed modifications: a clause corresponding to the negated literal must be specialized to fix the misclassification of positive examples, since this is going to make the clause not to be satisfied and therefore its negation succeed. Similarly, in situations where a negative example has been proved because the negation of a literal succeeds, it is necessary to generalize such a clause so that it can be satisified and its negation fails.

We thus introduced a procedure for handling a faulty negated literal during the revision process. Roughly speaking, if the negated literal is responsible for a failed proof of positive examples, it is treated as a specialization revision point. On the other hand, if the negated literal conducts to a proof of a negative example, it is treated as a generalization revision point. This is a preliminary attempt at introducing non-monotonic reasoning in YAVFORTE.

Experimental Results

Experimental methodology To show experimental results obtained with the framework discussed in the paper, we generated datasets for 3 different chess variants. The variants are described as follows, selected from (PRITCHARD, 2007).

- Gardner minichess. Minichess comprises a set of chess variants played with regular pieces and standard rules, but on smaller boards. There are games played on boards of size 3X3, 4X4, 4X5, 5X5, 5X6 and 6X6. The goal of this family of games is to make the game simpler and shorter than international chess. Figure 11 shows several initial boards for minichess games. Here we

focus on Gardner's minichess, which is the smallest chess game (5X5) in which all original chess pieces and legal moves are still used, including pawn double move, castling and en-passant.

- Reform chess, also known as Free capture chess differs from the international chess because either side may capture its own men, as well as the opponent's. Only the friendly king cannot be captured.

- Neunerschach chess is played on a board 9X9, with two extra pieces but removing the queen. The first one is called Marshall and moves like a queen. The second extra piece moves like a queen, but only two squares and it is named $Hausfrau$. The pieces are arranged in the board following the order: Rook-Knight-Knight-Marshall-King-Hausfrau-Bishop-Bishop-Rook.
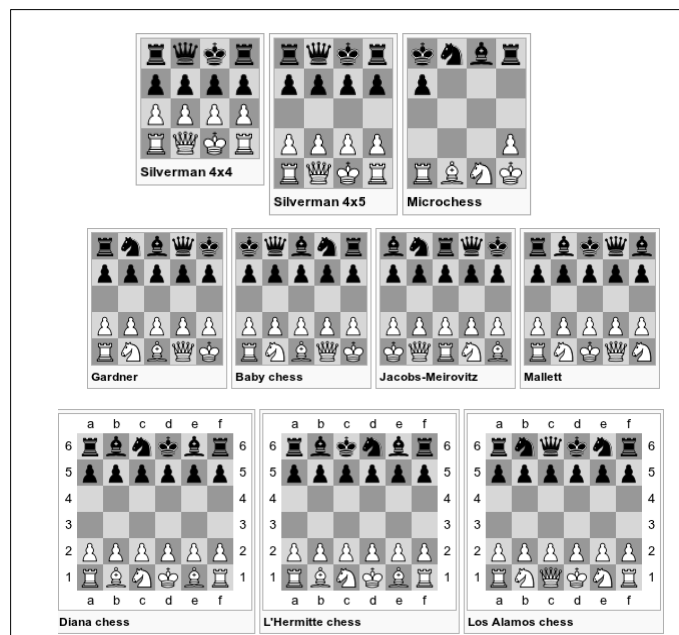

Figura 11: Initial boards of minichess games, taken from Wikipedia

We employed the framework described in this chapter to obtain theories describing each one of the above variants. We performed 5X2-fold cross validation and scored the revisions using f-measure. The datasets are arbitraly composed of 5 simulated game (example), where each round has 1 positive and 5 negative examples

181

and the maximum round for each game is 25.

Next we present the evaluation measures obtained from each variant in Table 13.

Tabela 13: Evaluation of the revision on chess variants

Variant	# Pos. examples	# Neg. examples	# Facts in Context	# Initial clauses	# Final clauses	Initial Acc (%)	Final Acc (%)
Gardner minichess	97	500	3294	151	147	59	99.83
Reform chess	100	500	3360	151	154	93	100
Neunerschach chess	100	500	12651	151	160	87	100

Note that the variant with a smaller board had the size of the theory decreased but the others variants, which include all the rules of chess and also additional ones, had the size of the theory increased. We could say that the first case is a specialized version of the chess while the others are a more general version of the game. The number of clauses include the initial (revised) theory and the FDT, which is fixed. Notice that using only the initial theory can still achieve a good accuracy, as most rules are shared between different variants. On the other hand, accuracies were significantly improved through revision in every case, suggesting that the revision makes the dataset be correctly reflected.

The next subsection discusses specifically the revisions performed by the system in each variant together with remarks about them.

Discussions about the automatic revisions performed by the revision system

Gardner's Mini-chess

The system performed the following revisions on the initial theory to obtain a correct theory for Gardner minichess:

1. The *delete rule step* was able to remove the following clauses from the theory: $file(f)$, $file(g)$, $file(h)$, $rank(6)$, $rank(7)$, $rank(8)$. After that, any negative example coming from an invalid position or going to an invalid position becomes non-proved, since they do not attend one of the conditions 1.(a) , 1.(f), 2.(a) or 2.(h) of section 5.3.2.

2. The *add rule* generalization operator added the following clauses to the theory:

- *basic_move(pawn, black, File, 4, File, 2)*, that allows the black pawn to move 2 squares. This is necessary because in the chess theory the black pawn moves 2 squares only from rank 7 (within the condition 2.(*i*)), which is not a valid rank in the Gardner Chess.

- *promotion_zone(pawn, white, File, 5)*, that allows the white pawn be promoted when reaches the last rank of the Gardner mini-chess board.

Remarks about the revision/learning process in Gardner Mini-chess

- The final accuracy was on average 99.83%, since not all folds contained examples for promotion and black pawn moving 2 squares, as these moves are scarcely executed during the game.

  - Promotion occurs rarely in both chess and in Gardner mini-chess, since usually the pawn is captured before reaching the last rank.

  - The initial double moves of pawns are difficult in Gardner's smaller board, since, and differently from chess, the intended positions (rank 4 for white pawn and rank 2 for black pawn) are already occupied by opponent pieces and the pawn cannot capture a piece through its usual move.

- The best final theory is able to correctly classify all the moves of this variant, although it did not remove the clauses defining pawn double move from rank 7 to rank 5 and black pawn promotion when it reaches the rank 8, as to perform the former moves the pawn would have to be on rank 7. As previously stated, the clause defining this rank was removed from the theory and therefore it became an invalid rank. In this way, the condition 2.(*a*) of section 5.3.2 is not satisfied and the piece cannot move anyway. A post-pruning procedure should remove such useless clauses from the theory, so that the best final theory would perfectly correspond to the target theory of Gardner Mini-chess.

- We have tried to run Aleph and Progol 5 to find the rules of Gardner's minichess. As they do not revise a theory, we let the initial theory and FDT be

the background knowledge. Aleph abduction procedure was not working properly. Progol, on the other hand, was able to abduct predicates basic_move and promotion_zone. However, it did not let such a predicate be in the final theory, as according to its evaluation function, there had been no improvement in score. The revision system is more fortunate in this question, since it focuses on the misclassified examples to propose revisions. Progol could neither delete rules, as this is not one of its refinements operators.

Reform chess: Unusual capture rule

The system performed the following revisions on the initial theory:

1. According to condition 1.(e) of section 5.3.2, the king must not go to a position where there is already a friendly piece. This is declared through subgoals of the form $\mathtt{not}(land\_on(..,Colour,...,Colour,...))$ in the clauses defining how a king may move. As free capture chess allows the king to capture a friendly piece, the *delete antecedent* operator removed these literals from their respective clauses.

2. In the original theory, a piece may move to an occupied position only if an opponent piece is there. This is defined as a move by the attack concept. In Reform chess the piece is also allowed to attack a friendly piece. Because of that, the revision chosen to be implemented in most of the folds is proposed by the *add rule* operator. The main literals in the body of such a rule, allows an attack to a piece of the same color, only if such a piece is not a king. Note that the original rule defining the attack concept remains on the theory.

3. Progol does not proposed to create new rules for the same predicate as the revision. Instead, it tried to learn a new definition of the top-level move predicate, without success.

Unusual pieces and larger board: Neunerschach

The system performed the following revisions on the initial theory:

1. The delete rule step discussed in section 5.3.2 removed the clause $piece(\mathtt{queen})$ from the theory;

2. Abduction when searching revision points (third introduced type) added the facts $piece(\mathtt{marshall})$ and $piece(\mathtt{hausfrau})$ to the theory and, just after that,

3. From the rules defining the basic moves of the queen, the add rule operator created rules for the $\mathtt{marshall}$ by replacing the constant $\mathtt{queen}$ from the head of such rule by $\mathtt{marshall}$. This was done by first treating the constants in the clause, substituting them by equality predicates. FORTE always does that before searching for revision points. Then, the literal was deleted and the constant $\mathtt{marschall}$ was made to replace $\mathtt{queen}$;

4. The add new rule operator included the following new rules in the theory

   - Rules defining the basic move of $\mathtt{hausfrau}$, so that this piece can move diagonally and orthogonally, but only two squares;
   - New ground clauses, produced by abduction of the second type, to make the last file and rank valid, namely, $file(\mathtt{i})$ and $rank(9)$.

Remarks about the revision process in Neunerschach chess Note that without the abduction procedure it would be difficult for the revision process to induce the rules for the new pieces, since first of all, the piece must be valid to match condition $2.(a)$ of section 5.3.2 and only after that the piece can try to move. Thus, only including the clauses defining the pieces would not bring any benefit to the theory, since there were not any clauses defining how such pieces should move. Additionally, as the pieces did not exist on the theory, the search for revision points procedure would not be able to reach the literals defining the basic moves of the pieces and then identify them as revision points. Even if it could, defining the basic moves of the clauses is not enough for they to move, since they must be valid in the theory.

The move generator procedure was not able to generate games with promotion cases for this dataset, probably due to the size of the board X maximum number of rounds in the game. Thus, the revision process failed on correcting the rules of promotion, which would allow the pawn to be promoted to the new pieces and the white pawn be promoted in rank 9, instead of in the rank 8. We expect that using

185

games with a larger number of rounds will allow us to represent them, but at the cost of much larger datasets.

Finally, Progol was able of proposing abduction of predicates file/1, rank/1 and piece/1, but it could not create clauses defining basic moves of the new pieces.

Conclusions

We presented a framework for learning variants of a game through automatic theory revision from examples. The framework is composed of a Theory Revision system, the chess initial rules (expressed as the initial theory which is allowed to be modified) and fundamental domain theory (assumed to be correct) and a move generator for obtaining the examples. We described the modifications performed on the revision process to best address the revision of chess problem, including (1) the introduction of an initial step for deleting rules responsible for misclassified negative examples, (2) the use of abduction in three different moments and (3) the use of negation. The experimental results encompassed 3 variants of chess, ranging from a specialized version of chess (minichess) to a more general version of chess (including a larger board and new pieces). The revision was able to return final theories correctly describing most of the rules of the variants. The missing cases were due to the lack of examples of rare events during a game, such as the promotion. Also, the final theory would benefit from a post pruning procedure to remove clauses that have become useless after the revision.

We are aware that the datasets were generated in a quite arbitrary way, since the number of examples and depth of the games were chosen according to the need of generating situations necessary for the revision. A better experimental setting should include several datasets of at least varying sizes. Notice however that our primary goal was to demonstrate the capability of the revision system acquiring the rules of variants of the game, using the rules of the traditional game as starting point. Also, we would like to show that it was necessary to change the base revision system to achieve that. In this way, new issues were introduced in the revision system that can also be useful for others domains besides chess.

Thus, as future work, we intend to further experiment the framework with

datasets varying on the number of maximum rounds and simulated games. We would like to apply the framework to other more complex variants, namely regional variants, such as Shogi. Shogi and several others variants of chess require the knowledge of an entirely new concept. The *Einstein chess* variant, for example, has the concept of demotion, where each time a piece moves without capturing it is "demoted" to a less valuable piece. This concept does not exist in international chess and if one would like to obtain the theory for this variant, literals for demotion would have to be included in the language by hand. This is not the best scenario, since it is assumed the previous knowledge concerns the situations occurring in traditional chess only. Therefore, the revision system would greatly benefit from predicate invention operators (MUGGLETON, BUNTINE, 1988).

A more ambitious future work is learning playing strategies (BAIN, MUGGLETON, 1994). In this case, we would like to obtain strategies for playing the variants of chess from strategies of chess. To do that we believe we would benefit from probabilistic theory revision, so that uncertainty about the strategies could be represented.

Finally, we would like to apply theory revision to others tasks involving transfer learning, such that the domains share the same predicates language. We believe in this way it would not be necessary to map between predicates of one domain to predicates of another domain.

Appendix D: Stochastic Local Search

Searching over large spaces is a recurrent problem in Computer Science. In order to get good hypotheses while still keeping the search feasible, one may take advantage of local search algorithms, which start generating candidate hypothesis at some location of the search space and afterward moving from the present location to a neighbouring location in the search space. Each location has a relatively small numbers of neighbours and each move is determined by a decision based on local knowledge (HOOS, STÜTZLE, 2005). In this way, they abandon completeness to gain efficiency.

It is also possible to further improve efficiency and also escape from local optima by making use of randomised choices when generating or selecting candidates in the search space of a problem, through the *Stochastic Local Search Algorithms*(SLS). One major motivation and successful application of SLS has been in satisfiability checking of propositional formulae, namely through the well-known GSAT (SELMAN, et al., 1992) and WalkSAT (SELMAN, et al., 1996) algorithms. A large number of tasks in areas such as planning, scheduling and constraint solving can be encoded as a satisfiability problem, and empirical observations show that SLS often can substantially improve their efficiency (CHISHOLM, TADEPALLI, 2002; RÜCKERT, KRAMER, 2003).

Randomisation may also be used to improve other search strategies, such as *backtracking* search. For example, Rapid Randomised Restarts (RRR) introduces a stochastic element intro backtrack-style search in order to introduce restart search from scratch if we are not making progress (GOMES, et al., 1998; GOMES, et al., 2000). This led to an interest in applying such techniques on data-mining applications, and more specifically on multi-relational data-mining. Initial work on the area has indeed shown promising results for stochastic techniques when learning

theories from scratch in ILP systems (PAES, et al., 2006b; ŽELEZNÝ, et al., 2006). For two very different algorithms, results showed very substantial improvements in efficiency, with little or no cost in accuracy.

Motivated by these work, this thesis also contributes in the development of SLS methods when revising first-order and probabilistic logical models. Thus, in order to ground the contributions of next chapter, this chapter presents an overview of SLS algorithms and SLS methods employed in Inductive Logic Programming. The chapter starts by describing standard SLS methods in section 5.3.2. Then, well-known and vastly used SLS algorithms, such as GSAT and WalkSAT, are reviewed in section 5.3.2. Finally, in section 5.3.2, SLS algorithms designed to efficiently learn first-order logic theories are discussed. In this last section it is included one technique developed with our contribution, where first-order logic theories are learned using propositionalization combined with a SLS algorithm (PAES, et al., 2006b).

Stochastic Local Search Methods

The key ideas of the search process performed by a Stochastic Local Search Algorithm are as follows.

1. Initialisation: An initial candidate solution is selected, usually by generating a candidate at random;

2. Move step: Iteratively, the process (at random) decides to move from the present candidate solution to a local neighbouring candidate solution, usually (but not always) considering a function to evaluate the neighbours.

3. Stop criteria: The process is finished when it attends a termination criteria, which could be a maximum number of iterations or a solution has been found.

Suppose for example, a *Stochastic Hill Climbing* (or Iterative Descent) strategy (RUSSELL, NORVIG, 2010). It starts from a randomly selected point in the search space (initialisation) and tries to improve the current candidate solution by choosing with uniform probability distribution a neighbour of the current candidate (move), but requiring the value of the evaluation function is improved. The process

finishes when none of the neighbours improves the evaluation function (stop criteria). A *Greedy Stochastic Hill Climbing* strategy (or Discrete Gradient Descent) would try to perform the best move by choosing uniformly the next candidate solution in the set of maximally improving neighbours (the best possible improvement). As such a method requires a complete evaluation of all neighbours in each iteration, one may prefer a *First Improvement* neighbour selection strategy, which moves to the first neighbour encountered improving the value of the evaluation function.

SLS Methods Allowing Worsening Steps

It is easy to note that the performance of a SLS algorithm (as it also occurs in other local search methods) strongly depends on the size of the neighbourhood. The larger is the neighbourhood, the more potentially better candidate solutions it contains. In fact, the ideal case is to have an exact neighbourhood: the neighbourhood relation for which any locally optimal candidate solution is guaranteed to be the globally optimal solution. Obviously, taking into account an exact neighbourhood relation prevents the search method to be stuck in very low-quality local optima. However, it is also much more expensive to determine search improving steps in exact neighbourhoods.

It is possible to escape from local optima and avoid an expensive search considering a fairly simple neighbourhood and allowing the search strategy to perform worsening steps. Usually, a strategy following this idea alternates with a fixed frequency between selecting an improving neighbour and selecting a neighbour at random. In order to avoid cycles, which may happen if the random walk is undone in subsequent improvement steps, the algorithm can probabilistically decide in each step whether to apply an improvement step or a random walk step. The family of algorithms following this strategy is called *Randomised Iterative Improvement (RII)* and its top level search step is exhibited as Algorithm 26. A RII algorithm does not terminate as soon as a local optima is encountered. Instead it may stop the execution when it reaches a number of iterations or when a number of search steps has been performed without making progress in improvement. It is proved that when the search process runs long enough, eventually an optimal solution to any problem instance is found with arbitrarily high probability (HOOS, STÜTZLE, 2005).

This method had been successfully applied to SAT problems, through the WalkSAT algorithm (SELMAN, et al., 1996), which we will discuss more thoroughly in next sections.

Algorithm 26 Top-level Step Function Performed by an Algorithm using Randomised Iterative Improvement method (HOOS, STÜTZLE, 2005)

Input: problem instance π, candidate solution s, walk probability wp
Output: candidate solution s'
 1: $u \leftarrow \text{random}(0,1)$ # returns a number between zero and one
 2: **if** u $\leq$ wp **then**
 3:     $s' \leftarrow \text{random\_walk\_step}(\pi, s)$
 4: **else**
 5:     $s' \leftarrow \text{improvement\_step}(\pi, s)$
 6: **return** the result of step 3

Another mechanism for allowing worsening steps it to accept a worsening step depending on the deterioration in the evaluation function value, i.e., the worse a step is, the less likely it would be performed. The algorithms following such strategy compose the family of *Probabilistic Iterative Improvement (PII)*. In each step of the search process a PII algorithm selects a neighbour according to a given function $p(g, s)$, which determines a probability distribution over neighbouring candidate solutions of s based on their evaluation function values g.

A vastly used strategy, which is closely related to PII is the *Simulated Annealing (SA)* method (KIRKPATRICK, et al., 1983; GEMAN, GEMAN, 1984; CERNY, 1985; LAARHOVEN, ARTS, 1987) (Top-level algorithm in 27). SA starts from a random initial solution s and in each iteration of the search a neighbour s' of s is selected at random. Usually, the search makes the decision of moving to s' or staying in s based on a value T, which is adjusted at each step t by an *scheduling function*. Standard SA always accepts the candidate s' in case it has a score better than s. When it does not happen, s' is accepted with a probability calculated as the exponential of difference between both scores divided by T. Thus, the worse is the move and the less is the value T, the less exponentially is the probability. Thus, bad moves are chosen more frequently at the beginning of the search, when the value of T is high, and they become more difficult to be accepted as the value of T decreases. It has been proved that if the value of T is reduced slowly enough, the algorithm

191

finds a global optima with probability close to 1 (RUSSELL, NORVIG, 2010).

Algorithm 27 Simulated Annealing algorithm (GEMAN, GEMAN, 1984)

Input: Integer number k and *limit*; a float number *lam*
Output: a solution s

1: $s \leftarrow$ random generated candidate solution
2: **for** t from 1 to ∞ **do**
3:      $T \leftarrow$ scheduling($t, k, limit, lam$)
4:      **if** $T == 0.0$ **then**
5:         **return** s
6:      $s' \leftarrow$ a neighbour of s, chosen at random from the neighbourhood relation
7:      $\Delta E \leftarrow$ score(s') - score(s)
8:      **if** $\Delta E > 0$ **then**
9:         $s \leftarrow s'$
10:     **else**
11:        $s \leftarrow s'$ only with probability $e^{\Delta E/T}$

Rapid Random Restarts

Another way of escaping from local optima is to simply restart the search algorithm
whenever it reaches a local minimum (maximum). Such a strategy works reasonably
well when the number of local optima is rather small or reinitialising the process is
not very costly. The development of this approach was motivated by the recognised
variability in performance found in combinatorial search methods (GOMES, et al.,
1998; GOMES, et al., 2000), such as satisfaction constraint problems. Often, the cost
distributions of a complete backtracking search have very long tails and an average
erratic behaviour. Search algorithms with RRR include a stochastic component
in backtracking style search. The key idea is to restart the search, possibly from
another random initial seed, if it is not making any progress after a number of tries.
To do so, a *cutoff* specifies a number of attempts of finding the best solution before
restarting from another point.

A simple example of RRR strategy applied to a hill-climbing algorithm con-
ducts a series of improving moves from a random initial candidate solution. The
algorithm is complete with probability close to 1 for the simple reason that it will
eventually generate the goal solution as the initial solution. If each iteration of
the hill climbing has a probability p of success, the expected number of restarts is

$1/p$ (RUSSELL, NORVIG, 2010).

Other SLS approaches

Others SLS methods exist. *ILS* (LOURENĆO, et al., 2002) combines two types of SLS methods, one in each step. One step reaches local optima as fast as possible and the other step escapes from local optima. At each iteration, *Iterated Local Search* (ILS) first applies a perturbation to the current candidate solution s, yielding a modified candidate solution s'. A local search is performed from s' until a local optimum s'' is obtained. Finally, an acceptance criteria decides if the next candidate solution is s' or s''. *Greedy Randomised Adaptive Search Procedures (GRASP)* (FEO, RESENDE, 1995) randomise the construction method of candidate solutions such that it can generate a large number of good starting points for a local search procedure. Evolutionary algorithms are a large and diverse class of stochastic search methods strongly inspired by models of the natural evolution of biological species (BÄCK, 1996). Generally speaking, evolutionary algorithms such as genetic algorithms (MITCHELL, 1996) start with a set of candidate solutions and repeatedly apply three genetic operators, namely selection, mutation and recombination, replacing partially or completely the current population by a new set of candidate solutions. For the knowledge of the much many others SLS approaches and a better understanding of the methods briefly described in this section, we refer the reader to (HOOS, STÜTZLE, 2005).

Stochastic Local Search Algorithms for Satisfiability Checking of Propositional Formulae

Stochastic local search has been used since the early nineties to solve hard combinatorial search problems, starting from the seminal algorithm published independently by Selman (SELMAN, et al., 1992) and Gu (GU, 1992). This SLS algorithm was able to solve hard satisfiability problems in only a fraction of the time required by the most sophisticated complete algorithms. The satisfiability problem in propositional logic (SAT) must decide whether there exists an assignment of truth values to the variables of a formula F under which F evaluates to true (a model of F). SLS

are among the most successful methods for solving the search variant of SAT, i.e., to find models for a given formula rather than to decide if such a model exists. In the following we discuss the seminal GSAT algorithm (SELMAN, et al., 1992) and its most arguably prominent derivative, the WalkSAT algorithm (SELMAN, et al., 1996).

The GSAT Algorithm

GSAT (SELMAN, et al., 1992) original algorithm is a SAT solver based on a greedy hill climbing procedure with a stochastic component. It is based on a one-flip-variable neighbourhood in the space of all complete truth values assignments of a given formula, written in conjunction normal form (CNF), i.e., a conjunction of clauses, where each clause is a disjunction of literals. Thus, two variable assignments are neighbours iff they differ in the truth assignment of exactly one variable. GSAT uses an evaluation function $g(F, va)$ that maps each variable assignment va to the number of clauses of the given formula unsatisfied under va. At each iteration of the algorithm, $g(F, va)$ must be minimised by flipping the value of one variable, since the goal is to find a model of F, therefore evaluated to zero under $g(F, va)$. The variable to be flipped is selected at random from the neighbourhood of current candidate solution minimising the number of unsatisfied clauses (HOOS, STÜTZLE, 2005).

Algorithm 28 presents the basic GSAT algorithm. It has two nested loops, with the inner loop starting from a randomly chosen truth assignment of the variables in CNF formula F. Then, it iteratively flips the variable resulting in a maximal decrease in the number of unsatisfied clauses. If there is a tie, the variable to be flipped is chosen at random from the set of variables improving the score at most. The inner loop continues until it finds an assignment of variables satisfying F or when it reaches a user defined number of steps. In case no model is found after a maximum number of steps, GSAT reinitialises the search at another randomly chosen truth assignment of F. This is strictly necessary, since the inner loop gets easily stuck in local minima. The outer loop follows trying to find a model of F until a user defined number of tries. After the maximum number of tries without finding a solution, the algorithm ends with "no solution found".

Algorithm 28 GSAT Algorithm (SELMAN, et al., 1992)

Input: Positive integers *maxFlips* and *maxTries*; a CNF formula F
Output: model of F or "no solution found"

1: **for** *try* from 1 to *maxTries* **do**
2:      *va* ← a randomly generated assignment of the variables in formula F
3:      **for** *step* from 1 to *maxSteps* **do**
4:         **if** *va* satisfies F **then**
5:            **return** *va*
6:         *v* ← randomly selected variable flipping which minimises the number of unsatisfied clauses
7:         *t* ← *t* with *v* flipped
8: **return** "no solution found"

Due to the greedy hill climbing nature of GSAT, it suffers from a severe stagnation behaviour, getting easily stuck in local optima for any fixed number of restarts. Thus, GSAT has been extended into to other search strategies in order to improve its performance. One of the most prominent is the GWSAT algorithm (SELMAN, KAUTZ, 1993) which decides at each step with a fixed probability np whether to do a standard GSAT step or to flip a variable selected uniformly at random from the set of all variables occurring in unsatisfied clauses. The probability np is called *walk probability, noise setting or noise level.* The WalkSAT algorithm, which we discuss next is derived from GWSAT.

The WalkSAT Algorithm

WalkSAT (SELMAN, et al., 1994; SELMAN, et al., 1996; SELMAN, et al., 1997) changes GSAT based algorithms mainly by considering only a dynamically determined subset of the GSAT static neighbourhood relation. This is effectively done by first considering only variables occurring in unsatisfied clauses. Then, in order to find the next assignment, the variable to be flipped is selected from that set in two steps. In the first step, a clause c, which is unsatisfied under the current assignment of truth values, is selected at random. Next, in a second step, the new assignment results by flipping one of the variables appearing in c. This general procedure of the WalkSAT architecture is exhibited as Algorithm 29. Note that as GSAT, WalkSAT starts from a randomly generated assignment of the variables in initial formula. Also

as GSAT, it considers a maximum number of tries and a maximum number of steps in order to find the solution.

Algorithm 29 WalkSAT General Algorithm (SELMAN, et al., 1994)

Input: Positive integers *maxFlips* and *maxTries*; a CNF formula F; a heuristic function *hf*
Output: model of F or "no solution found"

1: **for** *try* from 1 to *maxTries* **do**
2:     *va* ← a randomly generated assignment of the variables in formula F
3:     **for** *step* from 1 to *maxSteps* **do**
4:         **if** *va* satisfies F **then**
5:             **return** *va*
6:         *unsat* ← the set of all unsatisfied clauses under *va*
7:         *c* ← a randomly selected clause from *unsat*
8:         *v* ← a variable selected from *c* according to the heuristic function *hf*
9:         *t* ← *t* with *v* flipped
10: **return** "no solution found"

The general procedure requires a heuristic function as input to decide which variable is going to be selected, in line 8 of the algorithm. The most commonly used heuristic function applied into the WalkSAT architecture first scores each variable v by counting the number of currently satisfied clauses that will become unsatisfied by flipping the variable. Then, it tries to perform a *zero damage step*: if there is a variable with score equal to zero, that is, if the the clause c becomes satisfied by flipping the variable v without damaging another clause, then v is flipped (if there is more than one variable in this situation, one of them is chosen at random). In case it is not possible to follow the zero damage step, WalkSAT must decide which step to follow: a *random walk step* or a *greedy step*. The decision is taken considering a walk probability wp as follows:

- With a certain probability wp one of the variables from the clause c is selected at random to be flipped (random walk step);

- With probability $1 - wp$ the variable with the minimal score calculated as above is selected to be flipped (greedy step)

This heuristic function is presented as Algorithm 30.

Algorithm 30 Most Used Heuristic Function into WalkSAT Architecture

---

Input: va a variable assignment of a formula F; a clause c; wp, a walk probability
Output: v a variable to be flipped

1: *scores* ← for each variable in c, the number of clauses that are currently satisfied, but become unsatisfied if the variable is flipped
2: **if** $min(scores) = 0$ **then**
3:      v ← a random variable from c whose score is $min(score)$
4: **else**
5:      **with** probability wp **do**
6:          v ← a random variable from c
7:      **otherwise**
8:          v ← a random variable from c whose score is $min(score)$

Following the success of GSAT/WalkSAT based algorithms to check the satisfiability of propositional formulae, a large number of randomised strategies were derived from them to solve tasks in areas such as planning, scheduling and constraint solving. Empirical observations show that SLS often can substantially improve their efficiency (CHISHOLM, TADEPALLI, 2002; RÜCKERT, KRAMER, 2003; RÜCKERT, KRAMER, 2004).

Stochastic Local Search in ILP

Most Inductive Logic Programming algorithms perform search on a large search space of possible clauses, leading to huge time and storage requirements and urging for clever search strategies (PAGE, SRINIVASAN, 2003). It is therefore unsurprising that research on stochastic search has taken place since early ILP days (KOVACIC, et al., 1992). Many of the ILP algorithms indeed include a limited amount of stochastic search. As an example, GOLEM system randomly select examples as *seeds* to start their search (MUGGLETON, FENG, 1990). Next we present recent work on stochastic search in ILP.

Stochastic Local Search for Searching the Space of Individual Clauses

A recent study in ILP implemented and evaluated the performance of several randomisation strategies in the ILP system Aleph (ŽELEZNÝ, et al., 2002; ŽELEZNÝ,

et al., 2004; ŽELEZNÝ, et al., 2006), using Aleph deterministic general-to-specific search as reference. Aleph runs a covering algorithm which obtain hypotheses inducing clauses one by one, until all the positive examples are covered. Roughly, Aleph's covering procedure is composed of two nested loops. The inner loop starts with the generation of the bottom clause from a seed example. Then, a clause with the best score is generated by adding antecedents from the bottom clause. The outer loop of the covering algorithm removes already covered positive examples from the set of examples and the inner loop restarts with this new set of examples. The procedure continues until there are no move not-covered positive examples or it is not possible anymore to generate clauses obeying the setting of parameters defined by the user. Note that the inner iteration returns a single clause generated independently from the clauses previously added to the theory. Thus, the stochastic strategies developed in that study were framed in terms of a single clause search algorithm.

They designed four randomised restart strategies to search the ILP subsumption lattice, namely (1) A simple randomised search strategy (RTD); (2) A Rapid Random Restart strategy (RRR); (3) A GSAT based strategy and (4) a WalkSAT based strategy. The randomised strategies differ on how to choose the saturated example, on which clause to start from, on which clause to try next, on whether to do greedy or full search, and on whether to do bidirectional refinement or not, following the main properties:

1. the saturant example is chosen at random in all randomised algorithms, instead of being the first positive example as in the deterministic reference strategy;

2. the search starts from a clause selected with uniform probability from the set of allowable clauses, except for the RTD search which which starts from the most general definite clause (SRINIVASAN, 2000);

3. GSAT and WalkSAT strategies update the list of possible modifications in the current hypothesis greedily, i.e., only the newly explored nodes are retained, whereas RRR and RTD maintain a list of all elements.

4. the selection of next clause follows a random choice in WalkSAT and in RTD, according to the following criteria: with probability 0.5 the clause with the

highest score is chosen at random and otherwise a random clause is chosen with probability proportional to its score. The others strategies choose the clause with the highest score.

5. GSAT, WalkSAT and RRR perform bidirectional refinement, combining specialisation and generalisation, instead of only performing specialisations of the clause being refined.

6. All strategies include restarts based on a cutoff parameter, defined as the maximum number of clauses evaluated on any single restart.

It was observed that if a near-to-optimal value of the cutoff parameter (the number of clauses examined before the search is restarted) is used, then the mean search cost (measured by the total number of clauses explored rather than by cpu time) may be decreased by several orders of magnitude compared to a deterministic non-restarted search. It was also observed that differences between the tested randomised methods were rather insignificant.

