PROGRAMAÇÃO EM LÓGICA INDUTIVA ATRAVÉS DE ALGORITMO ESTIMADOR DE DISTRIBUIÇÃO E CLÁUSULA MAIS ESPECÍFICA

Cristiano Grijó Pitangui

Tese de Doutorado apresentado 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.

Orientador: Gerson Zaverucha

Rio de Janeiro

Março de 2013

PROGRAMAÇÃO EM LÓGICA INDUTIVA ATRAVÉS DE ALGORITMO ESTIMADOR DE DISTRIBUIÇÃO E CLÁUSULA MAIS ESPECÍFICA

Cristiano Grijó Pitangui

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE 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. Valmir Carneiro Barbosa, Ph. D.

_____

Prof. André Ponce de Leon Ferreira de Carvalho, Ph. D.

_____

Prof[a]. Gisele Lobo Pappa, Ph. D.

_____

Prof[a]. Bianca Zadrozny, Ph. D.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2013

Pitangui, Cristiano Grijó

Programação em Lógica Indutiva através de Algoritmo Estimador de Distribuição e Cláusula mais Específica/Cristiano Grijó Pitangui. – Rio de Janeiro: UFRJ/COPPE, 2013.

X, 166 p.: il.; 29,7 cm.

Orientador: Gerson Zaverucha

Tese (doutorado) – UFRJ/ COPPE/ Programa de Engenharia de Sistemas e Computação, 2013.

Referências Bibliográficas: p. 144-149.

1. Programação em Lógica Indutiva. 2. Algoritmos Estimadores de Distribuição. 3. Cláusula mais Específica. I. Zaverucha, Gerson. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Agradeço ao meu orientador Gerson Zaverucha pela presteza e paciência durante estes quase nove anos de convívio.

Agradeço o apoio financeiro da CAPES durante o princípio de meu doutoramento.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciências (D.Sc.)

PROGRAMAÇÃO EM LÓGICA INDUTIVA ATRAVÉS DE ALGORITMO ESTIMADOR DE DISTRIBUIÇÃO E CLÁUSULA MAIS ESPECÍFICA

Cristiano Grijó Pitangui

Março/2013

Orientador: Gerson Zaverucha

Programa: Engenharia de Sistemas e Computação

De forma geral, Programação em Lógica Indutiva (ILP) é uma área da Inteligência Artificial que investiga a construção indutiva de teorias de cláusulas de Horn, de primeira-ordem, a partir de exemplos e de um conhecimento preliminar. A construção de teorias de cláusulas de Horn é uma tarefa de acentuada dificuldade devido ao extenso tamanho do espaço de busca, e, dessa forma, os Algoritmos Genéticos (AGs) foram, em certa extensão, aplicados a tal problema. Por sua vez, os Algoritmos Estimadores de Distribuição (AEDs), embora obtenham, na maioria das vezes, resultados superiores quando comparados aos AGs tradicionais, não foram ainda aplicados à ILP. Este trabalho apresenta o AED-ILP, um sistema ILP baseado em AEDs e em implicação inversa, bem como suas (duas) extensões, a saber: o RAED-ILP, que emprega o algoritmo *Reduce* para reduzir seu espaço de busca, e o HAED-ILP, que usa uma abordagem de busca completamente nova para desenvolver suas teorias. Resultados experimentais em problemas do mundo real mostram que os sistemas obtêm grande êxito em relação ao sistema Aleph (considerado estado da arte) bem como em relação ao AG-ILP (outra variante do AED-ILP criada pela substituição do AED no AED-ILP, por um AG padrão). Em problemas artificiais de Transição de Fase, o AED-ILP também obteve grande êxito quando comparado ao sistema Progol (e suas variantes). Adicionalmente, verificou-se que a variante do AED-ILP, o RAED-ILP, obtém, de forma mais eficiente, teorias mais simples e tão acuradas quanto às do AED-ILP, enquanto que o HAED-ILP é capaz de obter teorias significativamente mais acuradas e em tempo computacional competitivo quando comparado ao Aleph, AED-ILP, e RAED-ILP, e claramente supera o sistema AG-ILP.

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

INDUCTIVE LOGIC PROGRAMMING THROUGH ESTIMATION DISTRIBUTION ALGORITHM AND BOTTOM-CLAUSE

Cristiano Grijó Pitangui

March/2013

Advisor: Gerson Zaverucha

Department: Systems Engineering and Computer Science

In general, Inductive Logic Programming (ILP) is an area of artificial intelligence which investigates the inductive construction of theories of Horn clauses of first-order logic from examples and a preliminary knowledge. Constructing theories of Horn clauses is a task of considerable difficulty due to the extensive size of the search space, thus, Genetic Algorithms (GAs) have been, to some extent, applied to such problem. In turn, although Estimation of Distribution Algorithms (EDAs) generally perform better than the standard GAs, they have not been applied to ILP. This work presents EDA-ILP, an ILP system based on EDA and inverse entailment, and also its extensions, REDA-ILP, which employs the Reduce algorithm in bottom clauses to considerably reduce the search space, and HEDA-ILP, which uses a new hybrid approach to develop its theories. Experiments in real-world datasets show that the proposed systems were successfully evaluated against the standard ILP system Aleph, and GA-ILP (another variant of EDA-ILP created by replacing the EDA by a standard GA). EDA-ILP was also successfully compared to Progol-QG/GA (and its other variants) in phase transition benchmarks. Additionally, we find that REDA-ILP usually obtains simpler theories than EDA-ILP, more efficiently and with equivalent accuracies, while HEDA-ILP finds significantly more accurate theories, in competitive computational time, when compared to Aleph, EDA-ILP, and REDA-ILP, while clearly outperforming GA-ILP.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1: Introdução

Este capítulo apresenta o problema abordado neste trabalho, bem como um resumo das principais contribuições dos sistemas propostos para a resolução do problema apresentado. Adicionalmente, uma visão geral de cada capítulo desta tese é realizada e as publicações/submissões originadas a partir do trabalho proposto são brevemente apresentadas.

1.1 Introdução

Este trabalho trata de um assunto que pode ser considerado um dos principais tópicos da área de Aprendizado de Máquina, a Programação em Lógica Indutiva (ILP).

De forma geral, ILP (NIENHUYS-CHENG et al. 1997) (MUGGLETON, 1991) (MUGGLETON, DE RAEDT, 1994) é uma área da Inteligência Artificial que investiga a construção indutiva de teorias de cláusulas de Horn, de primeira-ordem, a partir de exemplos e de um conhecimento preliminar. ILP é definida como sendo a interseção entre o aprendizado indutivo e a programação em lógica. Do aprendizado indutivo, herda seu objetivo: construção de ferramentas e técnicas para induzir hipóteses a partir de observações (exemplos) e sintetizar novo conhecimento a partir da experiência. Da programação em lógica (LLOYD, 1984), ILP herda o formalismo representacional, como a representação de teorias, hipóteses, exemplos e conhecimento preliminar. Pode-se dizer que tal formalismo representacional é baseado em um fragmento da Lógica de Primeira Ordem e, dessa forma, teorias, hipóteses, e exemplos em ILP são todos representados usando-se um fragmento desta lógica. ILP é aplicada a vários problemas importantes (MUGGLETON, 2001), (LAVRAC, DZEROSKI, 1994) como: dimensionamento de malhas em métodos de elementos finitos, previsão de estrutura de proteínas, detecção de problemas de tráfego, processamento de linguagem natural, dentre outros. A importância do referido problema pode ser mensurada, até certo ponto, pela vasta quantidade de sistemas computacionais construídos para tratá-lo, dentre os quais se destacam: Progol (MUGGLETON, 1995) (MUGGLETON, 1996), FOIL (QUINLAN, 1990), TILDE (BLOCKEEL, DE RAEDT, 1998), e o Aleph (SRINIVASAN, 2003).

O aprendizado de hipóteses em lógica de primeira ordem é uma tarefa de acentuada dificuldade devido ao extenso tamanho do espaço de busca e, dessa forma, não demorou até que os primeiros sistemas baseados em Algoritmos Genéticos (AGs) (HOLLAND, 1975) fossem construídos para serem aplicados a este problema, uma vez que tais algoritmos são conhecidos por sua elevada capacidade de exploração de grandes espaços de busca. Os AGs usam uma analogia direta com o processo de evolução natural. Tipicamente, em um AG, um conjunto de soluções candidatas (indivíduos) é transformado (evoluído) ao longo de várias iterações (gerações), até que algum critério de parada seja satisfeito. As transformações executadas sobre as soluções candidatas são realizadas pelos operadores de recombinação e mutação. Estes operadores, juntamente com o operador de seleção, são os responsáveis pela simulação do processo de evolução natural e, dessa forma, por guiar o conjunto de soluções candidatas a regiões ainda mais promissoras do espaço de busca.

Os AGs podem ser aplicados a qualquer problema de otimização, uma vez que não dependem do domínio do problema para serem aplicados. Além disso, esses algoritmos possuem um grande potencial para explorar espaços de busca muito grandes, o que os capacita a encontrar melhores soluções onde outros tipos de técnicas provavelmente não seriam tão eficazes (MICHALEWICZ, 1999). Como afirmado, os AGs foram, em certa extensão, aplicados a ILP. Dentre tais aplicações, podem-se citar os seguintes sistemas: REGAL (*Relational Genetic Algorithm Learner*) (GIORDANA, NERI, 1996), SIA01 (*Supervised Inductive Algorithm version 01*) (AUGIER, VENTURINI, KODRATOFF, 1995), DOGMA (*Domain Oriented Genetic Machine*) (HEKANAHO, 1996), G-NET (*Genetic Network*) (ANGLANO et al., 1998), ECL (*Evolutionary Concept Learner*) (DIVINA, MARCHIORI, 2002), e QG/GA (*Quick Generalization/Genetic Algorithm*) (MUGGLETON, TAMADDONI-NEZHAD, 2006) (MUGGLETON, TAMADDONI-NEZHAD, 2007), com destaque para este último, não apenas por ser o mais atual, mas por obter resultados bastante interessantes em diversas instâncias do problema tratado.

Os Algoritmos Estimadores de Distribuição (AEDs) (MÜHLENBEIN, 1996) ou Algoritmos Genéticos Construtores de Modelos Probabilísticos (AGCMPs) (PELIKAN, GOLDBERG, LOBO, 1999) podem ser considerados variantes dos AGs tradicionais e, desta forma, guardam certas semelhanças e apresentam algumas diferenças em relação à técnica tradicional. Entre suas semelhanças, está o fato de que os AEDs também mantêm uma população de soluções para explorar o espaço de busca, juntamente com

algum mecanismo de seleção responsável por eleger, a cada geração, os melhores indivíduos da população. Sua diferença fundamental vem do fato de que estes algoritmos, ao contrário dos AGs tradicionais, geralmente suprimem os operadores de recombinação e de mutação que são usados para criar variabilidade genética na população. Para realizar tal efeito, os AEDs usam modelos probabilísticos que são geralmente aprendidos a cada nova geração do algoritmo. Tais modelos, uma vez aprendidos, são então amostrados para gerarem uma nova população.

De forma geral, pode-se dizer que os AEDs são projetados para captar as interações entre as variáveis dos problemas e, com isso, estimam diretamente a distribuição de boas soluções em vez de usar operadores genéticos. Estes algoritmos têm sido aplicados aos mais diversos tipos de problemas de otimização gerando resultados bastante interessantes e superando, na maioria das vezes, o método tradicional de AGs (PELIKAN, 2005). A obtenção de melhores resultados por parte dos AEDs frente aos AGs tradicionais se justifica devido ao aprendizado do modelo probabilístico que capta interações importantes entre as variáveis do problema. Apesar de apresentar melhores resultados quando comparados aos AGs tradicionais, os AED, até os estudos por hora realizados, não foram usados para a construção de sistemas de ILP.

1.2 Objetivos: ILP através de AED

O presente trabalho apresenta um sistema ILP baseado em AED, a saber, o AED-ILP (PITANGUI, ZAVERUCHA, 2010) (PITANGUI, ZAVERUCHA, 2011), bem como suas (duas) extensões, o RAED-ILP (PITANGUI, ZAVERUCHA, 2012), e o HAED-ILP[1]. De forma geral, tais sistemas possuem três objetivos principais: (I) construir teorias simples de alta acurácia usando razoável tempo computacional, sendo capazes de competir ou até mesmo superar alguns sistemas ILP considerados estados da arte; (II) serem aplicáveis a problemas de alta complexidade, tais como (BOTTA et al., 2003) e (ALPHONSE, OSMANI, 2009), uma vez que os sistemas clássicos não obtêm bons resultados quando aplicados a alguns destes problemas; (III) manter certa simplicidade, isto é, os sistemas devem possuir poucos parâmetros a serem configurados, uma vez que os sistemas com grande número de parâmetros, embora sejam flexíveis, são muito

[1] A expressão "sistemas-AED" será utilizada em substituição a expressão "AED-ILP, RAED-ILP, e HAED-ILP" na discussão de características que são comuns a todos os sistemas.

difíceis de serem parametrizados. Com estes objetivos em mente, são apresentadas a seguir algumas diretrizes assumidas para o desenvolvimento do AED-ILP, bem como uma explicação que justifica a razão destas serem adotadas. Tais diretrizes são também válidas para os sistemas RAED-ILP e HAED-ILP, uma vez que ambos os sistemas derivaram-se do AED-ILP.

I. Adoção de um AED como mecanismo de exploração padrão: os sistemas-AED propostos utilizam um AED como mecanismo de exploração de hipóteses. Atualmente, nenhuma outra forma de busca é utilizada, ou seja, o AED é responsável tanto por encontrar uma área promissora no espaço de busca quanto por efetivamente explorá-la.
Justificativa: a adoção desta classe de algoritmos se mostra promissora uma vez que, como afirmado, ela já foi aplicada a vários problemas de otimização e obteve melhores resultados quando comparados aos AGs tradicionais. Além disso, existe o fato de que esta "classe" de algoritmos ainda não foi aplicada ao problema de Programação em Lógica Indutiva, sendo que, caso verificada sua eficiência em tal problema, existe uma grande quantidade de variações da técnica que pode ser utilizada com objetivo de contribuir para o desenvolvimento na área do problema abordado.

II. Uso de *bottom clauses* (cláusulas mais específicas): os sistemas-AED buscam por cláusulas cujos corpos são subconjuntos dos literais das *bottom clauses*.
Justificativa: a implicação inversa através do uso de *bottom clauses* pode ser considerada uma das principais razões do sucesso de importantes sistemas ILP tais como o Progol (MUGGLETON, 1995), Aleph (SRINIVASAN, 2003), QG/GA (MUGGLETON, TAMADDONI-NEZHAD 2006), (OLIPHANT, SHAVLIK, 2007) (chamado a partir de agora pelas iniciais dos seus autores, i.e., OS) dentre outros.

III. Adoção de Redes Bayesianas como Modelos Probabilísticos: os sistemas-AED propostos utilizam redes Bayesianas para capturar as dependências entre os literais da *bottom clause* tal como ocorre no OS. Tais redes são, portanto, os modelos probabilísticos responsáveis por explorar o espaço de busca.
Justificativa: o uso de redes Bayesianas para modelagem do espaço de busca já se mostrou promissor no sistema OS, assim, espera-se que tais estruturas sejam capazes de explorar o espaço de busca de forma efetiva. É importante destacar que as estruturas das redes não se modificam durante a exploração, já que elas apenas capturam as dependências entre os literais da *bottom clause*. Assim, o passo de construção do modelo é efetuado apenas uma vez. Contudo, deve-se ressaltar que as probabilidades

armazenadas no modelo são atualizadas a cada geração, i.e., embora as estruturas das redes Bayesianas sejam mantidas fixas durante toda a execução dos sistemas, suas Tabelas de Probabilidades Condicionais (TPCs) são atualizadas a cada geração. Assim, pela adoção de um modelo estruturalmente estático, mas ainda assim poderoso, tem-se um bom mecanismo de exploração aliado a um baixo custo de tempo computacional.

IV. Simples Atualização do Modelo Probabilístico: as versões atuais dos sistemas-AED propostos utilizam uma simples "variação" da forma de atualização do modelo probabilístico do sistema PBIL (BALUJA, 1994).

Justificativa: a forma de atualização do modelo probabilístico presente no PBIL é bem simples e por isso demanda baixo custo de tempo computacional quando comparado a atualização de outros sistemas baseados em AED, como, por exemplo, o *Bayesian Optimization Algorithm* (PELIKAN, 2005). Como se almeja que o sistema proposto seja competitivo em relação à demanda por tempo, uma atualização simples do modelo probabilístico se faz necessária.

V. Evolução de Teorias: os sistemas propostos consideram uma abordagem diferente da maioria dos sistemas ILP, uma vez que eles evoluem teorias, ou seja, cada indivíduo da população representa um conjunto de cláusulas. Esta abordagem contrasta com a maioria dos sistemas ILP atuais, uma vez que tais sistemas, tanto os tradicionais como FOIL, Progol e Aleph, quanto os sistemas ILP baseados em AGs, como REGAL, G-Net, SIA01, ECL e QG/GA, buscam por cláusulas isoladas em vez de teorias. Algumas exceções a esta regra são o HYPER (BRATKO, 1999) e FORTE-MBC (DUBOC et al., 2009). Este último, embora seja um sistema de revisão de teorias, pode ser usado para aprendê-las "do zero" também.

Justificativa: quando a busca é realizada por cláusulas isoladas, geralmente emprega-se o algoritmo de cobertura para que este, progressivamente, monte a teoria. Um dos problemas já verificados com a utilização deste tipo de algoritmo é que ele gera teorias desnecessariamente grandes, i.e., teorias com um grande número de cláusulas (BRATKO, 1999). Com a evolução de teorias, espera-se que estas possuam um menor número de cláusulas quando comparadas a teorias desenvolvidas por sistemas que usam um algoritmo de cobertura.

Uma vez discutidos os pontos em comum que nortearam o desenvolvimento do AED-ILP e de suas extensões, é importante destacar os pontos principais que diferenciam estes sistemas.

De forma geral, o RAED-ILP pode ser entendido como o AED-ILP que utiliza o algoritmo Reduce (MUGGLETON, FENG, 1990) com objetivo de reduzir o seu espaço de busca. Grosso modo, pode-se dizer que o algoritmo Reduce retira alguns literais da *bottom clause* fazendo com que a mesma se torne menor. Em consequencia, o espaço de busca dos sistemas que utilizam *bottom clauses* "reduzidas" se torna menor em relação ao espaço de busca delimitado por *bottom clauses* não reduzidas. Pode-se dizer que o QG/GA influenciou diretamente a construção do RAED-ILP, uma vez que tal trabalho também utiliza o algoritmo Reduce. Contudo, como será visto durante a revisão do referido sistema (veja capítulo 3), os sistemas QG/GA e RAED-ILP apresentam diferenças fundamentais entre seus componentes.

Por sua vez, o HAED-ILP pode ser entendido como o AED-ILP com seu ciclo de busca completamente remodelado. Esta nova versão do AED-ILP é chamada AED-ILP Híbrido (por isso o nome de HAED-ILP) uma vez que ele continua evoluindo teorias (tal como motivado pela 5ª diretriz discutida acima), mas estas agora são construídas de maneira iterativa. Tal como será apresentado posteriormente, esta nova abordagem de evoluir teorias é completamente diferente da abordagem utilizada pelos sistemas AED-ILP e RAED-ILP, uma vez que em tais sistemas as teorias possuem um tamanho fixo (fornecido *a priori* pelo usuário) durante toda a execução do sistema.

Os pontos principais que, em conjunto, diferenciam os sistemas AED-ILP, RAED-ILP, e HAED-ILP de todos os outros sistemas ILP são:

- Utilização de um AED responsável por encontrar um lugar promissor no espaço de busca e por explorá-lo;
- Busca por teorias em vez de busca por cláusulas isoladas;
- Utilização de uma "variação" da regra de atualização do sistema PBIL para modificar as TPCs das redes Bayesianas;
- Utilização do algoritmo Reduce para reduzir o espaço de busca por teorias (apenas para o RAED-ILP);
- Utilização da nova abordagem híbrida para evolução de teorias (apenas para o HAED-ILP).

1.3 Visão Geral

Os principais tópicos discutidos em cada um dos capítulos do trabalho são apresentados em seguida.

- **O Capítulo 2** trata exclusivamente do problema de Programação em Lógica Indutiva e, desta forma, ele apresenta formalmente o referido problema, bem como uma breve motivação para o seu estudo. Adicionalmente, ao final do capítulo são revisados alguns sistemas ILP considerados estado da arte.
- **O Capítulo 3** trata de AGs aplicados a ILP. Este capítulo revisa os conceitos que permeiam os AGs tradicionais como também apresenta os AEDs, uma vez que estes podem ser vistos como uma variação dos AGs tradicionais. Adicionalmente, o capítulo revisa o sistema QG/GA, uma vez que este motivou o desenvolvimento do sistema RAED-ILP, e apresenta uma breve discussão, a título de ilustração, de outros sistemas ILP baseados em AGs. Finalmente, o capítulo revisa o sistema OS uma vez que tais autores usam a motivação por de trás dos AEDs para a construção de um sistema ILP, mas, como será apresentado, tal trabalho não pode ser visto como uma aplicação de AED a ILP.
- **O Capítulo 4** apresenta o AED-ILP bem como suas duas extensões, o RAED-ILP e o HAED-ILP. Seus principais componentes e funcionalidades são apresentados e detalhadamente discutidos.
- **O Capítulo 5** apresenta os resultados da aplicação dos sistemas propostos em 8 (oito) problemas ILP do mundo real, e a 7 (sete) problemas artificiais de Transição de Fase (*Phase Transition*) (BOTTA et al., 2003). Os resultados obtidos são comparados com dois sistemas considerados estado da arte em ILP, o Aleph, e o Progol (com algumas de suas variantes) bem como com o sistema denominado AG-ILP, obtido pela substituição do AED no AED-ILP por um AG "padrão".
- **O capítulo 6** apresenta as conclusões obtidas a partir dos experimentos computacionais, bem como propostas de trabalhos futuros.

1.4 Publicações e Submissões

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

- **PITANGUI, G. C., ZAVERUCHA, G.**, *Genetic Local Search for Rule Learning*. In: ACM Genetic and Evolutionary Computation Conference (GECCO-2008), 2008, Atlanta, Georgia. ACM Genetic and Evolutionary Computation Conference (GECCO-2008), 2008. p. 1427-1428.

- Este trabalho apresentou um método de busca local estocástico aplicado ao nosso sistema genético de aprendizado de regras proposicionais (PITANGUI, ZAVERUCHA, 2006) (PITANGUI, ZAVERUCHA, 2007). Tal método foi inspirado no trabalho (RÜCKERT, KRAMER, 2003). Uma adaptação deste método está sendo construída para ser aplicada aos sistemas-AED[2].