There are others approaches of SLS methods for searching the space of candidate clauses. Particularly, Muggleton and Tamaddoni-Nezhad have been conducting research based on the stochastic search performed by genetic algorithms (TAMADDONI-NEZHAD, MUGGLETON, 2000; MUGGLETON, TAMADDONI-NEZHAD, 2008). Their approach is built on the fact that to find desirable consistent clauses in ILP systems it is necessary to evaluate a large number of inconsistent clauses, and such consistent clauses are located at the fringe of the search space. The approach is composed of two components. The first one, called *Quick Generalisation* (QG), carries out a random-restart stochastic bottom-up search to efficiently generates a population of consistent clause on the fringe of the refinement graph search without needing to explore the graph in detail. The second component is a Genetic Algorithm which evolves and re-combines those seeded clauses, instead of performing the A* of Progol system. The experiments performed in that work indicate that QG/GA algorithm can be more efficient than the standard refinement graph search of Progol system, while generating similar or better solutions.

The approaches just reviewed were both framed in a single clause search algo-

rithm. One consequence of this is that the statistically assessed performance ranking of individual strategies may not be representative of their performance when used for an incremental entire-theory construction due to the statistical dependence between the successive clause search procedures.

Another issue concerns the time spent to evaluate a candidate hypothesis. Even though the search space is reduced because of SLS methods, there is a large amount of time used to check whether a hypothesis should be chosen as the next candidate solution. In the next section we present methods to learn theories which try to overcome both covering and hypothesis evaluation pitfalls.

Stochastic Local Search in ILP for Searching the Space of Theories and/or using Propositionalization

The standard greedy covering algorithm employed by most ILP systems is a short-coming of typical ILP search. There is no guarantee that greedy covering will yield the globally optimal hypothesis; consequently, greedy covering often gives rise to problems such as unnecessarily long hypothesis with too many clauses. To overcome the limitations of greedy covering, the search can be performed in the space of entire theories rather than clauses (BRATKO, 1999). However, there is a strong argument against this: the search space composed of theories is much larger than the search space of individual clauses. It is interesting then to apply an efficient search strategy such as SLS for searching hypothesis in the space of theories and hence uniting the benefit of both techniques.

Stochastic local search algorithms for propositional satisfiability benefit from the ability to quickly test whether a truth assignment satisfies a formula. As a result, many possible solutions (assignments) can be tested and scored in a short time. In contrast, the analogous test within ILP—testing whether a hypothesis covers an example—takes much longer, so that far fewer possible solutions can be tested in the same time.

Thus, considering both motivations above, in a recent work (PAES, et al., 2006b) we have applied stochastic local search to ILP, but not to the usual space of first-order Horn clauses. Instead, we used a propositionalization approach that

transforms the ILP task into an attribute-value learning task. In this alternative search space, we can take advantage of fast testing as in propositional satisfiability. Additionally, we use a non-covering approach to search in the space of theories since now we are dealing with a more efficient search strategy and we also turn the relational domains into a simpler propositional one. Then, we use a SLS algorithm to induce k-term DNF formulae, which performs refinements on an entire hypothesis rather than a single rule (RÜCKERT, KRAMER, 2003).

Propositionalization

Propositionalization is a method to compile a relational learning problem to an attribute-value problem, which one can solve using propositional learners (LAVRAC, DZEROSKI, 2001; KROGEL, et al., 2003). During propositionalization *features* are constructed from the background knowledge and structural properties of individuals. Each feature is defined as a clause in the form $f_i(X) := Lit_{i,1}, ..., Lit_{i,n}$ where the literals in the body are derived from the background knowledge, and the argument in the clause's head is an identifier of the example. The features are the attributes which form the basis for columns in single-table (propositional) representations of the data. If such a clause defining a feature is called for a particular individual and this call succeeds, the feature is set to "true" in the corresponding value column of the given example; otherwise it is set to "false".

There are several propositionalization systems such as RSD (ŽELEZNÝ, LAVRAC, 2006) and SINUS (LAVRAC, DZEROSKI, 1994), among others. In this work we used RSD as the base propositionalization system. RSD constructs features by discovering statistically interesting relational subgroups in a population of individuals[4].

Stochastic Local Search in k-term DNF Relational Propositionalised Domain Learning

After propositionalising the relational domain, we apply an SLS algorithm to learn k-term DNF formulae from the feature-value table. The aim in k-term DNF learning is to induce a formula of k terms in disjunctive normal form, where each term is a

---

[4]RSD is publicly available at http://labe.felk.cvut.cz// zelezny/rsd

conjunction of literals (KEARNS, VAZIRANI, 1994). k-term DNF learning is a NP-hard problem of combinatorial search. The SLS algorithm designed in (RÜCKERT, KRAMER, 2003) to solve k-term DNF learning is reproduced here in Algorithm 31.

The algorithm starts generating randomly a hypothesis, i.e., a DNF formula with k-terms and then refines this hypothesis in the following manner. First, it picks a misclassified example at random. If this example is a positive one the hypothesis must be generalised. To do so, a literal has to be removed from a term of the hypothesis. Now, with probability p_{g1} and p_{g2} respectively, the term and a literal in this term are chosen at random. Otherwise the term in the hypothesis which differs in the smallest number of literals from the misclassified example and the literal whose removal from the term decreases the score at most are chosen. On the other hand, if the example is a negative one, it means that the hypothesis must be specified. Therefore, a literal has to be added in a term. The term is chosen at random from those ones which cover the misclassified negative example. In a similar way to the last case, either with probability p_s the literal to be added in this term is chosen at random or a random literal which decreases the score at most is taken. This iterative process continues until the score is equal to zero or the algorithm reach a maximum number of modifications. All the procedure is repeated a pre-specified number of times.

It is important to mention that Algorithm 31 performs refinements of an entire hypothesis rather than a single rule. A detailed analysis of SLS performance compared to WalkSAT shows the advantages of using SLS to learn a hypothesis as short as possible (RÜCKERT, KRAMER, 2003).

Experiments and remarks about them   Two ILP benchmarks were considered in the paper (PAES, et al., 2006b): the East-West Trains (MICHALSKI, LARSON, 1977) and Mutagenesis Data (SRINIVASAN, et al., 1996). They were both propositionalised by RSD, producing a set of features in attribute-value form. Then, the K-term DNF SLS learner were compared to Aleph in its default mode and to Aleph using GSAT search as explained in section 5.3.2, both using the original relational dataset. Additionally, the propositionalised domains were given as input

---

Algorithm 31 A SLS algorithm to learn k-term DNF formulae (RÜCKERT, KRAMER, 2003)

Input: Integers k and $maxSteps$; probability parameters p_{g1}, p_{g2} and p_s; a set of examples E in attribute-value form
Output: A k-term DNF formula

1: $H \leftarrow$ a random generated DNF formula with k terms
2: $steps \leftarrow 0$
3: **while** $score(H) \neq 0$ and $steps < maxSteps$ **do**
4:      $steps + +$
5:      $ex \leftarrow$ a incorrectly classified example under H, get at random from E
6:      **if** ex is a positive example **then**
7:          **with** probability p_{g1} **do**
8:              $t \leftarrow$ a random term from H
9:          **otherwise**
10:             $t \leftarrow$ the term in H that differs in the smallest number of literals from ex
11:         **with** probability p_{g2} **do**
12:             $l \leftarrow$ a random literal in t
13:         **otherwise**
14:             $l \leftarrow$ the literal in t whose removal decreases $score_L(H)$ most;
15:         $H \leftarrow H$ with l removed from t
16:     **else if** ex is a negative example **then**
17:         $t \leftarrow$ a (random) term in H that covers ex;
18:         **with** probability p_S **do**
19:             $l \leftarrow$ a random literal m so that $t \wedge m$ does not cover ex;
20:         **otherwise**
21:             $l \leftarrow$ a literal whose addition to t decreases $score_L(H)$ most
22:         $H \leftarrow H$ with l added to t

---

to Part (FRANK, WITTEN, 1998) and Ripper (COHEN, 1995), two popular rule learners algorithms. Besides the fact that they do not use SLS, they also differ from K-term DNF learner in the use of a covering approach. The results were compared in terms of compression achieved, cpu time consumed and predictive accuracy. They indicated that DNF/SLS performs faster w.r.t. all other tested methods when it comes to short theories (in number of rules). Comparing to relational methods, the performance gap was significantly large (in orders of magnitude), while corresponding predictive accuracy does not favor either of SLS/DNF or the relational methods. Comparing to the propositional methods, this performance gap is much smaller, while SLS/DNF's short theories exhibit slight superiority in terms of predictive accuracy.

There are other approaches to induce hypothesis using SLS with or without propositionalization. For instance, (SERRURIER, PRADE, 2008) employs Simulated Annealing to induce hypothesis directly - do not using neither propositionalization nor covering. The candidate hypothesis are generated by a neighbourhood relationship derived from a refinement operator defined over hypothesis. In this case, the neighbourhood of a current hypothesis H is composed by adding or removing clauses from H or still by applying a refinement operator (downward or upward) on each one of its clauses. In (JOSHI, et al., 2008; SPECIA, et al., 2009) was developed an approach to construct features randomly in order to build modes to assist the task of Word Sense Disambiguation. A randomised search procedure based on GSAT and using theory-error-guided sampling is designed for dynamically construct the features and generate the output model. The procedure is composed of two nested steps: the outer loop iterates R times, where R is a pre-defined number of restarts and the inner loop executes M times, where M is a pre-defined number of local moves. In the outer loop, a sample of n acceptable features is generated, where a feature is considered as acceptable it covers at least s examples, has a minimal pre-defined precision p and obeys constraints of the language. A model is then constructed from this set of features. The inner loop iteratively selects a new subset of features based on the errors made by the current features on the current model.

Appendix E: Revising First-Order Logic Theories through Stochastic Local Search

Introduction

Usually, a First-Order Theory Revision system perform search in three steps. First, it searches for points in the theory responsible for misclassifying an example. Second, it searches for possible modifications to be implemented within each revision operator, including addition and deletion of antecedents in the body of clauses. And finally, it searches from a number of available revision for the one which will be responsible for implementing a modification to the theory. In each one of these searches, a system such as FORTE and its descendants systems follow an enumerative strategy, engendering large search spaces that may grow to be intractable, according to the factors below.

1. The number of misclassified examples, since the revision system traverses each example's proof looking for faulty points;

2. The size of the initial theory, since every clause on it might be potential revision points;

3. The number of clauses responsible for misclassifying an example, since the revision system proposes modifications to each one of them;

4. The size of the knowledge base and background knowledge, since antecedents must have to be generated and added to the body of clauses, as possible

modifications to be implemented on the theory.

Additionally, theory revisions systems do tackle whole theories instead of step-wise search for individual clauses as most ILP systems do. Search over whole theories is known to be a hard problem (BRATKO, 1999). As a result, traditional theory revision systems must search over extremely large spaces, and can become rather inefficient and even intractable.

The last years have shown that stochastic local methods, originally designed to solve difficult combinatorial propositional problems (SELMAN, et al., 1992; SELMAN, et al., 1996; RÜCKERT, KRAMER, 2003), can also perform well in a variety of applications. Moreover, combining Stochastic Local Search with inductive Logic Programming has shown very substantial improvements in efficiency, with little or no cost in accuracy (SRINIVASAN, 2000; PAES, et al., 2006b; ŽELEZNÝ, et al., 2006; MUGGLETON, TAMADDONI-NEZHAD, 2008). Such results motivate the contribution of this chapter. Further, we take Trefheten's Maxim No. 30 into high consideration, which states that if the search space is huge, the only reasonable way to explore it is at random (TREFETHEN, 1998). Thus, we aim to achieve a balance between efficiency and efficacy, by decreasing the negative impact of the mentioned factors on the running time of theory revision. To do so, we sacrifice completeness in favor of finding good solutions rather than optimal ones, by means of SLS techniques. Stochastic components are included in the key searches of the revision process, namely:

1. Search for revision points: A random decision may return a subset of the revision points instead always returning all of them.

2. Search for literals: as the proposals of modifications are dominated by addition and deletion of antecedents, one may benefit from randomising antecedent search (PAES, et al., 2007a; PAES, et al., 2007b).

3. Revision search: rather than proposing all revisions, one might enumerate the possible modifications and choose one to implement at random (PAES, et al., 2007b).

Preliminary experiments with the theory revision system FORTE showed good promise from introducing stochastic search at the last two searches above (PAES, et al., 2007a; PAES, et al., 2007b). This chapter revisits that work enhancing it by designing a number of stochastic search in every step of YAVFORTE system. Note that in (PAES, et al., 2007a; PAES, et al., 2007b) stochastic components were introduced in original FORTE system and in the present work they are built upon YAVFORTE system, including the bottom clause to bound the search space and mode declarations as the language.

The outline of this chapter is as follows. Firstly, section 5.3.2 brings the stochastic algorithm developed to search for the revision points. Next, stochastic algorithms applied within the revision operators, for choosing literals to be added to or removed from a clause, are devised in section 5.3.2. Then, we present a number of SLS algorithms for deciding which revision operator will be responsible for modifying the theory in section 5.3.2. Finally, experimental results are presented in section 5.3.2, followed by conclusions in section 5.3.2.

Stochastic Local Search for Revision Points

FORTE-based systems follow the key steps below in order to generate revision points:

1. Identify the misclassified instances;

2. Through the misclassified examples, find the clauses and/or antecedents responsible for such misclassifications. These points will compose the set of revision points;

3. Calculate the potential of each revision point. Remember that the potential is the number of examples which identify the necessity of modifying the revision point;

4. Identify the relevant examples for each revision point, i.e., the examples whose provability can be affected after proposing some revision in that point. This is essential to make the evaluation of revision operators more efficient by not

proving every example at each modification, since only the proofs of relevant examples must be re-done when proposing modifications on specific points of the theory.

It can be seen from the steps above that there are two major factors working together to possibly increase the cost of searching for revision points: the set of examples and the size of the theory. This happens because each example must be tested on the theory, either to identify faulty clauses or literals or to check whether the example is relevant to the revision point. Moreover, each clause in the theory may be tested as a potential revision point. Therefore, what it is done in this thesis is to decrease the work performed in those tests by introducing a stochastic component within the search for revision points. Rather than always looking for all revision points in the theory, the stochastic component allows only a subset of that group to be sought.

The strategy developed in this chapter does not only avoid searching in the whole theory for revision points but also avoids considering all misclassified examples. It works by alternating between stochastic and complete moves according to a certain probability, a method that can be seen as an instance of Randomising Iterative Improvement and WalkSAT techniques. Thus, with a probability p_{rp} the stochastic move is taken and the procedure will look for only a subset of all the possible revision points. The size of the subset is previously defined by the user, with its default value as 1. Otherwise, a complete move is taken just as in the original algorithm.

The stochastic move works as follows to gather the subset of revision points. First, it selects a misclassified example at random. Then, revision points are collected from such an example. In case this single misclassified instance already produces the required number of revision points, the procedure stops. If it produces more than that, the number is chosen at random from them. If the instance does not have enough revision points, the procedure proceeds to collect more revision points by choosing another misclassified example at random. After collecting the subset of random revision points, it is time to find out the relevant examples. This is necessary to compute the potential and also to consider only those examples to be proved

again when evaluating a modification on the revision point. Note that in case the probability p_{rp} is 100% and the subset is required to be composed of only one revision point, the approach employed here follows the same line of thought of (RÜCKERT, KRAMER, 2003), although there they address propositional learning. Algorithm 32 brings the procedure for collecting revision points using a stochastic component.

Algorithm 32 SLS Algorithm for generating revision points

Input: A set of positive and negative examples E, divided into ECC, the correctly classified examples and EIC, the incorrectly classified examples, Theory T, probability p_1, A integer k
Output: A set of revision points RP

1: $RP \leftarrow \emptyset$
2: **with** probability $p1$ **do**
3:     **while** $\#RP < k$ **do**
4:         $ex \leftarrow$ a misclassified instance chosen at random from EIC;
5:         $num\_rp = k - \#RP$
6:         $RP_{ex} \leftarrow$ at most num_rp revision points generated from ex;
7:         $RP \leftarrow RP \cup RP_{ex}$;
8: **otherwise**
9:     $RP \leftarrow$ the revision points generated from all misclassified examples in E
10: **for each** revision point $rp \in RP$ **do**
11:     identify the relevant examples for RP;
12:     calculate the potential of RP;
13:     sort RP by potential;
14: **return** rps;

It is important to stress that examples from both classes have the same chance to be chosen at random. This is particularly important when we have skewed datasets, with one class largely dominating the other on the number of examples. Algorithm 32 introduces the stochastic component as an alternative path in Algorithms 2.2 and 2.3, by avoiding in stochastic moves to search for revision points at each clause of the theory and not considering all misclassified examples. In this way, roughly speaking, instead of getting a complexity time bounded by the number of training examples times the size of the theory, we get for stochastic moves a time complexity limited to the number of required revision points, which in the best case is only 1. Note that in the complete case, the theory is traversed even to the correctly classified examples, because it is in this moment that the relevant examples are collected. Although the stochastic search still has to find the relevant

examples for the revision points in the set of all training examples, this is also a reduced search, since it is restricted to the set of revision points instead of traversing the whole theory. Additionally, in case the search returns only one kind of revision points, two distinct cases be addressed: (1) the revision point is a generalisation one and then only unprovable examples are considered (true negatives and false negatives) or (2) the revision point is a specialisation one and only provable examples (true positives and false positives) are considered.

Stochastic Local Search for Literals

The main goal of introducing antecedents to a clause is to stop proving negative examples while continues covering as much of the originally proved positive examples as possible. To do so, the operator can proceed from two approaches. Either it uses a hill-climbing procedure, where at each iteration the antecedent which improves the score at most is chosen to be added in the clause or it uses the relational pathfinding algorithm, where more than one antecedent can be added to a clause at once. These two approaches can also be combined, with the relational pathfinding algorithm being executed and, next, antecedents being added to a clause through the hill-climbing algorithm. Both approaches consider the bottom clause generated from a covered positive example as their search space.

Similarly, the delete antecedents operator has the goal to make the clause to start proving positive examples while still does not proving as much of the negative examples as possible. To achieve its goal, this operator either removes one antecedent at once from the clause, using a hill-climbing approach, or it can delete multiple antecedents at once to escape from maxima local. However, this last approach is only used when the latter does not produce any results, since it is expensive to list and test the combination of all possible literals to be removed from the clause. Both approaches require the modes language to be obeyed after a removal.

Add antecedentes and delete antecedents are the basic operations performed into all operators of YAVFORTE with the exception of delete rule: Specialisation is performed either by adding antecedents to a clause or deleting clauses from the theory; generalisation is achieved by either deleting antecedents from a clause or

creating new clause. In this last case it is possible to create a clause from another one, by leaving the original one in the theory and deleting antecedents followed by adding antecedents from its copy. It is also possible to create a completely new using the add antecedents operator in a clause with a predicate in its head, properly defined in a modeh declaration.

Regarding addition of antecedents there are three main factors impacting the search space of the bottom clause: (1) the size of the intentional and extensional background knowledge, (2) the amount of different modeb definitions to the predicates together with the recall of each one of them, and (3) the value set to variables depth. In fact, as it was shown in Algorithm 5.3, the cardinality of a bottom clause is bounded by $r(|M|j + j-)^{ij^+}$, where $|M|$ is the cardinality of the set of modes declarations, j^+ is the number of $+$ type occurrences in each modeb in M plus the number of $\breve{\ }$ type occurrences in each modeh, j^- is the number of $\breve{\ }$ type occurrences in each modeb in M plus the number of $+$ type occurrences in each modeh, r is the recall of each mode $m \in M$, and i is the maximum variable depth. Thus, the bottom clause generates a search space of exponential size w.r.t. the maximum variable depth. In case the recall r is defined as $*$, which is the most common case, all the possible instantiations of a literal are going to be collected in the BK. Because of that, the size of the BK also influences the cardinality of the bottom clause.

Possibly, each element of the bottom clause may be tested on each example relevant to the clause being specialised, excluding those who does not have variables compatible to the mode declarations of the clause (although they also slightly influence the running time since they must be tested to check the compatibility). Additionally, in the worst case, to specialize one single clause it is necessary to pick up literals from the bottom clause as many times as the maximum size set to a clause. Considering all these factors, addition of antecedents is an expensive operation and still performed many times during the whole revision process. Not counting the relational pathfinding algorithm which is expensive for itself (RICHARDS, MOONEY, 1992). Therefore, we intend to make the add antecedents operator, and consequently the revision process, more efficient by introducing stochastic components on this. Once again, we sacrifice completeness to gain efficiency when proposing

modifications on the theory by adding antecedents to clauses or creating new rules.

Deleting antecedents is obviously a less expensive operation than adding antecedents. The search space is composed of only the literals in the clause. Because of that at each iteration an antecedent is deleted the search space of the next iteration is reduced. Thus, in the worst case the search space of tested literals at each iteration has all antecedents in the body of the current clause. Note that sometimes the search space is is less than the size of the clause because some literals can make the clause incompatible to the modes definitions when removed from it. However, they still increases the running time, although slightly, since it is necessary to check if they can or cannot be a candidate to be deleted. Additionally, in the worst case, the operation of deleting antecedents may be performed $|clause|$ times in case deletions always improve the score. Thus, although benefiting less than when adding antecedents, we can also improve running time of the proposals of modifications by making the delete antecedent operator more efficient. Aiming this goal, we also introduce stochastic search when deleting antecedents, either when proposing generalisations on a single clause or when generalising the theory by creating a new rule from an existing one.

Stochastic versions of the delete antecedents and add antecedents algorithms were developed, performing according to the following strategy. They may perform either a random or a greedy move, depending upon a fixed probability. While the greedy move is the same for both approaches, since in this case the original algorithm is maintained, the random move differs for each approach. Next we devise each approach separately.

Stochastic Component for Searching Literals

The algorithm introducing a stochastic component follows a stochastic hill climbing technique and adopts a conservative strategy even when performing a random move, by maintaining the requirement of improving the value of the evaluation function, and by refining clauses in the same way as it is done in the original algorithm. The stochastic component is employed for choosing the next candidate clause. The decision of not allowing bad moves to escape from local maxima is arguably justified

because adding/deleting a sequence of literals at once, using the relational pathfinding algorithm or the algorithm for deleting multiple antecedents, may already be able of escaping from local maxima. Basically, a random walk is carried out by taking a random step for choosing the next candidate clause with a fixed probability p_l, In case the probability p_l is not achieved, the original greedy hill climbing algorithm is performed. This algorithm is built upon the following decisions.

- **Defining the search space:** Candidate clauses are formed by adding/deleting a single literal or a sequence of literals in the current clause, depending on the algorithm employed (greedy hill climbing or relational pathfinding/deletion of multiple antecedents). The literals to be added/deleted are elected differently, depending upon the move is greedy or stochastic. In case the move is greedy, the approach is pure hill climbing in the sense the current clause and candidate clauses differ by only one literal and the goal is to add antecedents, the set of possible antecedents to be added to a clause is composed by literals taken from the bottom clause. Similarly, when adding a sequence of literals, the paths are created considering the literals of the bottom clause. When deleting antecedents in the greedy move, candidate clauses are created either by removing a single literal or by removing a combination of literals. In case the move is random and the goal is to add single antecedents, a literal picked at random from the bottom clause is added to the current clause to form a candidate clause. Similarly, a path created from the literals of the bottom clause in relational pathfinding algorithm is chosen at random. To delete antecedents from a clause in a stochastic move, the body of the clause is randomised and then a literal is chosen. In a similar way, combinations of literals are randomised and one of them is taken at random. In all of the cases, the candidate clause must be valid according to the modes declarations.

- **Choosing the next clause:** Since this conservative approach requires the next clause improves the current score, in all cases listed above this demand must be attended. However, in greedy moves, all candidate clauses are tested on the examples in order to calculate their score and the one improving the

213

score at most is the chosen one, while in random moves, the first randomly generated candidate clause improving the score is chosen. In others words, to choose at random the next clause, a single candidate clause formed as explained in the last topic has its score computed using the set of examples. If this clause already improves the score, it is going to be the next clause. Otherwise, it is necessary to choose another candidate clause, which is clearly formed as explained above, by selecting a random literal or a sequence of literals. This procedure continues until finding a candidate clause improving the score or to find out that there is no clause able to do that, finishing the operator procedure. The only exception is the relational pathfinding algorithm, since its procedure allows a path to be chosen if the score is not changed (neither increased nor decreased). It only allows that because after it runs, hill climbing is employed to further specialize the resulting clause. Note that in the best case only one candidate clause is evaluated, which makes the run time of the procedure independent on the size of the bottom clause in case of specialisations, or on the current clause, in case of generalisations. In the worst case all possible candidate clauses are evaluated as in greedy steps. Choosing next clause in random moves greatly benefits from do not computing score for each possible candidate, which is the most expensive task performed inside the original algorithms since it is necessary to check the provability of each relevant example.

- **Stop criteria:** As usual in hill climbing approaches, the algorithm stops when there are no more candidate clauses improving the score, either because the set of generated candidate clauses are not able to do that, or because there are no further valid clause to be evaluated.

Algorithm 33 substitutes hill-climbing addition of antecedents, exhibited as Algorithm 18 in both add-antecedent specialisation operator and second phase of the add-rule generalisation operator. The algorithm starts by generating the bottom clause from a covered positive example, as it is done in the original Algorithm. After that, it performs a random walk, following the approach of algorithms such

as WalkSAT, and decides the type of the move, based on a fixed probability p_{ls}. In case p_{ls} is not reached, the algorithm performs a greedy hill climbing step, exactly as it done in the original algorithm: all valid (according to modes) candidate clauses formed by adding the literals from the bottom clause to the current clause are evaluated on the examples. Then, the candidate clause improving the score at most is selected. If there is no such improving clause, the procedure returns nothing. If the probability p_{ls} is reached, a random step is taken: a literal is selected at random from the bottom clause and added to the current clause. After the candidate clause is validated relative to the modes, it is evaluated using the examples. In case such a clause improves the current score, it is chosen to replace the current clause. Otherwise, it is discarded and another candidate clause is selected. This procedure continues until finds a clause improving the score or until exhausting all the possibilities. Finally, the candidate clause replaces the current clause (if there is one) and the algorithm proceeds to the next iteration. This procedure is performed until there is no further clause improving the score or if is reaches the maximum size defined to clauses.

Relational pathfinding algorithm provides a sequence of antecedents to be introduced in a clause. The algorithm searches for all possible sequences and chooses the one with the highest score. In case of a tie, the smallest sequence is chosen. A stochastic version of this algorithm selects the sequence to be added to the current clause according to a stochastic decision: with a probability p_{ls} it chooses a sequence at random from all the possible generated paths; otherwise it proceeds as in the original algorithm. We do not generate paths at random, since this algorithm tries to find a sequence of literals connecting the variables in the head of the clause, and to introduce a randomness component into this process could either disregard a possible valid sequence or to force the procedure to backtrack to several previous points. As it was said before, this algorithm is quite expensive by itself and therefore introduce more backtracks on it goes contrary to our primary objective of reducing run time. Therefore, the benefit the stochastic algorithm brings is to avoid computing score considering the set of examples for each possible sequence, which is obviously an expensive task. Algorithm 34 shows such procedure.

Algorithm 33 Algorithm for adding antecedents using hill-climbing SLS

Input: A clause C, CL, maximum size of a clause, p_{ls}, the probability for deciding which move is going to be taken
Output: A (specialised) clause C'

1: **repeat**
2:     $currentScore \leftarrow$ compute score of C;
3:     $BC \leftarrow$ createBottomClause(...);
4:     **with** probability p_{ls} **do**
5:         **repeat**
6:             $ante \leftarrow$ an antecedent chosen at random from BC, whose input variables are already in C (therefore it obeys modes);
7:             $C' \leftarrow C$ with $ante$ added to it;
8:             $candidateScore$ score of C';
9:             **if** $candidateScore > currentScore$ **then**
10:                 $C \leftarrow C'$
11:                 $currentScore \leftarrow candidateScore$
12:             **else**
13:                 $BC \leftarrow BC - ante$
14:         **until** $C = C'$ or $BC \neq \emptyset$
15:     **otherwise**
16:         **for each** antecedent $ante \in BC$ **do**
17:             $C' \leftarrow C$ with $ante$ added to, in case $C + ante$ obeys the modes declarations;
18:             $candidateScore$ score of C';
19:         $bestClause \leftarrow$ candidate clause with the highest $candidateScore$
20:         **if** $candidateScore > currentScore$ **then**
21:             $C \leftarrow bestClause$
22:             $currentScore \leftarrow candidateScore$
23:     remove $ante$ from BC
24:     $FPC \leftarrow FPC-$instances in FPC not proved by C;
25: **until** $FPC = \emptyset$ or there are no more antecedents in BC or it is not possible to improve the score of the current clause or $|C| = CL$
26: **return** C

Algorithm 34 Stochastic Relational-pathfinding

1: generate all possible sequence of antecedents through relational_pathfinding algorithm and the Bottom clause;
2: **with** probability p_{l2} **do**
3:     choose a sequence at random;
4: **otherwise**
5:     choose a sequence with the highest score or the one with less antecedents in case of a tie;

The delete-antecedent operator benefits less from stochastic local search than add-antecedent, since the search space is restricted to goals in the clause, and is therefore much smaller. The hill-climbing stochastic algorithm for antecedent deletion is shown in Algorithm 35. Notice that delete-antecedent is also part of add-rule, with the latter using it in its first phase. The algorithm follows exactly the same random walk approach as previous algorithms seen in this section. First, it decides which type of move it is going to take, namely, a greedy move or a random move, based on a fixed probability p_{lg}. In case the move is greedy, it uses the original algorithm to propose deletions of antecedents. Otherwise, it selects at random a literal to be removed from the clause. Both cases require an improvement on the score to indeed remove a literal from the clause.

Algorithm 35 Algorithm for deleting antecedents using hill-climbing SLS

Input: A clause C, p_{lg}, the probability for deciding which move is going to be taken
Output: A (generalised) clause C

 1: **repeat**
 2:     $currentScore \leftarrow$ compute score of C;
 3:     $antes \leftarrow$ antecedents from the body of C;
 4:     **with** probability p_{lg} **do**
 5:         **repeat**
 6:             $ante \leftarrow$ an antecedent chosen at random from $antes$, whose removal from C still makes it valid relative to modes;
 7:             $C' \leftarrow C$ with $ante$ deleted from it;
 8:             $candidateScore \leftarrow$ compute score of C';
 9:             **if** $candidateScore > currentScore$ **then**
10:                 $C \leftarrow C'$
11:                 $currentScore \leftarrow candidateScore$
12:             **else**
13:                 $antes \leftarrow antes - ante$
14:         **until** $C = C'$ or $antes = \emptyset$
15:     **otherwise**
16:         **for each** antecedent $ante \in antes$ **do**
17:             $C' \leftarrow C$ with $ante$ deleted from;
18:             $candidateScore \leftarrow$ compute score of C';
19:         $bestClause \leftarrow$ candidate clause with the highest $candidateScore$
20:         **if** $candidateScore > currentScore$ **then**
21:             $C \leftarrow C'$
22:             $currentScore \leftarrow candidateScore$
23: **until** no antecedent can improve the score;
24: **return** C

A stochastic version of the delete multiple antecedents algorithm was also developed. However, this algorithm is executed only when the hill-climbing approach for deleting antecedents is not capable of deleting any antecedent. It is quite simple and it only differs from the original one by the stochastic decision introduced to decide which movement to make, greedy or random. A greedy move previews all possible combinations of literals from the body of the clause to choose the best among them, while a random move chooses the first combination improving the current score. Note that both cases only delete antecedents if after the deletion the clause continues to obey modes declarations. This algorithm is better visualised in 36.

Algorithm 36 Stochastic version of the algorithm for deleting multiple antecedents

---

1: **with** probability p_{lg} **do**
2:     generate all possible sequence of antecedents from the clause;
3:     choose a sequence at random;
4: **otherwise**
5:     choose a sequence with the highest score;

Stochastic Local Search for Revisions

YAVFORTE employs two specialization revision points, namely delete rule and add antecedent, and three generalisation revision points, namely delete antecedent, add rule (creating a rule by an existing one, by copying the original, deleting antecedents from it, followed by adding antecedents on it) and add new rule (creating a clause from scratch). Even employing all those revision operators, it may be the case that the set of revision points has only few components, either because the theory has only few failure points or because the stochastic search for revision points was applied and only a subset of the revision points is given back. In such a situation, the search space for selecting the operator which will be indeed responsible for modifying the current theory can be small enough to not largely influence the runtime of the revision process. However, this small search space does not always occur, making the choice of the revision operator be an important factor of cost during the revision process, since it is necessary to evaluate each possible revision yielded by the proposal

of each revision operator on each matching revision point.

In this way, the revision process can also benefit from stochastic search to reduce runtime when selecting the revision operator which will be the responsible for implementing some modification on the theory. Additionally, the revision can also take advantage of stochastic local search techniques to escape from local maxima. Aiming that goals, we designed four stochastic version of the top level algorithm of YAVFORTE. They differ mainly in the decision made to implement a revision, since in some algorithms a bad move can be executed. The algorithms are called as follows.

1. Stochastic greedy search with random walk

2. Stochastic hill-climbing search with random walk

3. Stochastic hill-climbing with stochastic escape

4. Simulated annealing search

Next we discuss each one of these algorithms.

Stochastic Greedy Search for Revisions with Random Walk

This approach follows Randomised Iterative Improvement technique and its most famous instance, namely WalkSAT algorithm, and accordingly, it performs random walks, alternating between greedy and stochastic moves, based on a fixed probability. In case this fixed probability is not reached, the algorithm performs a greedy move, by proposing all possible modifications on the theory though the application of each matching revision operator to each found revision point, in the same fashion the original algorithm does. Then, the best proposed revision is chosen to be implemented. Note that this move is greedy only, rather than greedy hill climbing as in original algorithm and therefore the score may be worse than the current one.

In case the fixed probability is reached, the algorithm performs a stochastic move, selecting at random a revision to be implemented from all the possible ones. There is no requirement on the selected revision: it is neither mandatorily the best possible revision nor it is required to improve current score. It is just a revision

proposed on a revision point by a matching revision operator. Because of that, it is not necessary to explicitly propose and evaluate all the possible revisions, since no assumptions are made on the score. Thus, it is enough to *enumerate* the possible revisions through the revision points and the revision operators suitable to be applied on each of those revision points. For each revision point, the list of possible revisions gets an entry containing a tuple with the revision point plus a matching revision operator. For instance, regarding a single specialisation revision point SRP and assuming both revision operators are employed in the revision process, there is going to be two tuples in the list: one containing SRP plus add-antecedent and another containing SRP plus delete-rule. The same is done for generalisation revision points. From such enumeration, a revision can be chosen at random and then be implemented.

Note that bad moves are always allowed in this greedy stochastic approach. On one hand, this ability makes the algorithm capable of always escaping from local maxima. On the other hand, some moves can conduct the theory to such a damaged state that the revision process will not be able to recover from it. However, modifications are performed on failure points and revision operators are designed to propose worthwhile revisions on those points, which makes the risk of really deteriorating the theory very small. To summarise, the key ideas of Stochastic Greedy Search with Random Walk algorithm are:

- **Composing the space of candidate hypothesis:** the search space is formed differently depending on the type of the move. If the move is greedy, the search space is composed of all possible proposals of revisions applied on each revision point by the matching revision operators. If the move is stochastic, the candidate hypothesis are also all the possible revisions, however, as one of them are going to be chosen at random, the candidates are not in fact proposals of revisions, but instead a representative tuple of the real revision. Each tuple contains the revision point and a possible revision operator to be applied on it.

- **Choosing the revision to be implemented:** In a greedy move, all revi-

sions composing the search space of candidate hypothesis are evaluated by an evaluation function and the best revision is indeed implemented on the current theory. In a stochastic move, a tuple representative of one real revision is chosen at random to be implemented. As the revision has not been already proposed before to be chosen as it happens in greedy moves, it is necessary to check if it is really possible to modify the theory with that revision. In other words, it may be the case the chosen revision cannot produce any modification in the theory. This is the case, for example, of trying to delete a rule and it is the last one for a top level predicate, or still if is not possible to delete or add antecedents on the clause. In such a situation, the procedure must choose another revision from the list until the list is empty. If it is possible to propose a revision from the tuple, then it is implemented on the theory.

- **Stop criteria:** The revision process stops under three circumstances: (1) when there are no more revisions to be implemented, (2) when the revision reaches a maximum score on the training set (for example, in case the evaluation function is accuracy, the maximum score could be 1.0) or (3) when the procedure implements a maximum number of revision (performs a maximum number of steps).

The procedure is exhibited in Algorithm 37 as a simplified and replacing version of Algorithm 16. By simplified version we mean some lines are omitted, specially those concerning the application of only required revision operators. The algorithm starts by computing the score of the current theory, since one of the stop criteria is the score reaches a maximum value. Next, it starts a loop which is going to stop according to the criteria established just above. Inside the loop, the first matter to worry about is finding revision points. Note that either the original algorithm or the stochastic algorithm may be used to this task, depending on the user preference. Then, as usual, the move must be chosen. In a stochastic move, it is necessary to find a random "implementable"revision. To do so, the revisions are enumerated considering each revision point and revision operators that are applicable on them. A revision is then selected at random and the system tries to implement in on the

current theory. If it is possible, the procedure proceeds to the next iteration. On the contrary, another revision is chosen at random, until one of them can indeed be implemented or there are no more possible revisions. In case the move is greedy, proposals of revisions are generated and scored in the same way the YAVFORTE top-level algorithm does. However, here the best revision is implemented even though the score is not improved.

Algorithm 37 Stochastic Greedy Search for Revisions with Random Walk

---

Input: An initial theory T, A Background Knowledge FDT, a set of examples E, integer $maxSteps$, real $maxScore$
Output: A revised theory T'

1: $score \leftarrow$ score of T
2: $steps \leftarrow 0$
3: **repeat**
4:      generate revision points
5:      **with** probability p_{rev} **do**
6:          $possibleRevisions$ possible revisions enumerated from the revision points and respective revision operators
7:          **repeat**
8:              $nextRevision \leftarrow$ a revision chosen at random from $possibleRevisions$
9:              $T' \leftarrow$ implements $nextRevision$
10:          **until** $T \neq T'$ or $possibleRevisions = \emptyset$
11:      **otherwise**
12:          generate all possible revisions from the revision points and respective revision operators
13:          compute score of each proposed revision
14:          $nextRevision \leftarrow$ revision with the highest score
15:          $T \leftarrow$ implements $nextRevision$
16:          $score \leftarrow$ score of T
17:      steps $++$;
18: **until** $score >= maxScore$ or $steps = maxSteps$ or T has not been modified

---

While Stochastic Greedy strategy is able to escape from local maxima, it is also a risky approach, since as we discussed before there is a chance of the theory deteriorates in one iteration and never recover again. Next section we present a second algorithm for selecting a revision to be implemented that is based on a Stochastic Hill-Climbing strategy, and therefore does always request a revision improving the current score.

Stochastic Hill-Climbing Search for Revisions with Random Walks

The Stochastic Hill Search with Random Walks algorithm, as the previous algorithm, also alternates between greedy and stochastic moves. However, However, whatever the move is, it is required an improvement in the score to go ahead. In this way, it is the most similar to the original algorithm comparing to all the approaches designed in this section. As usual, with a fixed probability p_{rev} the algorithm chooses at random a revision to be implemented from the list of possible revisions. Such a list is an enumeration of $revision\ point - operator$ tuples formed exactly in the same way that the previous discussed algorithm.

In order to guarantee the improvement in the score the revision chosen at random is evaluated according to some function and it is verified if its score is better than the current one. If so, the revision is implemented. If do not, another revision is chosen at random until the procedure finds a revision with a score better than the current one, or there are no more possible revisions to be implemented. The benefit in runtime brought by this approach relies on the fact that in random moves it is not necessary to explicitly propose and compute the score of all possible revisions. Proposing and computing all revisions in random moves will only happen in very unlikely cases where there is only one revision improving the score and it is the last to be chosen.

When the fixed probability p_{rev} is not reached, the move is greedy. As the approach is greedy hill climbing, it is necessary to propose and evaluate all the possible modifications on the theory and then to choose the best one, just as the original algorithm does. The revision process stops when there are no more revisions capable of improving the score. The key components of the strategy are as follows.

- **Composing the space of candidate hypothesis:** The set of candidate hypothesis is composed of the same elements as in the previous stochastic algorithm.

- **Choosing the revision to be implemented:** In a greedy move, all revisions composing the search space of candidate hypothesis are evaluated and the best

revision is indeed implemented on the current theory, only if it improves the current score. In a stochastic move, a tuple representative of one real revision is chosen at random to be implemented. As the revision has not been already proposed before to be chosen as it happens in greedy moves, it is necessary to check if it is really possible to modify the theory with that revision and also if it is able of improving the current score. In case a revision chosen at random is implementable and improves the score, it is indeed implemented. Otherwise, another revision must be chosen from the list of possible revision, until the list is empty.

- **Stop criteria:** The revision process stops under two circumstances: (1) when there are no more revisions to be implemented so that the score is improved or (2) when the revision reaches a maximum score on the training set.

The algorithm is exhibited as Algorithm 38 and as before we omit some obvious lines concerning the applicability of revision operators. It starts by computing the score of the current theory, since is is the base for verifying any improvement. Then, it performs an iterative hill climbing procedure, where at each iteration the revision points are generated and either a greedy or a stochastic move is accomplished, depending on a fixed probability p_{rev}. If a stochastic move is selected, then representatives of possible revisions are collected, encompassing each revision point together with each revision operator applicable to that point. An element from that collection is chosen at random and after proposing the selected revision in a temporary variable, the new score is computed. In case there is an improvement on the score, the proposed revision is accepted as next revision. On the contrary, it is required to pick up another revision, until finding one able of improving the score or giving up because there is no such a revision. If the move is chosen to be greedy, then, as in the original algorithm, the revisions are generated by a number of matching revision operators and scored with an evaluation function, until the maximum potential of a revision point is achieved by some revision. Afterwards, the revision with the highest score is selected and implemented in case the current score is improved with that revision. The procedure stops when the theory under revision

224

reaches a maximum score or when there are no revision capable of modifying the current theory so that the score is improved.

Algorithm 38 Stochastic Hill-climbing Search for Revisions with Random Walk

---

Input: An initial theory T, A Background Knowledge FDT, a set of examples E, integer $maxSteps$, real $maxScore$
Output: A revised theory T'

1:   $score \leftarrow$ compute score of T
2:   **repeat**
3:      generate revision points
4:      **with** probability p_{rev} **do**
5:         $possibleRevisions$ possible revisions enumerated from the revision points and respective revision operators
6:         **repeat**
7:            $nextRevision \leftarrow$ a revision chosen at random from $possibleRevisions$
8:            $T' \leftarrow T$ after implementing $nextRevision$
9:            $scoreNextRevision \leftarrow$ score of T'
10:           **if** $scoreNextRevision > score$ **then**
11:              $T' \leftarrow T$
12:              $score \leftarrow scoreNextRevision$
13:           **else**
14:              $possibleRevisions \leftarrow possibleRevisions - nextRevision$
15:         **until** $T' = T$ or $possibleRevisions = \emptyset$
16:      **otherwise**
17:         generate all possible revisions from the revision points and respective revision operators
18:         compute score $scoreNextRevision$ of each proposed revision
19:         $nextRevision \leftarrow$ revision with the highest score
20:         **if** $scoreNextRevision > score$ **then**
21:            $T \leftarrow$ implements $nextRevision$ on T
22:            $score \leftarrow scoreNextRevision$
23: **until** $score = maxScore$ or T has not been modified

Next section we present an intermediate strategy between both of the stochastic algorithms discussed here. It is able to perform bad moves aiming to escape from local maxima, but only under a stochastic move.

Hill-Climbing Search for Revisions with Stochastic Escapes

The stochastic greedy algorithm first presented here may deteriorate the theory since it always allows bad moves to be performed. The previous algorithm, on the other hand, never allows a bad move, which can not avoid it gets stuck in local maxima.

Besides, even when the move is stochastic it has to search for a revision improving the score, adding an extra factor of cost, compared to the former algorithm. Aiming to overcome those issues, a third stochastic version of the algorithm 16 is designed on this section. The strategy exploited here is to try escape from local maxima when performing a stochastic move and implement the best candidate otherwise. Thus, as usual, with a certain probability p_{rev}, the algorithm performs a stochastic escape, choosing a revision to be implemented at random even if its score is not better than the current one. On the other hand, when the move is greedy, this algorithm proceeds as originally, selecting the revision with the highest score and demanding such a score is better than the current one. If there is no such a revision, the algorithm continued to the next iteration in an attempt to reach the probability p_{rev} and then to perform a stochastic move. The procedure stops when it reaches a maximum number of steps or when it reaches a maximum score. As before, we summarise the key components.

- **Composing the space of candidate hypothesis:** The set of candidate hypothesis is composed of the same elements as in the previous stochastic algorithms.

- **Choosing the revision to be implemented:** In a greedy move, it behaves as the greedy component of Stochastic Hill-Climbing and hence the original revision algorithm. Thus, all revisions composing the search space of candidate hypothesis are evaluated and the best revision is implemented on the current theory, only if it improves the current score. However, in case no revision improves the score, instead of terminating like those algorithms, Hill-climbing with Stochastic Escape continues to the next iteration. In a stochastic move, it may implement a revision with a bad score, but only if it does not degradate that much the score. To decide how much degradation it is allowed, we use a function based on a pertubation strategy (HOOS, STÜTZLE, 2005), defined as $score + 0.5 * (Potential + score)$. Potential is the number of examples that indicated the need of revising a point and therefore is the maximum score that a revision could have.

226

- **Stop criteria:** The revision process stops under two conditions, namely: (1) when the revision reaches a maximum score or (2) when the procedure performs a maximum number of steps.

Algorithm 39 exhibits the Hill-climbing with Stochastic Escape procedure. The first component of the main loop, which is executed when the move is stochastic, chooses a revision at random. In case its score obeys the "perturbation"constraint, it is implemented. Otherwise, the algorithm proceeds to a next iteration. The second component, the greedy move, behaves as the Stochastic Hill-climbing. Nevertheless, the algorithm has a stopping criteria differing from previous algorithms, as it stops only when a maximum score is reached or when a maximum number of iterations is executed. Such a criteria consents the procedure proceeds to a next iteration without implementing any revision, which happens when the greedy move cannot find any revision capable of improving the current score or a randomized revision degradating the score more than it is allowed.

Finally, we would like to have an algorithm which does not take greedy decisions and also accepts bad moves under a certain condition. As this idea is quite similar to what is achieved with *Simulated annealing* techniques, we implemented a version of this strategy on the base of the top-level revision algorithm.

Simulated Annealing Search for Revisions

Simulated annealing chooses at each iteration a candidate hypothesis at random and in case the current score is not improved, it uses a scheduling function to decide if such a hypothesis can be accepted as next hypothesis. As simulated annealing always selects a revision at random, it can be arguably faster than the previous algorithms developed in this section, since it never evaluates all the possible modifications on the theory. However, it can take more iterations to converge. It is proved that, if the value returned by the scheduling function is reduced slowly enough, the algorithm will find the optimal global (RUSSELL, NORVIG, 2010). Here we follow exactly the same idea and implement the simulated annealing as one of the stochastic algorithms for selecting revision to be implemented. The key components of the algorithm are as follows.

Algorithm 39 Hill-Climbing Search for Revision with Stochastic Escape

Input: An initial theory T, A Background Knowledge FDT, a set of examples E, integer $maxSteps$, real $maxScore$

Output: A revised theory T'

1:   $score \leftarrow$ compute score of T
2:   $steps \leftarrow 0$
3:   **repeat**
4:      generate revision points
5:      $possibleRevisions$ possible revisions enumerated from the revision points and respective revision operators
6:      **with** probability p_{rev} **do**
7:        **repeat**
8:          $nextRevision \leftarrow$ a revision chosen at random from $possibleRevisions$
9:          $score' \leftarrow$ compute score of $nextRevision$
10:          **if** $score' + 0.5(Potentil + score') > 0$ **then**
11:            $T' \leftarrow$ implements $nextRevision$
12:          **else**
13:            $T' \leftarrow T$
14:            $possibleRevisions \leftarrow possibleRevisions - nextRevision$
15:        **until** $T \neq T'$ or $possibleRevisions = \emptyset$
16:      **otherwise**
17:        generate all possible revisions from the revision points and respective revision operators
18:        compute score $scoreNextRevision$ of each proposed revision
19:        $nextRevision \leftarrow$ revision with the highest score
20:        **if** $scoreNextRevision > score$ **then**
21:          $T \leftarrow$ implements $nextRevision$ on T
22:          $score \leftarrow scoreNextRevision$
23:      $steps + +$
24: **until** $score >= maxScore$ or $steps = maxSteps$

- **Defining the search space of candidate hypothesis:** Candidate hypothesis are exactly the same of the stochastic components of previous algorithms. Thus, the candidate hypothesis are all the possible revisions applied to each revision provided revision point, but without proposing them to compose the search space. Instead, they are representative tuples of each real revision, where each tuple is a revision point and a possible revision operator to be applied on it.

- **Choosing the revision to be implemented:** The revision to be implemented is chosen at random from the search space of candidate hypothesis. A proposal of the revision is generated and its score is computed. If such a score is better than the current one, the revision is implemented on the current theory. Otherwise, the revision is implemented only if a certain probability is reached. This probability is defined from the difference between the current score and the score of the revision, divided by the value computed with the scheduling function. The scheduling function, among others parameters, takes into account the number of steps performed so far. It may be the case that an iteration gets to the end without implementing any revision, precisely when a revision got at random neither can improve the current score, nor it is acceptable as a bad move.

- **Stop criteria:** The revision process stops under three conditions, namely: (1) when there are no possible revision to be implemented or (2) when the scheduling function returns zero or (3) when the theory achieves a maximum score.

Algorithm 40 brings Simulated strategy performed on the search for revisions. Besides the usual parameters, it requires a parameter indicating the maximum number of iterations *limit* and a reduction factor *lam*. The main loop starts by generating the revision points as usual, followed by the enumeration of tuples of possible revisions to be implemented upon such revision points. As the revisions are not really proposed before one of them has been chosen, it is necessary to check out whether it is possible to implement that revision on the theory. In possession of

an implementable revision, the algorithm must decide whether it is going indeed be implemented on the current theory. To do so, first it is verified if the score can be improved after implementing the revision, which in an affirmative case, makes the revision be implemented. If the score is not improved the revision might be implemented anyway, but only when a scheduling function returns a value higher than a random probability. The scheduling function guarantees that the more iterations the algorithm performs the less is the chance of a bad move be selected. The algorithm may proceed without implementing any revision on that iteration and it finishes the revision process when the scheduling function return 0.0.

Algorithm 40 Simulated Annealing Search for Revisions

Input: An initial theory T, A Background Knowledge BK, a set of examples E
Output: A revised theory T'

1: $score \leftarrow$ score of T
2: $S \leftarrow 1$
3: **repeat**
4:     generate revision points
5:     $T' \leftarrow T$
6:     $possibleRevisions$ possible revisions enumerated from the revision points and respective revision operators
7:     **repeat**
8:         $nextRevision \leftarrow$ a revision chosen at random from $possibleRevisions$
9:         **if** it is possible implement $nextRevision$ on T **then**
10:             $T' \leftarrow$ implements $nextRevision$
11:         **else**
12:             $possibleRevisions \leftarrow possibleRevisions - nextRevision$
13:     **until** $possibleRevisions = \emptyset$ or $T' \neq T$
14:     **if** $T' \neq T$ **then**
15:         $scoreNextRevision \leftarrow$ compute score of T'
16:         $\Delta E \leftarrow scoreNextRevision - score$
17:         **if** $\Delta E > 0$ **then**
18:             $T \leftarrow T'$
19:             $score \leftarrow scoreNextRevision$
20:         **else**
21:             $S \leftarrow$ scheduling$(t, limit, lam)$
22:             **if** $S \neq 0.0$ **then**
23:                 **with** probability $e^{\Delta E/S}$ **do**
24:                     implements $nextRevision$
25:                     $score \leftarrow scoreNextRevision$
26:     $t + +$
27: **until** $S = 0$

Experimental Results

In this chapter, we would like to investigate if it is possible to reduce runtime of the revision process, even when the search space is larger than usual, by the use of Stochastic Local Search techniques. Thus, the major question we would like to answer is whether the runtime of the revision process guided by SLS algorithms, can be faster than the traditional revision but without harming accuracy. A secondary question is if we can also be comparable to learning from scratch in terms of learning time, while reaching better accuracies than this approach. In this way, we have selected three ILP datasets that YAVFORTE had a hard time to revise and compared accuracy and runtime obtained from Aleph (), YAVFORTE (see chapter 3) and stochastic algorithms. The datasets are as follows.

Datasets

- Pyrimidines: this is a Quantitative Structure Activity Relationships (QSAR) problem, concerning the inhibition of E. Coli Dihydrofolate Reductase by *pyrimidines*, which are antibiotics acting by inhibiting Dihydrolate Reductase, an enzyme on the pathway to forming DNA (KING, et al., 1992; HIRST, et al., 1994a). The dataset we used in this work is composed of 2361 positive examples and 2361 negative examples.

- Proteins: This is a task of secondary structure protein prediction. The task is to learn rules to identify whether a position in a protein is in an alpha-helix (MUGGLETON, et al., 1992). We considered a dataset with 1070 positives and 970 negatives.

- Yeast_sensitivity (SPELLMAN, et al., 1998; KADUPITIGE, et al., 2009): This is a dataset concerning the problem of gene interaction of the *yeast Saccharomyces cerevisiae*. It is composed of 430 positive examples and 680 negative examples. It has a huge background with approximately 170,000 facts.

Experimental Methodology   The datasets were splitted up into 10 disjoint folds sets to use a K-fold stratified cross validation approach. Each fold keeps the rate

of original distribution of positive and negative examples (KOHAVI, 1995). The significance test used was corrected paired t-test (NADEAU, BENGIO, 2003), with $p < 0.05$. As stated by (NADEAU, BENGIO, 2003), corrected t-test takes into account the variability due to the choice of training set and not only that due to the test examples, which could lead to gross underestimation of the variance of the cross validation estimator and to the wrong conclusion that the new algorithm is significantly better when it is not. All the experiments were run on Yap Prolog (SANTOS COSTA, 2008).

The initial theories were obtained from Aleph system using parameters default, except for clause length, which is defined as 10 , noise, defined as 30, nodes set to 10000 and minpos, set to 2. The use of those parameters has been inspired on the work of (MUGGLETON, et al., 2010). To generate such theories, the whole dataset was considered but using a 10-fold cross validation procedure. Thus, a different theory was generated for each fold and each one of these theories is revised considering its respective fold (the same fold is used to generate and revise the theories). Theories returned by Aleph have about 200 clauses for each dataset, which makes them difficult for YAVFORTE to revise. Stochastic algorithms were run 5 times because of the random choices.

Behavior of the Stochastic Local Search Algorithms with Different Parameters

First, we would like to observe how the different stochastic strategies behave with different parameters. To do so, we used the datasets Pyrimidines and Proteins and plot curves for each individual algorithm, with different appropriate parameters.

Varying Number of Revision Points   We start by comparing the revision time and accuracy results of Algorithm 6.1, which includes a stochastic component when searching for revision points, with different amounts of maximum revision points returned. Figures 12 and 13 exhibit the results for Pyrimidines and Proteins, respectively, with probability parameters fixed to 100% and 50% and number of revision points defined as 1, 5, 10 and 20. As expected, runtime increases as more revision points are returned. However, accuracies are not significantly different when 5 or

more revision points are found out. Accordingly, choosing only one revision point makes the revision system extremely fast, but at the expense of worse accuracies. Accuracies are not significantly different when the probability is either 100% or 50%.
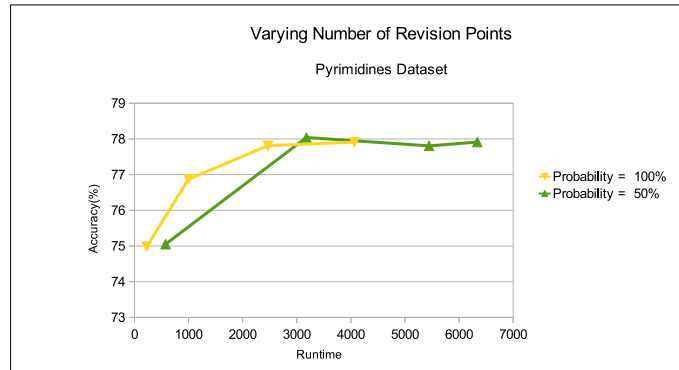

Figura 12: Comparing runtime and accuracy of Algorithm 6.1 in Pyrimidines Dataset, with number of revision points varying in 1, 5, 10, 20. Probabilities are fixed in 100% and 50%.
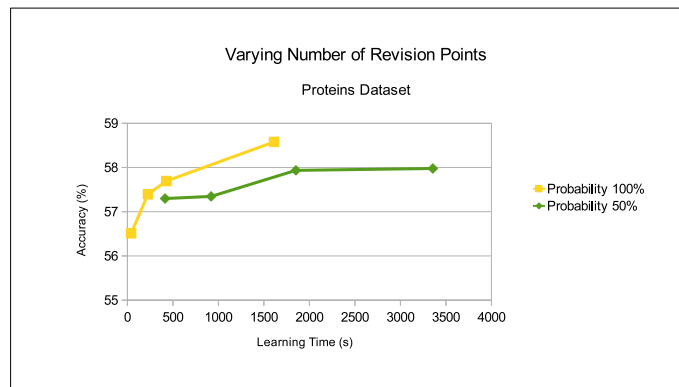

Figura 13: Comparing runtime and accuracy of Algorithm 6.1 in Proteins Dataset with number of revision points varying in 1, 5, 10, 20. Probabilities are fixed in 100% and 50%.

Varying Number of Iterations   Algorithms Stochastic Greedy(6.6) and Hill Climbing with stochastic Escape (6.8) considers as stop criteria a maximum number of iterations. Simulated annealing also takes into account a limit to decide stopping, which is the parameter *limit* in Algorithm(6.9). In order to check the performance

of these approaches when facing different maximum iterations, we fixed the probability of random walks in 100% and 50% for the two first cases and plotted the accuracy versus runtime achieved by the algorithms. The graphics are exhibited in Figures 14 and 15, for Pyrimidines and Proteins datasets, respectively.

In the Pyrimidines dataset, Simulated annealing is the fastest algorithm but it only achieves accuracy equivalent to the other algorithms when the number of iterations is 40. We perform some additional tests to see if by increasing the number of iterations, simulated annealing would perform better, but we only verified higher runtime, while accuracies were stationed.

Fixing the probabilities as 100% for the two others algorithms makes them to execute very fast. However, always to perform stochastic moves does not improve accuracies of the theories as it could. We see that by looking at the performance of these algorithms when the probability is fixed in 50%: although the revision process takes more time, since in greedy moves all revisions are evaluated so that the best one is chosen, the accuracies are also significantly higher. Hill Climbing stochastic escape algorithm with 50% of probability is the algorithm achieving best accuracies, in the same runtime as the second best algorithm, which is the Greedy Stochastic algorithm, also considering 50% of probability.

In the proteins dataset, it is interesting to see that the best accuracy was achieved by Simulated Annealing in less time than most of the cases. All the other algorithms behaves as the Pyrimidines dataset: Considering probabilities of 50% yields better accuracies, achieved in slower revision time. However, Hill climbing stochastic escape with 100% in this case is also able to achieve accuracies statistically equivalent to the 50% cases, in less time.

Those curves suggest that when increasing the number of iterations the systems are slower, as expected, but their accuracies are not significantly changed.

Varying Probability Values   Probability parameters are responsible for deciding the type of the move the algorithm is going to follow: either the move is greedy, and the best hypothesis found from the set of all generated hypothesis, is chosen to be implemented, or the move is stochastic and possible hypothesis are randomized. All
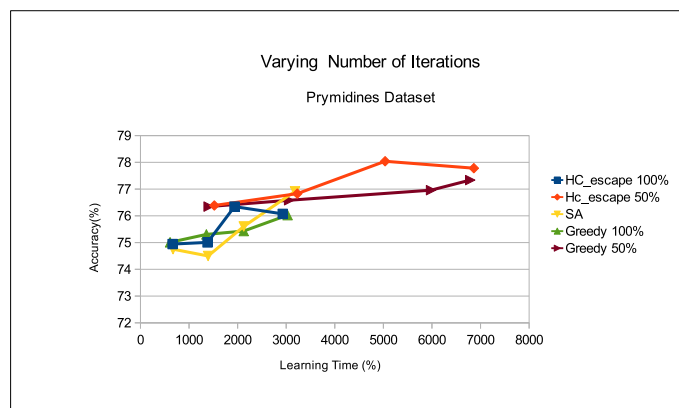
Figura 14: Comparing runtime and accuracy of SLS algorithms in Pyrimidines Dataset, varying maximum number of iterations, which is set to 10, 20, 30, 40. Probabilities are fixed in 100% and 50%, when it is the case.
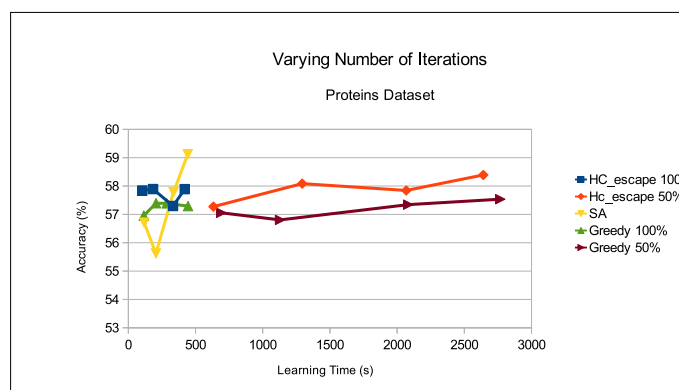

Figura 15: Comparing runtime and accuracy of SLS algorithms in the proteins dataset, varying maximum number of iterations, which is set to 10, 20, 30, 40. Probabilities are fixed in 100% and 50%, when it is the case.

algorithms but Simulated Annealing follow this strategy. To see the performance of the algorithms when facing different probability parameters, we plot accuracy versus runtime curves for each algorithms, considering probabilities as 100%, 80%, 60% and 40%. Maximum number of iterations for Hill climbing with stochastic escape and Stochastic Greedy algorithms for choosing revisions are set to 20, since in previous section we see this value has the best balance between accuracy and runtime. Maximum number of revision points is set to 10.

In Pyrimidines dataset, the performance of Greedy and Stochastic Escape

235

algorithms follow the same pattern: the smaller is the probability, higher accuracies are achieved at the expense of higher runtime. However, while Stochastic Escape can reach accuracy very close to the other algorithms, in the best case, accuracies of Greedy algorithm are always significantly slower than the other algorithms. The accuracies achieved by the rest of the algorithms slightly changes, but they are not significantly different, no matter the probability value is. On the other hand, the smaller is the chance of choosing a stochastic move, the slower is the revision process. It is interesting that in these cases, even always following a stochastic move (probability set to 100%), accuracies are not significantly affected. Thus, we can conclude from this dataset that, if the stochastic move does not demands an improvement on the score, which is the case of Greedy and Stochastic Escape, probabilities play a fundamental role. On the other hand, if both moves require improvement on the score (stochastic Hill climbing), high probabilities can yield the same results as low probabilities, but in less time.

Proteins dataset has similar results. In this case, the stochastic hill climbing search for revisions and stochastic search for literals achieve the best accuracy results, but at the expense of higher runtime.
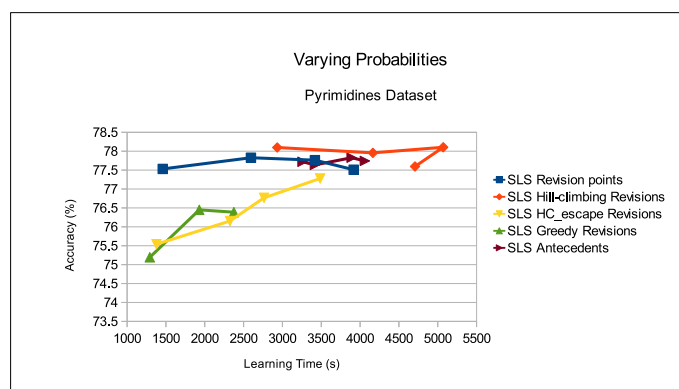

Figura 16: Comparing runtime and accuracy of SLS algorithms in Pyrimidines Dataset, with probabilities varying in 100, 80, 60, 40. Maximum number of iterations is fixed in 20, when it is the case.
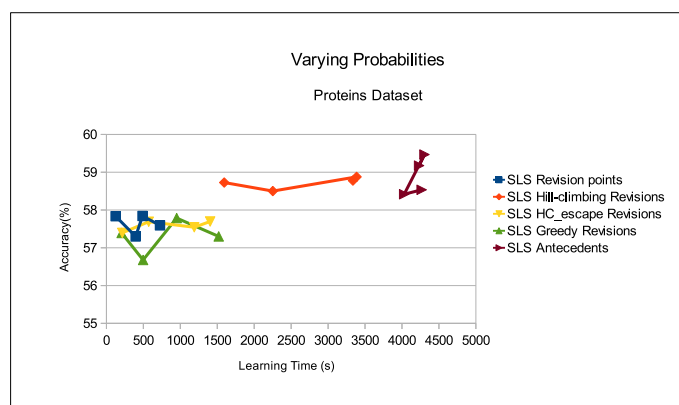
Figura 17: Comparing runtime and accuracy of SLS algorithms in Proteins Dataset, with probabilities varying in 100, 80, 60, 40. Maximum number of iterations is fixed in 20, when it is the case.