- **PITANGUI, G., C., ZAVERUCHA, G.**, *Algoritmo Genético Construtor de Modelo Probabilístico Aplicado à Programação em Lógica Indutiva*. In: Joint Conference SBIA/SBRN/JRI 2010 / III Workshop on Computational Intelligence, 2010, São Bernardo do Campo. Joint Conference SBIA/SBRN/JRI 2010 / III Workshop on Computational Intelligence, 2010. p. 1-6.
    - Este trabalho pode ser considerado o embrião do sistema AED-ILP em que foram apresentadas as idéias fundamentais do sistema bem como resultados preliminares.
- **PITANGUI, G., C., ZAVERUCHA, G**. *Inductive Logic Programming through Estimation of Distribution Algorithm*. In: IEEE Congress on Evolutionary Computation (CEC), 2011, New Orleans. IEEE Congress on Evolutionary Computation (CEC 2011), 2011. p. 54-61.
    - Este trabalho introduziu formalmente o sistema AED-ILP bem como apresentou resultados do sistema em algumas bases de dados.
- **PITANGUI, G., C., ZAVERUCHA, G**. *Learning Theories Using Estimation Distribution Algorithms and (Reduced) Bottom Clauses*. In: 21st International Conference on Inductive Logic Programming (ILP 2011), 2012, Windsor. Leccture Notes in Artificial Intelligence. Heidelberg: Springer, 2012. v. 7207. p. 286-301.
    - Este trabalho apresentou o sistema RAED-ILP e avaliou mais profundamente o sistema AED-ILP.
- **PITANGUI, G., C., ZAVERUCHA, G.** *Learning First-order Logic Theories Through the use of Estimation of Distribution Algorithm and Bottom- clauses.*
    - O sistema **HAED-ILP**, bem como os experimentos adicionais (em relação aos artigos já publicados) e os sistemas AED-ILP e RAED-ILP foram

[2] O referido método (em sua "versão" proposicional) a ser adaptado pode ser encontrado no Apêndice A desta tese.

submetidos (como um único trabalho completo) a um periódico internacional de alta relevância – ainda aguarda-se o processo de revisão.

Capítulo 2: Programação em Lógica Indutiva

Este capítulo apresenta o problema de Programação em Lógica Indutiva. Seus principais eixos, a saber, Lógica de Primeira Ordem e Aprendizado Indutivo de Conceitos são brevemente apresentados. O capítulo apresenta também a motivação para o estudo do problema e alguns dos sistemas computacionais considerados estados da arte aplicados ao problema abordado.

2.1 Introdução

Programação em Lógica Indutiva (ILP) (NIENHUYS-CHENG et al. 1997), (MUGGLETON, 1991), (MUGGLETON, DE RAEDT, 1994) é uma área da Inteligência Artificial que investiga a construção indutiva de teorias de cláusulas de Horn, de primeira-ordem, a partir de exemplos e de um conhecimento preliminar. Do aprendizado indutivo, herda seu objetivo: construção de ferramentas e técnicas para induzir hipóteses a partir de observações (exemplos) e sintetizar novo conhecimento a partir da experiência. Da programação em lógica (LLOYD, 1984) ILP herda o formalismo representacional, como a representação de teorias, hipóteses, exemplos e conhecimento preliminar.

De forma geral, o objetivo de um sistema ILP é induzir teorias a partir de exemplos e de um conhecimento preliminar expressos por cláusulas de Horn. Devido à ampla aplicação a vários problemas importantes, tais como, dimensionamento de malhas em métodos de elementos finitos, previsão de estrutura de proteínas, detecção de problemas de tráfego, processamento de linguagem natural, dentre outros, pode-se afirmar que ILP é um dos principais tópicos da área de Aprendizado de Máquina (MUGGLETON, 2001), (LAVRAC, DZEROSKI, 1994).

Aprendizado pode ser visto como um problema de busca num espaço de hipóteses (MITCHELL, 1997). Sob este ponto de vista, pode-se dizer, mais especificamente, que o objetivo de um sistema de ILP é encontrar uma teoria (expressa por cláusulas de Horn) que implique logicamente todos os exemplos positivos e não implique logicamente nenhum exemplo negativo, utilizando, para isso, um conhecimento preliminar dado *a priori* e aceito como correto.

Este capítulo apresenta uma visão geral sobre ILP. A seção 2.2 se encarrega de apresentar os conceitos preliminares essenciais ao entendimento do trabalho proposto. Dentre os conceitos apresentados na próxima seção, estão a questão de Aprendizado Indutivo de Conceitos e uma breve revisão sobre Lógica de Primeira Ordem. A seção 2.3 introduz formalmente o problema de ILP. A seção 2.4 apresenta, de forma resumida, três sistemas para ILP que podem ser considerados o estado da arte, são eles: Progol (MUGGLETON, 1995), (MUGGLETON, 1996), FOIL (QUINLAN, 1990) e Aleph (SRINIVASAN, 2003). Finalmente, a seção 2.5 apresenta um resumo do que foi apresentado no capítulo.

2.2 Conceitos Preliminares

Como apresentado anteriormente, ILP combina Aprendizado Indutivo de Teorias (AIT) e Programação em Lógica (PL), e, desta forma, antes de se definir formalmente o problema de ILP, se faz necessária a apresentação dos conceitos fundamentais que o regem. Por conseguinte, esta seção apresenta primeiramente uma revisão sobre Indução de Teorias e, posteriormente, uma breve revisão sobre PL e Lógica de Primeira Ordem (LPO), uma vez que é sobre um fragmento desta lógica que ILP se baseia.

2.2.1 Aprendizado Indutivo de Conceitos

O conceito de Aprendizado Indutivo de Conceitos (AIC) pode ser formalmente definido como segue (MITCHELL, 1997).

Definição 2.1: (Aprendizado Indutivo de Conceitos)[1]: dada uma linguagem de descrição usada para expressar possíveis hipóteses, um conhecimento preliminar (BK), um conjunto de exemplos positivos, e um conjunto de exemplos negativos, Aprendizado Indutivo de Conceitos pode ser formulado como o problema de se encontrar uma hipótese que cubra todos os exemplos positivos e não cubra nenhum dos exemplos negativos.

De forma geral, derivar conclusões gerais a partir de observações específicas é chamado de indução. Dessa forma, no AIC, os conceitos são de fato induzidos, uma vez

[1] Optou-se por apresentar a definição do AIC uma vez que o AIT trata-se, de fato, do AIC aplicado ao caso em que se representa a hipótese por meio de um fragmento da lógica de primeira ordem. Dessa forma, pode-se dizer que o AIT é uma especificação do conceito de AIC. Esta diferença ficará mais clara no decorrer deste trabalho.

que as hipóteses (conclusões) são induzidas a partir dos exemplos (observações específicas). Este processo de indução pode ser visto como um processo de busca em que se almeja encontrar a melhor hipótese (de acordo com certa função de avaliação) no espaço de todas as possíveis hipóteses expressas na linguagem de representação (MITCHELL, 1997). A conjectura fundamental do aprendizado indutivo é que qualquer hipótese descoberta que aproxima bem um determinado conceito (função alvo) usando um conjunto suficientemente grande de exemplos de treinamento, também aproximará bem a função alvo sobre os outros exemplos não observados (generalização) (MITCHELL, 1997).

Uma linguagem de representação deve ser escolhida para representar os conceitos, as hipóteses, os exemplos, e o conhecimento preliminar. A escolha da linguagem de representação é muito importante, uma vez que ela limita o tipo de conceito que pode ser aprendido (MITCHELL, 1997). Com uma linguagem de representação detentora de baixa expressividade, pode-se não ser possível representar algum problema, por ele ser muito complexo para a linguagem adotada. Em contrapartida, uma linguagem com grande expressividade pode ser capaz de representar um maior conjunto de domínios de problemas. Entretanto, esta solução pode fornecer muita liberdade no sentido de ser possível construir hipóteses de muitas formas diferentes, o que poderia levar a uma impossibilidade de se encontrar o conceito correto. Como linguagem de representação, ILP utiliza um fragmento da LPO, e esta será revisada a seguir.

2.2.2 Lógica de Primeira Ordem e Programação em Lógica[2]

A Lógica de Primeira Ordem (LPO) pode ser vista como uma extensão da Lógica Proposicional (LP) sendo capaz de representar relações entre objetos, concluir particularizações de uma propriedade geral dos indivíduos de um universo de discurso, assim como derivar generalizações a partir de fatos que valem para um indivíduo arbitrário do universo de discurso. Como era de se esperar, por possuir maior expressividade quando comparada à Lógica Proposicional, a LPO usa símbolos mais complexos quando comparados aos símbolos utilizados na LP. Apresentam-se a seguir alguns conceitos relativos à LPO.

[2] O conteúdo desta seção foi retirado de: (CASANOVA, 1987), (LLOYD, 1987), (ZAVERUCHA, 2008), e (DUBOC, 2008).

Um alfabeto de LPO consiste de:

1. **Símbolos Lógicos:**
    a. Pontuação: consiste de parêntese aberto, parêntese fechado e vírgula.
    b. Conectivos: $\neg$ (negação), $\wedge$ (conjunção), $\vee$ (disjunção), $\leftarrow$ (implicação), e $\leftrightarrow$ (bi-condicional).
    c. Quantificadores: $\forall$ (universal), e $\exists$ (existencial).
    d. Variáveis: se iniciam com letras maiúsculas, tais como *A*, *X*, *Var*.
    e. Símbolo de igualdade (opcional): =
2. **Símbolos não-lógicos:**
    a. Funções: se iniciam por letras minúsculas.
    b. Constantes: são símbolos funcionais de aridade (número de argumentos) zero.
    c. Predicados: expressam relações entre seus argumentos.

Definição 2.2 (Conjunto de termos de primeira-ordem): O conjunto dos termos de primeira-ordem é definido recursivamente pelas seguintes regras:

1. Qualquer constante é um termo.
2. Qualquer variável é um termo.
3. Toda expressão $f(t_1, \ldots, t_n)$ de $n \geq 1$ argumentos (onde cada argumento t_i é um termo e f é um símbolo de função de aridade n) é um termo.
4. Nada mais é um termo.

Exemplo 2.1: $f(t_1,\ldots, t_n)$ é um termo em que $t_1,\ldots, t_n$ são termos.

Definição 2.3 (Fórmula Atômica ou Átomo): Se $t_1,\ldots, t_n$ são termos e P é um símbolo predicativo n-ário (com n argumentos), então $P(t_1,\ldots, t_n)$ é chamado de fórmula atômica ou átomo.

Exemplo 2.2: *pai(helio, cristiano)* é um átomo, em que o predicado pai possui aridade dois e expressa a relação de paternidade entre *hélio* e *cristiano* (duas constantes).

Um átomo é um literal positivo, e a negação de um átomo é um literal negativo. Se A e B são fórmulas, então ($\neg$A), (A $\wedge$ B), (A $\vee$ B), (A $\leftarrow$ B), e (A $\leftrightarrow$ B) são fórmulas. Se B é uma fórmula e X é uma variável, então $\forall$X (B) e $\exists$X (B) também são fórmulas.

Definição 2.4 (Cláusula): Uma cláusula é uma disjunção de literais precedida por um prefixo de quantificadores universais, um para cada variável que aparece na disjunção, na seguinte forma:

$$\forall X_1 \forall X_2 ... \forall X_s (L_1 \vee L_2 ... \vee L_m)$$

onde cada L_i é um literal, e $X_1 X_2 ... X_s$ são todas as variáveis ocorrendo em $(L_1 \vee L_2 ... \vee L_m)$. O conjunto $\{A_1, A_2, ..., A_h, \neg B_1, \neg B_2, ..., \neg B_b\}$ onde A_i e B_i são átomos, indica a cláusula:

$$(A_1 \vee ... \vee A_h \vee \neg B_1 \vee ... \vee \neg B_b)$$

que é equivalentemente representada por: $A_1 ... A_h \leftarrow B_1, ..., B_b$ onde $A_1 ... A_h$ é a cabeça e $B_1, ..., B_b$ é o corpo da cláusula. Normalmente, em ILP é usado ':-' em vez de '←'.

Definição 2.5 (Teoria): Uma teoria é um conjunto de cláusulas, representando a conjunção destas cláusulas.

Definição 2.6 (Cláusula de Horn): Uma cláusula de Horn é uma cláusula que possui no máximo um literal positivo, ou seja, possui, no máximo, um literal na cabeça.

Exemplo 2.3: A cláusula $A \leftarrow L_1, L_2, L_3$ é uma cláusula de Horn.

Definição 2.7 (Cláusula Definida): Uma cláusula Definida é uma cláusula de Horn com exatamente um literal na cabeça.

Exemplo 2.4: A cláusula $A \leftarrow L_1, L_2, L_3$ é uma cláusula Definida.

Definição 2.8 (Cláusula Objetivo): Uma cláusula Objetivo é uma cláusula de Horn sem o literal na cabeça.

Exemplo 2.5: A cláusula $\leftarrow L_1, L_2, L_3$ é uma cláusula Objetivo.

Definição 2.9 (Fato): Um Fato é uma cláusula Definida que possui corpo vazio.

Exemplo 2.6: a cláusula A ← é um Fato. Fatos são normalmente representados sem o símbolo de implicação.

Definição 2.10 (Programa Lógico Definido): Programa Lógico Definido é um conjunto de cláusulas definidas.

Exemplo 2.7: A figura 2.1 ilustra um Programa Lógico Definido. As cláusulas de 1 até 5 são fatos. As cláusulas de 6 a 10 ilustram algumas relações familiares que podem (ou não) ocorrer entre os fatos expressos no programa lógico. Assim, por exemplo, a

cláusula de número 8 informa que um indivíduo *X* qualquer será irmão de um indivíduo *Y* qualquer, caso *X* seja irmão de *Y* (relação definida pela cláusula de número 10), e *X* seja do gênero *masculino*. As outras relações podem ser interpretadas de forma equivalente.

```
01 - pais(jorge, julio).
02 - pais(jorge, lucia).
03 - genero(lucia, feminino).
04 - genero(jorge, masculino).
05 - genero(julio, masculino).
06 - pai(X,Y) :- pais(X,Y), genero(X, masculino).
07 - mãe(X,Y) :- pais(X,Y), genero(X, feminino).
08 - irmão(X,Y) :- irmãos(X,Y), genero(X, masculino).
09 - irmã(X,Y) :- irmãos(X,Y), genero(X, feminino).
10 - irmãos(X,Y) :- pais(Z,X), pais(Z,Y), X \= Y.
```

Figura 2.1: Exemplo de Programa Lógico

Definição 2.11 (Substituição θ): Uma substituição $\theta = \{X_1/t_1,..., X_n/t_n\}$ é um mapeamento finito de variáveis em termos que atribui a cada variável X_i um termo t_i, para $1 \leq i \leq n$.

Exemplo 2.8: $\theta = \{X/a, Y/b\}$ é um exemplo de substituição θ em que as ocorrências da variável *X* serão simultaneamente substituídas pela constante *a*, e as ocorrências da variável *Y* serão simultaneamente substituídas pela constante *b*.

Definição 2.12 (Termo, átomo ou cláusula básica(o)): Um termo, átomo ou cláusula é dito básica(o) caso não exista nenhuma variável ocorrendo nele(a).

Exemplo 2.9: o literal *pai(helio, cristiano)* é dito básico, uma vez que nele não ocorre nenhuma variável.

Definição 2.13 (Unificador): Assuma que L_1 e L_2 são literais. Pode-se dizer que a substituição θ é um unificador para L_1 e L_2, se e somente se, $L_1\theta = L_2\theta$. Pode-se dizer também que L_1 e L_2 são unificáveis via θ.

De forma geral, podem existir vários unificadores, e, dentre eles, o unificador mais geral pode ser encontrado.

Definição 2.14 (Generalidade de Substituição θ): Considere duas substituições θ_1 e θ_2. Diz-se que θ_1 é mais geral que θ_2, $\theta_1 \preceq \theta_2$, se existir uma substituição θ_3 tal que $\theta_1\theta_3 = \theta_2$.

Definição 2.15 (Unificador Mais Geral): Pode-se dizer que θ é o Unificador Mais Geral (UMG) para L_1 e L_2, se θ é mais geral que todos os outros unificadores de L_1 e L_2.

Exemplo 2.10: Assuma que os literais L_1 e L_2 são, respectivamente, *p(X,b)* e *p(a,b)*. Dessa forma, θ = {*X/a*} é um unificador para L_1 e L_2 e, além disso, θ é o UMG para L_1 e L_2.

Definição 2.16 (θ-subsume ($\preceq$)): Uma cláusula *C* θ-subsume a cláusula *D*, ou seja, $C \preceq D$, se existe uma substituição θ tal que $C\ \theta \subseteq D$, isto é, todos os literais de *C* com a substituição θ formam um subconjunto dos literais de *D*.

Exemplo 2.11: A clausula *C*: *f(A,B)* ← *p(B,G), q(G,A)* θ-subsume a cláusula *D*: *f(a,b)* ← *p(b,g), q(g,a), t(a,d)*, através da substituição θ = {*A/a, B/b, G/g*}.

Apresentados alguns dos principais conceitos que regem a LPO, pode-se definir o paradigma de Programação em Lógica da seguinte maneira (SEBESTA, 1996):

Definição 2.17 (Programação em Lógica): Programação em Lógica é o uso de uma notação lógica formal para comunicar processos computacionais a um computador.

A LPO é a notação formal utilizada nas atuais linguagens de programação em lógica, como Prolog, por exemplo. As linguagens de programação em lógica necessitam de um processo de inferência para computar os resultados desejados (SEBESTA, 1996). O método de prova chamado de resolução[3] (ROBINSON, 1965) é utilizado no processo de inferência. A definição seguinte deve ser interpretada dentro do escopo de programação em lógica.

Definição 2.18 (Consulta a um Programa Lógico): Uma Consulta a um Programa Lógico é uma cláusula na forma ← $L_1, L_2, \ldots, L_n$, em que L_i, para $1 \leq i \leq n$, são literais.

[3] Resolução é uma regra de inferência que leva a uma técnica de prova de teoremas por refutação para sentenças em lógica proposicional e em lógica de primeira ordem (GALLIER, 1986).

Quando se consulta um programa lógico, um procedimento de resolução chamado de SLD[4] (*Selection rule driven Linear resolution for Definite clauses*) é aplicado com objetivo de verificar se a consulta é satisfeita pelo programa lógico. A resolução SLD aplica uma série de passos de derivações para verificar se a consulta é satisfeita. Um passo de derivação consiste das seguintes operações:

1. **Selecione um literal L_i $1 \leq i \leq n$, da consulta.**
2. **Selecione uma cláusula *C* no programa lógica tal que sua cabeça possa ser unificada com L_i.**
3. **Selecione o UMG para a consulta e a cabeça da cláusula *C*.**
4. **Substitua L_i na consulta pelo corpo da cláusula *C*, e aplique o UMG à consulta resultante.**

Nos passos 1 e 2, a ordem em que os literais da consulta devem ser resolvidos (regra de seleção) e a ordem em que as cláusulas do programa lógico são usadas na derivação precisam ser especificados para fazer com que a resolução seja determinística. Caso a derivação acabe com a cláusula vazia (cláusula sem nenhum literal), denotada por □, então a derivação foi um sucesso e a resposta à consulta é um "sim". Caso a derivação não encontre a cláusula vazia, a derivação não obteve sucesso e retorna um "não" a consulta.

De forma geral, se uma consulta *C* pode ser derivada de um programa lógico PL em zero ou mais passos de derivação, denota-se, este fato por PL |- *C*. Uma importante propriedade da resolução é que apenas consequências lógicas podem ser derivadas. Esta propriedade é chamada corretude *(soundness)* da resolução. De forma geral, uma fórmula *F* implica logicamente uma fórmula *G* (*F* |= *G*) se e somente se todo modelo[5] de *F* for também modelo de *G*. Outra importante propriedade da resolução é chamada de completude (*completeness*), que afirma que, para um dado programa lógico PL e A um fato básico, então, PL |= A se e somente se (PL ∪ {← A}) |- □. Pode-se provar que

[4] Resolução SLD é um caso especial de refinamento do método de resolução que possui características importantes quando aplicado a cláusulas de Horn. Resolução SLD é também muito importante visto que ela é o procedimento principal de computação em Prolog.

[5] A definição de modelo está ligada à definição de interpretação. Interpretação é um processo que mapeia uma sentença em alguma declaração em relação a um determinado domínio. Se a interpretação resultar o valor verdadeiro para uma sentença então ela satisfaz esta sentença e tal interpretação é chamada um modelo da sentença.

a resolução SLD possui as propriedades de corretude e de completude quando as cláusulas trabalhadas se resumem às cláusulas de Horn.

Uma vez que se tenha definido a regra de seleção, a totalidade das derivações SLD para uma dada consulta e um dado programa lógico pode ser representada em uma árvore SLD. Cada ramo da árvore é uma derivação SLD que ocorre via regra de seleção. Os nós da árvore são consultas com um literal selecionado, sendo que cada nó da árvore tem exatamente um filho para cada cláusula que unifica com o literal selecionado da consulta contida no nó. Em Prolog, a árvore SLD é explorada pela busca em profundidade (RUSSEL, NORVIG, 2003), e dessa forma, são construídas todas as possíveis derivações para a consulta até se encontrar uma cláusula vazia, ou até todas as possíveis derivações terem sido tentadas. No último caso, como já afirmado, a consulta falha. A título de ilustração do processo descrito, atente para o exemplo 2.12 a seguir.

Exemplo 2.12: considere uma simplificação do programa lógico da figura 2.1 que é apresentado na figura 2.2.

```
01 - pais(jorge, julio).
02 - pais(jorge, lucia).
03 - genero(lucia, feminino).
04 - genero(jorge, masculino).
05 - genero(julio, masculino).
06 - pai(X,Y) :- pais(X,Y), genero(X, masculino).
07 - mãe(X,Y) :- pais(X,Y), genero(X, feminino).
```

Figura 2.2: Exemplo Simples de Programa Lógico Definido

Suponha que a seguinte consulta foi realizada: ← *pai(X, lucia)*, ou seja, deseja-se saber quem é o pai de lucia. No primeiro passo, com θ_1 = {*X/X* e *Y/lucia*}, o único literal da consulta inicial se unifica com a cabeça da cláusula de número 6, fornecendo, após aplicação do passo 4 da derivação, a consulta ← *pais(X, lucia)*, *genero(X, masculino)*. O primeiro literal desta nova consulta é unificado pela substituição θ_2 = *{X/jorge}* com a cabeça da cláusula de número 2, fornecendo, após aplicação do passo 4 da derivação, a consulta ← *genero(jorge, masculino)*. O único literal desta nova consulta é unificado pela substituição θ_3 = {∅} com a cabeça da cláusula de número 4 resultando na cláusula vazia □, ou seja, a derivação obteve sucesso e o sistema retorna um "sim" como resultado deixando instanciada a variável *X* com a constante *jorge*, ou seja, *X = jorge*.

Este é o único caminho possível na árvore SLD devido à simplicidade do programa lógico. Contudo, se existissem, por exemplo, várias cláusulas com o literal pai(X, Y) na cabeça, certamente ter-se-iam vários outros ramos na árvore SLD para esta consulta específica.