Comparing Runtime and Accuracy of SLS Algorithms to Aleph and YAVFORTE

In this section we would like to compare the performance of Stochastic Local Search Algorithms to state-of-the-art learning from scratch and revision systems. We compare the accuracy and runtime of Aleph and YAVFORTE (see chapter 3) to the SLS algorithms presenting a better trade-off between accuracy and runtime. In addition to Pyrimidines and Proteins, we also consider here the Yeast sensitivity dataset, but in this case, concerning the search for revisions, we show the results only for stochastic Hill climbing, since this has the best accuracy results for the other two datasets. Besides running individually the stochastic algorithms for each key search, we also combine stochastic algorithms as follows.

1. Stochastic search for revision points with stochastic search for literals.

2. Stochastic search for revision points with stochastic Hill climbing search for revisions.

3. Stochastic Hill climbing search for revisions with stochastic search for literals.

4. Stochastic search for revision points with stochastic search for literals and stochastic Hill climbing search for revisions.

Figures 18, 19 and 19 presents the accuracies returned by the systems for Pyrimidines, Proteins and Yeast sensitivity datasets, respectively. The parameters used to each system are presented in parenthesis, where the first value is a probability parameter and the second value (when it is the case) it is the number of iterations/revision points. Next, we discuss the most relevant cases observed in the graphic.

- All revision algorithms in the three datasets achieve accuracies significantly better than Aleph system.

- Except for Greedy Search for revisions, in Pyrimidines dataset there is no statistical significant difference between the accuracy returned by YAVFORTE and the stochastic algorithms.

- Best accuracy result for Pyrimidines is achieved by stochastic search for literals combined with Hill climbing stochastic search for revisions, although it is not significantly better than the others.

- In Proteins dataset, best results are achieved by stochastic search for literals and stochastic search for literals combined with Hill climbing stochastic search for revisions.

- In Proteins dataset, stochastic search for revision points and stochastic greedy search for revisions achieves significantly worse accuracies, compared to YAV-FORTE and best SLS results.

- There is no significant difference between YAVFORTE and the rest of the SLS algorithms.

- In Yeast Sensitivity dataset, stochastic search for literals and the combination of the three stochastic components achieve significantly better accuracies than YAVFORTE.

- The other SLS algorithms do no present significant difference compared to YAVFORTE

238

From these results, we can see that SLS algorithms either provide better or equivalent accuracies compared to the baseline revision system YAVFORTE. In addition, they are always significantly better than Aleph system.
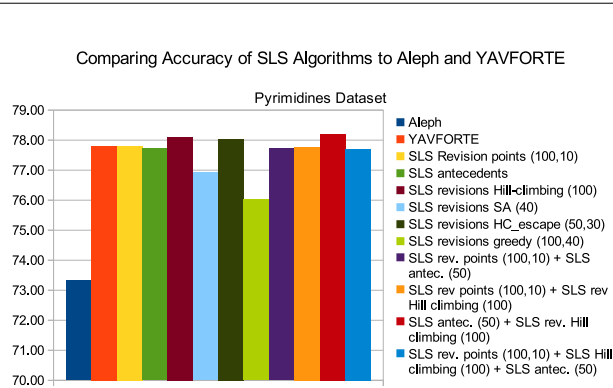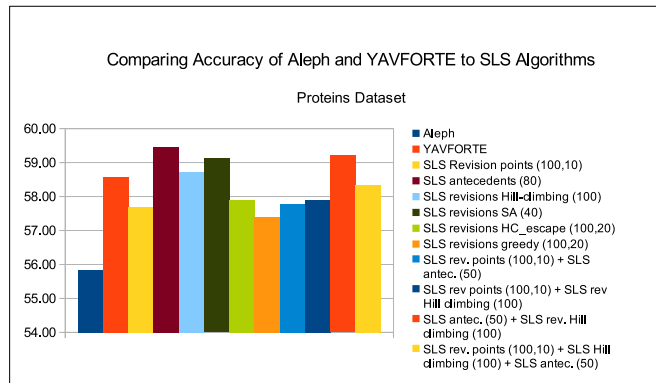

Figura 18: Comparing accuracy of SLS algorithms to Aleph and YAVFORTE in Pyrimidines Dataset. Results of SLS algorithms are the ones with a best trade-off between accuracy and runtime. Parameters of probability, number of iterations and number of revision points are in parenthesis.

Figura 19: Comparing accuracy of SLS algorithms to Aleph and YAVFORTE in Proteins Dataset. Results of SLS algorithms are the ones with a best trade-off between accuracy and runtime. Parameters of probability, number of iterations and number of revision points are in parenthesis.

Figures 21, 22 and 23 exhibit runtime of Aleph, YAVFORTE and SLS algorithms for Pyrimidines, Proteins and Yeast sensitivity, respectively. The following issues are observed from the results graphically represented in the figures.
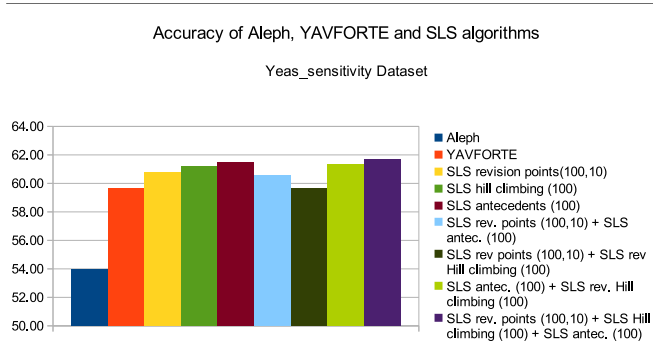
239

Figura 20: Comparing accuracy of SLS algorithms to Aleph and YAVFORTE in Yeast Sensitivity Dataset. Results of SLS algorithms are the ones with a best trade-off between accuracy and runtime. Parameters of probability, number of iterations and number of revision points are in parenthesis.

- Pyrimidines Dataset

    - Revision time of YAVFORTE is significantly slower than all other algorithms in Pyrimidines dataset.

    - In Pyrimidines dataset, the only algorithm significantly slower than Aleph is SLS with revisions escape. This is likely due to the high number of iterations.

    - Stochastic Hill climbing search for revisions combined with stochastic search for literals does not present difference compared to Aleph. In this case, the combination of these SLS algorithms takes more time to converge.

    - All the other SLS cases are significantly faster than Aleph.

    - Higher speed of a SLS algorithm, compared to YAVFORTE has a factor of 17, while smallest speed up is 4.

- In Proteins dataset all SLS algorithms are significantly faster than Aleph and YAVFORTE, except for stochastic search for literals, which is faster than YAVFORTE but it is not significantly different than Aleph. Higher speed of a SLS algorithm, compared to YAVFORTE has a factor of 16, while smallest speed up is 2.

- In Yeast sensitivity, Aleph is significantly faster than all revision algorithms. We believe this is due to the huge background knowledge this dataset has, that makes coverage tests slower and revision must perform much more such kind of tests than learning from scratch approaches. On the other hand, all SLS algorithms are significantly faster than YAVFORTE. Higher speed of a SLS algorithm, compared to YAVFORTE has a factor of 26, while smallest speed up is 2.
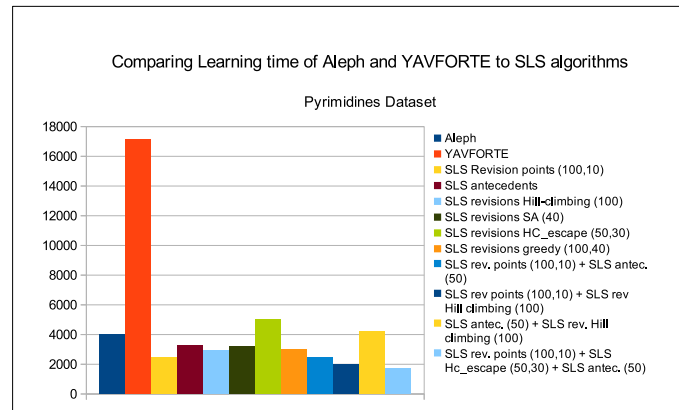

Figura 21: Comparing runtime of SLS algorithms to Aleph and YAVFORTE in Pyrimidines Dataset. Results of SLS algorithms are the ones with the best trade-off between accuracy and runtime. Parameters of probability, number of iterations and number of revision points are in parenthesis.

From the results, we can conclude that using stochastic local search algorithms individually and specially combining them, it is possible to greatly reduce the revision time and also to be faster or competitive with learning from scratch. Moreover, accuracies achieved by SLS strategies are always better than learning from scratch approach and competitive with traditional revision approach. Thus, we positively answer both questions posed in the beginning of this section.

Conclusions

In this chapter we designed a set of stochastic local search algorithms for exploring the key search spaces of the revision process more efficiently. The algorithms abandon completeness in favor of finding good solutions in a reasonable time. Most of
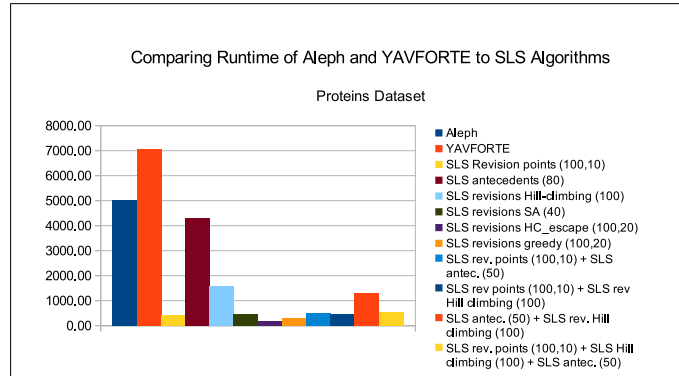
Figura 22: Comparing runtime of SLS algorithms to Aleph and YAVFORTE in Proteins Dataset. Results of SLS algorithms are the ones with the best trade-off between accuracy and runtime. Parameters of probability, number of iterations and number of revision points are in parenthesis.
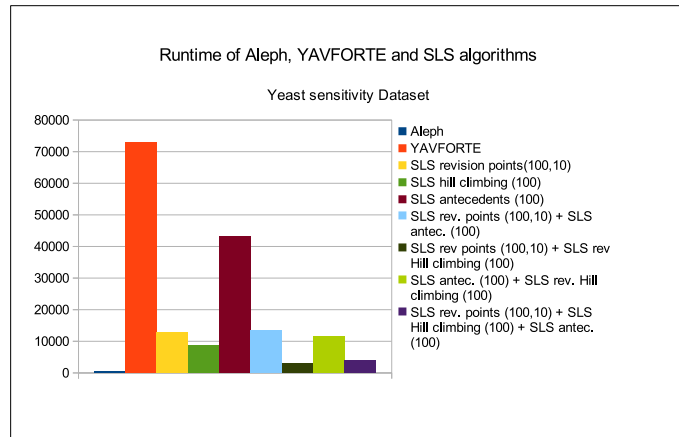

Figura 23: Comparing runtime of SLS algorithms to Aleph and YAVFORTE in Yeast Sensitivity Dataset. Results of SLS algorithms are the ones with the best trade-off between accuracy and runtime. Parameters of probability, number of iterations and number of revision points are in parenthesis.

all are based on random walks, so that the choice of pursuing a greedy or a stochastic move is made according to a probability parameter. Stochastic algorithms were implemented in YAVFORTE system (see chapter 3 and (DUBOC, et al., 2009)) in every key search of the revision process.

First, a SLS algorithm was built to avoid collecting all revision points from all misclassified examples. With a probability p, misclassified examples are randomized

and a pre-defined number of revision points is generated. The search is alternated with greedy moves, since without the probability p all revision points found in the theory from each misclassified instance are collected. However, through experimental results we found out that in several cases there is no need of employing greedy moves, since defining the probability parameter as 100% already achieves good results: the revision time is greatly reduced and accuracies are not statistically significant different compared to the baseline system. The performance of stochastic component in the search for revision points is more influenced by the parameter defining the number of revision points that should be returned.

Second, SLS components were included in the search for literals to be added to or removed from a clause. Stochastic search was included in both hill climbing and relational pathfinding algorithms for specializing clauses. In the first case, when the move is stochastic, literals from the Bottom Clause are randomized and the first literal found that improve the score is added to the clause. In relational pathfinding, paths created from the Bottom Clause and a positive instance are randomized. In this work, we used both algorithms cooperating to each other, when relational pathfinding was applicable. We noticed from the empirical evaluation that, although the stochastic search for literals is able to reduce runtime compared to hill climbing greedy approach of YAVFORTE, if it is executed without the other stochastic components, it is not much effective as the other SLS algorithms. The reasons for that are mainly due to the Bottom clause: either it is small and the stochastic component does not make a huge difference, or it is large, but in this case the use of stochastic move takes more iterations to converge than the original approach. Novel approaches can be investigated to further improve this stochastic component, such as pre-process literals using Bayesian networks or genetic algorithms (OLIPHANT, SHAVLIK, 2008; MUGGLETON, TAMADDONI-NEZHAD, 2008; PITANGUI, ZAVERUCHA, 2011).

Third, four different stochastic components were included to decide which revision is going to be implemented. Three of them are based on random walks and one of them performs a simulated annealing algorithm. In the results we could see that the stochastic greedy algorithm for searching revisions, which in both stochastic

and greedy moves allows a revision with score worse than the current one be implemented, does not performs well. As an improvement in score is always not required, the accuracy deteriorates along the iterations. The stochastic escape approach, that accepts bad moves but only if it does not degrade so much the score, performs better than greedy, with good accuracy and runtime results in several cases. However, the simplest strategy, that with a certain probability randomizes revisions and implements the first one improving the score, achieved the best overall results.

The three strategies above were compared both individually and combining the best strategies. The vast majority of the cases showed that stochastic approaches achieve better accuracies than the learning from scratch Aleph system, with faster or competitive runtime. Moreover, accuracies in most of the cases are significantly equivalent to YAVFORTE but the runtime is always much faster. In the best case, an SLS algorithm is $25X$ faster than YAVFORTE, with equivalent accuracy.

We see at least two different topics for further investigation concerning SLS in first-order logic revision: the use of rapid randomized restart strategies, already employed in ILP in (ŽELEZNÝ, et al., 2004; ŽELEZNÝ, et al., 2006), that would avoid being stuck in unproductive refinements by restarting after a certain criteria from another random point; and the use of stochastic component in coverage tests, as it is done in (SEBAG, ROUVEIROL, 1997; KUZELKA, ZELEZNÝ, 2008b).

Appendix F: Probabilistic Logic Learning

Traditional statistical machine learning algorithms deal with uncertainty by assuming the data are independent and identically distributed (i.i.d). On the other hand, relational learning algorithms have the capability to represent multi-typed related objects, but they impose severe limitations for representing and reasoning in the presence of uncertainty. However, in most real-world applications, data is multi-relational, heterogeneous, uncertain and noisy. Examples include data from web, bibliographic datasets, social network analysis, chemical and biological data, robot mapping, natural language, among others (LACHICHE, FLACH, 2002; BATTLE, et al., 2004; GETOOR, et al., 2004; JAIMOVICH, et al., 2005; DAVIS, et al., 2005a; NEVILLE, et al., 2005; WANG, DOMINGOS, 2008; RAGHAVAN, et al., 2010). Therefore, in order to extract all useful information from those datasets it is necessary to use techniques dealing with multi-relational representations and probabilistic reasoning. *Probabilistic Logic Learning (PLL)*, also called *Probabilistic Inductive Logic Programming (PILP)* (DE RAEDT, et al., 2008a) and *Statistical Relational Learning (SRL)* (GETOOR, TASKAR, 2007) is an emerging area of artificial intelligence, lying at the intersection of reasoning about uncertainty, machine learning and logical knowledge representation (DE RAEDT, 2008). As such, PLL is able to deal with machine learning and data mining in complex relational domains where information may be missed, partially observed and/or noisy.

A large number of PLL systems have been proposed in the last years, giving rise to several formalisms for representing logical and probabilistic knowledge. The formalisms can be divided into several general classes (GETOOR, 2007), yet here we choose to put them in the two more relevant axes (DE RAEDT, et al.,

2008a): *Logical Probabilistic Models (LPM)* are extensions of probabilistic models that are able to deal with objects and relations by including logical or relational elements. Typically, they build a directed or undirected probabilistic graphical model to reasoning about uncertainty using logic as a template. Inside this category there is a group of formalisms based upon directed probabilistic graphical models, composed of Relational Bayesian Networks (JAEGER, 1997), Probabilistic Logic Programs (NGO, HADDAWY, 1997; HADDAWAY, 1999), Probabilistic Relational Models (PRM) (KOLLER, PFEFFER, 1998; KOLLER, 1999; FRIEDMAN, et al., 1999; GETOOR, et al., 2001), *Bayesian Logic Program (BLP)* (KERSTING, DE RAEDT, 2001d; KERSTING, DE RAEDT, 2001b; KERSTING, DE RAEDT, 2001a; KERSTING, DE RAEDT, 2007), *Constraint Logic Programming with Bayes Nets* (CLP(BN)) (SANTOS COSTA, et al., 2003a), Hierarchical Bayesian Networks (GYFTODIMOS, FLACH, 2004), Logical Bayesian Networks (FIERENS, et al., 2005), Probabilistic Relational Language (GETOOR, GRANT, 2006), etc, and a group composed of systems built upon undirected probabilistic graphical models, including Relational Markov Networks (TASKAR, et al., 2002), Relational Dependency Networks (NEVILLE, JENSEN, 2004) and *Markov Logic Networks* (MLN) (SINGLA, DOMINGOS, 2005; KOK, DOMINGOS, 2005; RICHARDSON, DOMINGOS, 2006; DOMINGOS, LOWD, 2009).

In the other axis are the *Probabilistic Logical Models (PLM)*, which are formalisms extending logic programs with probabilities, staying as close as possible to logic programming by annotating clauses with probabilities. In this class, the logical inference is modified to deal with the parameters of probabilities. Formalisms following this approach include Probabilistic Horn Abduction (POOLE, 1993) and its extension *Independent Choice Logic* (ICL) (POOLE, 1997), *Stochastic Logic Program (SLP)* (MUGGLETON, 1996; MUGGLETON, 2000; MUGGLETON, 2002), PRISM (SATO, KAMEYA, 1997; SATO, KAMEYA, 2001), Logic Programs with Annotated Disjunctions (VENNEKENS, et al., 2004), SAYU (DAVIS, et al., 2005a; DAVIS, et al., 2005b; DAVIS, et al., 2007), nFoil (LANDWEHR, et al., 2007), kFoil (LANDWEHR, et al., 2006), *ProbLog* (DE RAEDT, et al., 2007; KIMMIG, et al., 2008; KIMMIG, 2010), among others.

The present work aims to contribute to the rule-based formalism using the directed probabilistic graphical model of Bayesian networks. BLP was chosen to conduct the research of theory revision in PLL since it elegantly instantiates both logic programs and Bayesian networks. Next we bring some details on the fundamentals of BLPs, starting from Bayesian Networks in section 24, immediately followed by the ideas behind BLPs in section 5.3.2. Finally, we review our revision system PFORTE in section 5.3.2.

For some basic concepts from probability theory we refer to (FELLER, 1970; ROSS, 1988; PEARL, 1988; DEGROOT, 1989).

Bayesian Networks: Key Concepts

Complex systems involving some sort of uncertainty may be characterized through multiple interrelated random variables, where the value of each variable defines an important property of the domain. In order to probabilistically reason about the values of one or more variables, possibly given evidence about others, it is necessary to construct a joint distribution over the space of possible assignments to a set of random variables. Unfortunately, even in the simplest case of binary-valued variables, the representation of a joint distribution over a set of not assumed independent random variables $\chi = X_1, ..., X_n$ requires the specification of the probabilities of at least 2^n different assignment of values $x_1, ..., x_n$. It is obviously unmanageable to explicitly represent such a joint distribution. *Probabilistic Graphical Model*s (PGMs) provide mechanisms for encoding such high-dimensional distributions over a set of random variables, structuring them compactly so that the joint distribution can be utilized effectively (KOLLER, FRIEDMAN, 2009). They use a graph-based representation, where the nodes correspond to the random variables in the domain and the edges correspond to direct probabilistic interactions between the variables. There are two most common used families of PGM, one representing the domain through undirected graphs and the other through directed graphs. In the first case lies the *Markov networks* (aka *Markov random field*), consisting of an undirected graph G and a set of potential functions ϕ_k (PEARL, 1988). The graph has a node for each random variable and the model has a non-negative real-valued potential function for each

clique in the graph. Markov networks are useful in modeling problems where one cannot naturally define a directionality to the interactions between the variables. The representative of the second case are the Bayesian networks (PEARL, 1991) whose edges of the graph have a source and a target. Next we describe Bayesian networks thoroughly, since it is the underlying probabilistic graphical models of BLPs.

In order to review the main concepts of Bayesian networks we use the following convention: X denotes a random variable, x a state, $\mathbf{X}$ a set of random variables and $\mathbf{x}$ a set of states.

Bayesian networks represent the joint probability distribution $P(X_i, ..., X_n)$ over a fixed and finite set of random variables $\{X_i, ..., X_n\}$. Each random variable X_i has a $domain(X_i)$ of mutually exclusive states. They allow a compact and natural representation of the set of random variables by exploiting conditional independence properties of the distribution.

We say a variable X is conditionally independent of a variable Y, given a variable E, in a distribution P if $P(X|Y, E) = P(X|E)$. Conditional independence is denoted by $(X \perp Y|Z)$.

In order to represent the connections between random variables, as well as their probability distributions, a Bayesian network is composed of two components, as follows. *Qualitative or logic component of a Bayesian network*: it is an augmented *Directed Acyclic Graph* (DAG) G whose nodes are the random variables in the domain and whose edges correspond to direct influence among the random variables. The *parents* of a variable X_i are the variables represented by the nodes whose edges arrive in X_i. Similarly, the *children* of a variable X_j are variables represented by the nodes whose edges come from X_j. Henceforward, we use the terms "variables" and "nodes" interchangeably. The *local independence assumption* in a Bayesian network states that a variable X_i is conditionally independent of its non-descendants in the network, given a joint state of its parents, i.e.:

$$(X_i \perp Non - Descendants_{X_i}|Parents(X_i)) \qquad (3)$$

where $Parents(X_i)$ denotes the states of the parents of node X_i, and if the

node has no parents, then $P(X_i|Parents(X_i)) = P(X_i)$

The independence assumption allows it to write down the joint probability distribution as

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i| \ Parents(X_i)) \tag{4}$$

by applying the independence assumption to the *chain rule* expression of the joint probability distribution.

Quantitative component of a Bayesian network. Because of the independence assumption, each node is associated to a *local probability model* that represents the nature of the dependence of each variable on its parents. Thus, each node has a *Conditional Probability Distribution*s (CPDs), $cpdX_i$, specifying a distribution over the possible values of the variable given each possible joint assignment of values to its parents, i.e, $P(X_i|parents(X_i))$. If a node has no parents, then the CPD turns into a marginal or prior distribution, since is conditioned on an empty set of variables.

Consider, for instance the following problem from genetics (FRIEDMAN, et al., 1999; KERSTING, et al., 2006): *It is a genetic model of the inheritance of a single gene that determines the blood type of a person. Each person has two copies of the chromosome containing this gene, one inherited from her mother and another inherited from her father. Occasionally, a person is not available for testing, and yet because of the clarification of crime, test of paternity, allocation of (frozen) semen etc. it is often necessary to estimate the blood type of the person. A blood type can still be derived for that person through an examination and analysis of the types of family members.*

To represent this domain we would have for each person three random variables: one representing her blood type (bt_{person}), another one representing the gene inherited from her father (pc_{person}) and the last one representing the gene inherited from her mother (mc_{person}). The possible values for bt_{person} are in the set $domain(bt_{person}) = \{a, b, o, ab\}$ and the domain for mc and pc are the same and composed of $\{a, b, o\}$. In this example the independence assumptions are clear due to the biological rules: once we know the blood type of a person, additional evidence about others members of the family will not provide new information about the blood type. In the same way, once we know the blood type of both parents,

249

we know what each of them can pass on to their descendants. Thus, finding new information about a person's non-descendants does not provide new information about blood type of the person's children. Those local dependencies are better visualized through the Bayesian network of the Figure 24. For example, regarding the biological point of view, the blood type of *Susan* is correlated with the blood type of her aunt *Lily*, but once we know the blood type of *Allen* and *Brian*, the blood type of *Lily* is not going to be relevant to the blood type of *Susan*. Each variable has an associated CPD, coding the probability of a person has one of the values of the domain as her blood type, given each possible assignment of the genes inherited from her parents.
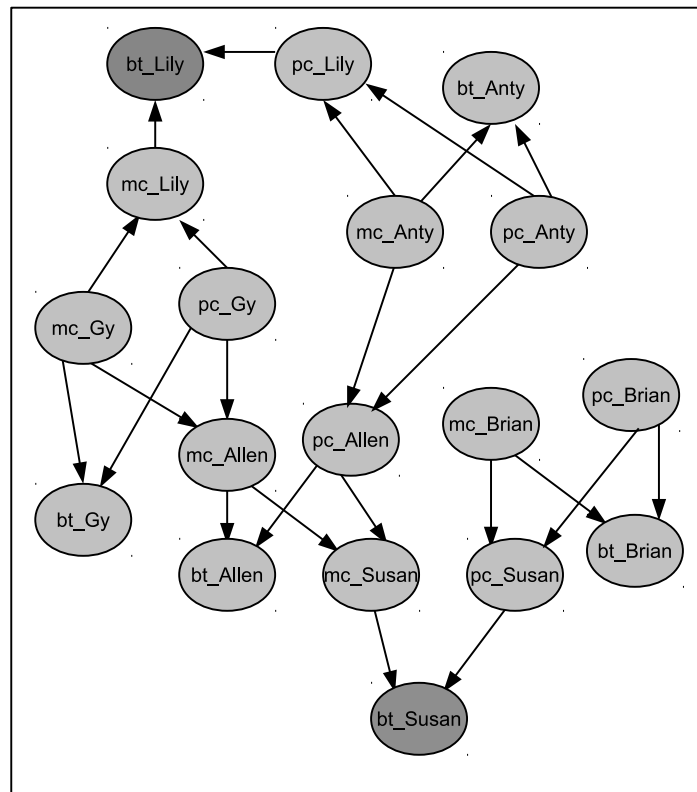

Figura 24: Bayesian network representing the blood type domain within a particular family

D-separation

Independence properties in probabilistic graphical modes can be exploited in order to reduce the computation cost of answering queries process. If one guarantees that

a set of nodes **X** is independent from another set of nodes **Y** given **E**, then in the presence of evidence about the nodes in **E**, an observation regarding variables in **X** cannot *influence* the beliefs about **Y**. The independence assumptions satisfied by a Bayesian network can be efficiently identified using a graphical test called *d-separation* (GEIGER, et al., 1989). The intuition above this test encompasses four general cases, illustrated in Figure 25, from where we would like to analyze whether knowing an evidence about a variable X can change the beliefs about a variable Y, in the presence of evidence about variables **E**. Naturally, when variables X and Y are directly connected, they are able to influence each other. Now, consider the four cases where variables X and Y are connected through E.

- The first case represents an indirect causal effect, where an ancestor X of Y could pass influence to it via **E**. Consider, for example, the Bayesian network of Figure 24. If one wants to know the blood type of *Susan* and do not know the gene she receives from its mother (mc_Susan is unknown), the gene its mother received from its grandmother (mc_Allen variable is known) is able to influence the beliefs on which blood type Susan is. On the other hand, if one already knows the gene Susan received from its mother, the gene passed from Susan'grandmother to Susan' mother no longer influences her blood type. Going back to the general case, X can only influence Y in the presence of E if E is not observed. In this case, we say the evidence E *blocks* influence of X over Y, as defined in Definition .3.

- The next case is the symmetrical case of the last one: we want to know whether evidence above a descendant may affect an indirect ancestor. In our running example, this is the case of trying to know whether the gene *Allen* received from its mother (random variable mc_Allen) is affected by the knowledge of the blood type of Susan (bt_Susan). As before, once the gene Susan received from Allen is known (random variable mc_Susan is observed), bt_Susan is not able to affect the beliefs on mc_Allen. However, in case mc_Susan is missing, bt_Susan has a free path to reach mc_Allen and therefore to affect its value.

- A common cause case is represented in Figure 25(c), where the node E is a common cause to nodes X and Y. In the Bayesian network of the example, this is the case of variables mc_Allen, mc_Susan and bt_Allen, where mc_Allen is a common cause for the others variables. The bloodtype of $Allen$ (bt_Allens) and the gene $Susan$ received from her mother $Allen$ (mc_Susan) are correlated, since knowing the bloodtype of $Allen$ helps us to predict the gene she received from her mother (mc_Allen) and consequently, the gene $Allen$ has transmitted to her daughter $Susan$. However, in case mc_Allen is observed, knowing the bloodtype of $Allen$ provides no additional information to predict the gene she transmitted for her daughter, as the information of which gene she has is stronger than her bloodtype. Thus, in the general case, variables sharing a common cause are able to influence each other if the path of common causes between them if free of evidences, in another words, the path is unblocked.

- Figure 25(d) represents a common effect trail. Structures in the form $X \rightarrow E \leftarrow Y$ are called *v-structures*. The three cases previously discussed share a pattern: X can influence Y via E is and only if E is not observed. In constrast, a common effect trail is different as X can influence Y via E if and only if E *is* observed. This is easier to see through an example: consider the random variables mc_Susan, pc_Susan and bt_Susan in Figure 24 and suppose we would want to know the gene $Susan$ received from her father (random variable pc_Susan). If the blood type of $Susan$ is known, the evidence about mc_Susan is able to influence the beliefs on pc_Susan. Knowing the gene she received from her mother *and* her blood type affects the beliefs about the gene she received from her father. For example, knowing her blood type is A and the gene her mother transmitted to her is o, the only possible gene she received from his father is A. However, if we do not know the blood type of $Susan$, it is not possible that the gene she received from her mother affects our beliefs on the gene she received from her father. Thus, if the common effect variable is not observed, knowing about a parent variable cannot affect our expectation about the others parents.
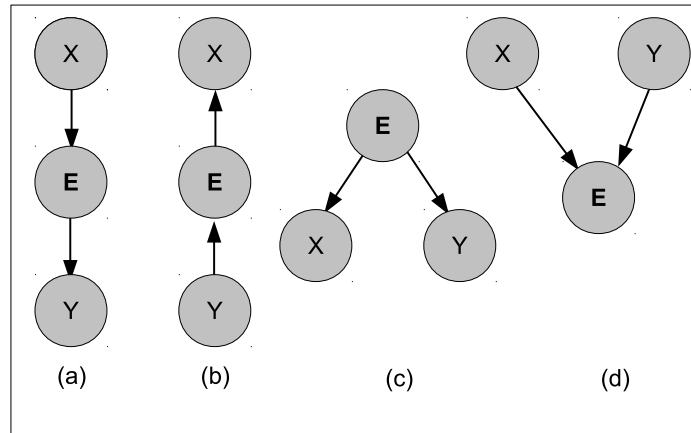
Figura 25: The four possible edge trails from X to Y, via E. Figure (a) is an indirect causal effect; Figure (b) is an indirect evidential effect; Figure (c) represents a common cause between two nodes and Figure (d) shows a common effect trail.

Definition .3 *Block: Let X and Y be random variables in the graph of a Bayesian network. We say an undirected path between X and Y is blocked by a (set of) variable E if E is in such a path and influence of X cannot reach Y and change the beliefs about it because of evidence (or lack of it) of E.*

The d-separation test guarantees that **X** and **Y** are independent, given **E** if every path between **X** and **Y** is blocked by **E** and therefore influence cannot flow from **X** through **E** to affect the beliefs about **Y** (KOLLER, FRIEDMAN, 2009; DARWICHE, 2010). If the path is not blocked, we say there is an *active trail* between the two sets of nodes.

Definition .4 *Active trail: Given an undirected path $X_1 \rightleftharpoons ... \leftrightharpoons X_n$ in the graph component of a Bayesian network, there is an* active trail *from X_1 to X_n given a subset of the observed variables* **E***, if*

- *whenever we have a v-structure $X_{i-1} \rightarrow X_i \leftarrow X_{i+1}$, then X_i or one of its descendants are in* **E***;*

- *in all the others cases no other node along the trail is in* **E***.*

An active trail indicates precisely a path in the graph where influence can flow from one node to another one. Thus, we say that one node can influence another if

253

there is any active trail between them. The definition below provides the notion of separation between nodes in a directed graph.