2.3 Programação em Lógica Indutiva

Primeiramente, esta seção apresenta a definição formal do problema de Programação em Lógica Indutiva e, posteriormente, apresenta a principal motivação para o estudo do problema.

2.3.1 Definição do Problema de Programação em Lógica Indutiva

Pode-se dizer que ILP pode ser vista como o estudo de métodos de aprendizado de hipóteses que são expressas por meio de predicados lógicos de primeira ordem. Mais formalmente, pode-se dizer que o problema básico de ILP é (LAVRAC, DZEROSKI, 1994):

Definição 2.19 (Programação em Lógica Indutiva): Dados (a) um conhecimento preliminar invariante denotado por BK, (b) um conjunto de exemplos positivos E+ e negativos E-, deve-se encontrar uma teoria[6] H que, junto com o conhecimento preliminar BK, implique logicamente todos os exemplos positivos (completude), ou seja, $BK \cup H \models E+$, e nenhum dos exemplos negativos (consistência), $\forall e^- \in E- : BK \cup H \not\models e^-$, e que esteja de acordo com o critério de qualidade estabelecido, tal como minimalidade do tamanho da teoria.

Na prática, os requisitos de completude e consistência são relaxados e, desta forma, deve-se encontrar uma teoria que implique logicamente a maior quantidade possível de exemplos positivos, ao mesmo tempo em que implica logicamente a menor quantidade possível de exemplos negativos.

Um termo bastante utilizado em ILP é conhecido como cobertura, sendo, este, definido como segue:

[6] Usa-se o termo "teoria" em detrimento ao termo "hipótese" uma vez que se fala, agora, de aprendizado de uma hipótese que é representada por um conjunto de cláusulas de primeira ordem, que nada mais é que uma teoria de acordo com a definição 2.5.

Definição 2.20 (Cobertura): Dados H, BK e E, como definidos em 2.19, a teoria H cobre (ou prova) um exemplo e $\in$ E em relação a BK, se BK $\wedge$ H |= e.

Conforme apresentado, pode-se dizer que o objetivo de um sistema ILP é o de inferir uma teoria (conjunto de cláusulas) quando lhe são dados uma base de dados com fatos e outras relações lógicas (BK). Por exemplo, um sistema ILP pode ser usado para aprender a relação *avo(X,Y)*, chamado de predicado alvo, dado um conjunto de exemplos positivos e negativos para esta mesma relação, e um conjunto de fatos para outras relações, tais como *genero(X)*, *pai(X,Y)*, *mãe(X,Y)*, chamadas de predicados de conhecimento preliminar. Em outras palavras, o sistema deve aprender a relação exposta no predicado alvo, utilizando, para isto, exemplos definidos sobre esta relação e um conjunto de outras relações. Em sistemas ILP, os exemplos de treinamento, o conhecimento preliminar e as hipóteses induzidas são todos expressos na forma de programas lógicos definidos, sendo que os exemplos de treinamento são representados como fatos da relação alvo a ser aprendida.

2.3.1 Motivação para Estudo do Problema

Quando se deseja resolver um problema com auxílio de um computador, o primeiro passo a ser realizado é traduzir o problema para termos computacionais. Desta forma, deve ser escolhida uma linguagem de representação para o problema.

Sob o ponto do AIC, a linguagem de representação utilizada para representar as hipóteses determina, até certo ponto, a complexidade do problema de aprendizado. A grosso modo, pode-se dizer que representações mais simples originarão problemas de busca mais simples quando comparados a problemas de busca gerados quando se utiliza representações mais complexas (MITCHELL, 1997). As linguagens de representação variam de fragmentos da lógica proposicional à lógica de segunda ordem, sendo a primeira detentora de baixo poder de expressividade e a última muito complexa, e por isso raramente usada. A seguir apresentam-se duas das várias formas de representação de hipóteses, a saber: Representação Proposicional e Representação em Lógica de Primeira Ordem[7].

[7] Embora seja dito que a linguagem de representação utiliza Lógica Proposicional ou Lógica de Primeira Ordem, é importante notar que a representação propriamente dita do problema é feita utilizando-se apenas um fragmento da Lógica Proposicional ou da Lógica de Primeira Ordem.

Uso de Lógica Proposicional: usa-se a representação proposicional quando o problema pode ser representado por um número fixo de atributos, sendo que cada um representa uma característica específica do problema. Talvez o exemplo mais clássico de aprendizado de hipóteses utilizando a representação proposicional seja o de buscar por uma hipótese que defina quando jogar tênis (MITCHELL, 1997) baseando-se em algumas condições climáticas do dia. Para isso, anotam-se informações climáticas dos dias (tempo, temperatura, umidade, e vento) em que se joga e em que não se joga tênis. Após esta anotação, estas informações podem ser facilmente dispostas em uma tabela. A tabela 2.1 exibe parte destas anotações.

Tabela 2.1: Situações Favoráveis para se Jogar Tênis (MITCHELL, 1997)

	Atributos			**Classe**	
Dia	Tempo	Temperatura	Umidade	Vento	Joga Tênis?
1	Sol	Quente	Alta	Fraco	Não
2	Sol	Quente	Alta	Forte	Não
3	Nublado	Quente	Alta	Fraco	Sim
4	Chuva	Moderado	Alta	Fraco	Sim

Como apresentado na tabela 2.1, o problema de se determinar a ocorrência ou não de um jogo de tênis pode ser facilmente representado utilizando-se a representação proposicional. Assim, uma possível hipótese aprendida (também expressa de forma proposicional) poderia ser: Se (tempo = nublado ou chuvoso) então jogue tênis.

Uso da Lógica de Primeira Ordem: a motivação para o uso de Lógica de Primeira Ordem vem do fato de que para alguns problemas, a lógica proposicional não pode representar adequadamente as estruturas de dados nele presentes. Como exemplo, considere a figura 2.3 que ilustra alguns exemplos de problemas (GOADRICH, 2009).

Os domínios de aplicação de ILP podem ser divididos em dois grupos, a saber: o grupo que trata das relações entre os objetos (exemplos a e b da figura 2.3) e o grupo que trata das relações entre as conexões dos objetos (exemplo c da figura 2.3). No primeiro grupo, o objetivo é discriminar os objetos em classes distintas, enquanto que no segundo grupo, o objetivo é discriminar as conexões entre os objetos em vez de os objetos propriamente ditos.

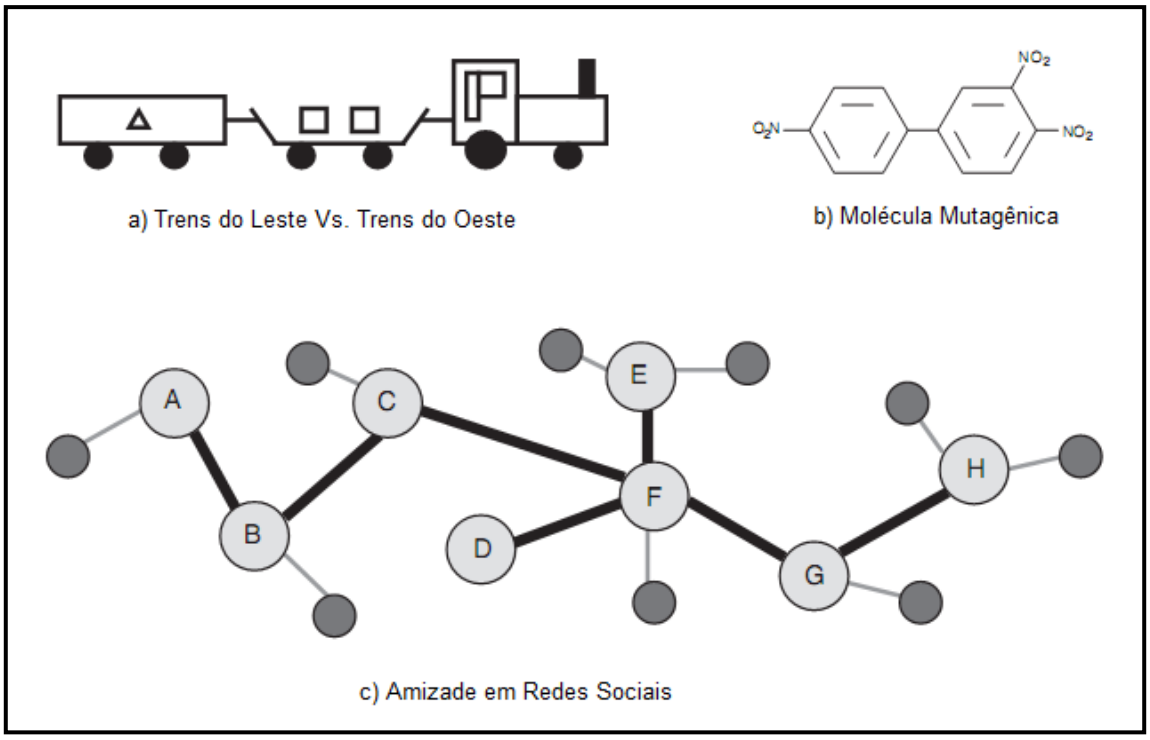

Figura 2.3: Exemplos de Problemas

Considere, a título de ilustração das limitações da representação proposicional, o problema de determinar se uma dada molécula é mutagênica ou não. Uma molécula consiste de vários átomos, cada qual descrito por algumas propriedades. Além das propriedades de cada átomo, existem relações entre os átomos. Caso dois átomos sejam conectados entre si, dizemos que existe uma conexão (*bond*) entre eles. Átomos podem estar associados a várias conexões e, além disto, existem diversos tipos destas.

Caso tente-se representar uma molécula usando Lógica Proposicional, primeiramente deve-se determinar o número máximo de atributos que descrevem as propriedades dos átomos, contudo, nem todos os átomos da molécula possuem as mesmas propriedades, e, consequentemente, vários destes atributos, para alguns átomos, não irão possuir valor algum, uma vez que existem propriedades que não são relativas a todos os átomos. Além disso, para cada relação entre os átomos, deve existir um atributo para cada possível tupla da relação. Seguindo esta abordagem, verifica-se que o número de atributos para relação é polinomial de acordo com o número de objetos disponíveis. Outro importante problema reside no fato de que para se representar uma molécula, deve-se impor uma ordem aos seus átomos, uma vez que, sem uma ordem predefinida, existe um número exponencial de representações equivalentes da mesma estrutura.

Claramente, estes problemas proíbem uma representação eficiente por par atributo-valor e torna clara a necessidade de se apelar para uma linguagem com maior capacidade de expressão.

Com a utilização de Lógica de Primeira Ordem, não é necessário fixar um número máximo de atributos, tão pouco criar um atributo para cada possível tupla da relação. Cada átomo pode ser descrito usando apenas as propriedades que lhes são relativas, e relações podem ser representadas por predicados cujos argumentos são os átomos envolvidos na relação e também o tipo de relação. Considere, por exemplo, uma conexão entre dois átomos a_1 e a_2. Esta relação poderia ser representada de forma clara e objetiva por: *conexão(a_1, a_2, tipo_de_conexão).*

Pela discussão exposta, fica claro que, para certos problemas, o uso de uma linguagem mais expressiva é fundamental para uma boa representação do problema, o que acaba por culminar em uma tradução mais simples do problema em termos computacionais.

2.3.2 A Estrutura do Espaço de Hipóteses

Como apresentado, o problema de ILP pode ser visto como um problema de busca no espaço de hipóteses que respeitem a definição 2.19 (ou uma relaxação desta). Contudo, uma das desvantagens de se utilizar a representação por LPO é que o espaço de busca é geralmente muito maior quando comparado a representações que usam LP (MITCHELL, 1997). Por esta razão, a maioria dos sistemas ILP limita sua busca a uma parte do espaço de hipóteses (como poderá ser verificado na próxima seção em que se revisam alguns sistemas ILP). Outro importante aspecto, usado por alguns sistemas ILP quando exploram o espaço de busca, é a implícita ordenação das hipóteses neste espaço. De fato, o espaço de busca pode ser estruturado por uma ordenação das hipóteses mais gerais para as mais específicas. Para um melhor entendimento desta estruturação, considere a definição a seguir.

Definição 2.21 (Generalidade de Hipóteses) (MITCHELL, 1997)**:** Sejam H_1 e H_2 duas hipóteses. H_1 é mais geral que H_2 ou tão geral quanto H_2, denotado por $H_1 \geq^g H_2$, se e somente se cada exemplo coberto por H_2 for também coberto por H_1.[8]

[8] Pode-se também falar no conceito de H_1 ser estritamente mais geral que H_2 se e somente se ($H_1 \geq^g H_2$) e ($H_2 \geq^g H_1$).

Pode-se imaginar o espaço de hipóteses organizado a partir da definição acima. Por exemplo, caso tenha-se apenas um predicado *p* de aridade 2, duas variáveis *X* e *Y*, e duas constantes *a* e *b*, o espaço de hipótese, consistindo de apenas um átomo, pode ser visto como um látice com uma ordenação das hipóteses mais gerais para as específicas. Tal látice é apresentado na figura 2.4 (DIVINA, 2004).

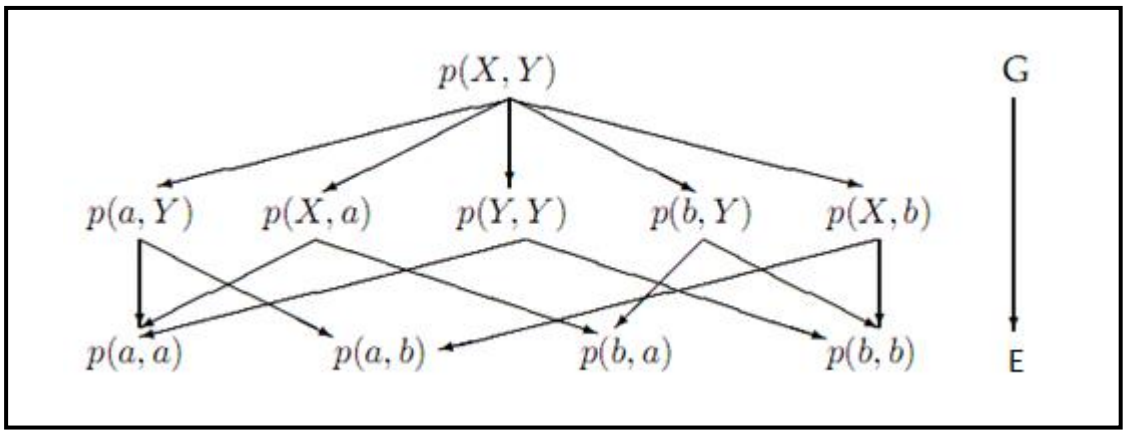

Figura 2.4: Espaço de Hipóteses

De forma geral, quanto menos restrições uma hipótese apresenta, mais geral ela é (MITCHELL, 1997). Especificamente em ILP, pode-se perceber que a generalidade de uma cláusula está ligada diretamente à quantidade de literais em seu corpo e à quantidade de variáveis que compõem os argumentos do literal. De forma geral, pode-se dizer que quanto menos literais uma cláusula possuir no corpo, mais geral esta cláusula será. Um raciocínio equivalente vale para os argumentos dos literais das cláusulas, ou seja, quanto mais variáveis em lugar de constantes existirem nos argumentos do literal, mais geral será este literal e, consequentemente, mais geral será a cláusula que possui este literal em seu corpo.

Muitos sistemas ILP usam esta ordenação do espaço de hipótese nos operadores usados para explorar o espaço de busca. Resumidamente, um operador de exploração recebe uma hipótese, a modifica, e retorna a hipótese modificada. Alguns sistemas começam sua busca partindo da hipótese mais específica a qual é generalizada (pela aplicação de operadores de generalização) durante o processo de aprendizado. Esta abordagem é chamada de *bottom-up*. Em contrapartida, sistemas que utilizam a abordagem *top-down* começam sua busca com a hipótese mais geral que é então especializada (pela aplicação de operadores de especialização) durante o processo de aprendizado. Existem, ainda, sistemas que não realizam especificamente uma busca *top-down* ou *bottom-up*, mas sim uma mistura de ambas, começando com uma hipótese

qualquer (não necessariamente a mais geral ou a mais específica) generalizando-a e/ou especificando-a de tempos em tempos, sem se ater a apenas uma forma de exploração.

2.4 Alguns Sistemas de Programação em Lógica Indutiva

Esta seção apresenta uma breve revisão de alguns dos sistemas considerados estado da arte na área de ILP. Como estes sistemas utilizam algumas definições específicas, seja para definir como sua busca é realizada, ou mesmo para restringi-la, faz-se necessária a apresentação destas definições. Dessa forma, esta seção apresenta, primeiramente, algumas definições necessárias ao entendimento dos sistemas considerados, para, posteriormente, apresentá-los.

2.4.1 Conceitos Preliminares

I. Algoritmo de Cobertura

Grande parte dos sistemas ILP utiliza um algoritmo de cobertura (MITCHELL, 1997) em seu ciclo básico de execução. Este algoritmo está exposto no algoritmo 2.1. Mais tarde, verificar-se-á que os três sistemas de ILP apresentados nesta seção, isto é, o FOIL (QUINLAN, 1990), o Progol (MUGGLETON, 1995) (MUGGLETON, 1996), e o Aleph (SRINIVASAN, 2003) utilizam este algoritmo em seu ciclo de execução.

A idéia geral de todos os algoritmos de cobertura para ILP pode ser resumida nos seguintes passos:

1. Aprenda uma cláusula que prove certa quantidade de exemplos positivos.
2. Remova todos os exemplos positivos provados pela cláusula.
3. Repita passos 1 e 2 até que todos os exemplos positivos estejam provados ou até que outro critério de parada seja satisfeito.

Algoritmo 2.1 – Típico Algoritmo de Cobertura

```
Entrada:
     (E+, E-): conjunto de exemplos de treinamento formado por
     exemplos positivos E+ e por exemplos negativos E-;
Variáveis Locais:
     teoria (conjunto de cláusulas);
Saída:
     teoria modificada;
Começo-Algoritmo-Cobertura:
01– teoria := ∅;
02– Enquanto (|E+| ≠ 0) faça:
03–   cláusula := Aprenda_cláusula((E+,E-));
04–   teoria := teoria ∪ cláusula;
05–   E+ := E+ - exemplos provados por cláusula;
06- fim-enquanto;
07– retorne teoria;
Fim-Algoritmo-Cobertura.
```

Pela utilização do algoritmo de cobertura, fica claro que os sistemas aprendem a teoria passo a passo, isto é, aprendem uma cláusula por vez. Cada cláusula aprendida é então adicionada à teoria, até que todo o conjunto de exemplos positivos seja provado ou até outro critério de parada qualquer ser satisfeito.

II. Modos, *Determinations*, e a *Bottom Clause*[9]

Enquanto o algoritmo de cobertura apresentado na seção anterior dita, de certa forma, *como* a teoria será construída, os modos, os *determinations* e a *bottom clause* (BC), ditam, de forma geral, *quais* são as possíveis cláusulas a serem consideradas durante a busca. De forma geral, pode-se dizer que estes restringem o espaço de busca a certas cláusulas que satisfaçam certos tipos de restrições. Estas restrições ficarão mais claras nas seções seguintes.

a) Modos:

As declarações de modos descrevem relações (predicados) entre objetos de dados e tipos desses dados. Adicionalmente, estas declarações informam se a relação estipulada pode ser utilizada na cabeça (declarações do tipo *modeh*) ou no corpo (declarações do

[9] Esta seção se baseou em (DUBOC et al., 2009) e (DUBOC, 2008).

tipo *modeb*) das cláusulas a serem geradas. Na declaração de modos, também são descritos o tipo dos argumentos e o número máximo de instanciações de cada predicado. Tais declarações têm o formato *mode(Número_Chamadas, Modo)*.

O parâmetro *Número_Chamadas*, ou, *recall_number*, determina o limite máximo do número de instanciações alternativas de um predicado, sendo que uma instanciação é a substituição dos argumentos do predicado por variáveis ou constantes. O *recall_number* pode ser qualquer número positivo $n \geq 1$, ou um *, indicando que não existe limite de instanciações. Para se atribuir um número positivo específico para o *recall_number*, existe a necessidade de se conhecer o número de instanciações possíveis para um determinado predicado. Por exemplo, para uma declaração do predicado ancestral(Ancestral, Pessoa), poderia ser fornecido um *recall_number* igual a *, pois, a princípio, não se sabe quantos ancestrais uma pessoa pode ter.

O *Modo* indica o formato do predicado que será utilizado. Esta declaração é realizada da seguinte forma:

predicado(Tipo_do_Argumento_1, ... , Tipo_do_Argumento_n).

Considere o exemplo a seguir que mostra algumas declarações de modos.

Exemplo 2.13: Considere o aprendizado da relação *sogra(Sogra, Genro)*, com BK descrito pelas relações *progenitor(Mãe, Filha)* e *esposa(Esposa, Marido)*. As declarações de modo poderiam ser (já na sintaxe correta de como devem ser realizadas):

:- modeh(1, sogra(+mulher, +homem)).
:- modeb(, progenitor(+mulher, -mulher)).*
:- modeb(1, esposa(+mulher, +homem)).

onde os símbolos + e - serão explicados a seguir. A primeira declaração, *modeh(1, sogra(+mulher, +homem))*, indica o predicado que irá compor a cabeça das cláusulas, neste caso o predicado *sogra/2* (/2 significa que o predicado possui aridade 2, ou seja, dois argumentos). O valor de *recall_number* é igual a 1 para este predicado, pois para uma dado *homem* tem-se uma única *esposa* e, portanto, uma única *sogra*. Continuando com a interpretação desta declaração, tem-se que o primeiro argumento do predicado *sogra(Sogra, Genro)*, i.e., *Sogra*, deve ser do tipo *mulher*, e *Genro* deve ser do tipo *homem*. A segunda e a terceira declaração indicam, respectivamente, que as cláusulas geradas devem ter no corpo o predicado *progenitor(Mae, Filha)* e *esposa(Esposa, Marido)* nos quais, *Mãe*, *Filha* e *Esposa* são do tipo *mulher* e *Marido* é do tipo *homem*. O valor do *recall_number* para o predicado *progenitor(Mae, Filha)* é igual a *, uma vez

que não se sabe quantas filhas uma mãe pode ter. Para o predicado *esposa(Esposa, Marido)*, este parâmetro tem o valor 1, uma vez que para uma dada *esposa* tem-se um único *marido* e vice-versa.

Os tipos dos argumentos podem ser +, - ou #. O símbolo '+' indica que o argumento do predicado é uma variável de entrada. O símbolo '-' indica uma variável de saída enquanto que '#' indica que este argumento é uma constante. A utilização destes tipos de variáveis deve seguir as seguintes regras:

- Variáveis de Entrada (+): qualquer variável de entrada do tipo *T* em um literal do corpo B_i deve aparecer como uma variável de saída do tipo *T* em um literal do corpo antes de B_i, ou como uma variável de entrada do tipo *T* na cabeça.
- Variáveis de Saída (-): qualquer variável de saída do tipo *T* em um literal na cabeça deve aparecer como uma variável de saída do tipo *T* em um literal do corpo B_i. Qualquer literal que contenha uma variável de saída deve ter, no mínimo, uma variável de entrada. Isto pode ser verificado na declaração de modos especificada acima.
- Constantes (#): qualquer argumento denotado por # só pode ser substituído (instanciado) por constantes.