Definition .5 *d-separation:* **X** *and* **Y** *are* d-separated *by a set of evidence variables* **E** *if there is no active trail between any node* $X \in \mathbf{X}$ *and* $Y \in \mathbf{Y}$ *given* **E**, *i.e, every undirected path from* **X** *to* **Y** *is blocked.*

Consider, for example the network in Figure 24, and the random variables $bt(Lily)$ and $bt(Susan)$. There is a path between them composed for the nodes

$$bt(Lily) \leftarrow mc(Lily) \leftarrow mc(Gy) \rightarrow mc(Allen) \rightarrow mc(Susan) \rightarrow bt(Susan)$$

that is an active trail: the path is a common cause trail, and there is no observed node between them blocking the influence of $bt(Lily)$ over $bt(Susan)$. Note, however, in case there is an observed node in the path between them, say, $mc(Susan)$, the path would be no longer an active trail, since this evidence would block the influence of $bt(Lily)$.

It has been proved that $\mathbf{X} \perp \mathbf{Y}|\mathbf{E}$ if and only if **E** d-separates **X** from **Y** in the graph G (GEIGER, et al., 1989; GEIGER, et al., 1990).

Bayes Ball

In order to answer a probabilistic query more efficiently, it is useful to identify the minimal set of relevant random variables, which are the ones influencing the computations. (SHACHTER, 1998) developed a linear time algorithm to identify conditional independence and requisite information in a Bayesian network, named the *Bayes Ball* algorithm, based on the concept of D-separation. Requisite information is composed of the nodes for which conditional probability distributions or observations might be needed to compute the probability of query nodes, given evidence. Bayes Ball works by using an analogy of bouncing a ball to visit the relevant nodes in the net, starting from the query nodes. From there, the ball may pass through the node from one of its parents to its children and vice-verse, may bounce back from any parent to all the others parents of the node or from any child to all children of the node or may be blocked. The move the ball takes also depends

whether the node is observed or not and/or whether the node is deterministic or not. Precisely,

- An observed node, deterministic or not, always bounces balls back from one parent to all its others parents, in order to identify common effect trails. However, such a node blocks balls from children, i.e., if a ball comes from one of its children it is not passed anymore from this node.

- An unobserved non deterministic node passes balls from parents to its children and from a child to both its parents and to others children.

- An unobserved deterministic node always passes the ball coming from its child to its parents and from a parent to its children.

Algorithm 41 formalizes what we have just said, with some important additions to guarantee that the same action is not repeated and that the algorithm finishes:

- It marks visited nodes on the top(bottom) when the ball is passed from a node to its parents(children). When a node is marked in the top (bottom) the algorithm has no need to visit the node's parent (children) anymore.

- It maintains a schedule of nodes to be visited from parents and from children, so that the move of the ball can be determined;

- It makes sure that the ball visits the same arc in the same direction only once.

After visiting all scheduled nodes, the algorithm returns as the minimum set of requisite information to answer a query $P(\mathbf{X}|\mathbf{E})$ the observed nodes which were visited and all residual nodes marked on the top, with the visit starting from the nodes in $\mathbf{X}$. It also identifies the nodes not marked on the bottom as irrelevant to estimate X, that is those nodes conditionally independent on $\mathbf{X}$, given $\mathbf{E}$.

It is proved that $\mathbf{X} \perp \mathbf{Y}|\mathbf{E}$ if and only if $\mathbf{Y} \subseteq \mathbf{I}$, where I is the set of conditionally independent nodes from $\mathbf{X}$ given $\mathbf{E}$, as determined by Algorithm 41. As any edge is traversed at most once in each direction, the complexity of the algorithm

Algorithm 41 The Bayes Ball Algorithm to Collect Requisite Nodes in a Network (SHACHTER, 1998)

Input: a Bayesian network B, a set $\mathbf{F}$ of deterministic nodes; A set $\mathbf{X}$ of query nodes; A set $\mathbf{E}$ of observed nodes

Output: A minimum set $\mathbf{R}$ of requisite nodes, which might be needed to compute $P(\mathbf{X}|\mathbf{E})$ and the set $\mathbf{I}$ of irrelevant nodes, where $\mathbf{X} \perp \mathbf{I}|\mathbf{E}$

1: Initialize all nodes in B as neither visited, nor marked on the top, nor marked on the bottom.
2: Create a schedule of nodes to be visited, initialized with each node in $\mathbf{X}$ to be visited as if from one of its children.
3: **while** there are still nodes scheduled to be visited **do**
4:      Pick any node K scheduled to be visited and remove it from the schedule. Either K was scheduled for a visit from a parent, a visit from a child, or both.
5:      Mark K as visited.
6:      **if** $K \notin \mathbf{E}$ and the visit is from a child **then**
7:          **if** the top of K is not marked **then**
8:              mark its top and schedule each of its parents to be visited;
9:          **if** $K \notin \mathbf{F}$ and the bottom of K is not marked **then**
10:             then mark its bottom and schedule each of its children to be visited.
11:      **if** the visit to K is from a parent **then**
12:          **if** $K \in \mathbf{E}$ and the top of K is not marked **then**
13:              mark its top and schedule each of its parents to be visited;
14:          **if** $K \notin \mathbf{E}$ and the bottom of K is not marked **then**
15:              mark its bottom and schedule each of its children to be visited.
16: $R \leftarrow$ nodes in $\mathbf{E}$ marked as visited $\cup$ nodes marked on top
17: $I \leftarrow$ nodes not marked on the bottom
18: **return** R and I

---

is $O(n + m)$, where n is the number of nodes and m is the number of edges in the graph.

Figure 26 shows an example of the execution of Bayes Ball algorithm in a Bayesian network extracted from Figure 24. The visit starts from the random variable bt_Susan, as it had been visited before from a child. Next,

- As bt_Susan is not observed, it is marked in the bottom, in the top and passes the ball to its parents. Then, nodes mc_Susan and pc_Susan are scheduled to be visited. This corresponds to lines $6 - 10$ of Algorithm 41.

- Node pc_Susan is picked in the schedule. As it is an observed node, visited from a child, the ball is not passed anymore from it. In another words, its
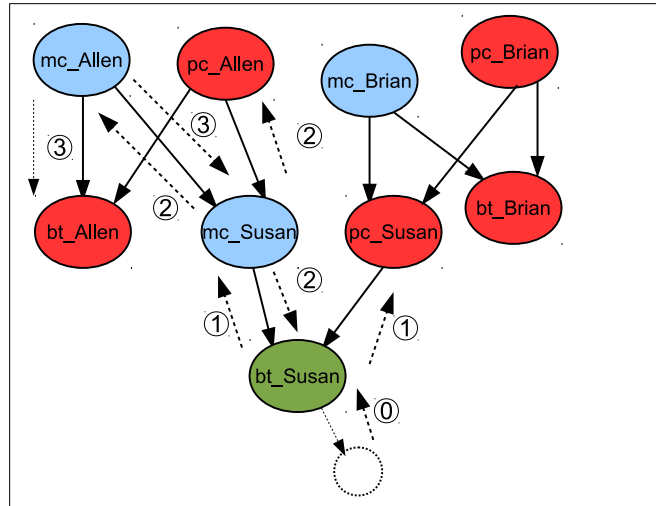
Figura 26: The ball visiting nodes with Bayes Ball algorithm. Red nodes are observed and green node is the query node.

evidence blocks influence to its child that could pass through it.

- Node *mc_Susan* is marked in the bottom and in the top and passes the ball from its child to both its parents (*mc_Allen* and *pc_Allen*) and its child (*bt_Susan*), since it is not observed.

- Node *bt_Susan* is visited again, this time from its parent, but there is nothing further to do, since it has already been marked in both directions.

- Node *pc_Allen* receives the ball, but it does not pass the ball anymore, since it is an observed node visited from a child.

- Node *mc_Allen* is marked in the top and in the bottom and bounces the ball to its children, *mc_Susan* and *bt_Allen*.

- Node *mc_Susan* is visited again, this time from its parent, but there is nothing further to do, since it has already been marked in both directions (bottom and top).

- *bt_Allen* in an observed node, visited from a parent, and therefore is marked in the top (line 13 of the algorithm). Then, it bounces the ball back to all its parents, in an attempt to find out a common effect trail between its parents.

257

mc_Allen is marked in both directions: there is nothing more to do with it. *pc_Allen* is again visited from a child. As there is no nodes scheduled to be visited anymore, the algorithm finishes.

Requisite nodes to compute the probability of *bt_Susan* are the set of nodes {*pc_Susan, mc_Susan, pc_Allen, mc_Allen, bt_Allen*}. Note that there is an indirect causal effect trail from *pc_Allen* to *bt_Susan*, via *mc_Susan*. Also, there is a common cause trail between *bt_Allen* and *bt_Susan* via {*mc_Allen, mc_Susan*}. Additionally, *pc_Allen* is able to influence the beliefs in *mc_Allen* via *bt_Allen*, as they form a common effect trail (v-structure). Influence cannot flow from *bt_Brian* and *pc_Brian* to *bt_Susan*, as they are blocked by the evidence of *pc_Susan*.

Inference in Bayesian Networks

Typically, a Bayesian network is used to compute marginal distributions of one or more query nodes, given some of the others nodes are clamped to observed values. Inference is the computation of these marginal probabilities, defined as

$$P(\mathbf{X}|\mathbf{E}) \propto \sum_{\mathbf{Z} \neq \mathbf{X}, \mathbf{E}} \mathbf{P}(\mathbf{X}, \mathbf{E}, \mathbf{Z}) \tag{5}$$

where $\mathbf{E}$ represents a set of observed variables (the evidence), $\mathbf{X}$ is the set of unobserved variables whose values we are interested in estimating and $\mathbf{Z}$ are the variables whose values are missing (the hidden nodes). For very small Bayesian networks it is easy to marginalize sums directly. But unfortunately, the number of terms in the sum will grow exponentially with the number of missing values in a network, making the exact computation of marginal probabilities intractable for arbitrary Bayesian networks, In fact, this is known to be a NP-hard problem (COOPER, 1990; DAGUM, LUBY, 1993). The good news is that some cases it is possible to exploit the graph structure so that the exact computation of marginal probabilities has complexity linear in the size of the network. This is precisely the case of *polytree* networks. In this case, exact inference such as variable elimination (LI, D'AMBROSIO, 1994), which exploits the chain-rule decomposition of the joint distribution and marginalizes out the irrelevant hidden nodes can be used efficiently.

There is also a family of algorithms named *junction tree* which work by joining variables in cluster nodes so that the resultant network becomes a polytree.

Learning Bayesian Networks from Data

Typically, learning in Bayesian networks has as ideal goal to return a model $B^* = (G^*, \theta^*)$, where G^* is a Directed Acyclic Graph and θ^* is a set of probability parameters. They both together should precisely capture the underlying probability distribution P^* governing the domain. To do so, both the structure of the Bayesian network (the DAG) and local probability modes (CPDs) may be learned using an *Independently and Identically Distributed* (IID) data set $D = \mathbf{d[1]}, ..., \mathbf{d[M]}$ of M examples sampled independently from P^*. However, the ideal goal is generally not achievable because of (1) computational costs and (2) limited data set providing only an approximation of the true distribution. Thus, learning algorithms attempt to return the best approximation to $B*$, according to some performance metric. As the metric is often a numerical criterion function, the learning task can be seen as an optimization problem, where the *hypothesis space* is the set of candidate models and the criterion for qualifying each candidate is the *objective function*. One common approach is to find B that maximizes the *likelihood* of the data, or more conveniently its logarithm, since the products are converted to summation. The *Log-Likelihood* (LL) is defined as

$$LL(B|D) = \sum_{k=1}^{M} \log P(\mathbf{d[k]}) = \sum_{\mathbf{k=1}}^{\mathbf{M}} \sum_{\mathbf{j=1}}^{\mathbf{V}} \log \mathbf{P(d[k]_j | parents(d[k]_j))} \qquad (6)$$

where each example $\mathbf{d[k]}$ is composed of a set of V random variables $d[k]_j$. Note that the likelihood function has the property of *decomposability*, where each variable is decomposed in a separate term. From a statistic point of view, the higher the likelihood, the better the Bayesian network is of representing $P*$. When the model is learned by maximizing the likelihood or a related function, we have a *generative* training, since the models is trained to *generate* all variables (FRIEDMAN, et al., 1997). Alternatively, if it is known in advance that the model is going to be used to *predict* values to random variables $\mathbf{X}$ from $\mathbf{Y}$, the training may be done *discriminatively* (ALLEN, GREINER, 2000; GREINER, ZHOU, 2002; GROSSMAN,

DOMINGOS, 2004), where the goal is to get $P(\mathbf{X}|\mathbf{Y})$ to be as close to the real distribution $P^*(\mathbf{X}|\mathbf{Y})$ as possible. In this case, the objective function typically used is the *Conditional Log-likelihood (CLL)*, computed as

$$CLL(B|D) = \sum_{k=1}^{M} \log P(d[k]_i | d[k]_1, ..., d[k]_{i-1}) \qquad (7)$$

where $d[k]_i$ is the class variable and $d[k]_1, ..., d[k]_{i-1}$ are the others variables. Notice that

$$LL(B|D) = CLL(B|D) + \sum_{k=1}^{M} \log P(d[k]_1, ..., d[k]_{i-1})$$

Because of that, maximizing CLL leads to better classifiers than when maximizing LL, since the contribution of $CLL(B|D)$ is likely to be swamped by the generally much larger (in absolute value) $\log P(d[k]_1, ..., d[k]_{i-1})$ term. In fact, (FRIEDMAN, et al., 1997) shows that maximizing the CLL is equivalent to minimize the prediction error, since this metric takes into account the confidence of the prediction. However, different from LL, CLL does not decompose into a separate term for each variable, and as a result there is no known closed form for computing optimal parameters.

There are three situations concerning the observability of the dataset one may encounter when learning Bayesian networks (KOLLER, FRIEDMAN, 2009):

1. The dataset is *complete* or *fully observed* in such a way that each example has observed values for all the variables in the domain.

2. The dataset is *incomplete* or *partially observed,* i.e, some variables may not be observed in some examples.

3. The dataset has *hidden or latent variables*, whose value is never observed in any example. It may be the case that we are even unaware of some variables (we do not know they exist), since it is never observed in the data. Despite of it, they might play a central role to understand the domain.

In respect to what must be learned from data to result in a Bayesian network, we have two distinct cases :

1. The graph is known (although it is not necessarily the correct one) and it is only required that the parameters are learned from data;

2. Both the graph structure and parameters are unknown and it is necessary to learn them both from the dataset.

Parameters estimation

The problem of CPDs estimation for a Bayesian network is concerned with estimating the values of the best parameters θ of a fixed graph structure. It is particularly important because, although estimating the graph structure is arguably easy for an expert of the domain, eliciting numbers is difficult for people and may be unreliable. Besides the graph structure, the random variables and the values they can take are assumed to be known. The problem is usually solved by finding the *Maximum Likelihood Estimator (MLE)*, which is the set of parameters with the highest likelihood. However, others measure functions can be used.

When the dataset is complete, learning CPDs by MLE decomposes into separate learning problems, one for each CPD in the Bayesian network, and maximum likelihood estimation reduces to frequency estimation. Thus, to calculate the probability of a variable X assume value x when its parents assume the set of values *parents* it is used the formula

$$P(X = x|\ Parents(X) = \mathbf{parents}) = \frac{\mathbf{N(X = x, Parents(X) = parents)}}{\mathbf{N(Parents(X) = parents)}} \quad (8)$$

where $N(X = x, \mathbf{Parents} = \mathbf{parents})$ is the number of training instances in the dataset where X has the value x and its parents have the values **parents**.

When the dataset is only partially observed, the maximum likelihood estimation cannot be written in a closed form, since it is a numerical optimization problem with all the known algorithms working under nonlinear optimization. Algorithm such as *Expectation-Maximization (EM)*(DEMPSTER, et al., 1977), (LAURITZEN, 1995), (MCLACHLAN, KRISHNAN, 1997) and gradient descent (BINDER, et al., 1997) are used in this case. Here we give an overview of the popular EM algorithm.

EM algorithm   EM is based on the idea that since it is not possible to calculate the real frequency counting, because some of the examples may have missing values, such counting should be estimated from the data. Then, these estimates can be used to maximize the likelihood. The algorithm assumes an initial set of parameters, which may be initialized at random, and iteratively performs the two steps below until convergence:

E-step: Based on the dataset and the current parameters, the distributions over all possible completion of each partially observed instance are calculated (expected counts). In order to calculate such distributions, the procedure must infer the probability of each variable for each value of its domain, using the current CPDs.

M-step: Each completion estimated above is treated as a fully observed data case, weighted by its probability. Then, each (weighted) frequency counting is used to calculate the improved parameters. The log-likelihood is calculated and in case it converges, the procedure stops. Otherwise, it iterates to E-step, but now using the newly computed CPDS.

Structure Learning

Often it is easy for a domain expert to provide the structure of the network, since it represents basic causal knowledge. However, this is not always the case and sometimes the causal mode may be unavailable or subject to disagreement. In these situations, it is necessary to search for a graph fitting the dataset. Learning a whole Bayesian network gets more complicated in the presence of incomplete data, since inference is necessary to produce the expected counts, and the problem is compounded in the presence of hidden variables: in this case the domain and number of hidden variables must also be selected from dataset.

Using a naive approach, structure learning may start with a graph with no edges and iteratively add parents to each node, learning the parameters for each structure and measuring how good is the resulting model using an objective function. Or still, it may start with a random generated initial structure and use a search approach to modify the current structure, by adding, deleting or reversing edges. Since the resulting graph structure cannot have cycles, it is necessary to either define

beforehand an ordering to the variables or search for possible orderings (RUSSELL, NORVIG, 2010). To measure how well a model explains the data, one may use a probabilistic function such as log-likelihood, perhaps penalizing complex structures by using for example the MDL principle (LAM, BACCHUS, 1994). Is is important to note that for each graph candidate parameters may change and therefore they will need to be re-estimated. In case the dataset is complete, only the variables involved with the modification must have theirs CPDs re-learned (decomposability property). Unfortunately, this does not happen when the dataset is incomplete, since local changes in the structure may result in global changes in the objective function and because of that the parameters of CPDs may change. In this case it is imperative to apply a heuristic solution such as Structural EM (FRIEDMAN, 1998) to return the final Bayesian network.

The situation gets even worse in the presence of hidden variables. Including hidden variables in a network can greatly simplify the structure by inducing a large number of dependencies over a subset of its variables. However, it is important to analyze the trade-offs, since learning a structure with hidden variables is far from trivial: it is necessary to decide how many of them are going to be included, their domains and also where to include them in the structure. If the number of variables and their domain is previously informed, it is possible to treat this problem as an extreme case of learning with incomplete data, where such variable(s) are never observed. There are several approaches developed to learning with hidden variables, typically based on the idea of using algebraic (KEARNS, MANSOUR, 1998; TIAN, PEARL, 2002) or structural (ELIDAN, et al., 2000) signatures. Recently, we also contributed with a method for adding hidden variables in the structure of Bayesian networks, based on a discriminative approach and theory revision (REVOREDO, et al., 2009).

Bayesian Logic Programs: A Logical Probabilistic Model Extending Bayesian Networks

Although Bayesian networks are one of the most important efficient and elegant formalism for reasoning about uncertainty, they are a probabilistic extension of propo-

sitional logic (LANGLEY, 1995). Indeed the qualitative component of a Bayesian network corresponds essentially to a propositional logic program. Consider, for example, the network exhibited in Figure 24. The influence relations of such a Bayesian network can be represented trough the propositional logic program in Table 14, where the random variables in the Bayesian network correspond to logical atoms and the direct influence relation corresponds to immediate consequence operator.

Tabela 14: Propositional Logic Program representing the network in Figure 24

bt_Susan		$mc\_Susan, pc\_Susan.$
bt_Allen		$mc\_Allen, pc\_Allen.$
bt_Brian		$mc\_Brian, pc\_Brian.$
bt_Lily		$mc\_Lily, pc\_Lily.$
bt_Gy		$mc\_Gy, pc\_Gy.$
bt_Anty		$mc\_Anty, pc\_Anty.$
mc_Susan		$mc\_Allen, pc\_Allen.$
pc_Susan		$mc\_Brian, pc\_Brian.$
mc_Allen		$mc\_Gy, pc\_Gy.$
pc_Allen		$mc\_Anty, pc\_Anty.$
mc_Lily		$mc\_Gy, pc\_Gy.$
pc_Lily		$mc\_Anty, pc\_Anty.$
$mc\_Brian.$		
$pc\_Brian.$		
$mc\_Gy.$		
$pc\_Gy.$		
$mc\_Anty.$		
$pc\_Anty.$		

Bayesian networks thus inherit the limitations of propositional logic, namely the difficulties to represent objects and relations between them. In the example just discussed, a new family would require a whole new graph. However, if we had the set of definite clauses in Table 15, we can take advantage of the domain regularities, by applying the rules upon different variables binding.

Bayesian Logic Program (BLP) (KERSTING, DE RAEDT, 2001d; KERSTING, DE RAEDT, 2001a; KERSTING, 2006) were developed to combine the advantages of the elegant and efficient formalism of the Bayesian networks with the expressive powerful representation language provided by Logic Programming. The goal is therefore to eliminate the disadvantages of propositionality in Bayesian networks

Tabela 15: First-order Logic Program representing the regularities in the network of Figure 24

$bt(X)$	|	$mc(X), pc(X).$
$mc(X)$	|	$mother(Y, X), mc(Y), pc(Y).$
$pc(X)$	|	$father(Y, X), mc(Y), pc(Y)$

and the lack of reasoning under uncertainty within ILP. The key idea is to associate the clauses above to a CPD, that characterizes the probability of the head of the clause, given the literals in the body. This is exhibited in Table 16.

Tabela 16: First-order Logic Program representing the regularities in the network of Figure 24, where each CPD is represented as a list of probability values.

$bt(X)$	| $mc(X), pc(X), [0.97, 0.01, 0.01, 0.01, ..., 0.01, 0.01, 0.01, 0.97].$
$mc(X)$	| $mother(Y, X), mc(Y), pc(Y),$
	$[0.93, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, ..., 0.01, 0.01, 0.01, 0.01, 0.01,$
	$0.01, 0.01, 0.93].$
$pc(X)$	| $father(Y, X), mc(Y), pc(Y),$
	$[0.93, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01, ..., 0.01, 0.01, 0.01, 0.01, 0.01,$
	$0.01, 0.01, 0.93].$
$mc(X),$	$[0.3, 0.3, 0.4]$
$pc(X),$	$[0.4, 0.3, 0.3]$

Next we define the BLPs language in more details.

Bayesian Logic Programs: Key concepts

Bayesian atoms and predicates A Bayesian predicate p/n is a first-order predicate with arity n and a a set of finite and mutually exclusive states associated to it (its domain). An atom of the form $p(t1, ..., tn)$ is a Bayesian atom if p/n is Bayesian, and both share the same domain. A Bayesian predicate p/l is generically a set of random variables, while a Bayesian ground atom $p\theta$ is a random variable over the states $domain(p/l)$.

Bayesian Clause A Bayesian clause c is an expression of the form $A|A_1, ..., A_n, n \geq 0$, where A is a Bayesian atom and each A_i is a Bayesian atom or a logical atom. The symbol | is used to highlight the conditional probability distribution. The set

of Bayesian atoms in the body of c directly influence the Bayesian atom in the head of c. Thus, there is a CPD $cpd(c)$ associated with each Bayesian clause c, encoding $P(head(c)|body(c)$ and representing the conditional probabilities distributions of each ground instance $c\theta$ of the clause c. Logical atoms do not have probabilistic influence in the Bayesian atom in the head of c, Note that as logical atoms do not correspond to random variables they do not have a representation in $cpd(c)$, serving only to instantiate variables in the clause.

The clause $mc(X)|mother(Y,X),mc(Y),pc(Y)$ in the genetic example is a Bayesian clause, where $mc/1$ and $pc/1$ are Bayesian predicates with $domain_{mc/1} = domain_{pc/1} = \{a,b,0\}$ and $mother(Y,X)$ is a logical atom, used to do the connection between logical variables X and Y. In this context, $mc(susan)$ represents the gene a person named $Susan$ inherits from her mother as a random variable over the states $\{a,b,0\}$.

Combining rule It may be the case that several ground clauses have the same head, for example, there are two clauses c_1 and c_2 such that $head(c_1\theta_1) = head(c_2\theta_2)$. The clauses specify $cpd(c_1\theta_1)$ and $cpd(c_2)\theta_2)$ but not the required probability distribution $P(head(c_1\theta_1)|body(c_1\theta_1) \cup body(c_2\theta_2)$. Typically, in order to get such a distribution *combining rules* are employed. A combining rule is a function that maps finite sets of conditional probability distributions $\{P(A|A_{i1},...,A_{ini})|i=1,...,m\}$ onto one combined conditional probability distribution $P(A|B_1,..,B_k)$ with $\{B_1,...,B_k\} \subseteq \cup_{i=1}^{m}\{A_{i1},...,A_{ini}\}$. For each Bayesian predicate p/l there is a corresponding combining rule $cr(p/l)$ such as *noisy-or* (JENSEN, 2001), *noisy-max* (DÍEZ, GALÁN, 2002), *(weighted)-mean* (NATARAJAN, et al., 2008).

Definition .6 *A BLP consists of a set of Bayesian clauses. For each Bayesian clause c there is exactly one conditional probability distribution cpd(c)and for each Bayesian predicate p/l there is exactly one combining rule cr(p/l).*

Well-defined BLP As a logical probability model, inference in BLP is executed in a Bayesian network generated from the BLP B and the least Herbrand model $LH(B)$ associated to the domain and the BLP. The nodes in the $DAG(B)$ correspond to

ground atoms in $LH(B)$, encoding the direct influence relation over the random variables in $LH(B)$. Thus, there is an edge from a node X to a node Y if and only if there is a clause $c \in B$ and a substitution θ such that $y = head(c\theta)$, $x \in body(c\theta)$ and for all ground atoms $Z \in c\theta$, $z \in LH(B)$. Although indeed the Herbrand base $HB(B)$ is the set of all random variables we can talk about, only the atoms in the least Herbrand model constitute the relevant random variables, as they are the true atoms in the logical sense, and because of that only they are going to appear in the $DAG(B)$. Each node in $DAG(B)$ is associated to a CPD $cpd(c\theta)$ where $head(c\theta) = y$, after applying a combining rule of the corresponding Bayesian predicate to the set of CPDs related to the corresponding variabilized clause. We say a BLP B is *well-defined* if and only if $LH(B) \neq \emptyset$, the graph associated to B is indeed a DAG and each node in $DAG(B)$ if influenced by a finite set of random variables. If B is well defined, it specifies a joint distribution $P(LH(B)) = \prod_{X \in LH(B)} P(X|parents(X))$ over the random variables in $LH(B)$.

Answering queries procedure

As well as in Bayesian networks, any probability distribution over a set of random variables can be computed in a BLP. A probabilistic query to a BLP B is an expression of the form

$$? - Q_1, ..., Q_m | E_1 = e_i, ..., E_p = e_p$$

where $m > 0$, $p \geq 0$, $Q_1, ..., Q_m$ are the query variables and $E_1, ..., E_p$ are the evidence variables and $\{Q_1, ..., Q_m, E_1, ..., E_p\} \subseteq HB(B)$. The answer for such query is the conditional probability distribution

$$P(Q_1, ..., Q_m | E_1 = e_i, ..., E_p = e_p)$$

The least Herbrand model and consequently its corresponding Bayesian network may become too large to perform inference. However, it is not necessary to compute the whole Bayesian network but instead only a part of it, called the *support network*. The support network N of a random variable $X \in LH(B)$ is the induced sub-network of $\{X\} \cup \{y | Y \in LH(B) \text{ and } Y \text{ influences } X\}$. The support network of a

finite set $\{X_1 ..., X_n\} \in LH(B)$ is the union of the support networks for each variable X_i. Thus, the support network constructed to answer a probabilistic query consists of the union of the support networks for each query variable and each evidence variable. (KERSTING, DE RAEDT, 2001b) proved that a support network of a finite set $\mathbf{X} \subseteq \mathbf{LH(B)}$ is sufficient to compute $P(\mathbf{X})$.

In order to construct the support network of a query variable X it is necessary to gather all ground clauses employed to prove $: -X$. The set of all proofs is all the information needed to compute the support network. The proofs are typically constructed by using the SLD-resolution procedure. After that, it may be necessary to combine multiple copies of ground clauses with the same head using the correspondent combining rule. Finally, to indeed calculate the probability distribution, the support networks of each Bayesian atom which is part of the probabilistic query are united, giving rise to a DAG. Such a DAG, together with the CPDs resulting or not of combining rules, and the states of each evidence variable, can be provided to any inference algorithm so that the probability of the query is computed.

Algorithm 42 Algorithm for Inducing a Support Network from a Probabilistic Query (KERSTING, DE RAEDT, 2001a)

Input: A probabilistic query $? - Q_1, ..., Q_n | Ev_1 = ev_1, ..., Ev_m = ev_m$
Output: A support network N related to the probabilistic query
1: **for** each variable $X_i \in \{Q_1, ..., Q_n, Ev_1, ..., Ev_m\}$ **do**
2:       compute all proofs for X_i
3:       extract the set S of ground clauses used to prove X_i;
4:       combine multiple copies of ground clauses $H|B \in S$ with the same H, generating the support network N_i for X_i;
5: $N \leftarrow \cup_{i=1}^{k} N_i$;
6: $N \leftarrow prune(N)$;

Consider, for example, the logical part of a BLP in Table 17 and the query

$$? - bt(susan) | bt(brian) =' a', pc(brian) =' a', pc(susan) =' o', pc(allen) =' o',$$
$$bt(allen) =' a'$$

Predicates *mother* and *father* are deterministic. In order to prove the query $bt(susan)$, the first clause in the table is instantiated with the substitution $X/susan$.

Tabela 17: Qualitative part of a Bayesian Logic Program

$bt(X)|\ mc(X), pc(X).$
$mc(X)|\ mother(Y, X), mc(Y), pc(Y).$
$pc(X)|\ father(Y, X), mc(Y), pc(Y).$
$mother(allen, susan).$
$father(brian, susan).$
$pc(allen).$
$mc(allen).$
$mc(brian).$
$pc(brian).$

Then, the procedure tries to prove $mc(susan)$ and gets $mother(Y, susan)$, $mc(Y)$, $pc(Y)$. The variable Y is substituted by *allen*, because of ground atom $mother(allen, susan)$, and then the algorithm reaches the ground atoms $mc(allen)$, $pc(allen)$. The same is done to prove $pc(susan)$. From this set of proofs, the support network in Figure 27(a) is built, by adding one node to each ground Bayesian atom in the proof and including an edge from an atom directly influencing another atom, following the Bayesian clauses. Next, following the same procedure, support networks are built for the evidence atoms, giving rise to the networks (b), (c), (d), (e) and (f) in Figure 27. Next step is to unite all those support networks, which is done by merging the random variables shared by more than one network. The final network is exhibited in Figure 28.