Os tipos devem ser especificados para cada argumento dos predicados utilizados na construção de uma cláusula. Para o Progol e o Aleph, por exemplo, os tipos são apenas nomes, e a declaração de tipos nada mais é que um conjunto de fatos. Por exemplo, a descrição dos objetos dos tipos *homem* e *mulher* poderiam ser: *mulher(maria), mulher(sueli), homem(paulo), homem(pedro).*

b) *Determinations:*

Alguns sistemas, como o Progol e o Aleph utilizam a relação *determination/2* para declarar os predicados que podem ser usados no corpo de uma cláusula. Esta declaração possui o seguinte formado:

determination(Predicado_Alvo/Aridade, Predicado_Corpo/Aridade)

em que *Predicado_Alvo* é o predicado que irá aparecer na cabeça da cláusula induzida, e *Predicado_Corpo* é o predicado que irá aparecer no corpo da cláusula induzida. Note que ambos os predicados devem ser sucedidos por sua aridade. Por exemplo, para o aprendizado do predicado alvo *sogra(Sogra, Genro)*, uma possível declaração seria:

:- determination(sogra/2, progenitor/2)

:- determination(sogra/2, esposa/2)

Estas declarações afirmam que a cláusula que possui o predicado *sogra/2* na cabeça, pode possuir em seu corpo os predicados *progenitor/2* e/ou *esposa/2*.

c) *A Bottom Clause:*

Como apresentado na seção 2.3.3, alguns sistemas ILP utilizam a ordenação do espaço de hipóteses para executar a sua busca. De forma geral, alguns sistemas ILP limitam inferiormente e superiormente o espaço de busca de uma única cláusula, formando, desta forma, um látice. Este látice é geralmente limitado superiormente pelo elemento mais geral, ou seja, a cláusula vazia, enquanto que o limite inferior é dado pela cláusula mais específica (*bottom clause*), i.e, o elemento menos geral do látice. Assumindo que L_i(M) denota a linguagem dos modos, e $|\text{-}_h \square$ denota a derivação da cláusula vazia em no máximo h resoluções, a *bottom clause* pode ser definida como segue (MUGGLETON, 1995).

Definição 2.22 (*Bottom Clause* $\perp_i$): Assuma que *h* e *i* são números naturais, *B* é um conjunto de cláusulas de Horn, $e = a \leftarrow b_1,..., b_n$, é uma cláusula definida, *M* é um conjunto de declaração de modos contendo exatamente um único modeh *m*, tal que $a(m) \preceq a$ e, $\perp$ sendo a cláusula definida mais específica (potencialmente infinita) tal que $B \wedge \perp \wedge \neg$ e $|\text{-}_h \square$. $\perp_i$ é a cláusula mais específica em $L_i(M)$ tal que $\perp_i \preceq \perp$.

O algoritmo 2.2 apresenta as etapas de construção da *bottom clause*. O *recall* é usado para determinar quantas vezes o interpretador Prolog será executado para cada instanciação da cláusula no passo 16. A profundidade máxima de variável determina quantas vezes o passo 16 será executado. A função *hash*: *Terms* → N é uma função que mapeia termos em variáveis diferentes.

Algoritmo 2.2: Passos para a construção de uma *bottom clause*

Entrada:
(E+, BK): o conjunto de exemplos positivos E+, e o conhecimento preliminar BK
(modos, determinations, e tipos): declarações de modos, *determinations* e tipos.
Variáveis Locais:
(inTerms): lista que armazena os termos que instanciam argumentos de entrada no predicado da cabeça, e os termos que instanciam argumentos de saída nos predicados do corpo.
Saída:
Bottom clause $\perp$;

Começo-Construção-*Bottom-clause*:
01- Adicione *e* $\in$ *E+* ao conhecimento preliminar;
02- *inTerms* := $\varnothing$, $\perp$:= $\varnothing$;
03- Ache a primeira declaração *mode* de cabeça *h* tal que *h* subsume *e* com substituição θ;
04- Para cada *v/t* em θ faça:
05- Se *v* corresponde a um tipo # então:
06- Substitua *v* em *h* por *t;*
07- fim-se;
08- Se *v* corresponde a um tipo + ou -, então:
09- Substitua *v* em *h* por v_k onde v_k é uma variável tal que *k* = *hash(t);*
10- fim-se;
11- Se *v* corresponde a um tipo + então:
12- Adicione *t* ao conjunto *inTerms*;
13- fim-se;
14- Adicione *h* a $\perp$.
15- fim-para;
16- Para cada declaração *mode* de corpo *b* faça:
17- Para toda substituição possível θ de argumentos correspondendo a tipo + por termos no conjunto *inTerms* faça:
18- Repita quantas vezes for o número *Recall*:
19- Se Prolog tem sucesso no objetivo *b* com substituição θ' então:
20- Para cada *v/t* em θ e θ' faça:
21- Se *v* corresponde a tipo # então:
22- Substitua *v* em *b* por *t*;
23- senão substitua *v* em *b* por v_k onde *k* = *hash(t)*;
24- fim-se;
25- Se *v* corresponde a tipo – então:
26- Adicione *t* ao conjunto *inTerms;*
27- fim-se;
28- Adicione *b* à $\perp$;
29- fim-para;
30- fim-se;
31- fim-repita;
32- fim-para;
33- Incremente a profundidade de variável;
33- Vá para o passo 4 se a profundidade máxima de variável não foi alcançada;
34- fim-para;
35- Retorne $\perp$;
Fim-Construção-*Bottom-clause*.

Para ilustrar o passo de construção da *bottom clause* considere o exemplo 2.14 (FERRO et al., 2007).

Exemplo 2.14: Considere as seguintes declarações a seguir que serão usadas para exemplificar a construção de uma bottom clause.

```
/* Modos e Determinations */
:- modeh(1, sogra_de(+mulher, -homem)).
:- modeb(*, progenitor_de(+mulher, -mulher)).
:- modeb(1, esposa_de(+mulher, -homem)).
:- determination(sogra_de/2, progenitor_de/2).
:- determination(sogra_de/2, esposa_de/2).

/* Declaração dos Tipos */
homem(pai1).
homem(marido1).
homem(marido2).
mulher(mae1).
mulher(filha11).
mulher(filha12).
mulher(neta11).

/* Conhecimento preliminar */
progenitor_de(mae1, filha11).
progenitor_de(pai1, filha11).
progenitor_de(mae1, filha12).
progenitor_de(pai1, filha12).
progenitor_de(filha11, neta11).
esposa_de(mae1, pai1).
esposa_de(filha11, marido1).
esposa_de(filha12, marido2).

/* Exemplos Positivos */
sogra_de(mae1, marido1).
sogra_de(mae1, marido2).

/* Exemplos Negativos */
sogra_de(mae1, pai1).
```

sogra_de(filha11, marido2).
sogra_de(filha11, marido1).

No algoritmo 2.2 a lista *inTerms* armazena os termos que instanciam argumentos de entrada no predicado da cabeça, e os termos que instanciam argumentos de saída nos predicados do corpo. A função *hash* associa a cada termo uma variável diferente.

A construção da *bottom clause* pode ser vista como uma árvore, em que cada vez que o passo 16 é executado, um nível desta é construído. A construção da *bottom clause* considerando o exemplo positivo *sogra_de(mae1, marido1)* segue os passos abaixo.

No início do algoritmo, a lista *inTerms* e a *bottom clause* estão vazias. O primeiro passo do algoritmo é encontrar uma declaração *modeh* tal que *h* subsume o exemplo *e* com uma determinada substituição θ. A primeira e única declaração *modeh* para este exemplo é:

modeh(1, sogra_de(+mulher, -homem)).

A substituição θ é dada por:

θ = {*+mulher/mae1, -homem/marido1*}

Como os argumentos de *sogra_de* correspondem a tipos + e -, os argumentos serão substituídos por duas variáveis novas *A* e *B*, retornando o predicado:

sogra_de(A ,B)

onde o *A* corresponderá a *mae1* e o *B* a *marido1*. O termo *mae1* é colocado na lista *inTerms* pois o argumento substituído por *mae1* é do tipo +. O predicado *sogra_de(A, B)* é colocado na *bottom clause*. Portanto tem-se:

inTerms = {*mae1*}, $\perp$ = {*sogra_de(A, B)*}

Neste momento, tem-se o primeiro nível da árvore mostrado na figura 2.5.

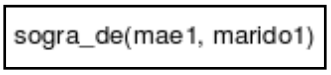

Figura 2.5: Árvore de construção da bottom clause – nível 1

Os próximos passos são feitos para cada declaração *modeb*. A primeira declaração é: *modeb(*, progenitor_de(+mulher, -mulher)).* As possíveis substituições θ de argumentos correspondentes ao tipo + por termos no conjunto *inTerms* são dadas por:

θ = {*+mulher/mae1*}

Como o *recall* de *progenitor_de* é *, o passo a seguir será repetido para cada fato do predicado *progenitor_de* que tenha o termo *mae1* como primeiro argumento. O primeiro fato encontrado é:

progenitor_de(mae1, filha11)

com substituição θ' = {*-mulher/filha11*}

Os argumentos de *progenitor_de* são substituídos por variáveis, sendo que *mae1* já corresponde à variável *A*, e a variável *C* é introduzida para o termo *filha11*. O predicado *progenitor_de(A, C)* é colocado na *bottom clause*. Como o termo *filha11* instancia um argumento do tipo -, ele é incluído na lista *inTerms*. Portanto tem-se:

inTerms = {*mae1, filha11*}, $\perp$ = {*sogra_de(A, B), progenitor_de(A, C)*}

O segundo fato encontrado é:

progenitor_de(mae1, filha12)

com substituição θ'= {*-mulher/filha12*}.

Os argumentos de *progenitor_de* são substituídos por variáveis, sendo que *mae1* já corresponde à variável *A*, e a variável *D* é introduzida para o termo *filha12*. O predicado *progenitor_de(A, D)* é colocado na *bottom clause*. Como o termo *filha12* instancia um argumento do tipo -, ele é incluído na lista *inTerms*. Portanto tem-se:

inTerms = {mae1, filha11, filha12}

$\perp$= {*sogra_de(A, B), progenitor_de(A, C), progenitor_de(A, D)*}

Não existem mais fatos para *progenitor_de* tendo *mae1* como primeiro argumento. Portanto passa-se para a próxima declaração *modeb*, dada por:

modeb(1, esposa_de(+mulher, -homem))

Considerando-se a substituição *+mulher/mae1*, como o *recall* de *esposa_de* é 1, será repetido o passo a seguir apenas uma vez. Assim, procura-se um fato com o predicado *esposa_de* que tenha *mae1* como primeiro argumento. O fato encontrado é:

esposa_de(mae1, pai1)

com substituição θ' = {*-homem/pai1*}

Os argumentos de *esposa_de* são substituídos por variáveis, sendo que *mae1* já corresponde à variável *A*, e a variável *E* é introduzida para o termo *pai1*. O predicado

esposa_de(A, E) é colocado na *bottom clause*. Como o termo *pai1* instancia um argumento do tipo -, ele é incluído na lista *inTerms*. Portanto tem-se:

inTerms = {*mae1, filha11, filha12, pai1*}

⊥= {*sogra_de(A,B), progenitor_de(A,C), progenitor_de(A,D), esposa_de(A,E)*}

Neste momento forma-se o segundo nível da árvore de construção da *bottom clause*, como mostrado na figura 2.6.

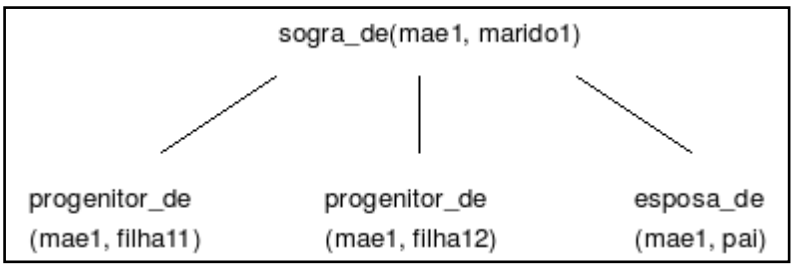

Figura 2.6: Árvore de construção da bottom clause – nível 2

Pela figura 2.6 pode-se ver que o nível 2 da árvore é formado por todos os fatos envolvendo os predicados do corpo *progenitor_de* e *esposa_de* que contenham como argumento de entrada o termo *mae1*, que é justamente o termo de entrada vindo da cabeça da regra dada por *sogra_de(mae1, marido1)*.

De acordo com a própria definição de variável de entrada, tem-se que uma variável de entrada em um literal do corpo deve ter aparecido antes como variável de saída em outro literal do corpo, ou como variável de entrada no literal da cabeça. Como neste momento do algoritmo formam-se os primeiros literais do corpo, são gerados literais cujas variáveis de entrada destes sejam variáveis de entrada do literal da cabeça, e até agora o único termo que representa uma variável de entrada no literal da cabeça é *mae1*.

Formado o segundo nível da árvore, passa-se para o passo 33 do algoritmo que corresponde a incrementar a profundidade de variável. Pode-se entender claramente agora o que significa esta profundidade. A profundidade de variável é um parâmetro que irá indicar quantos níveis a árvore de construção da *bottom clause* pode ter, e este parâmetro deverá ser definido no início do algoritmo. Em cada nível, novas variáveis serão introduzidas a partir dos termos do nível anterior. Se continuarmos o algoritmo, estaremos permitindo que novos literais do corpo sejam gerados a partir dos novos termos introduzidos no nível 1, dados por *filha11*, *filha12* e *marido1*. Estes termos foram introduzidos como argumentos de saída e, portanto, agora poderão ser

argumentos de entrada nos novos literais do corpo que serão gerados. Continuando o algoritmo e voltando para o passo 16 tem-se:
A primeira declaração *modeb* encontrada é

modeb(, progenitor_de(+mulher, -mulher))*

que terá o seu argumento de entrada substituído por *filha11* e *filha12*, ambos do tipo *mulher*. Como o *recall* é *, buscam-se quantos fatos existirem para as substituições dadas. Apenas o fato *progenitor_de(filha11, neta11)* será encontrado. O termo *filha11* já está vinculado à variável *C*, e o termo *neta11* será vinculada à nova variável *F*. Assim, tem-se:

$\perp$ = {*sogra_de(A, B), progenitor_de(A, C), progenitor_de(A, D), esposa_de(A, E), progenitor_de(C, F)* }

Para a próxima declaração *modeb* dada por

modeb(1, esposa_de(+mulher, -homem))

ter-se-á o argumento de entrada substituído por *filha11* e *filha12*, ambos do tipo *mulher*. Não se pode utilizar o termo *pai1*, pois este é do tipo *homem*. Como o *recall* é 1, só pode-se buscar um fato para *filha11* como primeiro argumento e um fato para *filha12* como primeiro argumento. Os fatos encontrados serão *esposa_de(filha11, marido1)* e *esposa_de(filha12, marido2)*. Apenas o termo *marido2* não havia sido introduzido antes, e, portanto, é vinculado à nova variável *G*. Tem-se, portanto:

$\perp$ = {*sogra_de(A, B), progenitor_de(A, C), progenitor_de(A, D), esposa_de(A, E), progenitor_de(C, F), esposa_de(C, G)* }

Neste momento forma-se o terceiro nível da árvore de construção da *bottom clause*, mostrado na figura 2.7.

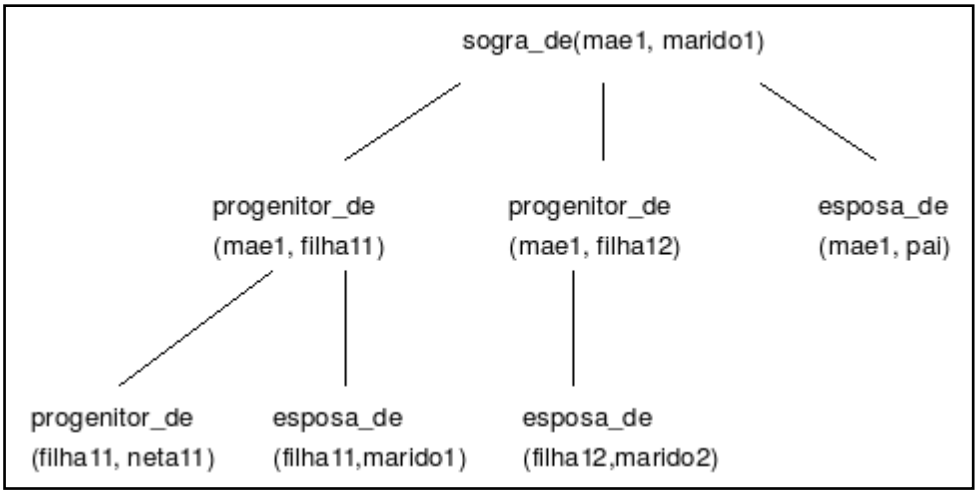

Figura 2.7: Árvore de construção da bottom clause – nível 3

Se continuássemos o algoritmo e fôssemos para o próximo nível da árvore, procuraríamos fatos que tivessem os termos *neta11*, *marido1* e *marido2* como argumentos de entrada nos literais do corpo, pois estes foram os novos termos introduzidos no nível anterior como argumentos de saída. Porém, olhando para o conhecimento preliminar deste exemplo, veremos que tais fatos não existem e, portanto o algoritmo irá terminar. Portanto a *bottom clause* encontrada é:

$\perp$ = {*sogra_de(A, B), progenitor_de(A, C), progenitor_de(A, D), esposa_de(A, E), progenitor_de(C, F), esposa_de(C, G)* }

Pragmaticamente, a *bottom clause* é construída utilizando um exemplo positivo, o BK, a declaração de modos, e as *determinations*. Esta cláusula tem a propriedade de provar, em alguns passos de resolução, o exemplo positivo utilizado em sua construção. De forma geral, pode-se dizer que sistemas que utilizam a *bottom clause* como limite inferior do látice e a cláusula vazia como limite superior desta estrutura, limitam seu espaço de busca a cláusulas que possuem em seu corpo um subconjunto dos literais presentes no corpo da *bottom clause*. Este procedimento ficará mais claro na próxima seção, onde serão apresentados alguns sistemas que utilizam tal limitação do espaço de busca de uma cláusula.

2.4.2 Sistemas ILP

I. FOIL (*First Order Inductive Learner*)

O FOIL (QUINLAN, 1990) realiza sua busca por cláusulas utilizando a estratégia *top-down* e um algoritmo de cobertura. O sistema começa com a cláusula mais geral que

possui como cabeça o predicado alvo e corpo vazio. Todos os argumentos da cabeça são variáveis diferentes e, desta forma, a cláusula inicial prova todos os exemplos, ou seja, classifica todos os exemplos como positivos. A cláusula é então especializada pela adição de alguns literais ao seu corpo. Vários literais são considerados com este propósito, sendo o que leva a um melhor ganho (considerando a métrica ganho de informação - explicada abaixo) adicionado ao corpo da cláusula. Literais são adicionados até que a cláusula não prove nenhum exemplo negativo. Uma vez induzida uma cláusula consistente (que não prova nenhum exemplo negativo) ela é armazenada e, com um típico algoritmo de cobertura, o sistema remove os exemplos positivos que são provados por esta cláusula, e todo processo é repetido, até que todos os exemplos positivos sejam provados. O algoritmo 2.3 descreve o funcionamento geral do sistema FOIL.

No passo 4, o FOIL usa um algoritmo *Hill Climbing* (RUSSELL, NORVIG, 2003) em busca de um "bom" literal. A qualidade de um literal é mensurada pela métrica chamada de ganho de informação, e o literal que maximiza este ganho é o escolhido para ser adicionado à cláusula. Especificamente, considere *C* a cláusula à qual um novo literal *L* será adicionado, e *C*' a cláusula criada a partir de *C* pela adição de *L* (*C*' = *C* ∧ *L*). A métrica ganho de informação é estipulada como mostra equação 2.1.

$$Ganho\_Informação = sc \times \left(log \frac{pc'}{pc' + nc'} - log \frac{pc}{pc + nc} \right) \qquad (2.1)$$

onde *pc*, *pc*', *nc*, *nc*' são os exemplos positivos e negativos cobertos, respectivamente por *C* e *C*', e *sc* é o número de exemplos positivos que são cobertos por *C* e que continuam sendo cobertos por *C* após *L* ter-lhe sido adicionado.

Algoritmo 2.3 – Funcionamento Geral do Sistema FOIL

```
Entrada:
     (E+, E-): conjunto de exemplos de treinamento formado por
     exemplos positivos E+ e por exemplos negativos E-;
     BK: Conhecimento preliminar;
Variáveis Locais:
     cláusula, teoria (conjunto de cláusulas);
Saída:
     teoria modificada;
Começo-FOIL:
01– teoria := ∅;
02– cláusula := □;
03– Enquanto (cláusula provar exemplos negativos) faça:
04-   Encontre um "bom" literal para ser adicionado à cláusula;
06- fim-enquanto;
05– teoria := teoria ∪ cláusula;
06– E+ := E+ - exemplos positivos provados por cláusula;
07– Se (|E+| ≠ 0) então
08–   retorne ao passo 2;
09- fim-se;
10– retorne teoria;
Fim-FOIL.
```

Os literais adicionados a uma cláusula C podem ser da seguinte forma (QUINLAN, 1990):

- $P(X_1,..., X_k)$, e $\neg P(X_1,..., X_k)$, onde X_i^s são variáveis já presentes na cláusula, ou novas variáveis.
- $X_i = X_j$, ou $X_i \neq X_j$, para variáveis presentes na cláusula.
- $X_i = c$, ou $X_i \neq c$, onde X_i é uma variável na cláusula e c é uma constante.
- $X_i \leq X_j$, $X_i > X_j$, $X_i \leq v$, $X_i > v$, onde X_i e X_j são variáveis da cláusula que assumem valores numéricos e v é um limite escolhido pelo FOIL.

Existe uma restrição sobre os literais que podem ser colocados na cláusula. Tal restrição dita que ao menos uma variável aparecendo no literal a ser adicionado deve já fazer parte da cláusula. Outra restrição adotada pelo FOIL se relaciona ao tamanho da cláusula. Esta restrição dita que se uma cláusula fica muito extensa (com muitos literais)

quando comparada ao número de exemplos positivos que esta cláusula prova, tal cláusula não é mais potencialmente considerada como parte da teoria.

Devido ao grande espaço de busca explorado pelo FOIL, tal sistema é geralmente superado pelos sistemas Progol e Aleph que usam a *bottom clause* como forma de limitar seus espaços de busca. Tais sistemas são brevemente explicados a seguir.

II. Progol

O sistema Progol (MUGGLETON, 1995), (MUGGLETON, 1996) usa o algoritmo de cobertura como forma de construir suas teorias juntamente com a construção da *bottom clause* que é usada para limitar inferiormente o látice que representa o espaço de busca de cláusulas isoladas. Dessa forma, o sistema Progol se restringe a buscar por cláusulas que são mais gerais que a *bottom clause* gerada, ou seja, este sistema busca por cláusulas cujo corpo é constituído por um subconjunto dos literais presentes no corpo da *bottom clause*. O algoritmo 2.4 ilustra o funcionamento geral do sistema Progol.

Algoritmo 2.4 – Funcionamento Geral do Sistema Progol

Entrada:
(E+, E-): conjunto de exemplos de treinamento formado por exemplos positivos E+ e por exemplos negativos E-;
BK: Conhecimento preliminar;
Variáveis Locais:
cláusula, *bottom clause* (⊥), teoria (conjunto de cláusulas);
Saída:
teoria modificada;

```
Começo-Progol:
01- teoria := ∅;
02- Enquanto (|E+| ≠ 0) faça:
03-   Selecione e ∈ E+;
04-   Construa ⊥ a partir de e;
05-   Encontre uma "boa" cláusula entre ⊥ e □ usando o A*;
06-   teoria := teoria ∪ cláusula;
07-   E+ := E+ - exemplos positivos provados por cláusula;
08- fim-enquanto;
08- retorne teoria;
Fim-Progol.
```

Primeiramente, é importante notar que para cada cláusula adicionada à teoria, uma *bottom clause* foi construída (passo 4) utilizando um dos exemplos positivos em *E+* que ainda não foi provado. Na linha 5, o algoritmo A* é usado para encontrar uma boa cláusula tendo como ponto de partida a cláusula mais geral. De acordo com esta estratégia, um número fixo de cláusulas é construído a partir da cláusula inicial usando um operador de refinamento que especializa a cláusula. A cláusula considerada a melhor é então escolhida e este processo é repetido. O Progol usa a definição de θ-*subsume* (definição 2.16) para buscar por cláusulas que irão constituir sua teoria. O operador de refinamento utilizado pelo Progol mantém a relação □ ⪯ *C* ⪯ ⊥ para qualquer cláusula *C* buscada, o que limita a busca por cláusulas *C* que tais que □ ⪯ *C* ⪯ ⊥.

De acordo com a definição 2.16, como *C* ⪯ ⊥, existe uma substituição θ tal que *C*θ ⊆ ⊥ e, dessa forma, para cada literal *L* em *C* existe um literal *L'* em ⊥ tal que *L*θ = *L'*. Percebe-se que o operador de refinamento deve manter um controle sobre θ e uma lista dos literais em *L'* em ⊥ que possuem um literal correspondente *L* em *C*. Toda cláusula *C* que *subsume* ⊥, possui em seu corpo um subconjunto dos literais de ⊥ com alguma substituição aplicada. O Progol escolhe o melhor entre todos os possíveis refinamentos utilizando uma função de avaliação que incide sobre a cláusula C e é dada por:

$$f(C) = pc - (nc + tc + hc) \tag{2.2}$$

onde *pc* é o número de exemplos positivos provados pela cláusula *C*, *nc* é o número de exemplos negativos provados por *C*, *tc* é o tamanho da cláusula *C* (dado pelo número de literais - 1), e *hc* é o número esperado de literais a ser adicionado ao corpo da cláusula. Como afirmado, o Progol busca apenas por cláusulas *C* que satisfaçam a seguinte relação □ ⪯ *C* ⪯ ⊥; este pode ser considerado um primeiro *bias* no espaço de busca. Outras restrições impostas ao espaço de busca de cláusulas são as declarações de modos e *determinations*, como apresentado na seção 2.4.1.

III. Aleph

De forma geral, pode-se dizer que o Aleph (SRINIVASAN, 2003) é uma implementação do Progol totalmente construída utilizando a linguagem Prolog (GOADRICH, 2009). Dessa forma, as declarações de modos, *determinations*, geração

da *bottom clause* e até mesmo a maneira sequencial de construção de teorias, que é realizada utilizando um algoritmo de cobertura e a *bottom clause*, são os mesmos do sistema Progol. O que torna o Aleph importante e até mesmo (de certa forma) mais popular que o Progol, é a sua potencial flexibilidade, uma vez que o Aleph possui dezenas de parâmetros que torna possível adaptá-lo a uma grande gama de problemas. Como um exemplo de sua notável flexibilidade[10], o Aleph, enquanto busca por cláusulas que formarão uma teoria, pode utilizar-se da busca em largura, ou da busca em profundidade, ou da busca iterativa por feixe (*iterative beam search*), ou da busca de aprofundamento iterativo (*iterative deepening*) (RUSSEL, NORVIG, 2003) dentre outras, inclusive usar métodos heurísticos que requerem a definição de uma função de avaliação.

2.5 Resumo do Capítulo

Este capítulo apresentou o problema de Programação em Lógica Indutiva, e para isto, uma breve revisão sobre Lógica de Primeira Ordem foi realizada. Viu-se que tal problema consiste em induzir uma teoria composta por cláusulas de Horn que prove todos os exemplos positivos e nenhum negativo, embora esta restrição possa ser (e frequentemente é) relaxada. O capítulo apresentou também uma breve motivação de se estudar ILP, uma vez que existem problemas que não podem ser representados facilmente com linguagens de baixa expressividade, como a Lógica Proposicional. Por fim, este capítulo apresentou uma breve revisão de alguns sistemas considerados estado da arte em ILP.

[10] Não é objetivo do trabalho apresentar todos os parâmetros do sistema Aleph e discutir todas as suas possíveis variações. O leitor interessado em tal tópico deve se referir a (SRINIVASAN, 2003).

Capítulo 3: Algoritmos Genéticos e Programação em Lógica Indutiva

Este capítulo apresenta o funcionamento básico de um Algoritmo Genético bem como alguns sistemas ILP baseados neste tipo de algoritmo. Uma variação do Algoritmo Genético padrão, conhecida como Algoritmo Estimador de Distribuição, é também apresentada juntamente com alguns motivadores para o seu uso em ILP, o que, de acordo com pesquisas efetuadas, não foi realizado anteriormente a este trabalho.

3.1 Introdução

Algoritmos Genéticos (AGs) (HOLLAND, 1975) são algoritmos de busca estocásticos inspirados na teoria moderna da evolução surgida em 1940. Esta teoria combina as descobertas sobre a hereditariedade feitas no inicio do século XX com a teoria da evolução descrita por Charles Darwin em a *Origem das Espécies* publicado em 1859. Os AGs usam uma analogia direta com o processo de evolução natural. Tipicamente, em um AG, um conjunto de soluções candidatas (indivíduos) é transformado (evoluído) ao longo de várias iterações (gerações), até que algum critério de parada seja satisfeito. As transformações executadas sobre as soluções candidatas são realizadas pelos operadores de recombinação e mutação. Estes operadores, juntamente com o operador de seleção, são os responsáveis pela simulação do processo de evolução natural e, dessa forma, por guiar o conjunto de soluções candidatas a regiões ainda mais promissoras do espaço de busca.

Os AGs podem ser aplicados a qualquer problema de otimização uma vez que não dependem do domínio do problema para serem aplicados. Além disso, estes algoritmos possuem grande potencial para explorar espaços de busca muito grandes, o que os capacita a encontrar melhores soluções onde outros tipos de técnicas provavelmente não seriam eficazes (MICHALEWICZ, 1999). Esta técnica foi (e é) aplicada com grande sucesso a uma enorme gama de problemas, como jogos, modelagem cognitiva, problemas de transporte, problemas de controle ótimo, problemas de roteamento e, obviamente, a problemas de aprendizado indutivo de conceitos (AGUILAR-RUIZ et al., 2007) (BACARDIT, GARREL, 2003).

Como visto no capítulo anterior, o aprendizado de hipóteses em Lógica de Primeira Ordem é uma tarefa de acentuada dificuldade, devido ao extenso tamanho do espaço de busca e, dessa forma, não demorou até que os primeiros sistemas ILP baseados em AGs fossem construídos. Alguns exemplos são: REGAL (*Relational Genetic Algorithm Learner*) (GIORDANA, NERI, 1996), SIA01 (*Supervised Inductive Algorithm version 01*) (AUGIER, VENTURINI, KODRATOFF, 1995), DOGMA (*Domain Oriented Genetic Machine*) (HEKANAHO, 1996), G-NET (*Genetic Network*) (ANGLANO et al., 1998), ECL (*Evolutionary Concept Learner*) (DIVINA, MARCHIORI, 2002), e QG/GA (*Quick Generalization/Genetic Algorithm*) (MUGGLETON, TAMADDONI-NEZHAD, 2006) (MUGGLETON, TAMADDONI-NEZHAD, 2007), com destaque para este último, uma vez que os primeiros citados se tornaram um tanto quanto obsoletos não possuindo publicações mais atuais. Estes são alguns dos principais sistemas ILP baseados em AGs, o que sugere a importância desta ferramenta no tratamento do problema em questão.

Os Algoritmos Estimadores de Distribuição (AEDs) (MÜHLENBEIN, 1996), ou Algoritmos Genéticos Construtores de Modelos Probabilísticos (AGCMP) (PELIKAN, GOLDBERG, LOBO, 1999) podem ser considerados uma variação dos AGs comuns. Os AEDs omitem totalmente ou parcialmente os operadores de recombinação e mutação presentes nos AGs comuns. Para evoluir sua população, tais algoritmos usam modelos probabilísticos aprendidos sobre os melhores indivíduos de sucessivas gerações. Estes modelos são aprendidos de bons indivíduos (mais adaptados), e então, amostrados para gerar os novos indivíduos da próxima geração. Os AEDs são modelados para captar as interações entre os genes, que representam a estrutura interna das soluções dos problemas e, com isso, estimam diretamente a distribuição de boas soluções em vez de usarem operadores genéticos. Estes algoritmos têm sido aplicados aos mais diversos tipos de problemas de otimização gerando resultados bastante interessantes e superando, na maioria das vezes, o método tradicional de AGs (PELIKAN, 2005). Apesar de apresentarem melhores resultados quando comparados aos AGs tradicionais, os AEDs, até os estudos por hora realizados, não foram aplicados para a construção de sistemas ILP[1].

[1] O trabalho (OLIPHANT, SHAVLIK, 2007), embora pareça ser a única exceção à regra, utiliza apenas a motivação dos AEDs, ou seja, a modelagem de soluções promissoras. Dessa forma, por utilizar apenas esta motivação, não se pode afirmar que se trata de um sistema ILP pautado em AEDs (veja seção 3.5).

Este capítulo se organiza como segue. A seção 3.2 apresenta uma revisão sobre os AGs tradicionais. Um pouco de sua teoria, bem como uma simples analogia com o processo de evolução natural são apresentados e seus principais operadores e operandos são definidos[2]. A seção 3.3 apresenta uma breve revisão de alguns dos sistemas ILP baseados nos AGs tradicionais. A seção 3.4 apresenta uma breve revisão sobre os AEDs, categorizando-os de acordo com a complexidade do modelo probabilístico usado para simular o processo de evolução dos indivíduos. A seção 3.5 apresenta uma revisão do sistema OS (OLIPHANT, SHAVLIK, 2007), uma vez que ele utiliza a idéia fundamental dos AEDs, mas difere em pontos fundamentais desta técnica, não podendo, portanto, ser considerado uma aplicação dos AEDs ao problema de ILP. Finalmente, a seção 3.6 apresenta um resumo do que foi abordado neste capítulo.

3.2 Algoritmos Genéticos: uma Revisão

Como afirmado, os AGs usam uma analogia direta com o processo de evolução natural. Basicamente, as seguintes observações e respectivas conclusões são a essência da teoria da evolução de Charles Darwin (AMABIS, MARTHO, 1994):

Observação 1: As populações naturais de todas as espécies tendem a crescer rapidamente, pois o potencial reprodutivo dos seres vivos é muito grande.

Observação 2: O tamanho das populações naturais, a despeito de seu enorme potencial de crescimento, se mantém relativamente constante ao longo do tempo. Isso se deve ao fato de que número de indivíduos numa população é limitado pelo ambiente (disponibilidade de alimento e locais de procriação, presença de predadores naturais, parasitas etc.).

Conclusão I: Em cada geração, morre grande número de indivíduos, muitos deles sem deixar descendentes.

Observação 3: Os indivíduos de uma população diferem quanto a diversas características, inclusive aquelas que influenciam na capacidade de explorar, com sucesso, os recursos naturais e de deixar descendentes.

[2] Para mais detalhes sobre tal técnica, recomenda-se a leitura do apêndice B desta tese que situa os AGs como um algoritmo de busca e também revisa a definição de esquemas e blocos construtores que são conceitos essenciais que explicam, de certa forma, o motivo do funcionamento dos AGs e motivaram a criação dos AEDs.

Conclusão II: Os indivíduos que sobrevivem e se reproduzem, a cada geração, são, preferencialmente, os que apresentam determinadas características relacionadas com adaptação às condições ambientais. Essa conclusão resume o conceito Darwinista de seleção natural ou sobrevivência do mais apto.

Observação 4: Grande parte das características apresentadas por uma geração é herdada dos pais.

Conclusão III: Uma vez que a cada geração os indivíduos mais aptos sobrevivem, eles tendem a transmitir aos descendentes as características relacionadas a essa maior aptidão para sobreviver, i.e., para se adaptar. Em outras palavras, a seleção natural favorece, ao longo de gerações sucessivas, a permanência e o aprimoramento de características relacionadas à adaptação.

Resumidamente, os indivíduos mais adaptados ao ambiente possuem maiores chances de produzirem descendentes. Estes descendentes herdam as características de seus pais e consequentemente tendem a ser mais adaptados ao ambiente. Dessa forma, depois de um logo período de tempo, as características que garantem uma melhor adaptação aos indivíduos tendem a permanecer, enquanto que as características que influenciam os indivíduos de maneira negativa tendem a desaparecer. Assim, a população aproxima-se de uma adaptação ótima ou sub-ótima ao seu ambiente.

A teoria moderna da evolução incorpora a tese central do Darwinismo (a teoria da seleção natural), mas é inovadora porque explica como surge a diversidade de características nos indivíduos de uma população, um ponto que Darwin não pôde explicar em sua época.

A teoria moderna da evolução admite que as variações hereditárias presentes nos indivíduos estão diretamente relacionadas aos genes e cromossomos. Dois mecanismos principais são responsáveis pela origem dessas variações: a recombinação e a mutação de genes. Novas combinações gênicas e novos genes originados através destes dois fenômenos garantem que os indivíduos de uma espécie sejam geneticamente variados a cada geração. Essa capacidade de gerar diversidade, denominada de variabilidade gênica, fornece a matéria prima sobre a qual atua a seleção natural.

Os AGs são projetados para simular o processo de evolução, e, para isto, utilizam os vários conceitos relacionados à teoria moderna da evolução. As próximas seções apresentam os componentes principais de um AG, sendo que estes foram divididos em operandos (gene, cromossomo, e população) e operadores (mutação,

recombinação e seleção). A função de aptidão, embora esteja diretamente relacionada a um cromossomo, será definida separadamente após a apresentação dos operandos, uma vez que ela não pode ser considerada nem um operando nem um operador. Após a apresentação dos operadores, operandos e da função de aptidão, é apresentado o ciclo básico de execução de um AG e, posteriormente, são apresentados e discutidos seus principais parâmetros.

3.2.1 Os Operandos

Os AGs mantêm um conjunto de soluções potenciais que são evoluídas ao longo de várias iterações. Na terminologia dos AGs, este conjunto de soluções manipuladas é chamado de população. O processo de evolução de um AG trabalha sobre cada solução denominada cromossomo ou indivíduo[3]. As unidades que constituem um cromossomo são denominadas genes. Assim, considere as seguintes definições:

I. Gene

Definição 3.1 (Gene α): Considere um alfabeto A, i.e., um conjunto de símbolos que são usados na codificação de um problema. Um gene α assume como valor um dos símbolos presentes em A.

II. Cromossomo

Definição 3.2 (Cromossomo χ): Um cromossomo χ é uma cadeia (*string*) de genes. O número de genes de um cromossomo, denotado por $|\chi|$, define o seu comprimento. Considerando A o alfabeto que representa os genes, o conjunto A^n com $n \in N$ é constituído de todos os cromossomos χ com $|\chi| = n$. Ainda, $\chi(i)$ denota o i-ésimo gene do cromossomo χ, com $1 \leq i \leq n$.

O Algoritmo Genético proposto por Holland em 1975 usava a codificação binária para representar os seus cromossomos, i.e., o alfabeto A que representava os genes era formado por apenas dois símbolos, assim A = {0,1}. Atualmente, diversas formas de alfabeto são usadas, como números naturais, números reais e até representações híbridas.

[3] Neste trabalho os termos cromossomo e indivíduo são usados de maneira intercambiável.

III. População

Definição 3.3 (População π): Uma população π é um conjunto de cromossomos ou indivíduos. O número de cromossomos numa população é denotado por $|\pi|$. Geralmente, $|\pi|$ é constante durante toda a execução do algoritmo, i.e., o tamanho da população não se modifica durante a evolução.

3.2.2 A Função de Aptidão

Para que o processo de evolução possa ocorrer, a distinção entre indivíduo mais adaptado e indivíduo menos adaptado deve ser feita. Desta forma, para cada problema define-se uma função de aptidão que atribui a cada individuo da população uma quantidade numérica que dita a sua adaptação ao ambiente. Portanto, considere a seguinte definição:

Definição 3.4 (Função de Aptidão): Atua sobre um cromossomo χ. Considere A^n o conjunto de todos os cromossomos de tamanho *n* formados com um alfabeto A. A função Apt: $A^n \rightarrow R$ é chamada função de aptidão e atribui um valor Apt(χ) para cada $\chi \in A^n$.

3.2.3 Os Operadores

As seções anteriores apresentaram os principais operandos de um AG bem como a função de aptidão. Esta seção apresenta os operadores que manipulam os operandos com o objetivo de simular o processo de evolução natural.

I. Operador de Recombinação

Durante a evolução natural, os indivíduos de uma população cruzam entre si produzindo descendentes que herdam o material genético dos pais. Este cruzamento é denominado recombinação gênica. De forma geral, é através da recombinação que os genes se organizam em novas combinações nos indivíduos sobre os quais atua a seleção natural.

Do ponto de vista de um AG, a recombinação auxilia na criação de novos indivíduos. A idéia da recombinação é a de que as combinações das características que garantiram uma boa aptidão aos pais podem resultar em combinações ainda mais eficazes nos descendentes produzidos. Dessa forma, considere a seguinte definição:

Definição 3.5 (Recombinação): Atua sobre um conjunto de cromossomos. Considere s_1 e s_2, subconjuntos de A^n, i.e., $s_1 \subset A^n$ e $s_2 \subset A^n$. Ainda, assuma que a cardinalidade $|s_1| = i$ e $|s_2| = j$, com $i, j \in N$. Assim, o operador de Recombinação Rec é um mapeamento Rec: $(s_1) \rightarrow (s_2)$. De maneira geral, este operador recebe *i* cromossomos, também chamados de pais, e retorna *j* cromossomos também chamados de filhos.

Comumente, o operador de recombinação recebe dois pais para a geração de dois filhos. Diversos tipos de operadores de recombinação foram propostos, os mais comuns serão apresentados em seguida.

- Recombinação de um Ponto: considere um alfabeto A e dois pais P_1 e P_2 de tamanho *n* tais que $P_1 \in A^n$ e $P_2 \in A^n$. Dado um ponto de corte *k* escolhido aleatoriamente no intervalo [0, n - 1], os filhos F_1 e $F_2 \in A^n$ serão dados por:

  - $F_1(i) = \begin{cases} P_2(i) \text{ se } i \in [0, k[\\ P_1(i) \text{ se } i \in [k, n] \end{cases}$
  - $F_2(i) = \begin{cases} P_1(i) \text{ se } i \in [0, k[\\ P_2(i) \text{ se } i \in [k, n] \end{cases}$

  com $0 \leq i < n$.

  Exemplo 3.1: Considere um alfabeto A = {0, 1} e dois pais P_1 e P_2 de tamanho 10, i.e., $|P_1| = |P_2| = 10$. Dado o ponto de corte k = 3, a figura 3.1 exibe a recombinação de um ponto.

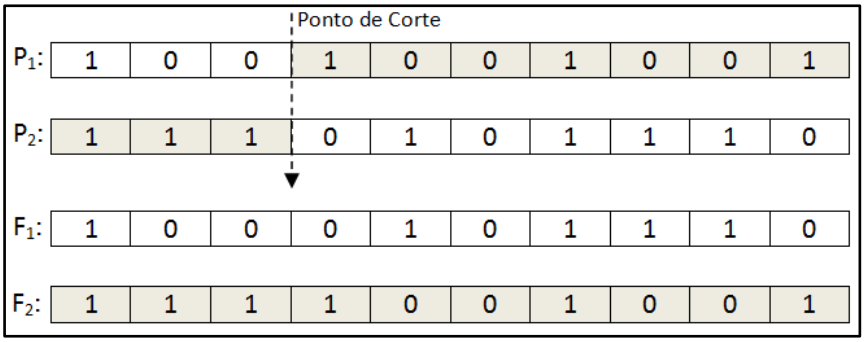

Figura 3.1: Recombinação de um Ponto

- Recombinação de *k* Pontos: considere um alfabeto A e dois pais P_1 e P_2 de tamanho *n* tais que $P_1 \in A^n$ e $P_2 \in A^n$. Dados *k* pontos de corte escolhidos aleatoriamente no intervalo [0, n-1], os filhos F_1 e $F_2 \in A^n$ serão dados por::
  - F_1: Comece copiando os símbolos de P_1 para F_1. Para cada ponto de corte encontrado, troque de pai e continue copiando os símbolos do "novo" pai para F_1.

- F_2: Comece copiando os símbolos de P_2 para F_2. Para cada ponto de corte encontrado, troque de pai continue copiando os símbolos do "novo" pai para F_2.

Exemplo 3.2: Considere um alfabeto A = {0, 1} e dois pais P_1 e P_2 tais que $|P_1| = |P_2| = 10$. Dados dois pontos de corte, $k_1 = 3$ e $k_2 = 7$, a figura 3.2 mostra a recombinação de dois pontos.

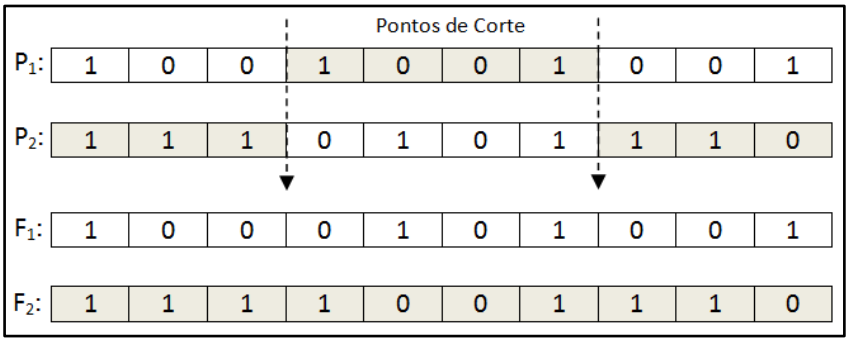

Figura 3.2: Recombinação de dois Pontos

- Recombinação Uniforme: este operador foi originalmente descrito em (SYSWERDA, 89). Considere um alfabeto A e dois pais P_1 e P_2 de tamanho n tais que $P_1 \in A^n$ e $P_2 \in A^n$. A recombinação uniforme usa uma máscara numérica para a produção dos descendentes. Esta máscara usa valores reais no intervalo [0,1] que determinam quais os genes serão copiados para os descendentes. Caso o valor da máscara na posição do i-ésimo gene ($1 \leq i \leq n$) seja menor que 0,5 então, $F_1(i) = P_1(i)$ e $F_2(i) = P_2(i)$, caso contrário $F_1(i) = P_2(i)$ e $P_2(i) = P_1(i)$. Na maioria das vezes, a máscara numérica é construída de maneira aleatória; assim cada gene possui a probabilidade de 0,5 de ser trocado.