Learning BLPs

As usual in machine learning, BLPs are learned from a set of examples **D**. Each example $\mathbf{D_l} \in \mathbf{D}$ has two parts, a *logical* part and a *probabilistic* part. The logical part is a *Herbrand interpretation*, more specifically the least Herbrand model of the BLP we want to learn. Thus, techniques used in *learning from interpretation* setting of ILP can be adapted for learning the qualitative part of the BLP, which implies that the hypothesis space is composed of sets H of Bayesian clauses such that all $\mathbf{D_l} \in \mathbf{D}$ is a model of H. Notice that, it is necessary to check whether the candidate hypothesis H is acyclic on the data, i.e, for each example of the data set the induced Bayesian network's graph must be acyclic. The probabilistic part is composed of a
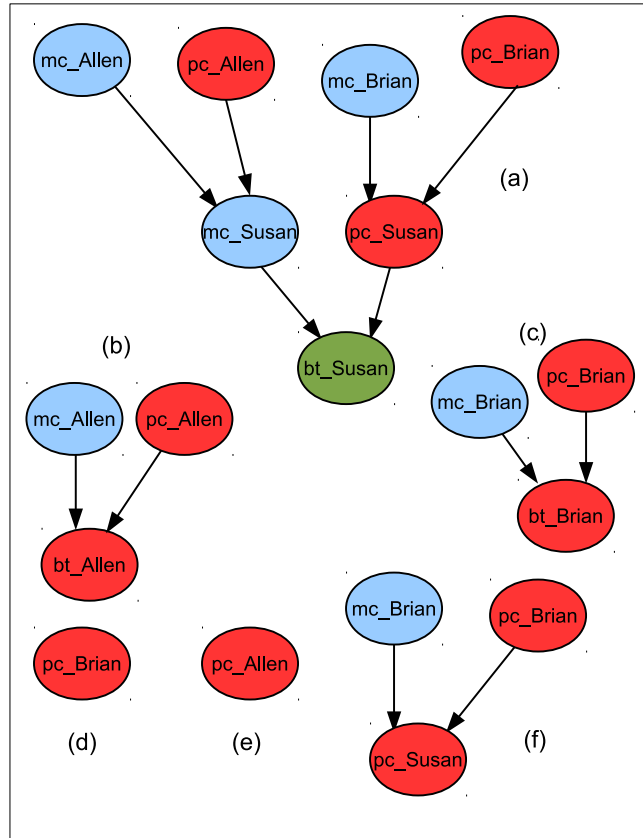
Figura 27: Support networks created from the query variable $bt(susan)$ and evidence variables $bt(brian)$, $pc(brian)$, $pc(susan)$, $pc(allen)$ and $bt(allen)$.

possibly partial assignment of values to the random variables in the least Herbrand model. One example of data case for the genetics example above would be:

$$\{bt(susan) =?, mc(susan) =?, pc(susan) =' o', pc(allen) =' o',$$
$$pc(brian) =' a', bt(brian) =' a', bt(allen) =' a', mc(allen) =?,$$
$$mc(brian) =?, mother(allen, susan), father(brian, susan)\}$$

where ? indicates a missing value for the random variable. Besides the logic part, a candidate hypothesis must also take into account the joint distribution over the random variables induced by the probabilistic part of the examples. To match this requirement, the conditional probabilities distributions are learned from the data set and a probabilistic scoring function. Thus, the final goal is to find the hypothesis H, acyclic on the data, with the examples $D_l \in \mathbf{D}$ being models of H in the logical sense and where H best matches the examples according to the
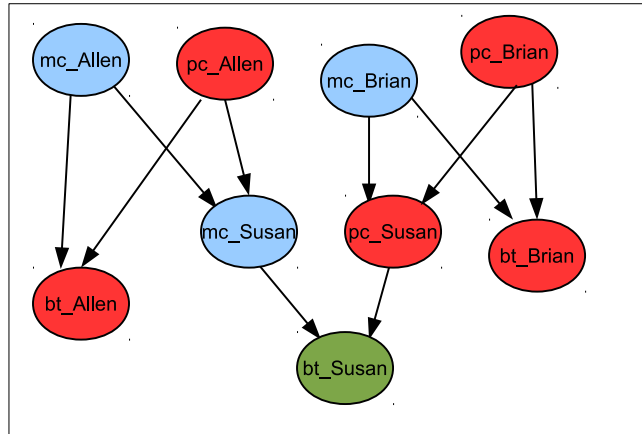
270

Figura 28: Bayesian network after joining same random variables from different support networks from Figure 27.

scoring function. To achieve this last requirement, the parameters of the associated conditional probability distributions in H must maximize the scoring function, which matches the learning setting of CPDs in Bayesian networks. Next, we see more details of how to estimate the parameters of BLPs and how to traverse the hypothesis space using refinement operators from ILP.

Parameter estimation in BLPs

In order to learn the parameters of a BLP, Bayesian networks are built from each example together with the current BLP. Thus, parameters estimation in BLPs follows the main ideas of parameters estimation in Bayesian networks. If the induced data set is completely observed, frequency estimation is achieved by counting. If there are missing values, it is necessary to use an algorithm capable of estimating the counts. However, parameters estimation differs from traditional algorithms of Bayesian networks because of the following reasons:

- In traditional Bayesian network setting, each node in the network has its separate CPT. In BLPs, CPDs are associated to Bayesian clauses rather than ground atoms what makes multiple instances of the same rule to share the same CPT. As a result, more than one node in the network may have the same CPT. Thus, parameters are learned considering Bayesian clauses instead of individual nodes in the network. Because of that, more than one node in the

same network may be taken into account to learn the same CPT. This situation is illustrated in Figure 29, where nodes in magenta are yielded from the same Bayesian clause $c_i = bt(X)|mc(X), pc(X)$. Each one of them is going to be considered as a separate "experiment" to estimate parameters of the Bayesian clause that originates them. Thus, considering EM algorithm, parameters are estimated using the formula

$$cpd(c_i)_{jk} = \frac{\sum_{l=1}^{m} \sum_{\theta} P(head(c_i\theta) = u_j, body(c_i\theta) = \mathbf{u_k}|\mathbf{D_l})}{\sum_{l=1}^{m} \sum_{\theta} P(body(c_i\theta) = \mathbf{u_k}|\mathbf{D_l})} \qquad (9)$$

where u_j stands for the values in the domain of $head(c_i)$, u_k stands for each combination of values of the Bayesian atoms in $body(c_i)$ and θ denotes substitutions such that the example D_l is a model of $c_i\theta$. The formula represents the action of computing the CPD for a clause c_i by taking into account each node produced from a substitution θ applied to the clause. In Figure 29, substitutions θ applied to the Bayesian clause c_i are $\{X/susan\}, \{X/Allen\}, \{X/Brian\}$.
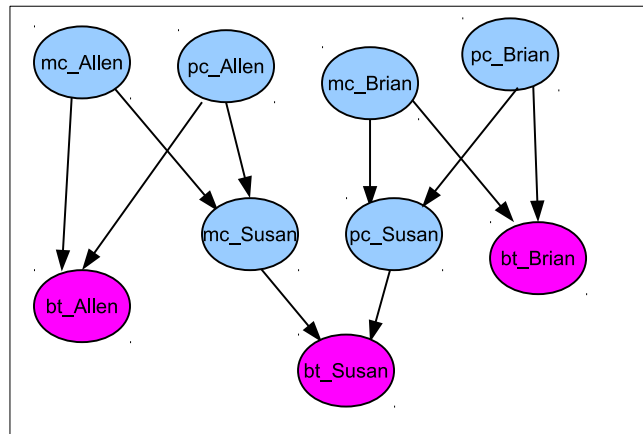

Figura 29: Bayesian network representing the blood type domain within a particular family. Nodes in magenta are yielded by the same Bayesian clause.

- The formula above computes the CPD of an *individual* clause c_i by estimating the joint probability of the head of the clause together with its body. The probabilities are computed through the Bayesian network produced by the examples. Thus, it is required that each node corresponds to a head of exactly one clause. In case combining rules are not handled properly, this is not going to happen with the Bayesian network produced from an example. Suppose,

for example, that we have in a BLP the three clauses below:

$pred(X)|pred1(X, Y).$

$pred(X)|pred2(X, Y).$

$pred(X)|pred3(X, Y).$

Then, suppose a substitution $\theta = \{X/1\}$. After the substitution, we have the ground clauses

$pred(1)|pred1(1, 2).$

$pred(1)|pred2(1, 3).$

$pred(1)|pred3(1, 4).$

The support network built from those ground clauses is going to be the one presented in Figure 30(a). Note that, as $pred(1)$ is the same random variable, the support network of each ground clause has been joined to produce the final network. The problem is, the requirement that each node is produced by exactly one clause is not attended if the support network is built like that. Therefore, it is necessary to assume independence of causal influences (ICI) (HECKERMAN, BREESE, 1994), (ZHANG, POOLE, 1996), which allows that multiple causes on a target variable can be *decomposed* into several independent causes, whose effects are combined to yield the final value. As combining rules are employed to compute the final required probability distribution of the random variable, it is necessary to assume they are *decomposable*, i.e., they allow to decompose each random variable corresponding to the same ground head into several possible causes. This is expressed by adding extra nodes to the induced network, which are copies of the ground head atom. With this modification in the network, each node is produced by exactly one Bayesian clause c_i and each node derived from c_i (the yellow nodes) can be seen as a separate experiment for computing $CPD(c_i)$. Those extra nodes are considered hidden, since the value given in the example is relative to the instance itself and not to each representative of a rule. A network with the extra nodes is exhibited in Figure 30(b).
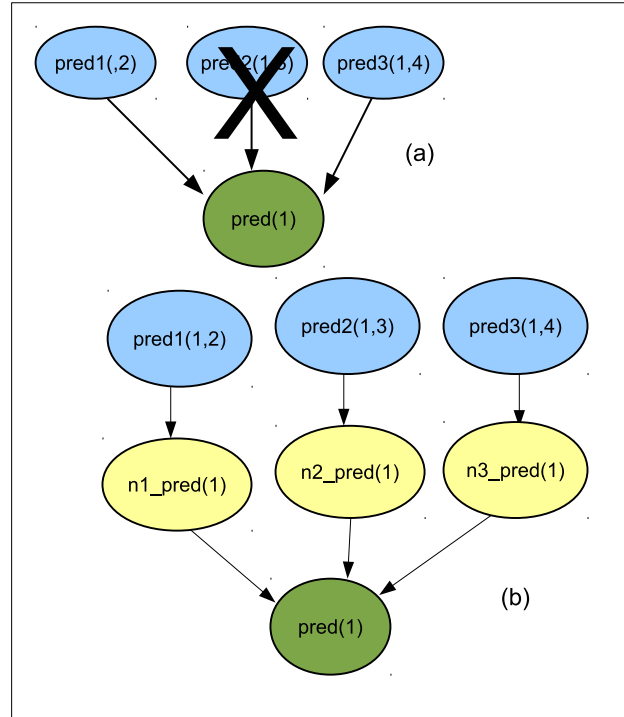
273

Figura 30: Figure (a) is a Bayesian network without adding extra nodes to represent decomposable combining rules. Node $pred(1)$ is produced by three different clauses. Figure (b) is the induced network reprsenting decomposable combining rules by adding extra nodes (yellow nodes) so that each node is produced by exactly one Bayesian clause. Nodes $n\_?pred(1)$ has the domain of $pred$ and cpd(c) associated.

In this work, we follow the representation of combining rules defined in (NATARAJAN, et al., 2008), where ground clauses are combined in two levels. The first level combines different instantiations of the same clause with a common head, while a second level combines different clauses with the same head. Suppose for example, in addition to the ground clause above, we would have also

$pred(1)|pred1(1,3)$

$pred(1)|pred1(1,4)$

Then, instances from the first clause are combined in one level and the others clauses are combined in the second level. Different combination functions may

274

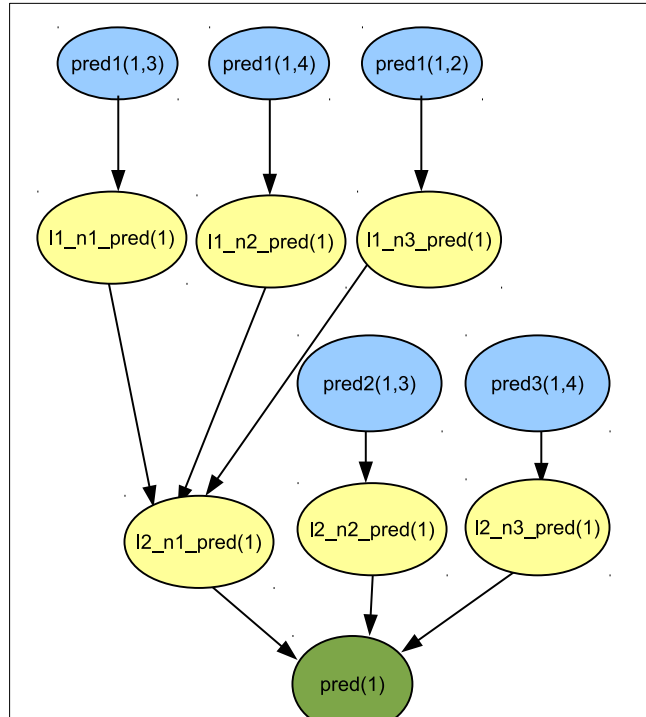be applied to each level. The final network is reproduced in 31.

Figura 31: Decomposable combining rules expressed within a support network. Instantiations of the same clause and different clauses are combined separately. Nodes $l1\_n?\_pred(1)$ represent different instantiations of the same clause and nodes $l2\_n?\_pred(1)$ represent different clauses.

Structure learning in BLPs

The algorithm proposed in (KERSTING, DE RAEDT, 2001b) for learning BLPs traverse the hypothesis space using two ILP refinement operators: $\rho_s(H)$, which adds constant-free atoms to the body of a single clause $c \in H$ and $\rho_g(H)$, responsible for deleting atoms from the body of a single clause $c \in H$. The algorithm is reproduced as Algorithm43 and work as follows. First, a hypothesis H_0, possibly generated from a learning from interpretation algorithm such as CLAUDIEN (DE RAEDT, BRUYNOOGHE, 1993; DE RAEDT, DEHASPE, 1997), is assumed as the starting point and the parameters maximizing the log-likelihood $LL(D, H)$ are computed. Then, following a basic greedy hill-climbing, the neighbors of H_0 are induced using the dataset D and the refinement operators $\rho_s(H_0)$ and $\rho_g(H_0)$, requiring that they induce acyclic Bayesian networks and proves all examples in D. The candidate

hypothesis are scored using the set of support networks and a probabilistic scoring function and the one with the best evaluation is considered as the next hypothesis if it has an evaluation better than the current hypothesis. This process continues until there are no further improvements in the scoring function.

Algorithm 43 Algorithm for learning BLPs (KERSTING, DE RAEDT, 2001b)

Input: A finite dataset D, a set of Bayesian clauses H_0
Output: A BLP B
1: $\Theta \leftarrow$ parameters maximizing $LL(D, H)$
2: $H \leftarrow H_0 \cup \Theta$
3: compute $Score_H$
4: **repeat**
5:     **for** each $H' \in \rho_g(H) \cup \rho_s(H)$ **do**
6:         **if** H' proves D **then**
7:             **if** the Bayesian networks induced from H' and D are acyclic **then**
8:                 $\Theta' \leftarrow$ parameters maximizing $LL(D, H')$
9:                 compute $score_{H'}$
10:                 **if** $score_{H'} > score(H)$ **then**
11:                     $H \leftarrow H'$
12:                     $score_H \leftarrow score_{H'}$
13:                     $\Theta \leftarrow \Theta'$
14: **until** there are no candidate hypothesis H' scoring better than H
15: **return** H

Note that the refinement operators may be applied in all Bayesian clauses of the current hypothesis in order to generate their neighbors, which arguably causes the structure learning to be very expensive. Next section, we present the first theory revision system designed to revise logical probabilistic models such as BLPs.

PFORTE: Revising BLPs from Examples

Revising first-order logic theories aims to start from an initial theory and modify it in order to return a more accurate model compared to learning from scratch techniques. Similarly, one may have an initial logic probabilistic model with only some points preventing it of correctly reflecting the domain. Such a model could be obtained from the following sources:

- A domain expert elicited the clauses which are not necessarily reflecting the dataset;

- An ILP or PLL system learned the model considering an old data set and now there are new examples not necessarily reflected by the old model;

- An ILP or PLL system learned the model considering the current dataset, but it could still be improved by revision techniques. In case it was learn from an ILP system, the uncertainty in the domain was not considered and the revision could improve the model.

Usually, it would be more valuable to search for those revision points and modify them, instead of discarding the original model or proposing modifications to all its points, as it is done in BLP structure learning algorithm. This approach should not only produce more accurate methods but also reduce the cost of searching in the space of clauses and parameters. In the probabilistic logic case, new questions arise, since the starting model is composed by two related parts: the clauses and the probabilistic parameters. Thus, in order to check whether the model should be revised one must take into account these two components when proposing modifications. Moreover, as the inference to answer queries is performed in the induced probabilistic graphical model, it is also through them that we should look for the problematic points. With this motivation, we developed the first probabilistic first-order revision system designed to revise Bayesian Logic Programs. It was named PFORTE since it was built upon FORTE system (REVOREDO, ZAVERUCHA, 2002; PAES, 2005; PAES, et al., 2005b; PAES, et al., 2005a; PAES, et al., 2006a). This section reviews the PFORTE system as published in (PAES, et al., 2006a).

PFORTE: Key concepts

Task definition

- **Given:** an incorrect initial logical probabilistic logic model, a consistent set of examples, background knowledge composed of Bayesian and/or logical clauses.

- **Find:** a revised logical probabilistic model that is complete considering the given examples and has the highest score given some metric.

In order to find this "minimally revised"and complete model, PFORTE works in two steps: the first component addresses theory incompleteness by using gen-

277

eralization operators only; the second component addresses classification problems using both generalization and specialization operators.

PFORTE terminology is defined as follows:

Logical probabilistic model: The logical probabilistic model is a BLP, composed of a a set of Bayesian clauses, as defined in 5.3.2.

In the Genetics domain, concepts might include the blood type, *bt(person)*.

Definition .7 (PFORTE Example) *The schema of an example (D_i) is:*

$$D_i = \{Instances, Deterministic, Evidences\}$$

An instance *is an instantiation of a predicate, with an associated value from the domain of the corresponding Bayesian predicate;* Deterministic *are literals do not representing random variables and* Evidences *are Bayesian atoms with a (partial) assignment of values from their respective domains.*

We built this definition following BLPs, where the logical part (qualitative) is each ground atom in $Instances, Deterministic$ and $Evidences$ sets and the assignment of values in $Instances$ and $Evidences$ is the probabilistic part (quantitative). The dataset might be composed of several examples. Instances and Evidences in one single example are mutually dependent, while different examples are assumed as independent from each other. In the Genetics domain, one example could be the one represented in Table 18.

Tabela 18: Format of an example in PFORTE system

[
   $[bt(susan) =' o', bt(brian) =' a', bt(allen) =' a']$
   $[mother(allen, susan), father(brian, susan)]$
   $[mc(susan) =?, pc(susan) =' o', pc(allen) =' o', pc(brian) =' a', mc(allen) =?,$
                                             $mc(brian) =?]$
]

Note that from a logical point of view all the instances are assumed as positive, so that the final model must cover them, and the class from which the instance belongs is indicated by the value associated to such an instance.

Definition .8 (Completeness) *Given a set of examples D, where $D_i \in D$ and $D_i = \{Insts, Dets, Evs\}$ we say that a BLP B is* complete *considering these examples if and only if*

$$\forall D_i \in D, \forall I_j \in logical\ part\ of\ Insts:\ B \cup Dets \cup logical\ part\ of\ Evs \vdash I_j$$

Provability is established using standard SLD-resolution, taking the instance to be the initial goal.

Definition .9 (Classification) *Given an instance we say that this instance is correctly classified if the probability value computed to its real class is higher than a pre-defined threshold. For example, consider D_i exemplified above. We say that the instance $bt(susan)$ is correctly classified, if using an inference method in its support network, the probability value computed to class a is higher than a pre-defined threshold. Otherwise we say that this instance was* misclassified.

The PFORTE Algorithm

PFORTE is a greedy hill-climbing system composed of two steps. The first step focuses on generating a theory as complete as possible, by addressing failed examples. The second step has as goal to induce a theory as accurate as possible, by addressing misclassified examples. Both steps follow the key ideas below in a greedy hill-climbing search:

1. Identification of *revision points*. In the first step, those are the clauses failing to prove instances and they are called *logical revision points*. In the second step, those are the points responsible for the misclassification of instances, discovered through a probabilistic reasoning mechanism (named as *probabilistic revision points*.

2. Proposal of modifications to the revision points using *revision operators*. As all the instances are positive and the first step aims to make the hypothesis proves all the examples, only generalization operators are necessary. This differs from the second step, where both generalization and specialization operators are applied, in an attempt to make the final hypothesis more accurate.

3. Score the proposed revisions. Each proposed revision, even the ones contemplated in the first step, is scored by a probabilistic evaluation function such as log-likelihood or conditional log-likelihood, since the ultimate goal is to reflect the joint probability distribution over the random variables in the data set.

4. Choice of the revision to be implemented. Both steps choose the revision with the highest score to be implemented, in case its score is better than the score of the current hypothesis. The first step requires the coverage is improved as well, while the second step requires the accuracy is also improved.

5. Stop criteria. The first step stops when there are no further generalizations capable of improving the coverage. The Second step stops the whole procedure when there are no further revisions proposed to the revision points capable of improving the accuracy.

Algorithm 44 shows how the key ideas are implemented. The first step focuses on generating a complete hypothesis by only addressing failed examples. To do so, it revises the initial hypothesis iteratively, using a hill-climbing approach. Each iteration identifies the logical revision points, where a revision has the potential to improve the theory's coverage. A set of revisions generalizing the current hypothesis are proposed and scored on the training set. The best one is selected and implemented in case the current coverage is improved. This process continues until the first step cannot generate any revisions which improve example coverage. It is expected that the revised hypothesis will be complete considering the data set.

This complete BLP is the starting point to the second step of PFORTE. This step tries to improve classification by modifying the probabilistic revision points. Both generalization and specialization operators are applied on such points and as in the first step, the revisions are scored, the best one is chosen and implemented if it improves the accuracy. This greedy hill-climbing process proceeds until either no further revision improves the scoring function or the probabilistic accuracy cannot be improved.

Next will detail the main parts of PFORTE's algorithm: generation of revision points, generation of possible revisions to this revision points and how to score the

---

Algorithm 44 PFORTE Algorithm (PAES, et al., 2006a)

Input: An initial logical probabilistic model B, the background knowledge BK, and a set of examples D

Output: A revised logical probabilistic model B

1: **repeat**
2:     generate logical revision points;
3:     **for** each logical revision point **do**
4:         generate and score possible revisions;
5:         update best revision according to the score and also improving coverage;
6:         **if** best revision improves coverage **then**
7:             implement best revision;
8: **until** no revision improves coverage
9: **repeat**
10:     generate probabilistic revision points;
11:     **for** each probabilistic revision point **do**
12:         generate and score possible revisions;
13:         update best revision found according to the scoring function and improving the probabilistic accuracy;
14:         **if** best revision improves the probabilistic accuracy **then**
15:             implement best revision;
16: **until** no revision improves score

---

proposed modifications.

Generating Revision Points

Revision points are places in a theory where errors may lie. PFORTE considered two types of revision points:

- **Logical revision points.** These points are generated in the same way FORTE does for generalization revision points, by annotating places in the theory where proofs of instances fail. These are places where the theory may be generalized in order to become complete. We follow the annotation process originally proposed by Richards and Mooney (RICHARDS, MOONEY, 1995): each time there is a backtrack in the proof procedure, the antecedent in which the clause failed is kept; this antecedent is a *failure point.* In addition, other antecedents that may have contributed to this failure, perhaps by binding variables to incorrect values (the *contributing points*) are also selected. Both failure and contributing points are considered logical revision points.

- **Probabilistic revision points.** These points are generated from misclassified instances, by following a prediction criteria which takes into account the probability distribution and the dependency among the Bayesian atoms. Thus, to generate the probabilistic revision points, the support networks built from the examples are collected and used to perform inference over the instances, now represented as query random variables. The value inferred in the network for each query variable is compared to the original value from the dataset and, in case they differ, the instance is considered as misclassified. The probabilistic revision points are then any clauses taking part in the network of the misclassified instance.

Algorithm 45 Algorithm for generating Probabilistic Revision Points in a Logical Probabilistic Model such as BLP

Input: A LPM B; Background knowledge BK; A set of examples **D**
Output: A set of revision points RP
 1: $RP \leftarrow \emptyset$
 2: **for** each example $D_i \in$ **D do**
 3:      $d \leftarrow$ Bayesian network built from B, BK, D_i
 4:      **for** each random variable v in D_i which is an instance (query Bayesian atom) $\in D_i$ **do**
 5:          $valueNode \leftarrow$ class of v inferred by a probabilistic inference engine
 6:          $valueExample \leftarrow$ class of the Bayesian atom $\in D_i$
 7:          **if** $\exists\ valueExample$ and $valueExample \neq valueNode$ **then**
 8:             $GRP \leftarrow$ nodes in d
 9: **for** each Bayesian clause $c \in B$ **do**
10:      **if** $head(c)$ unifies with a random variable in GRP **then**
11:          $RP \leftarrow RP \cup c$
12: **return** RP

Revision operators

In order to be able to repair arbitrarily incorrect theories, revision operators must be able to transform any theory in the language into another. Depending on the type of revision points different operators can be used.

- **Operators for logical revision points.** To propose modifications for logical revision points we use all FORTE generalization operators. The differences are in delete_antecedent and add_rule operator.

a) *delete_antecedent* operator in FORTE deletes as many antecedents as possible while negative examples are not proved. As we do not have negative examples, this operator stops removing antecedents when examples became proved or the scoring function cannot be improved.

b) Original *add_ruleoperator* works in two steps after copying the incorrect clause. First, it removes the failed antecedents in order to prove previously unproved positive instances and then it adds new antecedents in an attempt to not cover negative examples. In PFORTE this operator was changed so that after removing the failed antecedents in a rule, PFORTE adds antecedents in an attempt to improve the value of the probabilistic scoring function.

- **Operators for probabilistic revision points.** To propose modifications for probabilistic revision points PFORTE also uses FORTE operators, in this case generalization and specialization operators, with slight modifications. When specializing clauses, PFORTE differs from FORTE in three different actions:

  1. Antecedents with the highest score are added while they are able to improve the score of the current hypothesis, differing from FORTE which stops to *addantecedents* when there are no provable negative examples and all the provable positive examples continue to be covered.

  2. FORTE may create more than one specialized version of the revision point in one revision, since a refinement of a clause can make provable positive examples become unprovable. As PFORTE cares for more than provability, it returns one specialized version of the original clause.

Similarly, when deleting antecedents, the antecedent chosen to be removed is the one with the highest score. The algorithm for deletion/addition of antecedents is detailed in 46.

Finally, the add_rule operator works combining both operators described above: first it deletes antecedents and then it adds antecedents. All operators for probabilistic revision points must also enforce examples covering and

---

Algorithm 46 Algorithm for deletion/addition of antecedents in PFORTE

---

repeat

    for each antecedent in a clause/space of possible antecedents **do**

        if after deletion/addition covering of examples still holds **then**

            score this modification

    delete/add antecedent with the highest score

until no antecedent can be deleted/added without decreasing the score

---

range restriction of the clauses.

PFORTE_PI (REVOREDO, et al., 2006; REVOREDO, 2009) was developed to incorporate two novel revision operators based on predicate invention (MUGGLETON, BUNTINE, 1988; STAHL, 1993; MUGGLETON, 1994; KRAMER, 1995) in PFORTE system. The first operator, called *Compaction* creates new predicates to be head of clauses by absorbing a set of literals from the body of others clauses, in an attempt to decrease the complexity of literals and parameters in the LPM. The other operator, called *augmentation*, replaces a literal from the body of a clause by a new invented predicate, creates a clause with such a literal in the head and tries to specialize this new clause. As usual, the specialization and implementation of the predicate invention operators are only implemented if the score is improved.

Scoring possible revisions

Each proposed revision receives a score. Based on this score, PFORTE chooses the best one to be implemented. Since for each proposed modification the LPM is changed, it is necessary to re-learn the CPDs of the clauses. To do so, first PFORTE builds the support networks for each example generating a data set composed by these support networks. Having the set of support networks, the next step is to learn the CPDs for each clause, where *Maximum Likelihood Estimation*(MLE) approach is employed in the same way it is done in parameters estimation for BLPs. After learning and updating the CPDs for each clause, PFORTE calculates the score for the proposed modification using a probabilistic scoring function such as LL or CLL. The algorithm for scoring possible revision is detailed in 47

PFORTE was applied in three artificially generated datasets (PAES, et al.,

284

Algorithm 47 Algorithm to Score Possible Revisions in PFORTE

Input: A LPM B, a set of examples D, Background knowledge BK, a scoring
    function F
Output: Score $Score_B$ of B, considering D, BK and F
 1: $S \leftarrow emptyset$
 2: **for** each example $D_i \in D$ **do**
 3:     $S \leftarrow S\cup$ support network built for D_i
 4: $\Theta^* \leftarrow$ learned CPDs considering S;
 5: $score_B$ compute score using F and considering $Theta^*, S$;

2005b; PAES, et al., 2006a). The two-phase algorithm revising an initial BLP was successfully compared to the algorithm starting the learning from scratch (without an initial theory). However, when we tried to run PFORTE with more complex datasets, such as the ones used in SRL community, the system either could not finish the revision in reasonable time or it could not finish the revision at all, due to memory problems.

In (MIHALKOVA, et al., 2007) it was developed an algorithm for transfer learning between two Markov Logic Network models, where the first step perform mapping between predicates and the second step revises the first model learned. The second step uses revision operators in a fashion similar to PFORTE.

Appendix G: BFORTE: Addressing Bottlenecks of Bayesian Logic Programs Revision

Inference in Bayesian Logic Programs is performed on networks built from all the proofs of the set of examples. The algorithm for learning BLPs starts from a maximally general logic program satisfying the logical part of the examples and proposes refinements to each clause, by adding or deleting literals. In this fashion, learning BLPs requires searching over a large search space of candidate clauses, on the one hand, and the construction of all proofs to build the Bayesian networks, on the other. For each candidate theory one may have to perform full Bayesian inference, in order to compute scores in the presence of non-observed data.

The PFORTE system was developed to obtain accurate BLPs by revising an existing BLP and modifying it only in the rules used in Bayesian networks of misclassified examples. However, despite promising results on artificial domains, PFORTE faces similar bottlenecks as BLP and other SRL learning algorithms, as it must also address the large search space of logic programs and also perform inference.

The goal of this chapter is to address these bottlenecks of PFORTE system. First, the space of possible refinements can be reduced by limiting the candidate literals to be added to a clause to the ones present in the Bottom Clause (MUGGLETON, 1995b). Second, we show that collecting revision points through Bayes Ball (SHACHTER, 1998) ultimately reduces the number of clauses marked as candidates for being revised by revision operators, as well as the type of the operator applied on them. Third, we do not separate revision points as logical and probabilistic avoiding the cost of treating them in two separate steps of the revision

286

process. Instead, the algorithm performs a single iterative procedure by considering all revision points as probabilistic ones. Fourth, we observe that in SRL example networks can overlap due to shared entities, resulting in large joint networks. We apply methods to (1) to overlap ground clauses with identical features, (2) to separate the requisite nodes for a specific queryand (3) to overlap requisite nodes found for different instances. Such methods follow the ideas of recent developments in lifted inference (POOLE, 2003; SALVO BRAZ, et al., 2005; MEERT, et al., 2010). By focusing on the relevant modification points and enabling the revision process to explore its full potentialities more efficiently, we expect to obtain an indeed effective and feasible revision system, as pointed out in (DIETTERICH, et al., 2008) as a necessary development in SRL area. We named this new system BFORTE, referring to **B**LP, **B**ayes **B**all and **B**ottom clause.

The rest of this chapter is organized as follows. Section 5.3.2 address search space of new literals to the ones presented in the Bottom Clause. Section 5.3.2 address the search space of revision points by presenting the one step revision algorithm and the algorithm for collecting revision points with D-separation. Section 5.3.2 address the methods developed to reduce inference space, which is most often built to score each proposed revision. Section 5.3.2 shows experimental results obtained from BFORTE system and section 5.3.2 finally concludes the chapter.

Addressing Search Space of New Literals

The operators that generate the largest search space are the ones that search for new literals to be included in the body of clauses. This includes both add-antecedents and add-rule operators. PFORTE applies FOIL (QUINLAN, 1990) top-down strategy to make new literals. As expected of a top-down approach, the number of generated literals may be huge, depending on their definitions and the background knowledge. Moreover, the literals generated will often not be much significative, as no insight is given on which terms should be an input/output variable or a constant. Whenever we add a new literal to a body of a clause, each generated literal must be scored so that the best is chosen. Adding literals to Bayesian clauses causes a modification to

the qualitative part of the BLP, so it is necessary to re-learn the parameters affected by the revision. Taking everything into consideration, adding literals to clauses outweighs the time expended proposing modifications and dominates the runtime of the revision process.

In BFORTE we use the Bottom Clause to compose the search space of evaluated literals to be added to a clause. The use of the Bottom Clause greatly decreases the number of possible literals, by limiting the candidate clauses to subsets of the Bottom clause. The use of the Bottom Clause in BFORTE follows the procedure devised in (DUBOC, et al., 2009) with small differences in the implementation. The most significant difference is in that the Bottom Clause is generated from a positive example, covered by the current clause, since their goal when specializing clauses is to continue proving positive examples while making negative examples become unprovable. Here, the Bottom Clause is generated from a misclassified instance, no matter its class. It is still required that the instance is covered by the base clause, since the clause before being refined is a subset of the bottom clause.

As in (DUBOC, et al., 2009) literals in the clause must obey mode declarations. Additionally, through determinations declarations, it is possible to define an ordering to random variables. Remember that a determination declaration $determination(pred_1/N1, pred_2/N2)$ indicates that $pred_2/N1$ may appear in the body of a clause whose head comes from $pred_1/N2$. By exploring all determinations, the system can identify which random variables can be a parent to another random variable. Those are random variables produced by literals in the place of $pred_2/N2$. Moreover, it can also identify the opposite: which random variables may to be children of another random variable in the Bayesian network of an example. Those are random variables generated from predicates in the place of $pred_1/N1$ in determinations declarations.

Bayesian networks may be used to perform two reasoning strategies: bottom-up reasoning, when it goes from effects to causes, and top-down reasoning, when the reasoning goes from causes to effects (SPIRTES, et al., 2001; PEARL, 2009). We can compare these reasoning strategies to the steps performed by the operator when taking into account the Bottom Clause: first, the bottom clause is generated from

288

the instance, which is similar to search for causes from an effect (the instance); next, the literals in the bottom clause may be added to the body of a Bayesian clause – the algorithm tries to improve the BLP by adding causes to effects. In fact, the causal relationship is used as a guide to construct the graph structure. Note, however, that the process of constructing the Bottom Clause and taking literals from it comprises only one level of reasoning, since only literals *directly influencing* the instance, i.e., the ones with an edge going to the head random variable, are collected. Thus, the literals in the Bottom Clause representing random variables may be direct causes of the effect represented by the random variable in the head of the Bayesian clause.

It is important to notice that the Bottom Clause procedure is instance driven. Literals are collected considering only one instance, making the choice of literals biased by such an instance. Another instance might point out more worthwhile literals, i.e., literals whose corresponding random variables would produce a stronger influence and/or over more instances. One alternative strategy would be to collect the Bottom Clause for more than one single instance within an example. Arguably, the space of candidate literals would be larger. Thus, one could consider to reduce the search space by measuring the volume of information flowing between the two random variables involved – the one from candidate literals and the one from the head of the clause (CHENG, et al., 2002a). We intend to investigate this strategy in future work.

Addressing Selection of Revision Points

Bayesian Revision Point

The key step of the revision process is to identify revision points. BLP classification strategy is to build Bayesian networks, grounded from the set of examples and the current Bayesian clauses, and perform inference on them in order to compute probabilities for queries. Thereby, it is natural to use the same mechanism to find the points responsible for a misclassification. PFORTE considers two types of revision points: logical and probabilistic ones. In the first case, in the same fashion as logical theory revision, clauses failing to prove an instance are marked to be generalized. Recall that in PFORTE every instance is positive in the logical sense,

and its evidence value defines the class to which the instance belongs. If the example is unprovable, PFORTE could not compute a probability for it. In the present work we do not consider logical revision points separated from probabilistic revision points. Instead, in case the example is unprovable, we compute a probability for it by always including a *default node* in the Bayesian network, i.e., a node ground from a most general clause for the instance predicate. If such a default node gives a probability below a threshold to the correct class, then it will be marked as a revision point. Considering that we may now select revision points from the Bayesian network constructed to the example, we define *Bayesian revision points* as follows.

Definition .10 (Bayesian Revision Point) *Let B be the Bayesian network built from Bayesian clauses and an example D. In case an instance from D is misclassified after applying an inference engine in B, there is a maximum set S of nodes together with their respective CPDs in B relevant to assessing the belief in the instance. Bayesian revision points are the Bayesian clauses used to yield nodes in S.*

Following such a definition, BFORTE requires only one iterative step to fix the BLP: at each iteration Bayesian revision points are found and revision operators propose refinements on them. Note that default nodes are only introduced when there are unprovable instances. They can also lead to Bayesian revision points, but in this case, the only possible revision proposed to them is to create new rules. Algorithm 48 presents the top-level procedure of BFORTE, modified from Algorithm 7.4 by removing its first step (line 1 to line 8). As before, the algorithm follows an iterative hill climbing procedure, where at each iteration Bayesian revision points are collected at first, revisions are proposed to the revision points and their score are computed. The revision with the best score is chosen to be implemented, but only if it is capable of improving the current score of the BLP.

An observation about score computation is necessary here. As the proposed revisions are going to modify the qualitative part of the BLP, it is also necessary to reflect those modifications in the quantitative part of the model. Thus, at each proposed modification we re-learn the parameters involved in the modified structure. BLP revision presented in this work has an inherent discriminative behavior: we

are concerned with predictive inference and revision points are found out through misclassified instances. In this way, it is more appropriate to employ a discriminative evaluation function such as conditional log-likelihood, root mean squared error or precision-recall based functions. Additionally, the model would benefit from a discriminative training of the parameters. Because of that, we implemented the gradient descent learning of parameters to minimize the mean squared error, as initially designed in (NATARAJAN, et al., 2008). The method is based on representing combining rules in two levels: the first one combines different instantiations of the same rule, where they all share the same ground head, and the second one combines different clauses with the same ground head, as explained in section 7.2.3.

Algorithm 48 BFORTE Top-Level Algorithm

Input: A BLP BP, the background knowledge BK, a set of examples D, a set OP
    of revision operators to be considered to revise clauses
Output: A revised BLP BP'
  1: learn probability parameters of BP
  2: **repeat**
  3:     generate Bayesian revision points;
  4:     **for** each Bayesian revision point **do**
  5:         **for** each revision operator $\in OP$ **do**
  6:             propose revision
  7:             score revision
  8:             update best score revision found;
  9:     **if** best revision improves the current score **then**
10:         implement best revision;
11: **until** no revision improves score

Searching Bayesian Revision Points through D-Separation

Originally, PFORTE denoted as probabilistic revision points all clauses whose head took part in the Bayesian network where an instance had been misclassified. Although the set of candidate clauses to be modified is smaller than in Kersting's BLPs learning algorithm, since not all clauses appear in every network, depending upon the size of the network several Bayesian clauses are going to be marked to be modified by the revision operators. Howevver, not all clauses that are used to generate the network may be relevant to the classification of the query.

    Algorithms for learning Bayesian networks are generally grouped into two cate-

gories: scoring based methods (HECKERMAN, 1996; TIAN, 2000; CHICKERING, 2002), which uses a heuristic to construct the graph and a scoring function to evaluate it; and constraint-based methods (VERMA, PEARL, 1990; SPIRTES, et al., 2001; CHENG, et al., 2002b; CAMPOS, 2006), relying on conditional independece tests to build the model. There is also a recent effort in getting the best of both worlds, by developing strategies combining these two approaches (TSAMARDINOS, et al., 2006; PELLET, ELISSEEFF, 2008). In this work, we follow the second group of algorithms when identifying Bayesian revision points, by designing a procedure that analyses the dependency relationships of a misclassified instance. Accordingly, *active paths* are searched to identify the nodes in the network that exerts influence over misclassified instances. By a path we mean any consecutive sequence of edges, without regarding their directionalities. A path is active if it carries information or dependence from one node to another through this sequence of edges. Whenever there is an active path between two variables, we say they are *d-connected*. Formally, this concept is defined as follows.

Definition .11 (D-connection for directed graphs (PEARL, 1988)) *For disjoint sets of vertices,* $\mathbf{X}$*,* $\mathbf{Y}$ *and* $\mathbf{E}$*,* $\mathbf{X}$ *is d-connected to* $\mathbf{Y}$ *given* $\mathbf{E}$ *if and only if for some* $X \in \mathbf{X}$ *and* $Y \in \mathbf{Y}$*, there is an (acyclic) path* $\mathbf{U}$ *from* X*,* Y *such that:*

- $\mathbf{U}$ *is composed of a direct connection for* X *and* Y*:* $X \leftarrow Y$ *or* $X \rightarrow Y$*;*

- $\mathbf{U}$ *indicates an indirect causal/evidential effect:* $X \leftarrow W \leftarrow Y$ *or* $X \rightarrow W \leftarrow Y$*,* $W \notin \mathbf{E}$*;*

- $\mathbf{U}$ *represents a common cause:* $X \leftarrow W \rightarrow Y$*,* $W \notin \mathbf{E}$*;*

- $\mathbf{U}$ *represents a common effect (a v-structure):* $X \rightarrow W \leftarrow Y$*,* W *or any of its descendants* $\in \mathbf{E}$*.*

If no path between the set of nodes is active, the variables are *d-separated*. In this case, there is a variable in the undirected path blocking influence from one node to another. To detect the nodes d-connected to the misclassified node, we take advantage of linear-time Bayes Ball (Algorithm 7.1), starting from the misclassified

query node. Algorithm 49 presents the top-level procedure for collecting revision points, which is performed as line 3 of Algorithm 48. First, it is necessary to identify the misclassified instances in each example. To do so, each instance at each example has its marginal probability computed. Note that when computing a marginal probability for a specific query, others instances in the same example are considered as evidence, as it is usually done in collective inference (JENSEN, et al., 2004). In this case the algorithm proceeds to run Bayes Ball, so that the active paths leading to the misclassified nodes can be identified, and consequently the relevant nodes are selected. The set of relevant nodes are considered as the requisite nodes (observed nodes that had been visited and nodes marked in the top) and also the nodes marked in the bottom, i.e., all nodes that are not discovered as irrelevant by the Bayes Ball algorithm. Finally, the clauses corresponding to the relevant nodes are identified and marked to be revised.

Algorithm 49 Top-level Selection of Revision Points

Input: A value of threshold T, A set N of Bayesian networks, each corresponding to an example
Output: A set of Bayesian Clauses RP, marked as revision points
 1: $RP \leftarrow \emptyset$
 2: **for** each Bayesian network $N_i \in N$ **do**
 3:      **for** each query instance $id \in N_i$ **do**
 4:          compute the probability $P(id = class)$ using an inference engine, where $class$ is the value of id in the dataset
 5:          **if** $P(id = class) < T$ **then**
 6:             $Relevant \leftarrow$ nodes marked as relevant by Bayes Ball Algorithm
 7:             $RP \leftarrow RP\cup$ clauses corresponding to nodes in $Relevant$ set
---

Analyzing the Benefits Brought by Revision Operators According to D-Separation

In this section we analyze the behavior of revision operators by considering whether the modifications they propose are capable of modifying the influence flowing in the Bayesian network. The question that arises is whether the revision operators are capable of changing existing influence that is likely to make the instance misclassified and/or to create new connections influencing such instances. Additionally, to check whether the operator really brings benefits to the BLP, this last is scored after

293

modified using Bayesian networks built from the set of examples. In this way, the modifications proposed on the BLP form a hybrid mechanism combining independence tests and scored heuristic search methods.

To analyze the modifications proposed by the operators, we focus attention to the directions that the ball is bounced from/to nodes in Bayes Ball algorithm, aiming to disregard modifications that cannot bring additional or modify current influence over the query node. Additionally, we must also consider whether the modification can change the set of Bayesian clauses instantiated by the example to build the Bayesian network, since this also can change the influence coming to a node. Remember from section 7.2.2 that a directed edge in included between a pair of nodes if there is a ground clause whose head is represented by the node where the edge arrives and whose body contains a representative of the node from where the edge leaves. We proceed by discussing each possible situation ocurring in nodes marked as relevant separately.

Non-observed node visited from a child (NOC node) A non-observed node visited from a child passes ball from the child to (1) its parents, so that additional indirect causal influence might be brought, and to (2) its children, that can also bring back further indirect effect influence, as visualized in Figure 32. Naturally, the visiting child is also likely to influence the visited node. The node representing the misclassified instance itself is such a kind of this node, since the visit starts from it as it had been visited from a child, according to Algorithm 41. In this case, the set of operators are going to behave as follows, as graphically represented in Figures 33, 34, 35 and 36.

- Delete rule operator. As the clause whose head represents such a NOC node is deleted, the flow of influence through this path is withdrawn. Thus, this indirect causal/effect influence to the misclassified instance brought by this node is removed from the network. While this is a radical approach, it can reduce the size of the BLP and consequently of the Bayesian networks.

- Add antecedent operator. A NOC node passes ball to its parents, in an attempt to discover further influence to itself. Adding literals to the body of such a
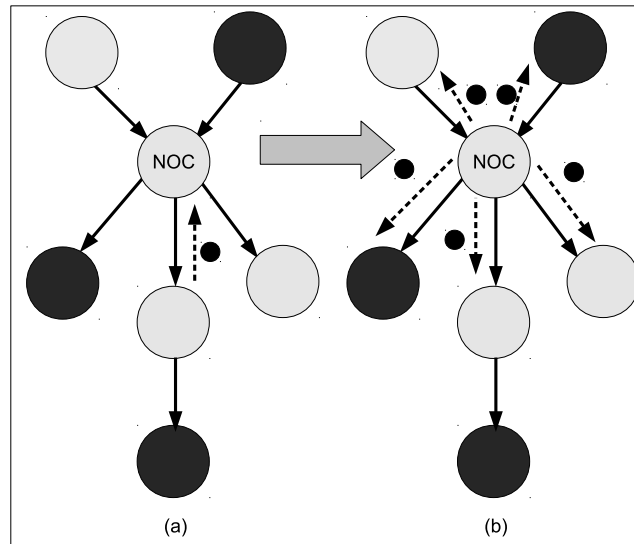
Figura 32: Non-observed node visited from a child, where shadow nodes are observed nodes. Figure (a) shows a non-observed node visited from a child. Figure (b) shows this node passing the received ball to its parents and children.
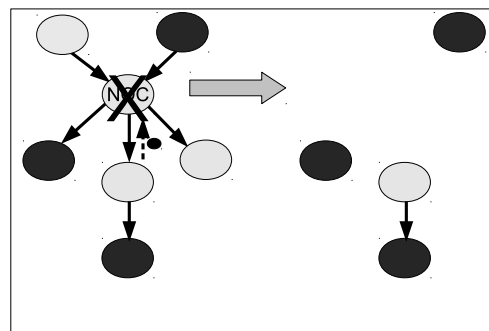

Figura 33: An example of the effect of deleting a rule corresponding to a non-observed node visited from a child, in a misclassification node visiting scenario.

clause is going to add additional parents to clause head, and consequently a new flow of influence might be brought in, in the case of the examples remain provable after the modification. On the other hand, influence passing through this node is removed in the case of an example that becomes unprovable, similar to what happens in del rule operator. Note that this is different from a standard Bayesian operation, since here changing the set of clauses covering an instance is also going to change the Bayesian network built from the example.
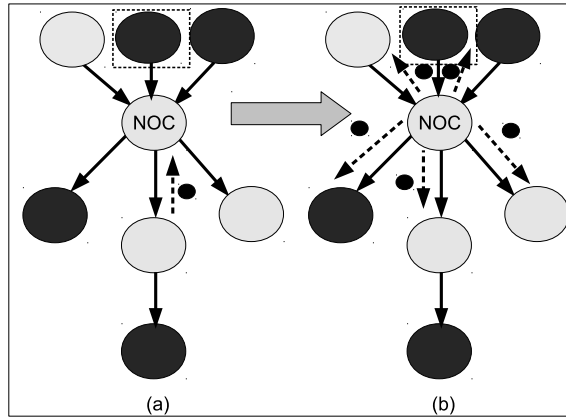
Figura 34: An example of the effect of adding antecedentes to a rule corresponding to a non-observed node visited from a child, in a misclassification node visiting scenario.

- Delete antecedents operator. In the same spirit to add antecedents might bring new profitable influence over the node, deleting antecedents may remove nodes whose influence through it from one of its parents contributes to wrong prediction.
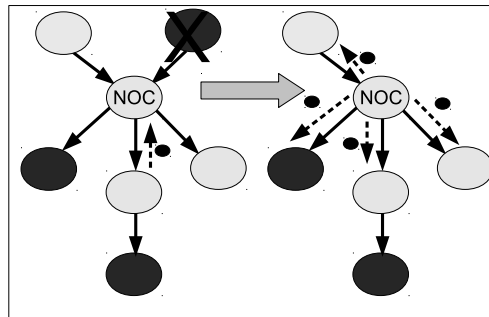

Figura 35: The effect of deleting antecedentes to a rule corresponding to a non-observed node visited from a child, in a misclassification node visiting scenario

- Add rule. As a new rule shares the set of satisfied examples with the old rule making rise to a NOC node, two situations may arise: (1) the new rule and old rule(s) give rise to the same ground atom in their head, which is going to be included in the support network. In BLPs, this require that the final probability value is obtained through a combination function. The second

possible situation (2) happens when the new and old rule do not share the same ground atom in their heads. In this case, the node coming from the head of the new rule may bring new influence, as it is likely to become a mate to the NOC node.
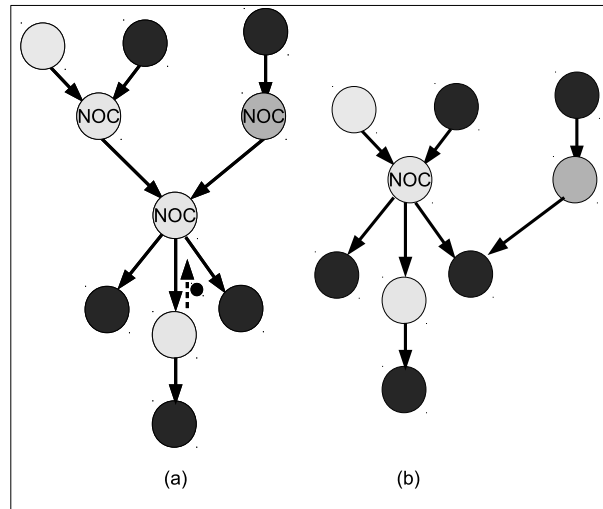

Figura 36: The effect of adding a new rule from a non-observed node visited from a child, in a misclassification node visiting scenario. Figure (a) shows a situation where the old and new rules had to be combined. Figure (b) shows the case where both rules were not combined.

Observed node visited from a child (OC node)   Observe in Algorithm 41 that an observed node visited from a child blocks balls from children, i.e., if a ball comes from one of its children it is not passed anymore from the OC node. This happens because the evidence of the node makes the path from its children through it inactive. Therefore, an OC node d-separates its child of the others nodes in the graph, i.e., nodes which could be reached if it was possible to pass through an OC node. This is exhibited in Figure 37, where the arrows indicate the direction of the visit. Although OC nodes are marked as a revision point, since they are not considered as irrelevant for the misclassified instance, not all operators are going to propose profitable modifications to change/modify the influence exerted by them.

- In case the rule whose head corresponds to an OC node is *deleted* from the BLP, its child would lose its influence. As this last node is considered a requisite

node, this operator may bring benefits to the model, in case the influence of such a node is bad to most of the query nodes. Therefore, this is an eligible operator to modify the set of influent connections to the misclassified instances.

- At first sight *adding antecedents* to the body of a clause whose head makes rise to an OC node, does not create new flow of influence, since any of its parents is blocked by its evidence. However, we must pay attention to the fact that those nodes are originated from Bayesian clauses, which are in essence logical clauses. In this way, add antecedents operator may remove some instances from the set of examples proved by the clause. The possible consequence is that the head of the clause is not able anymore to produce the OC node indicating a revision point. This is similar to the case of *delete rule* operator.

- Deleting antecedents from the body of a clause has two possible consequences in this case. First, the OC node is going to lose a parent. As its parents are blocked by its own evidence, this brings no changes in the flow of influence. Second, because a clause that has had one or mode antecedents deleted is more general than before, additional proof paths from the same clause may arise to build the Bayesian network. The ground heads of those additional paths either are the same as the previous OC node, or they induce new ground heads. In the first case, those equal nodes must be combined but besides sharing the same CPD (as they come from the same clause), they will also continue to contribute with the same evidence (as they come from the same ground fact). On the other hand, in case the modified clause produces new ground heads there is a chance that they are going to bring new influence to the child from whom the ball came. Consider, for example, a meaningless ground clause below that has given rise to an observed node in the network:

$$pred1(1,2) : -pred2(1,3), pred3(3,2), pred4(2).$$

Now, suppose $pred3(X,Y)$ is chosen to be deleted from the original clause. One possible consequence would be the appearance of a new ground clause with the same ground head as before, say

$$pred1(1,2) : -pred2(1, \_), pred4(2).$$

that had not appeared before because there is no $pred3(4,2)$ in the dataset. Note that both ground clauses have the same head and accordingly they are going to be combined. However, the random variable $pred(1,2)$, continues to be an observed node visited from a child, therefore, it is still a "blocking"node. On the other hand, it may be the case a new ground rule is produced after the deletion of the antecedent $pred3(X,Y)$, for example:

$$pred1(1,5) : -pred2(1, \_), pred4(5).$$

Such a clause could not be produced before because there is no $pred3(\_,5)$ in the background. This clause has a chance to bring new influence to the node from which the ball come, as $pred1(1,5)$ may become an ancestor of it.

- Add rule. Similarly to deleting antecedents case, adding new rules may bring additional evidence to the network.
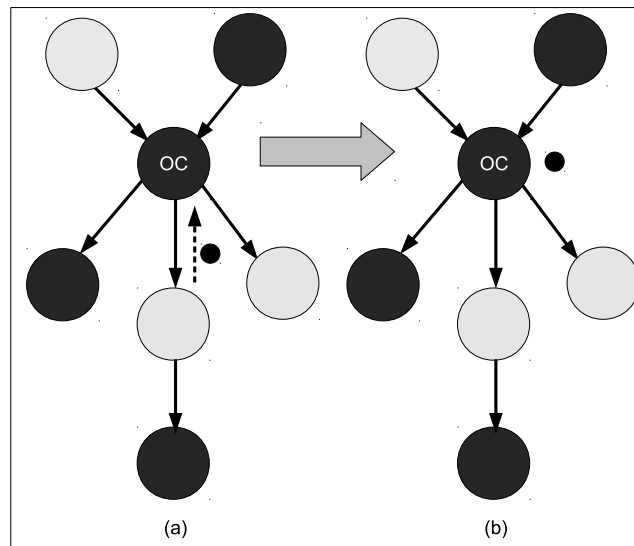

Figura 37: Observed node visited from a child, where shadow nodes are observed nodes. Figure (a) shows an observed node visited from a child. Figure (b) shows this node blocking the ball.

Non-observed node visited from a parent (NOP node)   Besides collecting evidence for itself, a non-observed node could be visited from its parent in an attempt to create an indirect effect path through one of its children, as can be seen in Figure 38. However, if this is really the case such a node would also be marked as non-observed visited from a child (NOC node) and therefore would attend a previous discussed case. Thus, deleting the rule corresponding to such a node head would only make the BLP, and consequently the Bayesian networks more compact. Similarly, adding antecedents to its body could only bring influence to their children, but this is also treated by the visit coming from a child case.

The operators left are the ones involving logic generalization. First, one may try to delete antecedents from such a clause in an attempt to generate new proof paths for some instance and consequently new ground heads forming a "brotherhood"relationship with the NOP node. In case the node visiting the NOP node remains as its parent (it is not one of the deleted antecedents), there will be a link to the node(s) coming from such new instantiations. There is a chance those new nodes bring together new evidence, as they are going to represent different ground facts of the dataset. The exact same situation may happen when adding new rules from such a NOP node. In this case, the link to the parent will exist if the visiting parent is also an element in the body of the just created clause.
Consider, for example the ground clause

$$pred(1, 2)|pred1(1, 3), pred2(3, 4), pred3(4, 2), pred4(2, 2).$$

and assume the node yielded from $pred(1, 2)$ is a NOP node, visited from the node produced from $pred1(1, 3)$. Now, consider that the Bayesian predicate $pred3$ is deleted from the clause. This could generate a new ground clause, say

$$pred(1, 4)|pred1(1, 3), pred2(3, 5), pred4(4, 4)$$

since $pred(1, 4)$ could not be proved by this clause before because there is no $pred3(5, 4)$ in the dataset. Note in Figure39 that there could be a new common cause trail in the network, due to this modification.
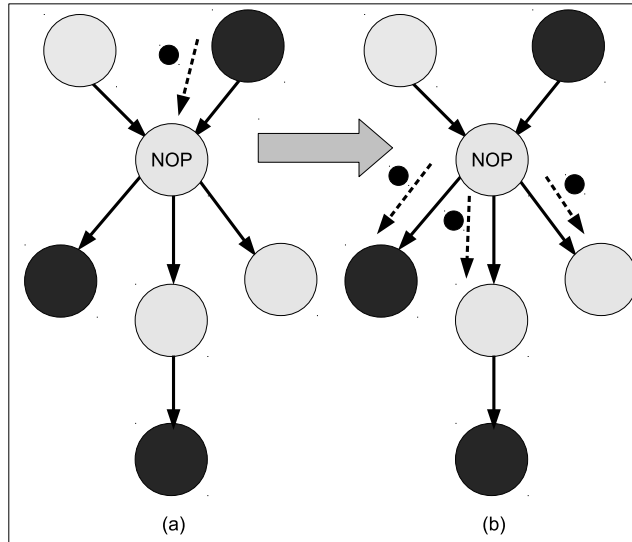
Figura 38: Non-observed node visited from a parent, where shadow nodes are observed nodes. Figure (a) shows a non-observed node visited from a parent. Figure (b) shows this node passing the ball to its children.
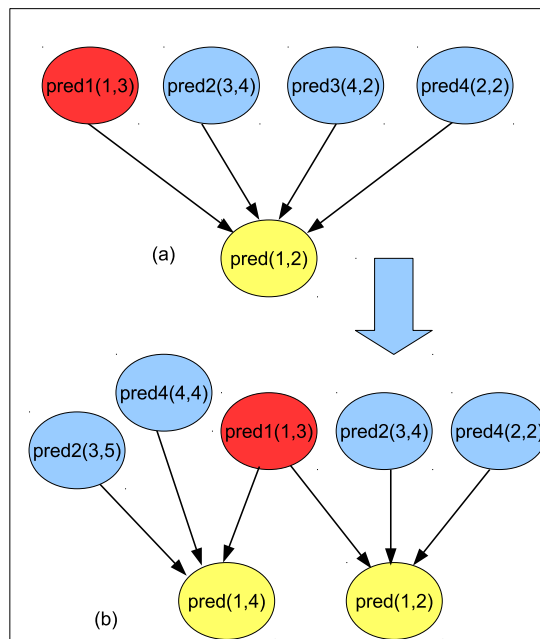

Figura 39: The effect of deleting antecedents from a clause whose head is a non-observerd node visited from a parent. Figure (a) is the network before deleting antecedents. Red node is the visiting parent and yellow node is the visited node. Figure (b) shows a possible resulting network, after the clause becomes more general.

Observed node visited from a parent (OP node)   The last case concerns an evidence node which is visited from a parent and bounces the ball back to its others parents, in an attempt to produce a common effect scenario, as the one presented in Figure 40. All operators may change the active paths as it is discussed next.

- Deleting the rule represented by such a node is going to cut it off the active path discovered by the visit. If the OP node contributes to a wrong prediction in this active path, removing the node from the network can repair misclassified instances.

- Deleting antecedents is beneficial for two reasons: firstly, as in previous cases, the generalization of the rule can bring additional active paths built from new instantiations of the rule, that did not exist before because of the antecedent(s) deleted. Secondly, deleting antecedents from the rule may remove the link to the visiting parent and, similar to the delete rule case, withdraw an influence contributing to the wrong prediction, exerted by the OP node. Moreover, deleting antecedents from such a node can remove common effect paths.

- Adding new rules has similar effects as the delete antecedents case: proofs not existing before may bring additional influence to the path, in case active path are also formed by the instantiations of the new rule and new common effect paths may arise from the added rule.

- Adding antecedents may bring two different consequences: the first one is similar to the result of deleting the rule, and may remove the link to the visiting parent, by making the clause more specific after some antecedents are added to it. Thus, the previous proof conducting to the parent node would fail. The second consequence is that adding antecedents in the body of such a clause might create further common effect influence path, which could be useful in correcting the misclassification of the example because of new evidence.

Remarks about Revision Operators and D-connection

One can see that are several issues concerning the real benefits a revision operator might bring. For example, some of the modifications are only likely to create/remove
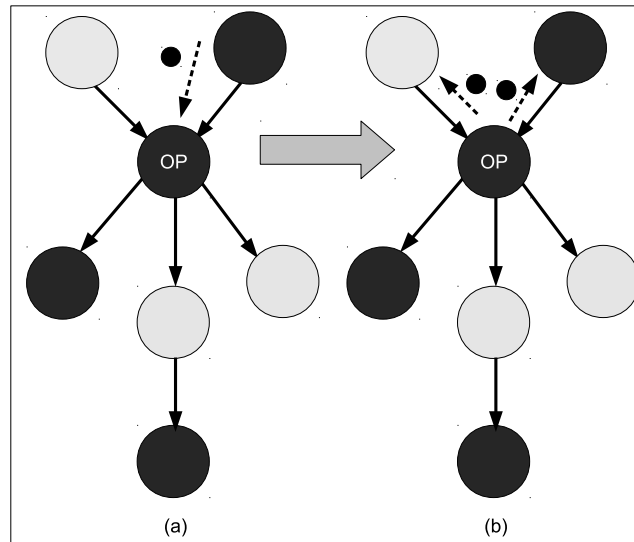
Figura 40: Observed node visited from a parent, where shadow nodes are observed nodes. Figure (a) shows a observed node visited from a parent. Figure (b) shows this node passing the ball back to its parents.

a connection when the set of ground Bayesian clauses changes after the revision. Moreover, some modifications are likely to bring/remove influence to fix misclassification, but this is not guaranteed, as they can/cannot make new active paths depending on the evidence. Moreover, some modifications depend whether a literal continues to be in the body of a clause, making sure a connection in the network is not removed. This can be true for one instance, but may not happen in other instances. As to check all possible cases is expensive, encompassing not only connections that are going to be removed but also of the active paths that are going to be created, we only disregard the operators guaranteed to not change influence that could fix the misclassified instance. Anyway, by computing the score of the revision it is verified whether the modification is going to improve the current BLP.

Notice that the revision operators used in this work do not compose a complete set of possible modifications to the BLP. To give an example, note that new active paths could be created by adding rules or antecedents leading a parent to visit and/or be visited from a child. To achieve this goal, the revision would be able to mark revision points not only throughout the Bayesian networks but also from the knowledge base itself, by identifying all possible determinations leading to new

connections coming from children to misclassified instances in the graph. Suppose, for example, that we have the Bayesian clause below, marked as revision point.

$$pred1(A, B) : -pred2(A, B).$$

It would be possible that a non existing clause could provide new information to $pred1/2$, say a clause such as

$$pred3(A, B) : -pred1(A, B).$$

As $pred3/2$ does not represent any instance in the examples and there is not an existing connection between pred3 and pred1, such a clause would not be proposed by the revision system.

BFORTE and its antecessor PFORTE cannot identify those possible new connections, as Bayesian revision points are identified through paths existing at the moment that an instance is misclassified. In this way, we cannot guarantee that we implement the complete set of possible modifications to improve classification in a BLP.

Finally, the BLP could be compacted by identifying clauses that are completely useless to any example in the dataset. Those correspond to nodes identified as irrelevant by Bayes Ball algorithm. To achieve this, Bayes Ball would have to run for each instance in the dataset. Then, the intersection of irrelevant nodes would correspond the clauses that could be safely removed from the BLP.

Algorithm 50 exhibits the procedure for applying revision operators to revision points, extended from lines 3-7 of Algorithm 48. It brings an optimization on the use of the four revision operators. First, when looking for revision points, it is necessary to keep the type each node matches, according to the four groups listed before. As the marked node is mapped to a Bayesian revision point (a Bayesian clause), the fact it is observed or not, visited from a child or from a parent, is stored together with the revision point. From our analysis, only deleting rules and adding antecedents to a non-observed evidence are guaranteed to not present any chance to change the connections to the misclassified instance. In all other cases, the proposed revision may change the Bayesian network. Therefore, before proposing the revision, it is

necessary to be sure that a Bayesian revision point is not identified only through that kind of visit.

Algorithm 50 Generation of Revision Points and Revisions Top-Level Procedure

---

Input: Set of Bayesian networks BN built from the dataset and current BLP
Output: Set of revisions Proposed to Bayesian revision points
 1: $BRP \leftarrow$ Bayesian revision points identified through Algorithm 49, slightly modified to keep around the type of the node (OC, NOC, OP, NOP)
 2: **for** each Bayesian revision point $BR \in BRP$ **do**
 3:     **if** type of BR is not only NOP **then**
 4:         propose modification to BR using all four operators
 5:     **else**
 6:         propose modifications to BR using delete antecedents and/or add rules operator
 7:     compute the score for each proposed revision

Next section we discuss how to reduce the amount of the time expended in the revision by reducing the inference space.

Addressing Inference Space

In order to build Bayesian networks from the set of examples, PFORTE collects *all* ground clauses taking part in every possible way of proving instances and evidences in an example. Those ground Bayesian atoms become nodes in the Bayes net, and they are connected when there is a direct influence between them. As the instances and evidences in a single example are not i.i.d, each Bayesian network may have a large number of nodes and edges connecting evidences, instances and non-evidence nodes, leading to high inference run times.