Exemplo 3.3: Considere um alfabeto A = {0, 1} e dois pais P_1e P_2 tais que $|P_1| = |P_2| = 10$. Dada a seguinte máscara de bits: {0,6 - 0,1 - 0,8 - 0,2 - 0,3 - 0,4 - 0,9 - 0,1 - 0,2 - 0,3}, a figura 3.3 exibe a recombinação uniforme.

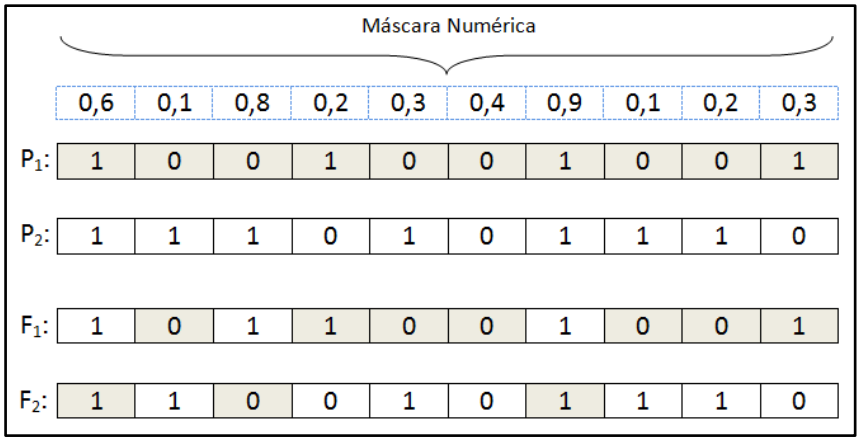

Figura 3.3: Recombinação Uniforme

III. Operador de Mutação

A mutação é a única maneira de se introduzir novos genes na população. Assim, se uma dada mutação favorece o indivíduo em sua adaptação ao ambiente, esse indivíduo possuirá uma maior chance de sobreviver e consequentemente de passar a nova característica adquirida por meio desta mutação para outros indivíduos da espécie. Caso a mutação seja desfavorável ao indivíduo, este indivíduo possuirá uma menor chance de se reproduzir e, portanto, de passar essa nova característica adquirida para outros indivíduos (AMABIS, MARTHO, 1994).

Do ponto de vista dos AGs, a mutação tem o objetivo de garantir que todos os possíveis cromossomos possam ser gerados. Já que o operador de recombinação apenas troca os genes já presentes na população como um todo, o operador de mutação é necessário para introduzir novos genes em um indivíduo e consequentemente garantir que todos os cromossomos possam ser gerados. Note que a introdução de novos genes na população não adiciona novos símbolos ao alfabeto A dos quais os genes assumem os seus valores. A mutação apenas modifica o valor de um gene fazendo com que o valor do gene modificado assuma outro símbolo de A. Dessa forma, considere a seguinte definição:

Definição 3.6 (Mutação de um ponto): Atua sobre um cromossomo. Considere A^n o conjunto de todos os cromossomos de tamanho *n* formados com um alfabeto A. O operador de Mutação de um ponto Mut é um mapeamento Mut: $A^n \rightarrow A^n$. Note que este operador recebe um cromossomo e retorna outro cromossomo.

Precisamente, o operador de mutação de um ponto atua em um gene de um determinado cromossomo. Assim, considerando um cromossomo χ com $|\chi| = n$ e $\chi(i)$ o

seu *i*-ésimo gene ($1 \leq i \leq n$), a mutação atua modificando o valor de $\chi(i)$ de forma que o seu valor seja diferente do seu valor atual. Para a codificação binária, i.e., A = {0,1}, a mutação tipicamente inverte o valor do gene, assim, se o seu valor era 0 antes da mutação, depois da aplicação deste operador, este gene possuirá o valor 1 e vice-versa.

Exemplo 3.4: Considere um alfabeto A = {0, 1} e um cromossomo C_1 formado a partir de A em que $|C_1| = 10$. Dado que o gene escolhido para mutação se localiza na posição 5, a figura 3.4 ilustra o processo de mutação. O cromossomo C_1' exibe a alteração do cromossomo C_1.

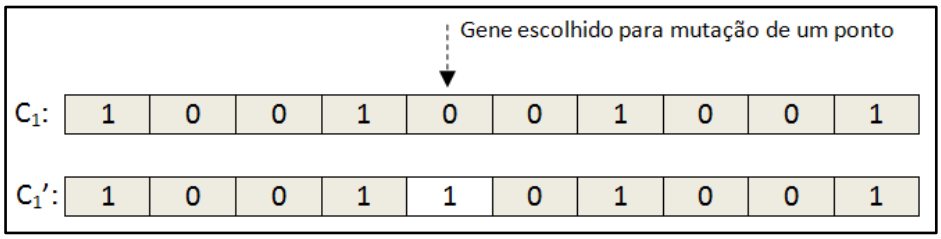

Figura 3.4: Mutação de um ponto

IV. O Operador de Seleção

(AMABIS, MARTO, 1994) fazem uma descrição precisa do fenômeno de seleção natural: "A seleção natural não é uma força misteriosa. Ela decorre simplesmente do fato de uma espécie produzir descendência quase sempre superior àquela que efetivamente tem condições de sobreviver. As restrições que o meio impõe à sobrevivência do individuo é que levam a seleção natural: quantidade de alimentos reduzida, competição, predadores, parasitas doenças, acidentes, etc. Estes são apenas alguns dos agentes seletivos que causam a morte do indivíduo antes da fase reprodutiva. Os mais aptos a sobreviver são aqueles que, graças à variabilidade gênica, herdaram combinações gênicas favoráveis à sobrevivência em determinado quadro de fatores ambientais."

Pela descrição, fica claro de que modo a seleção natural influencia diretamente o processo de adaptação dos indivíduos ao ambiente em que vivem.

Do ponto de vista dos AGs, o processo de seleção tem como objetivo selecionar os melhores indivíduos da população para que estes possam se reproduzir e sofrer mutações. Com isso, espera-se que novos indivíduos produzidos sejam mais aptos a solucionarem o problema abordado. Embora este processo de seleção seja estocástico, isso não implica que os AGs empregam uma busca sem direção. A probabilidade de um

indivíduo ser selecionado para recombinação e/ou mutação, está, de alguma forma, relacionada à sua aptidão. Assim, considere a seguinte definição:

Definição 3.7 (Seleção): Atua sobre uma população. Dada uma população π, o operador Seleção é um mapeamento Sel: $\pi \rightarrow \pi'$, em que π' é constituída pelos melhores indivíduos π escolhidos por algum critério.

Diversos critérios de seleção foram propostos, os mais comuns são apresentados em seguida.

- Seleção por Roleta: este é o método de seleção padrão e o mais comumente utilizado. Neste método, cada cromossomo tem a probabilidade de ser selecionado diretamente proporcional à sua aptidão. Assim, o seu efeito depende diretamente da abrangência dos valores de aptidão na população.

  Exemplo 3.5: Considere uma população de quatro cromossomos com as seguintes aptidões descritas na tabela 3.1. A probabilidade de seleção de cada um dos cromossomos, utilizando o método de seleção por roleta é também apresentada na tabela 3.1.

  Tabela 3.1: Seleção por Roleta

Cromossomo	Aptidão	Probabilidade de Seleção
χ_4	250	83,33%
χ_2	25	8,33%
χ_3	15	5%
χ_1	10	3,33%
Total	**300**	**100%**

  Como pode ser verificado, o cromossomo χ_4 possui uma probabilidade muito maior de ser selecionado do que os outros indivíduos. Essa diferença é indesejável em várias aplicações, já que pode facilmente levar a uma estagnação da busca, i.e., uma convergência prematura da população em direção a um cromossomo. Uma maneira de solucionar o problema é escalonar a aptidão antes da seleção (GOLDBERG, 1989).

- Seleção por *Ranking*: originalmente proposta por (BAKER, 1985), a probabilidade de um cromossomo ser selecionado é proporcional à sua posição ou *ranking* que ele ocupa em relação ao restante da população. Assim, esse tipo de seleção não se baseia diretamente na aptidão, já que esta é usada apenas para ordenar os indivíduos de uma população.

  Exemplo 3.6: Considere os mesmos quatro cromossomos apresentados na tabela 3.1. A tabela 3.2 mostra a probabilidade de seleção de cada cromossomo usando o

modelo de seleção em *ranking*. Neste exemplo, os indivíduos foram ordenados em função de sua aptidão sendo que a cada posição foi atribuída uma probabilidade de seleção.

Tabela 3.2: Seleção por Ranking

Cromossomo	Aptidão	Posição no Ranking	Probabilidade de Seleção
χ_4	250	1º	40%
χ_2	25	2º	30%
χ_3	15	3º	20%
χ_1	10	4º	10%
Total	**300**	**-**	**100%**

Em relação à tabela 3.1, é possível verificar que as probabilidades de seleção estão mais bem distribuídas entre os indivíduos. Neste caso, não é necessário escalonar a aptidão com o objetivo de se evitar uma convergência prematura, já que o *ranking* pode ser entendido como uma forma de escalonamento.

- Seleção por Torneio: este método de seleção foi proposto inicialmente por (BRINDLE, 1981) e escolhe $k \in \mathrm{N}$ indivíduos retornando o que possui o maior valor de aptidão. O número k fornece a pressão de seleção deste método, assim, quanto maior for este valor, maior será a pressão de seleção exercida na população. É importante notar que os indivíduos escolhidos para o torneio são selecionados de maneira aleatória e com reposição.

Esta seção apresentou os principais operadores dos AGs. A próxima seção dedica-se a apresentar como todos os seus mecanismos são usados para simular o processo de evolução natural.

3.2.4 O Ciclo Básico de um Algoritmo Genético

Até agora, pode-se resumir o que foi apresentado da seguinte forma: um AG manipula uma população que é constituída por um conjunto de soluções denominadas cromossomos e estes, por sua vez, são constituídos por unidades denominadas genes. Uma função de aptidão avalia a qualidade de adaptação dos indivíduos. A mutação insere novos valores para genes num cromossomo e a recombinação os mistura com os genes já existentes, originando, assim, os indivíduos geneticamente variados de uma população. A seleção natural favorece indivíduos portadores de determinados conjuntos gênicos adaptativos. Estes indivíduos tendem a sobreviver e a se reproduzir em maior escala que outros. Em função destes fatores, os indivíduos evoluem ao longo do tempo se tornando cada vez mais aptos a solucionar o problema abordado.

Essa seção descreve como todos estes componentes são usados para simular o processo de evolução natural. Dessa forma, considere em seguida o algoritmo 3.1 que apresenta o esquema geral dos Algoritmos Genéticos bem como a descrição de seus principais passos.

Algoritmo 3.1 - Algoritmo Genético: Esquema Geral

```
Entrada:
    população pop;
Saída:
    melhor indivíduo da população;
Começo-AG:
01- Inicialize pop de forma aleatória;
02- Avalie cada indivíduo de pop com a função de aptidão;
03- Enquanto (término == falso) faça:
04-   pop' := seleção de pais de pop;
05-   Aplique, de acordo com uma probabilidade, o operador de
      recombinação em alguns indivíduos de pop';
06-   Aplique, de acordo com uma probabilidade, o operador de
      mutação em alguns indivíduos de pop';
07-   Avalie cada indivíduo de pop' com a função de aptidão;
08-   pop := Substituição(pop, pop');
09- fim-enquanto;
10- Retorne o melhor indivíduo em pop;
Fim-AG.
```

Passo 1: Este passo é o responsável pela inicialização dos indivíduos da população. Tradicionalmente, os indivíduos da população são gerados de forma aleatória, tentando, dessa forma, abranger a maior parte possível do espaço de busca. Algumas vezes, os indivíduos podem ser "semeados" em áreas onde boas soluções são prováveis de serem encontradas. Para mais informações sobre métodos de inicialização, veja (CHIH-HSUN, JOU-NAN, 2000).

Passo 2: Este passo avalia cada indivíduo da população utilizando uma função de aptidão definida. De forma geral, esta função é uma medida da capacidade que o indivíduo possui de solucionar o problema.

Passo 3: Este passo define o critério de parada do algoritmo. Na terminologia dos AGs, a execução dos passos de 4 a 8 é denominada uma geração. O critério de parada mais

comum é definido como sendo um número específico de gerações. Dessa forma, o algoritmo é finalizado assim que ele é executado certa quantidade de vezes. Outro critério de parada está relacionado ao fato de a melhor solução não se modificar após certo número de gerações. Isso acontece quando o algoritmo encontra a solução ótima ou um ponto de ótimo local. Existe ainda outro critério de parada em que o algoritmo é finalizado quando a média da aptidão da população for muito próxima à aptidão do melhor indivíduo.

Passo 4: Este passo é de extrema importância para o AG, já que ele simula o processo de seleção natural proposto por Darwin. Neste passo, uma nova população, aqui chamada de *pop*', é construída selecionando-se os indivíduos mais adaptados de *pop*. Posteriormente, *pop*' será utilizada para geração de novos indivíduos.

Passo 5: Neste passo, alguns indivíduos de *pop*' são recombinados com o objetivo de se produzir novos indivíduos. Estes novos indivíduos são temporariamente armazenados em *pop*' para que depois sejam colocados em *pop*.

Passo 6: Este passo é responsável pela inserção de novo material genético em alguns indivíduos de *pop*'. Também neste caso, estes novos indivíduos são temporariamente armazenados em *pop*' para posteriormente serem inseridos em *pop*.

Passo 7: Este passo avalia a aptidão dos indivíduos que constituem *pop*'. Este passo é muito importante, já que em parte define quais indivíduos de *pop*' serão transferidos para *pop*.

Passo 8: Este passo substitui parte ou todos os indivíduos de *pop* pelos indivíduos de *pop*'. Aqui, várias possibilidades são possíveis, como por exemplo, substituir os piores indivíduos de *pop* pelos melhores indivíduos de *pop*', ou substituir os pais dos indivíduos de *pop*' pelos seus filhos (criados com operador de recombinação), ou ainda substituir indivíduos aleatórios de *pop* pelos indivíduos de *pop*'. Estes e outros métodos de substituição podem ser encontrados em (GOSH et al., 2000), que apresenta as vantagens e desvantagens de cada abordagem, já que não existe uma abordagem que seja considerada a melhor para os diversos problemas.

Passo 10: Este passo retorna a melhor solução encontrada após o critério de parada ser atingido.

Esta seção apresentou o esquema geral de todos os AGs bem como explicou os seus principais passos. A próxima seção discute os principais parâmetros presentes em um AG.

3.2.5 Principais Parâmetros de um Algoritmo Genético

Independentemente do problema abordado, os AGs apresentam uma série de parâmetros que determinam o seu comportamento e influenciam a qualidade da solução encontrada. Além disso, é importante analisar a influência destes parâmetros para que eles possam ser estabelecidos conforme as necessidades do problema e dos recursos disponíveis. Os principais parâmetros são descritos abaixo.

Tamanho da População: o tamanho da população afeta o desempenho global dos Algoritmos Genéticos. A busca pode ser ineficaz se a população for pequena, pois, deste modo, existe pouca diversidade genética para representar o espaço a ser explorado. Uma grande população geralmente fornece uma representação adequada do espaço de busca do problema, prevenindo, portanto, convergências prematuras para soluções locais. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais, ou alternativamente, que algoritmo trabalhe por um período de tempo muito maior, ou ainda que o mesmo seja paralelizado. Para mais informações sobre este parâmetro, consulte (GOLDBERG, 1989).

Taxa ou Probabilidade de Recombinação: este parâmetro controla a quantidade de indivíduos de uma população que irá sofrer recombinação**.** De maneira geral, quanto maior a taxa, mais rapidamente novos indivíduos serão produzidos. Se esta for muito alta, indivíduos com boas aptidões podem ser perdidos rapidamente. Com um valor baixo, o algoritmo pode tornar-se muito lento.

Taxa ou Probabilidade de Mutação: este parâmetro controla a quantidade de indivíduos de uma população que irá sofrer mutação. Argumenta-se que este parâmetro é o mais sensível dentre todos os outros (BÄCK, 1995). Uma baixa taxa de mutação previne que uma população fique estagnada em um valor, além de possibilitar que se alcance "qualquer" ponto do espaço de busca. Uma taxa muito alta a induz a uma busca essencialmente aleatória. Para mais informações sobre este parâmetro, consulte (OCHOA et al., 2000).

Taxa de Substituição: este parâmetro controla a porcentagem da população que será substituída durante a próxima geração. Com um valor alto, a maior parte da população será substituída, levando, possivelmente, a uma perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

3.3 Sistemas ILP baseados em Algoritmos Genéticos

Esta seção revisa brevemente alguns dos sistemas ILP que usam AGs para realizar a busca por teorias. Os sistemas revisados são REGAL, SIA01, DOGMA, G-NET, ECL, e QG/GA, sendo que será dada maior importância a este último, uma vez que ele pode ser considerado o mais importante dentre os destacados, já que, além de ser o mais novo, foi aplicado a uma ampla variedade de problemas ILP (o que não ocorreu com os outros sistemas). Além disso, como já afirmado, tal sistema influenciou diretamente o desenvolvimento do RAED-ILP.

3.3.1 QG/GA

De forma geral, o QG/GA (Quick Generalization /Genetic Algorithm) (MUGGLETON, TAMADDONI-NEZHAD, 2006), (MUGGLETON, TAMADDONI-NEZHAD, 2006) pode ser dividido em dois componentes principais, a saber: um operador, que pode ser entendido como um algoritmo, chamado de Quick Generalization (QG), e um método de busca, que nada mais é que um AG tradicional. Ambos os componentes são plugados no sistema Progol (revisado no capítulo 1) com objetivo de gerar melhores soluções comparadas às soluções encontradas quando tal sistema é usado sem o QG e com outro mecanismo de busca, como o A*, por exemplo.

O trabalho foi motivado pelo fato de que a maior parte dos sistemas ILP avalia uma grande quantidade de cláusulas inconsistentes (que provam exemplos negativos) e que, tipicamente, uma cláusula para fazer parte de uma teoria deve ser consistente (provar apenas exemplos positivos). Para justificar tal argumento, os autores apresentaram uma tabela consistindo de bases de dados em que o Progol foi aplicado e mostraram que apenas 3,14% das cláusulas avaliadas são consistentes. Dessa forma, a grande maioria das cláusulas avaliadas é inconsistente e não será, na maioria das vezes, adicionada à teoria aprendida pelo sistema. Notou-se, portanto, que o sistema perdia muito tempo avaliando cláusulas que seriam descartadas ao final de sua execução.

Para resolver o problema exposto, o trabalho propõe o QG que utiliza o algoritmo Reduce (MUGGLETON, FENG, 1990), com objetivo de gerar cláusulas consistentes que se localizam na borda do látice limitado superiormente pela cláusula vazia e inferiormente pela *bottom clause*. O operador QG tem a finalidade de construir cláusulas consistentes sem necessitar de explorar detalhadamente tal látice. De maneira geral, o QG recebe uma *bottom clause*, permuta seus literais aleatoriamente para

construir uma *bottom clause head-connected*, e então aplica o algoritmo Reduce nesta cláusula permutada. Como saída, o QG retorna uma cláusula consistente (e reduzida) construída sobre a *bottom clause* de entrada. Antes de apresentar o QG (veja algoritmo 3.4) e o Reduce (veja algoritmo 3.5), é importante revisar alguns conceitos usados por estes procedimentos (MUGGLETON, TAMADDONI-NEZHAD, 2006). Assim, considere as seguintes definições:

Definição 3.13 (*Head-connectness*): uma cláusula definida $h \leftarrow b_1, \ldots, b_n$ é dita *head-connected* sse cada átomo do corpo b_i contém ao menos uma variável que pode ser encontrada em h ou em um átomo do corpo b_j, onde $1 \leq j < i$.

Definição 3.14 (*Minimal support set*): considere $h \leftarrow B$ uma cláusula definida e B um conjunto de átomos. $S \subseteq B$ *é* um *minimal support set* para b de B sse $h \leftarrow S$, b é *head-connected* e não existe um conjunto $S' \subset S$ para o qual, $h \leftarrow S'$, b seja *head-connected*.

Definição 3.15 (*Profile* e *cutoff atom*): Considere $C = h \leftarrow b_1, \ldots, b_n$ uma cláusula definida, B o conhecimento preliminar, e E- o conjunto de exemplos negativos. $E_i \subseteq E$- É o i-ésimo *profile* negativo de C, onde $E_i = \{e: \exists\theta, e = h\theta, B \models (b_1, \ldots, b_i)\theta\}$. b_i é o *cutoff* atom sse i é o menor valor tal que $E_i = \varnothing$.

Algoritmo 3.4 – Procedimento Principal do QG

```
Entrada:
    (BC): uma bottom clause construída sobre um exemplo positivo;
Variáveis Locais:
    (perBC): uma permutação aleatória head-connected de BC;
Saída:
    (redBC): uma bottom clause reduzida pela aplicação do
             algoritmo Reduce sobre perBC;
Começo-QG:
01- perBC := Permute(BC);
02- redBC := Reduce(perBC);
03- Retorne redBC;
Fim-QG.
```

O QG recebe uma *bottom clause BC* que foi construída sobre um exemplo positivo selecionado aleatoriamente. No passo 1, o QG permuta aleatoriamente os literais do corpo da *BC* para gerar uma permutação aleatória e *head-connected* dela. O passo 2 do QG executa o procedimento determinístico Reduce (veja algoritmo 3.5) na

cláusula permutada. Finalmente, o passo 3 retorna a *bottom clause* reduzida que foi construída sobre *BC*.

Algoritmo 3.5 – Procedimento Principal do Reduce

Entrada:
Uma *bottom clause* permutada **perBC**, $h \leftarrow b_1,...,b_n$;
Variáveis Locais:
(**redBC**): uma cláusula sendo reduzida;
Saída:
(**redBC**): uma cláusula reduzida;

Começo-Reduce:
01- *redBC* := *perBC;*
02- Enquanto existir um cutoff atom b_i no corpo de *redBC* faça:
03- Para b_i encontre o *minimal support set* $S_i = \{b_1', ..., b_m'\} \subseteq \{b_1, ..., b_{i-1}\}$ tal que $h \leftarrow Si,bi$ seja *head-connected*;
05- *redBC* := $h \leftarrow S_i, b_i, T_i$, onde T_i é $b_1, ..., b_{i-1}$ com S_i removido;
06- fim-enquanto;
07- Retorne redBC;
Fim-Reduce.