Aiming to reduce the space where inference is performed, it is necessary to diminish the size of nodes to be considered by the inference engine. Making inference more efficient in relational probabilistic systems has been pursued by the development of lifted inference methods (POOLE, 2003; SINGLA, DOMINGOS, 2008; MILCH, et al., 2008; MEERT, et al., 2010), which mostly reduces the networks by exploring symmetries and grouping variables with identical behavior. Here, we decrease the search space of the inference procedure by disregarding non requisite nodes of an instance. Non requisite nodes can be deleted from the network and we

would still be able to resolve the query, as they are unable of influencing the instance. In addition, clauses and networks exhibiting identical behavior are grouped together so that they dot not have its score and parameters computed repeatedly unnecessarily.

Collecting the Requisite Nodes by d-separation

As said before, the networks built from examples are usually very large, since they are composed of the union of all possible proofs of each instance and each evidence. However, it is usually the case that for computing marginal probabilities for one instance only a subset of that large network is really relevant. We call the network composed of this subset of nodes as *minimum requisite network* for a query.

Definition .12 *The minimum requisite network g for a query node n is a subset of the original large network G built to the whole example, that contains only the requisite nodes necessary to compute the marginal distribution of n. All the others nodes in G can be safely disregard so that the query is still resolved by g.*

We make use of Bayes Ball algorithm (SHACHTER, 1998) to find out which nodes in the network are required to compose the minimum requisite network. In this way, instead of performing inference in one large network, the inference takes into account only a small subset of that network, namely the nodes which are really relevant for resolving the query. To collect such requisite nodes, the algorithm starts from one instance *id* in the original example and uses Bayes Ball to identify the minimum set of nodes required to compute the probability of *id*. This set is composed of visited observed nodes and those nodes marked in the top. Following collective inference (JENSEN, et al., 2004), the other queries instances in the example are treated as evidences when collecting the requisite nodes for *id*. Algorithm 51 brings the top-level procedure to identify requisite nodes and perform inference only over them.

Algorithm 51 is called at each time it is necessary to compute marginal probabilities, namely, to learn parameters and compute scores, when changes occur in the structure of the BLP.

Algorithm 51 Top-level Procedure for Performing Inference over Requisite Nodes
Only

1: **for** each example $D_i \in D$ **do**
2:      build the graph GD_i for D_i, joining support networks built from proof tree
        for each instance and evidence
3:      **for** each instance $id \in D_i$ which is a node nid in GD_i **do**
4:          consider nid as a query node
5:          visit nodes in GD_i starting from nid, using Bayes Ball algorithm
6:          $requisite \leftarrow$ visited observed nodes $\cup$ nodes marked in the top
7:          compute probability distribtion for id using only $requisite$

Bayes ball was also used in SRL context to build up a lifted inference engine in (MEERT, et al., 2010). There, they apply the Bayes Ball directly in the ground graph construction procedure. We are going to investigate in the future whether that algorithm can save further time to the inference process.

Grouping together Ground Clauses and Networks

When constructing Bayesian networks relative to examples, every successful proof of an Bayesian atom is gathered together to build the graph of the net. However, it is often the case that a large number of those ground clauses shares exactly the same features: they are instantiated from the same Bayesian clause, and therefore they have the same conditional probability distribution; additionally, they have the same evidence associated to each of its Bayesian ground atom. In this way, the only difference among them would be the terms of ground Bayesian atoms. However, for the Bayesian network, atoms are mapped to nodes, which cannot "see"those terms anyway. Therefore, it is a waste of resources to represent each Bayesian atom obeying these conditions as different nodes in the graph. Consider, for example, the following meaningless ground clauses.

$$pred(A, B) : -pred1(C, A), pred1(C, B).$$

and suppose that for a ground atom $t1, t2$ we have several possible instantiations for the variable C, producing, for instance

$pred(t1, t2) : -pred1(r1, t1), pred1(r1, t2).$

$pred(t1, t2) : -pred1(r2, t1), pred1(r3, t2).$

$pred(t1, t2) : -pred1(r3, t1), pred1(r3, t2).$

...

$pred(t1, t2) : -pred1(r30, t1), pred1(r30, t2).$

Now, suppose that, say 25, of these share the same evidence for $pred1(X, t1)$, $pred1(X, t2)$. Instead of mapping each one of those nodes above to a separate node in the Bayesian network, we could put together the $25X2$ nodes of the same evidence and $5X2$ separate nodes.

Thus, to prevent this waste of resources performed by BLP network construction, we developed a procedure for grouping together different instantiations of the same Bayesian clause that behaves in exactly the same way. This group of similar nodes make rise to a *mega node* in the network, instead of only a single node.

Definition .13 *Let SN be the set of nodes originated from different successful proof paths created using Bayesian clauses Bcc and an instance ins. A mega node is defined as the grouping of a set of nodes in SN, which are relative to the same Bcc, and that shares the same evidence.*

Because of that, a random variable in the graph is not necessarily a single ground Bayesian atom, but could be representative of a group of several Bayesian atoms. The number of atoms mapped to the same node is kept so that it is taken into account when learning parameters. Notice that we employed the same schema of combining rules as (NATARAJAN, et al., 2008), where instantiations of clauses with same head are combined in two levels: one level considers different instantiations of the same rule, while another level considers different rules with the same ground head, as explained in section 7.2.3. Thus, the clauses we mapped here to the same nodes would be combined anyway in the first level we just mentioned. However, although they would contribute in the same way for the final probability, since they share the same association of evidence values and CPDs, the inference engine could not have knowledge on that and it would compute their probability over and over again. Moreover, as decomposable combining rules are applied in BLPs, each different instantiation would be seen as a "separate experiment", by adding extra nodes in the network to represent them. By keeping the amount of ground atoms

308

overlapped together in a mega node, those nodes are still contributing as a separate experiment for the computation of parameters.

We also employ another way of overlapping components so that inference engines have to deal with a smaller space. This is the case when more than one graph have the same properties: the nodes are coming exactly from the same clauses, and evidence for those nodes are also the same. This is naturally a situation less frequent than the one we discussed immediately before, but still it can also happens mainly when Bayes Ball is applied to split the original networks in groups of smaller ones. To map more than one network to the same, it is required that (1) they have the same number of nodes; (2) they have the same edges connecting each pair of nodes (3) each group of child + parents are relative to the same Bayesian clause, so that the probability distribution associated to them in the different graphs are the same, and (4) the nodes have the same value of evidence, in case they are observed. The instances relative to networks grouped to one overlapped *mega network* are kept around, so that it can be taken into account when learning parameters or computing probabilities.

We show in Algorithms 52 and 53 the procedures for detecting and grouping together groups of nodes with the same behavior.

Algorithm 52 Top-level Procedure for Detecting Similar Ground Clauses and Mapping them to Only One

---
1: **for** each ground clause $clause_i\theta1 \in$ proof trees of the same example **do**
2:      **for** each ground clause $clause_j\theta2, i \neq j \in$ proof trees of examples **do**
3:          **if** $clause_i$ comes from the same Bayesian clause as $clause_j$ **then**
4:              **for** each Bayesian atom $a_{ki} \in clause_i\theta1$ and $a_{kj} \in clause_j\theta2$ **do**
5:                  **if** value associated to $a_{ki} \neq$ value associated to a_{kj} **then**
6:                      go to line 2
7:              remove $clause_j\theta2$ from proof tree
8:              increase number of mapped clauses associated to $clause_i\theta1$

Experimental results

This section we experiment our revision system BFORTE, by comparing each contribution developed in this chapter with a version of the system without the contribution. As there is no current implementation of BLP structure learning algorithm,

---
Algorithm 53 Top-level Procedure for Detecting Similar Networks and Mapping them to Only One
---
1: **for** each network N_i **do**
2:     **for** each network $N_j, i \neq j$ **do**
3:         **if** $DAG(N_i) == DAG(N_j)$ **then**
4:             **for** each node $n_{ki} \in N_i$ and node $n_{kj} \in N_j$ **do**
5:                 **if** n_{ki} comes from a different clause as n_{kj} or n_{ki} has an evidence value different from n_{kj} **then**
6:                     go to line 2
7:             increase number of group of nets associated to N_i
8:             associate instance(s) of N_j to corresponding node(s) of N_i
9:             disregard N_j
---

we use our own system to make the due comparisons. We opted by showing each contribution separately, so that it is possible to know the benefits brought by them.

Datasets We have considered datasets used in ILP and SRL communities. They are briefly described as follows.

1. UW-CSE is a vastly used dataset in SRL community (SINGLA, DOMINGOS, 2005; RICHARDSON, DOMINGOS, 2006; KOK, DOMINGOS, 2007). It consists of information about the University of Washington Department of Computer Science and Engineering. There are 5 examples, where each one of them contains instances representing a relationship of *advisedby* for a different research line of the department. There are 113 instances of the positive class and 2711 instances of the negative class, and 2673 ground facts.

2. Metabolism is based on the data provided by KDD Cup 2001 (CHENG, et al., 2002b). The data consists of 6910 ground facts about 115 positive instances and 115 negative instances. As part of the facts represents the interaction between genes, we gathered together in one examples all the positive and negative instances.

3. Carcinogenesis is a well-known domain for predicting structure-activity relationship (SAR) about activity in rodent bioassays (SRINIVASAN, et al., 1997a). There are 162 instances from the positive class and 136 instances from the negative class and 24342 ground facts about them. This dataset is

totally separable, as each example is composed of only one instance.

Experimental Methodology The datasets were splitted up into 5 disjoint folds sets to use a K-fold stratified cross validation approach. Each fold keeps the rate of original distribution of positive and negative examples (KOHAVI, 1995). To avoid overfitting during the revision process, similar to (BAIÃO, et al., 2003), we applied 5-fold stratified cross validation approach to split the input data into disjoint training and test sets and, within that, a 2-fold stratified cross-validation approach to split training data into disjoint training and tuning sets. The revision algorithm monitors the error with respect to the tuning set after each revision, always keeping around a copy of the theory with the best tuning set score, and the saved "best-tuning-set-score"theory is applied to the test set. The significance test used was corrected paired t-test (NADEAU, BENGIO, 2003), with $p < 0.05$. As stated by (NADEAU, BENGIO, 2003), corrected t-test takes into account the variability due to the choice of training set and not only that due to the test examples, which could lead to gross underestimation of the variance of the cross validation estimator and to the wrong conclusion that the new algorithm is significantly better when it is not.

The initial theories for Carcinogenesis and Metabolism were obtained from Aleph system using default parameters, except for clause length, which is defined as 5, noise, defined as 30, and minpos, set to 2. As UW-CSE is a highly unbalanced data, we use m-estimate as evaluation function and noise set for 1000. To generate such theories, the whole dataset was considered but using a 5-fold cross validation procedure. Thus, a different theory was generated for each fold and each one of these theories is revised considering its respective fold (the same fold is used to generate and revise the theories).

All the experiments were run on Yap Prolog (SANTOS COSTA, 2008) and Matlab. To handle Bayesian networks, we have re-used Bayes Net Toolbox (MURPHY, 2001), properly modified to tackle the particularities of BLPs. Combining rules are represented in two levels, where the first level (different instantiations of the same clause) uses *mean* as combining rule and the second level (different clauses with the same head) uses *weighted−mean* as combination function. To learn param-

eters, we implemented discriminative gradient descent described in (NATARAJAN, et al., 2008) that minimizes mean squared error. For the UW-CSE, we use a weighted MSE, with the weight inversely proportional to the distribution of the classes, so that the parameters do not totally favor negative class because of its amount of examples. To perform inference, we adapted variable elimination inference engine to handle combination rules. We impose a minimum number of misclassified examples as 2 to a clause be considered as revision point, in order to avoid outliers. Threshold for indicating if an instance is correctly predicted is defined as 0.5.

Comparing BFORTE to ILP and FOL Theory revision

First of all, we would like to know whether BFORTE achieves better score results than standard first-order logic systems. We compared BFORTE after learning the initial parameters (after line 1 of Algorithm 48) and BFORTE after revising structure to Aleph and FORTE. The same theories Aleph learn are provided to both revision systems. The first two columns of Table 19 present the results of conditional log-likelihood achieved after learning initial parameters and after the revision process is finished, respectively. Third and fourth column present the score of BFORTE after learning initial parameters and after the revision process is finished. The last two columns presents the score of Aleph and FORTE, respectively. Bold faces indicate the best score results obtained from all systems. The symbol $\diamond$ indicates the cases where score of BFORTE is significantly better than score of Aleph. The symbol $\bullet$ indicates the cases where BFORTE is significantly better than FORTE. Finally, $\star$ emphasizes the cases where it was possible to obtain an improvement after revising the structure.

Tabela 19: BFORTE compared to Aleph and FORTE

System/ Dataset	BFORTE Params CLL	BFORTE Struct CLL	BFORTE Params Score	BFORTE Struct Score	Aleph Score	FORTE Score
UW-CSE	-0.2142	-0.0746$\star$	0.3504	**0.5363**$\diamond \star \bullet$	0.3684	0.2864
Metabolism	-0.784	-0.6912$\star$	60.00	**64.76**$\diamond \star \bullet$	56.51	59.57
Carcinogenesis	-0.6712	-0.6618	59.38	**61.30**$\diamond \bullet$	55.0	55.70

From the table we can see that BFORTE, by revising the structure, can im-

prove the probabilities of examples, since conditional log-likelihood has significant increased in two of the three cases. The final value of the score function is better than the first-order systems in all cases, showing that it is possible to obtain more accurate systems when uncertainty is taken into account. Moreover, in two cases the revision in structure improved the initial score. Although learning parameters has improved the initial score of Carcinogenesis compared to first-order systems, the final score is not significantly better than the initial score.

Speed up in the revision process due to the Bottom clause

Now, we would like to verify whether introducing the Bottom Clause as space of literals can decrease the runtime without harming the score. To focus on this issue, we run BFORTE using the Bottom Clause procedure and BFORTE using FOIL algorithm to generate literals. Results of runtime and score are exhibited in Table 20. As expected, UW-CSE performs significantly faster when the Bottom Clause is the search space of literals. Surprisingly, the score is also better in the Bottom Clause case, although the difference is not significant. By analyzing the results of each fold, we found out that one fold has a higher score for the FOIL case, as it adds to a clause one literal that has not appeared in the Bottom Clause of the chosen example in BFORTE case. A similar situation also happens in another fold, but this time the literal added to the clause in one of the iterations makes the theory performs worse in the validation set. As a result, BFORTE has a better score for this fold.

Unfortunately, we were not able to collect FOIL results in the other two datasets, as the system runs out of memory after some time running. The reason, besides the much larger search space generated for FOIL approach, is that differently from UW-CSE, Metabolism and Carcinogenesis have several predicates with constants defined in mode declarations. The top-down approach of FOIL cannot generate literals with constants, and then a variable is put in the place of a possible constant. The problem that arises is the large number of instantiations of the same literal, yielding a huge amount of different proof paths for the same literal. Consider, for example, the following mode definition of a predicate in Carcinogenesis domain.

$modeb(*, atm(+drug, -atomid, \#element, \#integer, -charge)).$

Types *element* and *integer* are defined as constants. Bottom Clause construction procedure is able to find a constant to the place of such mode, since it is created from a particular instance. Those constants are going to limit the ground facts that can be unified with this literal. However, FOIL puts variables in third and fourth terms. As a consequence, every possible $atm/5$ ground literal in the dataset is able to unify with the variabilized new literal, producing in certain cases a huge amount of different proofs that Matlab cannot handle. Note that, this is an additional complexity of BLPs compared to the first-order case, as in this last it is not necessary to collect all possible proofs explaining an example.

Tabela 20: Comparison of runtime and accuracy of BFORTE with Bottom Clause and BFORTE using FOIL to generate literals.

| | BFORTE | | BFORTE without BC | |
| | Learning Time | Score | Learning Time | Score |
Dataset				
UW-CSE	8061.07⋆	0.5363	10884.16	0.5159
Metabolism	7184.15	64.76	N/A	N/A
Carcinogenesis	8812.16	61.30	N/A	N/A

Selection of Revision Points

In this experiment, we focus on the question of whether the BFORTE approach to select revision points can decrease learning time. Due to the limitations of PFORTE and BLP, specially concerning the search space of literals (previous section), we compare BFORTE to BFORTE simulating those algorithms with regard to the selection of revision points. Notice that this is necessary, specially because those systems cannot run in two cases without bounding the search space to the Bottom Clause, as we can see in Table 8.2. The compared settings are as follows.

- BFORTE: this is the system implemented upon all the contributions presented in this chapter: the Bottom clause is used to bound the search space of literals, revision points are selected using Bayes Ball and inference procedure is

314

optimized.

- PFORTE-like: this case considers Bottom clause to bound the search space and optimized inference. However, selection of the revision points follows PFORTE system, where all clauses appearing in the network of a misclassified instance are marked as revision points.

- BLP-like: this case considers Bottom clause to bound the search space and optimized inference. However, selection of the revision points follows BLP structure learning algorithm, where all Bayesian clauses in the BLP are refined.

We call the last two cases as PFORTE-like and BLP-like because original PFORTE and BLP learning algorithm *have none of the improvements* we designed in this chapter.

Table 21 shows the results of runtime, accuracy and number of revised clauses for each setting. The symbol $\star$ indicates the cases where there is significant difference between BFORTE and PFORTE-like and symbol • indicates a significant difference between BFORTE and BLP-like. Runtime of BFORTE is significantly better than both PFORTE-like and BLP-like. Note that PFORTE performs worse than BLP. This is due to the size of the network: as the network of UW-CSE is quite large and PFORTE selects all clauses taking part in a network of a misclassified instance, it always marks the whole theory as revision points. Thus, it has the same search space as BLPs. However, as BLPs modifies all Bayesian clauses and PFORTE performs inference to find out the misclassified instances, it expends more time to finish the revision process than BLP-like. On the other hand, the score of BFORTE is worse than PFORTE and BLP, although not significantly. The reason for that difference is that in one fold different theories are kept by the validation set during the learning.

Tabela 21: Comparison of BFORTE runtime and score with PFORTE and BLP algorithms for selecting points to me modified.

System/ Dataset	BFORTE			PFORTE-like			BLP-like		
	Learning Time	Score	#Revised clauses	Learning Time	Score	#Revised clauses	Learning Time	Score	#Revised clauses
UW-CSE	8061.07★ ●	0.5363	4.7	10517.49	0.5669	7.6	10734.23	0.5669	7.6
Metab	7184.15	-64.76	6.3	N/A	N/A	18	N/A	N/A	18
Carcino	8812.16 ★	61.30	3.8	8812.16	61.30	3.8	26922.49	61.76	8.0

316

Unfortunately, we cannot obtain results in reasonable time using either PFORTE or BLP settings for Metabolism dataset ($< 48h$). This is mainly due to the large search space that is explored, since, during the learning time that we could trace, they mark 18 clauses on average to be modified, which is the same number of clauses in the initial theory. The same situation of UW-CSE happens here comparing PFORTE and BLP: as this dataset yields a single large network, PFORTE marks every clause to be revised. BFORTE considers only 6.3 clauses on average to be revised, which is due to the smart selection of revision points conducted by Bayes Ball algorithm.

A different situation happens with Carcinogenesis. As the instances are not related in this dataset, there is a Bayesian network for each instance, yielding smaller individual networks than in previous cases. In this way, BFORTE and PFORTE-like select the same clauses as revision points. This is also because the clauses producing ascendant nodes to the top-level instances are derived from clauses in the fixed background knowledge, and therefore cannot be modified. Additionally, the ground facts produced by them are considered as non-observed. Hence, even though they were modifiable, they would non-observed parent nodes whose parents are observed and therefore Bayes Ball would also consider them as revision points. Finally, we see that runtime of BLP-like is much worse than revision systems, since in this case it alone marks all clauses from the theory as revision points.

Inference time

Finally, we show the reduction in runtime of the inference due to the use of Bayes Ball and the grouping of clauses and networks. Note that all cases previously discussed takes into account inference with Bayes Ball when learning or revising theories. Instead of running the whole revision process, we opted for running only the procedure that computes the discriminative score for each training set, considering the initial theories. In this way, we are able to see the *improvement in runtime inference alone*, without taking into account the particularities of the revision process. Additionally, running original *inference* of PFORTE and BLP makes the revision/learning process extremely slow.

Tables 22, 23 and 24 shows the runtime for computing accuracy in UW-CSE, Metabolism and Carcinogenesis, respectively, of five settings: BFORTE, with all improvements presented in section 5.3.2, BFORTE without collecting requisite nodes with Bayes Ball, BFORTE without detecting similar networks, BFORTE without overlapping same rules and inference performed in original PFORTE/BLP i.e, inference is performed without any of the algorithms presented in section 5.3.2. Let first focus on UW-CSE dataset. Observe that the runtime is reduced by a factor of 170 from the full BFORTE setting. The largest reduction is due to Bayes Ball, although grouping similar nodes also contributes. On the other hand, detecting similar networks does not help in the reduction of the runtime. In two folds Matlab runs out of memory for PFORTE/BLP. Metabolism dataset has similar results of UW-CSE, since they both are highly relational datasets, but in this case we were able to collect the values for all training sets.

Tabela 22: Inference runtime in seconds for UW-CSE dataset.

Setting/ Training set	BFORTE	BFORTE, without Bayes Ball	BFORTE without similar nets	BFORTE without grouping rules	PFORTE/ BLP
1	18.44	1247.72	20.28	74.62	N/A
2	4.16	89.36	4.40	15.34	443.62
3	4.56	92.80	4.74	23.92	693.98
4	22.86	1880.38	25.30	97.94	N/A
5	3.44	66.76	3.52	15.00	443.72

Tabela 23: Inference runtime in seconds for Metabolism dataset.

Setting/ Training set	BFORTE	BFORTE, without Bayes Ball	BFORTE without similar nets	BFORTE Without Grouping rules	PFORTE/ BLP
1	10.98	677.62	676.90	23.30	1505.58
2	9.16	541.82	542.44	19.90	1103.22
3	13.02	827.56	826	29.92	1990.48
4	8.24	487.14	485.78	15.68	958.98
5	11.44	697.44	696.70	21.30	1330.40

The inference runtime of Carcinogenesis it is not different from PFORTE and BLP. In fact, in most of the cases, the runtime is slightly worse, due to the cost of executing Bayes Ball. However, the fourth training set has a particular behavior

when running BFORTE without grouping rules. In this case, one of the Bayesian clauses yields a large amount of ground clauses to be part of the final network. Without grouping similar rules, the final Bayesian network is too large for Matlab to handle.

Tabela 24: Inference runtime in seconds for Carcinogenesis dataset.

Setting/ Training set	BFORTE	BFORTE, without Bayes Ball	BFORTE without similar nets	BFORTE Without Grouping rules	PFORTE/ BLP
1	5.24	4.36	4.12	6.66	5.08
2	5.34	4.44	4.30	7.50	5.78
3	4.64	3.92	3.70	6.04	4.68
4	9.70	8.96	8.64	N/A	N/A
5	4.48	3.76	3.46	9.48	6.96

Conclusions

We addressed in this chapter the bottlenecks of Bayesian Logic Programs revision. We showed through experiments that it is possible to obtain a feasible revision system that provides more accurate models than first-order logic systems, when the domain is uncertain.

First we focused on the reduction of new literals search space. The baseline system, PFORTE, generates literals following a FOIL's top down approach: all possible literals from the language that had at least one common variable with the current clause were considered to be added to a clause. We were able to reduce the search space of literals by defining it as the Bottom Clause of a misclassified instance. We show in the experiments that the search space is reduced without harming the score achieved by the system. As a single misclassified instance may not carry sufficient information to produce good literals, in the future we would like to investigate the use of more several instances to create the Bottom Clause. Additionally, we could also take into account the probabilistic information to create/remove literals from the Bottom clause, for example by measuring the information that literals could pass to the clause (CHENG, et al., 2002a; OLIPHANT, SHAVLIK, 2008; PITAN-GUI, ZAVERUCHA, 2011). The work in (MIHALKOVA, MOONEY, 2007) employs

Bottom-up learning, by considering the network of examples to reduce the search space of possible refinements. We intend to investigate how such an approach could also be followed when refining Bayesian Logic Programs. It does not seem that there is a direct way of doing that, since we construct the Bayesian networks from the ground clauses, instantiated by the examples, whereas Markov Logical Networks defines the graph from the constants in the domain.

Next, we addressed the search space of clauses to be refined. BLP learning algorithm starts from an initial set of Bayesian clauses and proposes modifications to each one. PFORTE considers all the clauses used to build the Bayesian network of an example where there was a misclassified instance. Both systems generate a large search space that can even become intractable. We showed that it is possible to reduce this search space by marking as revision points only the clauses that influence the probability distribution computed for the misclassified instance. We used the Bayes Ball algorithm to identify those clauses. Experiments suggested that the search space is indeed reduced by proposing refinements only to the clauses relevant to the misclassified instance. In the future, we would like to investigate in more details if it is possible to reduce the number of revision operators according to the revision point, as we started to analyze in this chapter. Additionally, search for revision points guided by examples may not identify all possible places bringing new influence to a misclassified instance. In the future, we intend to investigate whether it is possible to have a good balance on efficiency and score when choosing revision points in promising places, but still outside the set pointed out by the instance.

Finally, we addressed the large inference search space due to the large Bayesian networks produced by PLL examples. First, we developed a procedure to only consider the requisite nodes identified by Bayes Ball algorithm, when performing inference. Second, we reduced the size of the network by identifying ground clauses whose only difference are the terms replacing variables. We argue that it is not necessary to represent them as separate nodes in the network, what would make the inference procedure to repeat several probabilities computations. Third, we identified that after selecting the requisite nodes, there are "subnetworks"with exactly the same structure and evidences. We avoid to compute the probabilities for all those

networks, by taking into account only one representative of the group. Experiments showed that the inference runtime is in fact greatly reduced by the algorithms we proposed. Note that the we only took a simple step in the direction of recent work on lifted inference (POOLE, 2003; SALVO BRAZ, et al., 2005; SINGLA, DOMINGOS, 2008; MILCH, et al., 2008; KOK, DOMINGOS, 2009; NATH, DOMINGOS, 2010; MEERT, et al., 2010). There is much more to be done and investigate to fulfill the advantages of lifted inference. For example, the ground clauses that are grouped together could represent more than one Bayesian clause, if the information they have are the same.

Through preliminary experiments, we noticed that the revision greatly depends on the initial theories. Aleph system may not be appropriate to learn initial theories for giving rise to BLPs. Indeed, the authors of BLP argued that Claudien system (DE RAEDT, DEHASPE, 1997), which learns from interpretations, is more adequate to produce most general clauses. In the future we intend to create several different theories, from different first-order systems and compare the revised BLPs, starting from them.

In the experiments reported here, the threshold for marking an instance as misclassified was set at 0.5, which is usually considered in binary domains. That value may not be the best for evaluating the models. Thus, we intend to create curves showing the behavior of the system with different thresholds. Also, an evaluation function that is independent from the threshold should be applied, such as the ones considering the area under the curve. Last, it is essential to compare BLPs to others SRL languages, such as Markov Logic Networks.

We conclude by observing that as stochastic local search has been showed to be quite effective in the first-order revision case, we intend to implement in BFORTE the same SLS techniques considered in YAVFORTE system.

Appendix H: The Laws of International Chess

Here we summarize the main rules of the traditional game of chess, set up by Fédération Internationale des Échecs (FIDE), also known as the World Chess Organisation (FIDE,).

1. Chess is a game played by two people on a chessboard divided into 64 squares of alternating colour, with 32 pieces (16 for each player) of six types. Each player has control of one of two sets of coloured pieces (white and black). White moves first and the players alternate moves. Each type of piece moves in a distinct way. The goal of the game is to checkmate, i.e. to threaten the opponent's king with inevitable capture.

2. At the beginning of the game the pieces are arranged as shown in Figure 41.

3. Each square of the chessboard is identified with a unique pair of a letter and a number. The vertical *files* are labelled a through h. Similarly, the horizontal *ranks* are numbered from 1 to 8. Each square of the board is uniquely identified by its file letter and rank number. The white king, for example, starts the game on square $e1$.

4. During the game, a piece may be *captured* and then removed from the game and may not be returned to play for the remainder of the game. The king can be put in check but cannot be captured.

5. Pieces can move as follows:

   - The king can move exactly one square horizontally, vertically, or diagonally. At most once in every game, each king is allowed to make a special

move, known as castling (see below).

- The rook moves any number of vacant squares vertically or horizontally. It also is moved while castling.

- The bishop moves any number of vacant squares in any diagonal direction.

- The queen can move any number of vacant squares diagonally, horizontally, or vertically.

- The knight moves in an "L" or "7" shape (possibly inverted): it moves two squares like the rook and then one square perpendicular to that.

- A pawn can move forward one square, if that square is unoccupied. If it has not yet moved, the pawn has the option of moving two squares forward provided both squares in front of the pawn are unoccupied. A pawn cannot move backward. Pawns are the only pieces that capture differently from how they move. They can capture an enemy piece on either of the two spaces adjacent to the space in front of them (i.e., the two squares diagonally in front of them) but cannot move to these spaces if they are vacant.
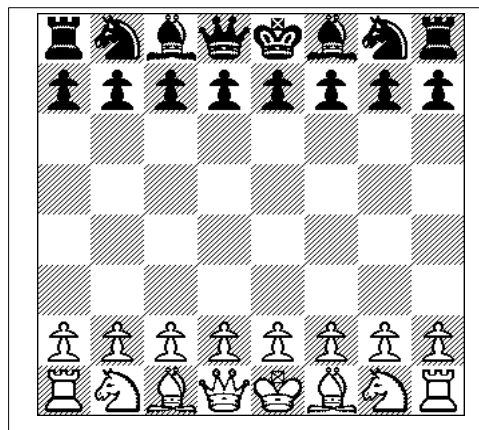
Figura 41: Initial position of a Chess board: first row: rook, knight, bishop, queen, king, bishop, knight, and rook; second row: pawns

6. *Castling* consists of moving the king two squares towards a rook on the player's first rank, then moving the rook onto the square over which the king crossed. Castling can only be done if the king has never moved, the rook involved

323

has never moved, the squares between the king and the rook involved are not occupied, the king is not in check, and the king does not cross over or end on a square in which it would be in check (see figure 42).
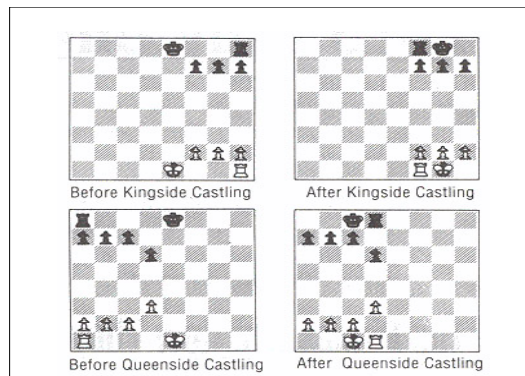

Figura 42: Castling from the queen side and from the king side (figure is due to $http://www.pressmantoy.com/instructions/instruct\_chess.html$)

7. *En-passant* happens if player A's pawn moves forward two squares and player B has a pawn on its fifth rank on an adjacent file and B's pawn capture A's pawn as if A's pawn had only moved one square. This capture can only be made on the immediately subsequent move. In Figure 43 the black pawn moves from b7 to b5 and the white at c5 capture it en passant, ending up on b6.
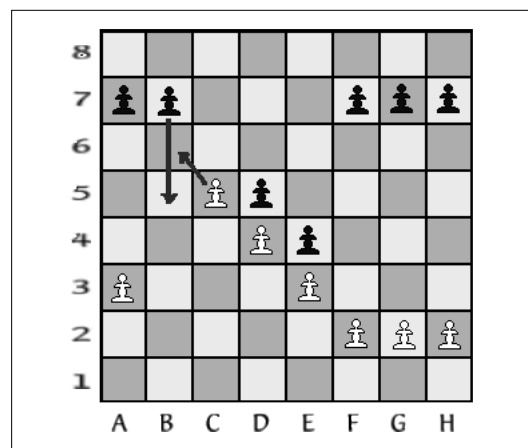

Figura 43: En-passant move. Figure is from $http://www.learnthat.com/courses/fun/chess/beginrules15.shtml$

8. If a pawn advances to its eighth rank it is *promoted* to a queen, rook, bishop,

or knight of the same colour, according to the players desire.

9. When a player makes a move that threatens the opposing king with capture, the king is said to be in check. The definition of check is that one or more opposing pieces could theoretically capture the king on the next move (although the king is never actually captured). If a player's king is in check then the player must make a move that eliminates the threat(s) of capture. The possible ways to remove the threat of capture are: (1) Move the king to a square where it is not threatened. (2)Capture the threatening piece (possibly with the king, if doing so does not put the king in check). (3) Place a piece between the king and the opponent's threatening piece.

10. If the king is in double check, i.e., threatened by two different pieces, the only piece allowed to move is the king itself, since no other piece is able to take away the king from the threat.

11. If a piece is protecting the king from a check it is only allowed to move to another position where it continues protecting the king (the piece is called as *absolute pin*).

12. If a player's king is placed in check and there is no legal move that player can make to escape check, then the king is said to be checkmated and the game ends.