O algoritmo Reduce recebe uma permutação aleatória da *bottom clause* e trabalha encontrando, a cada iteração, subconjuntos consistentes cada vez menores da *bottom clause* de entrada. A saída do algoritmo Reduce é uma cláusula reduzida e consistente.

Pela aplicação do QG, o sistema gera *n* cláusulas consistentes na borda do látice e estas *n* cláusulas são evoluídas (combinadas) por meio de um AG padrão. É importante perceber que as cláusulas geradas e suas possíveis variações são baseadas na *bottom clause* e que o AG utiliza a codificação binária para representar tais cláusulas. Especificamente, o AG manipula *strings* binárias que possuem o tamanho da *bottom clause* gerada, e realiza o seguinte mapeamento: caso o valor na posição *i* da string considerada seja 1, significa que o literal da *i*-ésima posição da *bottom clause* será usado quando a string binária for mapeada para a cláusula que ela representa. Esta maneira de codificar a *bottom clause* foi apresentada em (ALPHONSE, ROUVEIROL, 2000) e mais bem formalizada em (ALPHONSE, ROUVEIROL, 2006). A título de ilustração, considere o exemplo 3.13 a seguir.

Exemplo 3.13: Assuma que o sistema Progol gerou a seguinte bottom clause *BC*: q(X,Y) :- r(Z,Y), t(Z,X), w(X,Y). Como *BC* possui 4 literais, o AG irá trabalhar com uma *string* binária de 4 posições, em que, cada posição determina o uso (valor 1) ou não (valor 0) de um literal na posição relativa à *BC*. Dessa forma, a tabela 3.3 apresenta a *BC* juntamente com alguns indivíduos da população (S_1-*bin* e S_2-*bin*), seguidos, respectivamente, pelas cláusulas que eles representam (C_1 e C_2).

Tabela 3.3: *Strings* binárias e seus mapeamentos em cláusulas usando a *BC*

BC	**q(X,Y) :-**	**r(Z,Y)**	**t(Z,X)**	**w(X,Y)**
S_1-bin	1	0	0	1
C_1	q(X,Y) :-			w(X,Y)
S_2-bin	1	1	1	0
C_2	q(X,Y) :-	r(Z,Y)	t(Z,X)	

Verifica-se pela tabela 3.3 que a *string* (1001) dará origem a cláusula q(X,Y) :- w(X,Y) e a *string* (1110) dará origem a cláusula q(X,Y) :- r(Z,Y), t(Z,X).

O AG[4] usado por este trabalho manipula tais *strings* utilizando os operadores de mutação (de um ponto) e de recombinação (de um ponto), sendo que sua população inicial não é gerada aleatoriamente, mas sim por aplicações sucessivas do operador QG a uma *BC* específica. Sob este aspecto, é importante notar que a população inicial do AG somente apresentará literais presentes na *bottom clause* reduzida, contudo, a busca do AG é realizada na *bottom clause* sem a redução (antes de ser executado o algoritmo QG), i.e., a população inicial é gerada levando em consideração apenas os literais da *bottom clause* reduzida, enquanto que durante a exploração, todos os literais da *bottom clause* são considerados. Outro item importante a ser notado é que, quando o QG/GA é usado, a declaração de modos e *determinations* (que restringem as possíveis cláusulas geradas pelo Progol) se aplicam apenas a geração da *BC*, uma vez que devido à natureza estocástica da mutação e da recombinação de indivíduos, tais restrições não são mais respeitadas. Em outras palavras, o AG possui um espaço de busca mais amplo quando comparado ao A*, uma vez que este último método respeita as restrições impostas pelos modos e *determinations* quando gera suas cláusulas e o AG não pode fazê-lo, devido à natureza estocástica dos seus mecanismos de exploração. Finalmente, deve-se notar que o ciclo básico do Progol não é alterado, ou seja, o sistema continua aprendendo uma

[4] Os autores utilizam uma implementação do SGA (GOLDBERG, 1989) em tal trabalho, contudo, não deixam claro qual o operador de seleção foi adotado.

cláusula de cada vez como mostrado no capítulo 2, contudo, a busca por tais cláusulas em vez de ser realizada pelo A*, é realizada por um AG padrão.

A avaliação do QG/GA foi realizada como segue. Primeiramente, o sistema Progol usando o QG (chamando de Progol-QG) foi comparado ao Progol usando sua busca padrão que é realizada pelo A* (sistema chamado de Progol-A*). Os resultados mostraram que o Progol-QG obtém resultados de forma mais eficiente e com acurácia similar em relação aos resultados do Progol-A*. Posteriormente, duas outras variações do Progol usando o AG em vez do A* foram propostas. Na primeira versão, chamada de Progol-GA, a população inicial do AG foi gerada de maneira aleatória, enquanto que na segunda versão, chamada de Progol-QG/GA, a população inicial do AG foi semeada usando as cláusulas reduzidas que são geradas pelo QG. Com objetivo de avaliar ambas as versões, dois grupos de experimentos foram realizados. Primeiramente, o Progol-GA foi comparado ao Progol-A* e ao Progol-QG. Posteriormente, o Progol-QG/GA foi comparado ao Progol-A*, ao Progol-QG, e ao Progol-GA. O primeiro grupo de experimentos foi realizado usando bases de dados do mundo real, enquanto que o segundo grupo usou tanto bases de dados do mundo real quanto problemas de Transição de Fase (Botta et al., 2003). No primeiro lote de experimentos, o Progol-QG obteve acurácia similar ou mais elevada (não estatisticamente significante) avaliando menos cláusulas em relação ao Progol-A* e ao Progol-GA. No segundo lote de experimentos, o Progol-QG/GA claramente supera as outras variações dos sistemas à medida que o tamanho do conceito a ser aprendido se torna maior.

Baseando nos experimentos realizados, as seguintes conclusões são apresentadas pelos autores do QG/GA:

- Os resultados do QG/GA quando comparados ao Progol-A* são semelhantes, contudo, o QG/GA usa menos tempo (a redução de tempo é ainda mais substancial quando a densidade de cláusulas consistentes no látice é baixa);
- QG/GA necessita avaliar menos cláusulas para obter resultados semelhantes ao A*;
- Quando o QG/GA é usado para lidar com problemas de Transição de Fase (BOTTA et al., 2003), tal algoritmo claramente supera (tanto em acurácia quanto em tempo) o método de busca A*.

Embora ainda não se tenha apresentado formalmente o RAED-ILP, já é possível notar, baseando-se apenas na descrição de tal sistema, algumas diferenças fundamentais entre ele e o QG/GA. São elas:

- O RAED-ILP busca por teorias cujos literais são subconjuntos de cláusulas reduzidas (cláusulas nas quais o algoritmo Reduce foi aplicado). No Progol-QG/GA, a busca por cláusulas é realizada nas *bottom clauses* não reduzidas, i.e., o algoritmo Reduce é aplicado apenas com objetivo de semear a população inicial do AG, mas sua busca considera o espaço de busca definido pela *bottom clause* sem redução.
- O Progol-QG/GA e todas as suas outras variantes criadas usando somente o QG ou o GA usam o algoritmo de cobertura para construir suas teorias. Assim, todas as variantes do Progol buscam uma cláusula por vez, enquanto que o RAED-ILP busca por teorias (um conjunto de cláusulas).
- O Progol-QG/GA usa um AG convencional para realizar sua busca, enquanto o RAED-ILP usa um AED.

3.3.2 Outros sistemas ILP baseados em Algoritmos Genéticos

Esta seção apresenta um resumo dos sistemas REGAL, SIA01, DOGMA, G-NET, ECL. Não se entrará em grandes detalhes uma vez que tais sistemas não têm sido usados atualmente. Esta seção se justifica uma vez que é importante ter em mente algumas abordagens já utilizadas e que talvez possam ser readaptadas a algum sistema ILP-genético mais atual.

I. REGAL

O REGAL (*Relational Genetic Algorithm Learner*) (GIORDANA, NERI, 1996) explora o paralelismo implícito dos AGs. De fato, o sistema consiste de uma rede de nós genéticos totalmente interconectados que trocam os indivíduos a cada geração. Cada nó genético implementa um AG. Um nó supervisor é usado para coordenar estas sub-populações. O sistema adota a abordagem Michigan[5] de evolução de hipóteses e uma linguagem restritiva de cláusulas, o que limita o espaço de busca, e quatro operadores de

[5] As terminologias Michigan e Pittsburgh são muito utilizadas quando se aplica um AG ao problema de Aprendizado Indutivo de Hipóteses. Na abordagem Michigan, um indivíduo codifica uma única cláusula, enquanto na abordagem Pittsburgh, um indivíduo codifica toda uma teoria, ou seja, um conjunto de cláusulas.

recombinação: o uniforme, o de dois pontos, um de especialização de cláusulas e um de generalização das mesmas.

II. G-NET

O G-NET (*Genetic Network*) (ANGLANO et al., 1998) é um descendente do REGAL. Como o seu predecessor, o G-Net é também um sistema distribuído com uma coleção de nós genéticos e um supervisor. Basicamente, o G-Net se difere do REGAL por utilizar dois tipos de função de aptidão, bem como três operadores de mutação. As funções de aptidão atuam em um nível global (relativo a todos os nós do sistema) e em um nível local (relativo a um nó específico). Considerando os operadores de mutação, tem-se um para generalizar um indivíduo, outro para especializá-lo e um terceiro para gerar novas cláusulas. O operador de recombinação deste sistema é uma combinação dos operadores de recombinação de dois pontos e do uniforme, e tem o objetivo de especializar ou generalizar um indivíduo.

III. DOGMA

O DOGMA (*Domain Oriented Genetic Machine*) (HEKANAHO, 1996) usa dois níveis distintos para evolução de teorias. No nível mais baixo, o sistema usa a abordagem Michigan, enquanto que no nível mais alto, a abordagem Pittsburgh é usada. No nível mais baixo, o sistema utiliza os operadores de recombinação e mutação, enquanto que no nível mais alto, os cromossomos são combinados em famílias genéticas, sendo que operadores especiais foram desenvolvidos para unir ou quebrar famílias. A forma de representação dos indivíduos usada pelo DOGMA é a mesma utilizada pelo REGAL. Este sistema adota os operadores de recombinação do REGAL e o seu operador de mutação altera uma posição aleatória em um indivíduo pela modificação do bit escolhido.

IV.SIA01

O SIA01 (*Supervised Inductive Algorithm version 01*) (AUGIER et al., 1995) usa o algoritmo de cobertura explicado no capítulo 1 para evoluir suas teorias. O sistema adota a abordagem *bottom-up* de cobertura. Assim, ele começa com um exemplo positivo não coberto, cria uma cláusula que cobre este exemplo, e então busca a melhor generalização desta cláusula por meio de um AG. A população deste sistema pode crescer até um número máximo que foi definido pelo usuário. SIA01 usa uma

representação alto nível dos indivíduos, não introduzindo, portanto, nenhum *bias* relativo à linguagem.

V. ECL

No ECL (*Evolutionary Concept Learner*) (DIVINA, MARCHIORI, 2002) os indivíduos são representados em alto nível tal como no sistema SIA01. A maior diferença deste sistema para os outros se faz pela introdução de um módulo de otimização local estocástico assim que o indivíduo é selecionado e sofre mutação. Este sistema não utiliza operador de recombinação devido à dificuldade de elaboração do mesmo para representações em alto nível. O sistema apresenta quatro operadores de mutação, dois para generalização e dois para especialização.

3.4 Os Algoritmos Estimadores de Distribuição

Os Algoritmos Estimadores de Distribuição (AEDs) (MÜHLENBEIN, 1996) ou os Algoritmos Genéticos Construtores de Modelos Probabilísticos (AGCMPs) (PELIKAN, GOLDBERG, LOBO, 1999), podem ser considerados uma variação dos AGs tradicionais e, desta forma, guardam certas semelhanças e apresentam algumas diferenças em relação à técnica tradicional. Entre suas semelhanças, está o fato de que os AEDs também mantêm uma população de soluções para explorar o espaço de busca juntamente com algum mecanismo de seleção responsável por eleger, a cada geração, os melhores indivíduos da população. Sua diferença fundamental vem do fato de que estes algoritmos, ao contrário dos AGs tradicionais, geralmente suprimem os operadores de recombinação e de mutação que são usados para criar variabilidade genética na população. Para realizar tal efeito, os AEDs usam modelos probabilísticos que são aprendidos a cada nova geração do algoritmo. Tais modelos, uma vez aprendidos, são então amostrados (*sampled*) gerando uma nova população completa ou apenas uma parte desta.

De forma geral, pode-se dizer que os AEDs são projetados para capturar as interações entre os genes que representam a estrutura interna das soluções dos problemas e, com isso, estimam diretamente a distribuição de boas soluções em vez de usar operadores genéticos. Tecnicamente, os modelos probabilísticos presentes nos AEDS são responsáveis por aprender a estrutura dos blocos construtores presentes no problema que, uma vez identificados, serão usados para construção das soluções. Para

ilustrar tal fato, considere os dois exemplos a seguir (PELIKAN, 2005) que ilustram a estrutura dos blocos construtores a serem aprendidos.

Exemplo 3.14: Considere o problema de otimização a seguir denominado de *OneMax* (OM). Neste problema, o objetivo é encontrar uma *string* binária *X* de tamanho *n* que maximize a função definida como segue:

$$F_{OneMax}(X) = \sum_{i=1}^{n} x_i$$

em que x_i é o valor (0 ou 1) da i-ésima posição da string *X*.

Tipicamente, o comprimento da *string n* é um dado do problema, assim, uma instancia do *OneMax*, chamada de *OneMax-100*, seria encontrar uma *string* binária de 100 posições que maximizasse a função acima. Pode-se notar que a *string* que maximiza a função *OneMax* possui em todas as suas posições o valor 1.

Obviamente este problema é muito simples, uma vez que cada posição da *string* pode ser considerada independentemente pelo algoritmo, ou seja, à medida que a quantidade de valores 1's na string aumenta, o valor da função também aumenta, culminando em seu valor máximo quando todas as posições são iguais a 1. Assim, pode-se ver claramente o conceito de bloco construtor neste problema. Cada posição da *string* pode ser considerada um bloco construtor, uma vez que mantidas fixas *n – 1* posições da *string*, com exceção de uma posição *i* qualquer, o valor da função *OneMax* aumentará sempre que o valor desta posição *i* for 1.

Exemplo 3.15: Considere o problema de otimização a seguir denominado *Additively Separable Traps* de Ordem 5 (5-AST). Neste problema, o objetivo é encontrar uma *string* binária *X* de tamanho *n* que maximize a função definida como segue:

$$F_{5-AST}(X) = \sum_{i=1}^{\frac{n}{5}} trap_5(x_{bi,1} + x_{bi,2} + \cdots + x_{bi,5})$$

onde:

$$trap_5(u) = \begin{cases} 5, se\ u = 5 \\ 4 - u, caso\ contrário \end{cases}$$

Este problema é definido sobre uma *string X* de tamanho *n* (*n* deve ser múltiplo de 5) que é então divida em *n*/5 blocos de *strings* de tamanho iguais a 5. Assim, $X_{bi,j}$ é o valor (0 ou 1) da posição *j* ($1 \leq j \leq 5$) do bloco *i* da *string*.

Como ocorreu no caso do OM, a *string* que maximiza tal função possui todas as suas posições com valor igual a 1. Em contrapartida, este exemplo é bem mais complicado que o OM, uma vez que considerar as posições da *string* de forma independente não acarretará no ótimo global. Isto se deve a função $trap_5(u)$ que atribui um valor numérico a um bloco e não a uma posição isolada da *string*. Por esta função, caso todo o bloco contenha o valor 1, o valor de tal bloco é igual a 5. Caso o bloco não seja todo 1, o valor da função cresce à medida que o número de 1's no bloco cai. Dessa forma, caso cada posição da *string* seja considerada independentemente do bloco em que faz parte, o resultado final será, na maioria das vezes, um máximo local. É importante perceber que as partições dos blocos não são fornecidas ao algoritmo que irá resolver o problema, cabendo a ele, se for capacitado a isso, identificá-las. O conceito de bloco construtor pode ser visualizado também neste exemplo, sendo que cada bloco construtor é de fato um bloco de cinco posições do problema, ou seja, uma vez que mantidos fixos $(n/5) - 1$ blocos, com exceção de um bloco *i* qualquer, o valor da função AST aumentará sempre que o todas as posições deste bloco *i* forem iguais a 1.

Caso um algoritmo possa identificar a estrutura do problema a ser resolvido, ou usando a terminologia dos AGs, identificar os blocos construtores da solução, pode-se dizer que tal algoritmo terá mais sucesso em resolver o problema abordado (PELIKAN, 2005). Portanto, o objetivo dos AEDs é construir um modelo probabilístico que consiga sintetizar a estrutura do problema para que, desta forma, tenha mais chances de sucesso em resolver o problema proposto. De forma geral, os AEDs se diferenciam pelo tipo de modelo probabilístico usado para aprender a estrutura do problema, sendo que alguns usam modelos bastante simples, assumindo independência entre as variáveis do problema, e outros conseguem capturar relações complexas. Independentemente do modelo probabilístico usado, o ciclo básico dos AEDs pode ser descrito como mostra o algoritmo 3.6.

Algoritmo 3.6 – Esquema Geral dos AEDS

```
Entrada:
    (pop): população de indivíduos;
Variáveis Locais:
    (pop'): seleção dos melhores indivíduos de pop;
    (ind): conjunto de indivíduos gerados pela amostragem do
           modelo;
Saída:
    melhor indivíduo da população;
Começo-AEDs:
01- Inicialize pop de forma aleatória;
02- Avalie cada indivíduo de pop com a função de aptidão;
03- Enquanto (término == falso) faça:
04-   pop' := Seleção(pop);
05-   Construa um modelo probabilístico MP a partir de pop';
06-   Amostre MP para gerar novos indivíduos ind;
07-   Avalie cada indivíduo de ind com a função de aptidão;
08-   pop := Substituição(pop, ind);
09- fim-enquanto;
10- Retorne o melhor indivíduo em pop;
Fim-AEDs.
```

O passo 4, como ocorre no AG tradicional, seleciona os melhores indivíduos da população *pop*. O passo 5 constrói um modelo probabilístico usando os melhores indivíduos selecionados no passo 4. O passo 6 usa o modelo probabilístico construído para gerar novos indivíduos, em vez de usar os operadores de recombinação e mutação. Note que este conjunto de indivíduos foi representado por *ind*, pois não necessariamente precisa-se gerar uma quantidade de indivíduos igual ao tamanho da população usada pelo AED. Em semelhança ao AG tradicional, o passo 7 avalia os novos indivíduos gerados, e o passo 8 realiza uma substituição de alguns dos indivíduos de *pop* pelos indivíduos de *ind*. Todo o ciclo é repetido até que algum critério de parada seja satisfeito.

De forma geral, pode-se dizer que a busca nos AEDs é um procedimento iterativo para construção de um modelo probabilístico que gera um ótimo global. Inicialmente, o modelo probabilístico codifica uma distribuição uniforme sobre todas as soluções. Cada iteração atualiza o modelo para gerar soluções melhores. Um AED bem

sucedido deve acabar com um modelo probabilístico que gera (utopicamente) um ótimo global (PELIKAN, 2005).

Como afirmado anteriormente, os AEDs podem ser categorizados de acordo com a complexidade do modelo probabilístico que eles são capazes de aprender. A próxima seção descreve as principais classes destes algoritmos.

3.4.1 Principais Classes dos AEDs

Antes de apresentar as principais classes dos AEDs que, como afirmado, estão relacionadas à complexidade do modelo probabilístico a ser aprendido, é importante realizar uma breve revisão sobre redes Bayesianas, uma vez que estas podem ser usadas para representar grande parte destes modelos. Dessa forma, primeiro apresenta-se uma breve revisão sobre tais redes para que posteriormente sejam apresentadas as principais classes de AEDs.

3.4.1.1 Uma Breve Revisão sobre Redes Bayesianas

Esta seção revisa brevemente os tópicos relativos às redes Bayesianas que serão necessários ao entendimento dos AEDs. Dessa forma, primeiro define-se o conceito de redes Bayesianas, posteriormente revisa-se o *noisy-OR*, e finalmente apresenta-se a maneira pela qual uma rede Bayesiana pode ser amostrada para gerar uma amostra (ou observação).

I. Definição

Uma rede Bayesiana (RB) pode ser definida por duas componentes (PELIKAN, 2005):

1. **Estrutura:** a estrutura é representada por um grafo acíclico dirigido com os nós representando as variáveis do problema e os arcos representando probabilidades condicionais.
2. **Parâmetros:** os parâmetros são representados por um conjunto de tabelas de probabilidades condicionais (TPCs) que especificam a probabilidade para cada variável dada qualquer instanciação das variáveis das quais ela dependa diretamente.

Matematicamente, a rede Bayesiana codifica uma distribuição de probabilidade conjunta dada por:

$$p(X) = \prod_{i=1}^{n} p(X_i | \Pi_i) \tag{3.14}$$

em que $X = (X_1, X_2, ..., X_n)$ é um vetor que representa todas as variáveis do problema, Π_i é o conjunto dos pais de X_i (conjunto de nós para os quais existe um arco até X_i), e $p(X_i | \Pi_i)$ é a probabilidade condicional de X_i dados os seus pais Π_i.

Um arco direcionado relaciona duas variáveis, uma chamada de pai (a variável da qual o arco sai) e a outra chamada de filha (variável a qual o arco chega). As probabilidades das TPCs especificam a probabilidade condicional de uma variável dado um conjunto de valores para seus pais. Além de codificar dependências, as redes Bayesianas codificam um conjunto de independências. De forma geral, uma rede Bayesiana declara que cada variável é independente de qualquer um dos seus ancestrais não imediatos dados os valores de seus ancestrais imediatos (pais), i.e., uma variável é independente dos valores das demais variáveis da rede, dados os valores de seus pais. Para melhor ilustrar os conceitos principais das redes Bayesianas, considere o exemplo a seguir.

Exemplo 3.16: Considere a estrutura de uma rede Bayesiana mostrada na figura 3.5. Tal rede codifica uma série de dependências condicionais. Por exemplo, a rede dita que a velocidade de um carro depende do tempo estar chuvoso e/ou de ser uma área que contenha um radar de velocidade. Da mesma forma, é mais provável a estrada estar úmida se estiver chovendo. Em contrapartida, a rede codifica uma série de independências. Por exemplo, uma independência dita que a probabilidade de ocorrer um acidente dado que a estrada está molhada e uma determinada velocidade, é independente de estar chovendo.

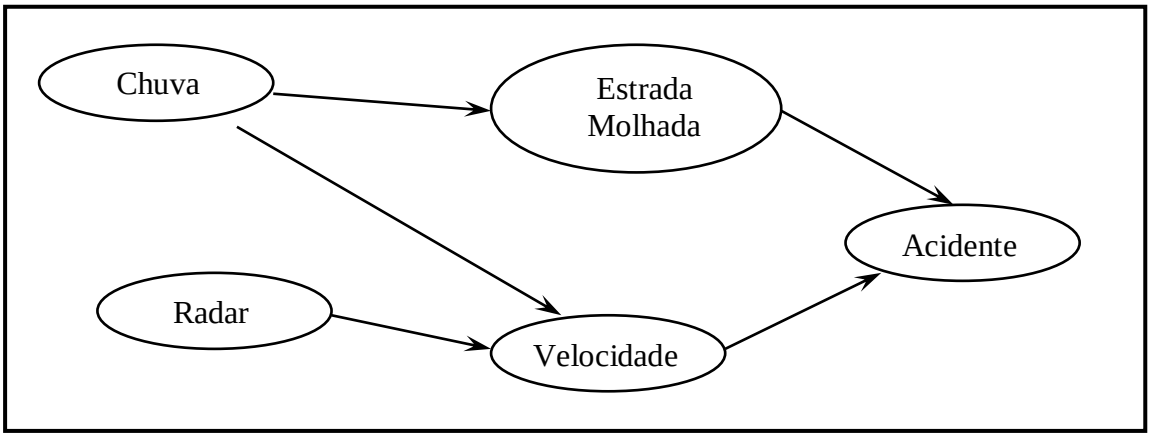

Figura 3.5: Rede Bayesiana Simples (PELIKAN, 2005)

Embora as variáveis representadas na rede Bayesiana possam, de forma geral, assumir vários valores (inclusive infinitos valores contínuos em um intervalo), neste trabalho considera-se o caso em que tais variáveis podem assumir apenas valores

binários (T para verdadeiro e F para falso). Dessa forma, o valor T indica a ocorrência do evento representado pela variável enquanto que F indica a não ocorrência deste evento. Assim, para cada variável X_i presente na rede Bayesiana, deve-se especificar uma TPC contendo $2^{|\Pi_i|}$, em que $|\Pi_i|$ representa a quantidade de pais da variável X_i. Para o exemplo 3.16, uma possível TPC para a variável acidente poderia ser dada pela tabela 3.4. Note que para cada variável, existe o valor T ou F indicando a ocorrência ou não do evento representado por ela.

Tabela 3.4: TPC para a variável Acidente

Estrada Molhada	**Velocidade**	**P(Acidente = T| Estrada Molhada, Velocidade)**
Sim (T)	Alta (T)	0.18
Sim (T)	Baixa (F)	0.04
Não (F)	Alta (T)	0.06
Não (F)	Baixa (F)	0.01

De forma geral, as TPCs especificam a probabilidade de ocorrência de um evento representado por uma variável X_i dados todos os possíveis valores dos pais de X_i. Uma vez especificada esta tabela, pode-se facilmente calcular a probabilidade da não ocorrência de um evento representado por X_i, dados os valores de seus pais. Dessa forma, como *P(Acidente = T| Estrada Molhada = T, Velocidade = T) = 0.18*, tem-se que *P(Acidente = F| Estrada Molhada = T, Velocidade = T) = 1 - P(Acidente = T| Estrada Molhada = T, Velocidade = T)*, o que significa dizer que a probabilidade de não ocorrer um acidente, dado que a estrada está molhada e que a velocidade é alta é igual a 0,72 (1 - 0,18). Embora este tipo de cálculo reduza a quantidade de linhas (probabilidades) que devem ser especificadas em uma TPC, percebe-se que o crescimento do número de linhas é exponencial com o número de pais de uma variável. Isto porque, como visto, devem-se especificar $2^{|\Pi_i|}$ probabilidades para uma dada variável X_i. Obviamente, especificar $2^{|\Pi_i|}$ probabilidades é impraticável caso $|\Pi_i|$ seja muito grande. Uma alternativa é assumir a hipótese do *noisy-OR* (PEARL, 1988).

II. Redução das TPCs pelo uso do *noisy-OR*

Basicamente, o *noisy-OR* dita que a ocorrência dos pais influencia independentemente a não ocorrência da variável filha. O *noisy-OR* reduz o número de probabilidades a ser especificado para uma quantidade linear com o número de pais. Para exemplificar seu uso, considere o simples exemplo a seguir.

Exemplo 3.17: Considere a seguinte rede Bayesiana que representa a influência de algumas doenças (Gripe, Malária e Resfriado) em um sintoma (Febre). Da mesma forma que no exemplo anterior, aqui se assume que todas as variáveis podem assumir apenas valores binários (verdadeiro ou falso).

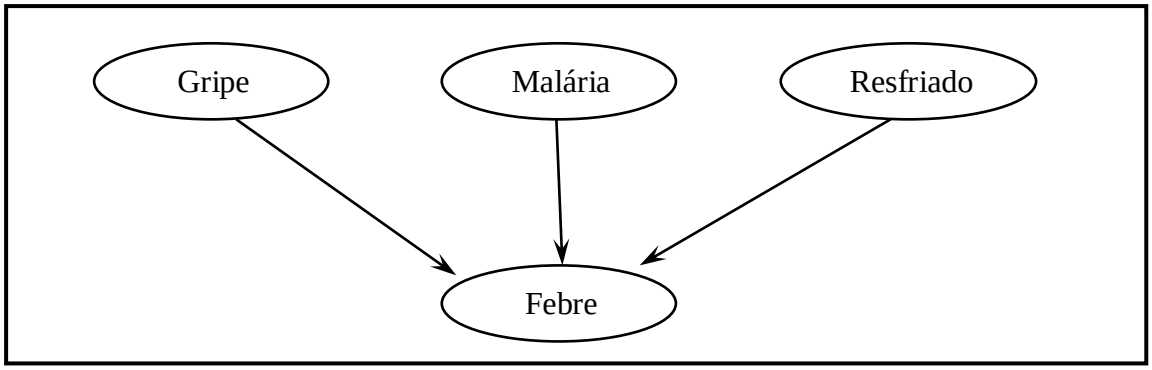

Figura 3.6: Rede Bayesiana para Doenças

Para os nós que não possuem pais, não existem probabilidades condicionais associadas a eles, apenas probabilidades marginais. Assim, assuma que, *P(Gripe = T) = 0,3*, *P(Malária = T) = 0,1* e, *P(Resfriado = T) = 0,6*. Para o nó Febre, que possui 3 pais, caso não se utilizasse o *noisy-OR*, seria necessário especificar 8 probabilidades distintas. Em contrapartida, assumindo-se o *noisy-OR*, será necessário especificar apenas 4 probabilidades que estão destacadas na tabela 3.5. As linhas não destacadas apresentam as probabilidades calculadas a partir das probabilidades especificadas.

Tabela 3.5: TPC para a variável Febre

Gripe	**Malária**	**Resfriado**	**P(Febre = F| Gripe, Malária, Resfriado)**
T	T	T	$0.2 \times 0.1 \times 0.6 = 0.012$
T	T	F	$0.2 \times 0.1 = 0.02$
T	F	T	$0.2 \times 0.6 = 0.12$
T	F	F	0.2
F	T	T	$0.1 \times 0.6 = 0.06$
F	T	F	0.1
F	F	T	0.6
F	F	F	1

Observe que a quarta coluna da tabela especifica a probabilidade de não ocorrer Febre (Febre = F) quando são dadas as ocorrências de certas doenças. Como ocorreu no exemplo anterior, pode-se facilmente notar que *P(Febre = T | Gripe, Malária, Resfriado) = 1 - P(Febre = F | Gripe, Malária, Resfriado)* para qualquer configuração

das variáveis pais. É importante notar como a ocorrência dos pais influencia independentemente a não ocorrência da variável filha (*noisy-OR*), assim, por exemplo, a probabilidade de um indivíduo não ter febre uma vez que ele tenha Gripe e Malária, representada por *P(Febre = F | Gripe = T, Malária = T, Resfriado = F)*, pode ser calculada pela probabilidade dele não ter Febre, dado que ele só tem Gripe, *P(Febre = F | Gripe = T, Malária = F, Resfriado = F)*, vezes a probabilidade dele não ter Febre dado que ele só tem Malária, *P(Febre = F | Gripe = F, Malária = T, Resfriado = F)*. Tal probabilidade está calculada na segunda linha da tabela 3.4.

Intuitivamente o *noisy-OR* faz sentido, uma vez que como as variáveis pais determinam a ocorrência da variável *X*, quanto mais ocorrências das variáveis pais acontecerem, maior será a probabilidade de a variável filha ocorrer. Mais uma vez, é importante notar que com a utilização do *noisy-OR*, é necessário especificar um número linear de probabilidades em vez de um número exponencial destas, fato muito importante quando um nó da rede possui muitos pais.

III. Amostragem de uma rede Bayesiana

Pode-se dizer, informalmente, que amostrar uma RB significa gerar uma amostra (também chamada de observação) que é produzida pelo processo de escolher, baseando-se nas probabilidades armazenadas na RB, um único valor para cada variável codificada na rede. Existem vários métodos de amostrar uma RB (PEARL, 1988), e, dentre eles, o método chamado *Forward Sampling (FS)* (HERION, 1988) amostra as variáveis da rede em uma sequencia na qual os valores das variáveis pais de um nó são gerados antes da geração do valor deste nó. Uma vez que este método de amostragem é bastante clássico e também é usado em nossos sistemas-AED, o exemplo 3.18 ilustra a aplicação desta técnica para amostrar a RB do exemplo 3.17.

Exemplo 3.18: Considere a estrutura da RB bem como as probabilidades (inclusive as calculadas usando-se o *noisy-OR*) estipuladas no exemplo 3.17. Como afirmado, para amostrar um nó da rede, o FS precisa antes amostrar todos os pais deste nó. Assim, no caso da RB do exemplo 3.18, o FS precisa amostrar primeiro os nós Gripe, Malária, e Resfriado, antes de amostrar o nó Febre. Para se amostrar o nó Gripe, sorteia-se uniformemente um número aleatório $x \sim [0,1]$. Após o sorteio de *x*, e assumindo que ele possui o valor 0,2, então o nó Gripe irá assumir o valor T (verdadeiro), uma vez que *x* é menor que 0,3 que é o valor para P(Gripe = T). Para amostrar os nós Malária, e Resfriado, sorteiam-se uniformemente mais dois números no intervalo [0, 1]. Agora,

considerando que *x* assumiu, respectivamente, os valores 0,5 e 0,4 para a amostragem de Malária e Resfriado, e, seguindo a mesma lógica usando para o nó Gripe, tem-se que Malária irá assumir o valor F (falso) enquanto o nó Resfriado irá assumir o valor T (verdadeiro). Uma vez que todos os pais do nó Febre estão amostrados, pode-se, finalmente, amostrar tal nó. Novamente, sorteia-se uniformemente um número $x \sim [0,1]$. Considerando que *x* assumiu o valor 0,8, pode-se afirmar que o nó Febre assumirá o valor T (verdadeiro), uma vez que 0,8 é menor que *P(Febre = T | Gripe = T, Malária = F, Resfriado = T) = 1 - P(Febre = F | Gripe = T, Malária = F, Resfriado = T) = 1 - 0.12 = 0,88*. Aqui é importante notar que *P(Febre = F | Gripe = T, Malária = F, Resfriado = T)* foi calculada usando-se o *noisy-OR* (verificando os valores amostrados para os nós pais de Febre) uma vez que este foi adotado com objetivo de reduzir as TPC.

3.4.1.2 Os AEDs de Classe I: Modelos Univariados

Os AEDs desta classe não consideram nenhuma dependência entre as variáveis do problema ou, em outras palavras, consideram blocos construtores de ordem um. Desta forma, a distribuição de probabilidade conjunta é simplesmente o produtório da probabilidade marginal de cada variável do indivíduo, que pode ser expressa por:

$$p(X) = \prod_{i=1}^{n} p(X_i) \tag{3.15}$$

em que *n* é o número de variáveis do problema (tamanho do cromossomo) e $p(X_i)$ é a probabilidade marginal de ocorrência da variável X_i.

Caso tais modelos fossem representados como uma rede Bayesiana, tal rede poderia ser expressa de acordo com a figura 3.8. Note que não existem arcos ligando qualquer variável do problema; isto indica que as variáveis são independentes umas das outras.

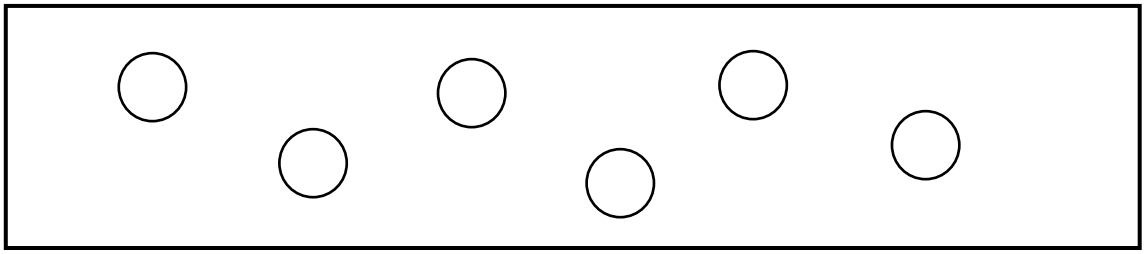

Figura 3.7: Modelos de Classe I representados como uma rede Bayesiana

Os AEDs de Classe I são muitos simples e por isso são muito eficientes, uma vez que não se faz necessário o aprendizado da estrutura do modelo probabilístico. Além disso, eles obtêm resultados interessantes em problemas cujas variáveis não são significativamente interdependentes. Como era de se esperar, algoritmos desta classe falham em problemas que existam fortes dependências entre as variáveis, uma vez que tais dependências não serão consideradas. Dentre alguns sistemas que utilizam tal modelo destacam-se o *Population Based Incremental Learning* (PBIL) (BALUJA, 1994) e o *compact Genetic Algorithm* (cGA) (HARIK et al., 1998).

O PBIL mantém um vetor de probabilidades cujo tamanho é igual ao número de variáveis do problema. Considerando o caso em que as variáveis assumem valores binários, cada posição *i* do vetor armazena a probabilidade da variável na posição *i* assumir o valor 1. Inicialmente, todo o vetor será preenchido com o valor 0.5 e, a cada geração, *M* indivíduos são gerados por amostragem do modelo. O vetor de probabilidades é então corrigido usando o melhor indivíduo atualmente gerado através da seguinte regra:

$$p_i \coloneqq p_i + \lambda \times (x_i - p_i) \tag{3.16}$$

onde p_i é a probabilidade de figurar o valor 1 na posição *i*, $\lambda \in (0, 1)$ é a taxa de aprendizado, e x_i é o valor da *i*-ésima posição da *string* binária que representa o melhor indivíduo da geração atual.

Por esta regra de atualização, é possível verificar que se o melhor indivíduo possui o valor 0 em alguma posição *i*, a p_i do modelo irá diminuir, o que significa aumentar a probabilidade de figurar um 0 naquela posição. De maneira semelhante, caso figure o valor 1 na posição *i* do melhor indivíduo da população, a p_i será acrescida. Outras variações de regras de atualização são possíveis, como modificar o vetor de probabilidades de acordo com uma taxa de mutação ou até mesmo modificar o vetor de probabilidade em relação aos $N < M$ indivíduos, em vez de atualizá-lo apenas de acordo com o melhor.

Exatamente como no PBIL, o cGA também mantém um vetor de probabilidade, contudo, ele constrói a cada geração apenas dois indivíduos. Para atualizar o vetor de probabilidade, ele compara estes dois indivíduos determinando qual é o pior e qual é o melhor (em relação ao valor de aptidão). Cada posição do vetor de probabilidade é atualizada usando-se uma taxa de atualização fornecida como parâmetro de entrada do sistema, sendo que caso o melhor e o pior indivíduo possuam em uma mesma posição *i* valores iguais, tal posição no vetor não será atualizada, uma vez que, intuitivamente, tal

posição não está contribuindo nem positivamente nem negativamente para a aptidão do indivíduo. A regra de atualização do cGA atualiza, desta forma, apenas as posições do vetor em que os valores do melhor e do pior indivíduo são diferentes.

3.4.1.3 Os AEDs de Classe II: Modelos Bivariados

Os AEDs desta categoria consideram dependências par a par entre as variáveis do cromossomo, em outras palavras, consideram blocos construtores de ordem dois. Quando comparados aos modelos Univariados, estes modelos são mais complexos e tomam a forma de uma rede Bayesiana, sendo que cada variável tem no máximo um único pai. Como esperado, esta classe de algoritmos alcança melhores resultados em problemas que apresentam interações entre as variáveis quando comparados aos AEDs de classe I. Entretanto, eles continuam falhando em problemas que apresentem interações entre múltiplas variáveis.

Alguns dos AEDs mais clássicos como *Mutual Information Maximization for Input Clustering* (MIMIC) (DE BONET et al., 1997), *Combining Optimizers with Mutual Information Trees* (COMIT) (BALUJA, DAVIES, 1997), e *Bivariate Marginal Distribution Algorithm* (BMDA) (PELIKAN, MÜHLENBEIN, 1999) estão representados na figura a seguir (PELIKAN, 2005).

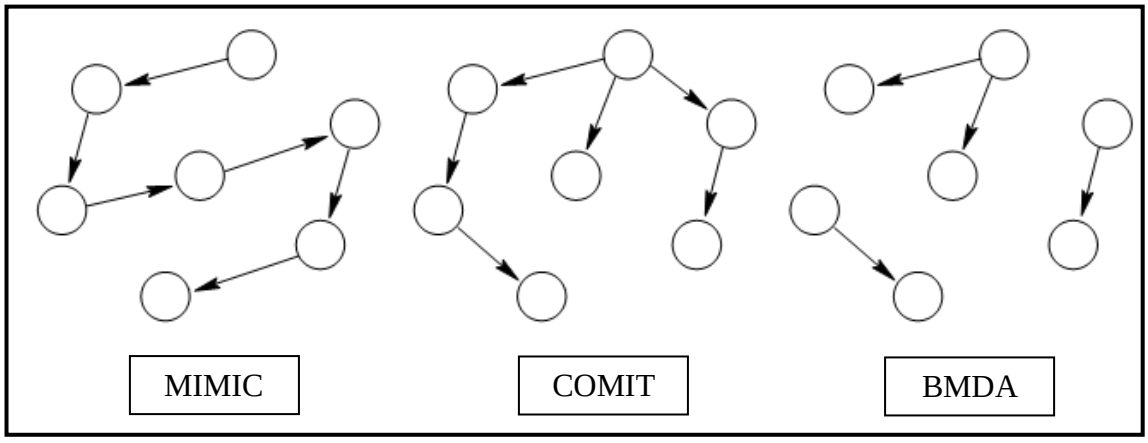

Figura 3.8: Modelos de Classe II representados como Redes Bayesianas

Note que apesar de todos os sistemas se categorizarem na mesma classe, a estrutura da rede Bayesiana é diferente para cada sistema apresentado. O MIMIC usa uma estrutura do tipo corrente, o COMIT usa uma estrutura de árvore enquanto que o BMDA usa uma estrutura de floresta. Contudo, a restrição de interações par a par é mantida, ou seja, cada nó da rede possuirá, no máximo, um único pai. É importante perceber também que este é um modelo mais complexo que os de Classe I, assim, os

algoritmos desta classe exigem que a estrutura da rede seja aprendida além do aprendizado das TPCs, o que claramente acarreta em um maior tempo computacional em relação aos AEDs de classe I. De qualquer forma, os AEDs de classe II, embora precisem mais tempo que os de classe I, alcançam resultados melhores que estes últimos em problemas cujas interações entre as variáveis não podem ser ignoradas.

3.4.1.4 Os AEDs de Classe III: Modelos Multivariados

Qualquer AED que considere a interdependência (entre as variáveis) de ordem superior a dois é classificado nesta categoria. Obviamente o modelo probabilístico fica bastante complexo nesta classe de algoritmos, fazendo com que a busca exaustiva por todos os modelos se torne impraticável. Assim, a maior parte dos sistemas implementada nesta categoria usa uma busca gulosa (RUSSELL, NORVIG, 2003) para construir a estrutura do modelo probabilístico. Entre os sistemas que mais se destacam nesta categoria estão o *Extended Compact Genetic Algorithm* (ECGA) (HARIK, 1999) e o *Bayesian Optimization Algorithm* (BOA) (PELIKAN et al., 1999); eles estão representados na figura 3.10 (PELIKAN, 2005).

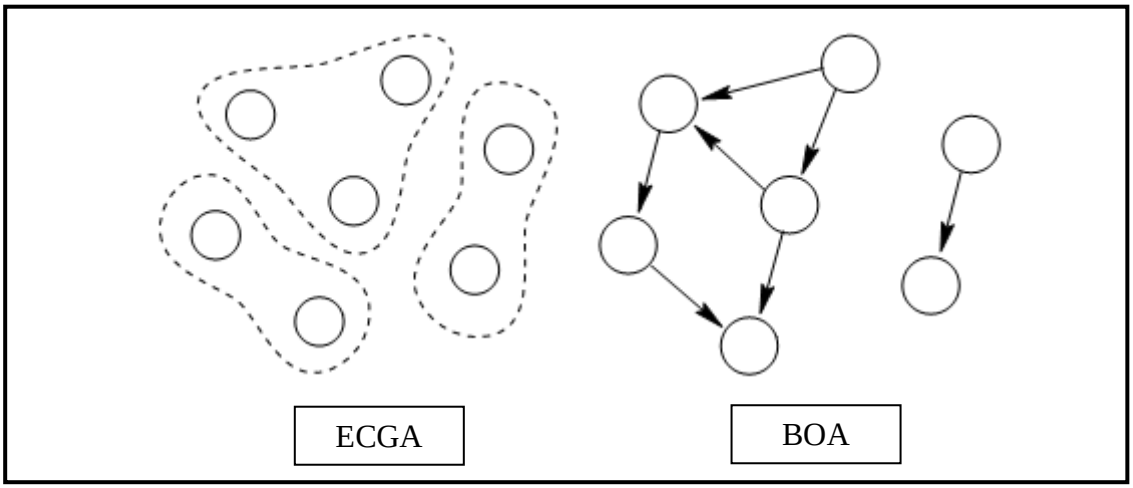

Figura 3.9: Modelos de Classe III

O ECGA usa um modelo probabilístico denominado de *Marginal Product Model* (MPM). Esse modelo é diferente de todos os modelos até agora apresentados no sentido de que ele não codifica dependências condicionais entre as variáveis do problema, ou seja, ele não assume a dependência direcional entre as variáveis (como assumido em uma rede Bayesiana). Tal modelo considera a probabilidade marginal de todo um conjunto de variáveis (circundados por linhas pontilhadas na figura 3.10). O ECGA é responsável por aprender estes grupos de dependências e, para isso, usa um algoritmo guloso que considera inicialmente tantos grupos quanto são as variáveis do

problema, mas que vai progressivamente juntando grupos, fazendo com que o número total de grupos diminua enquanto que o número de variáveis em cada grupo aumente.

O BOA, com era de se esperar, constrói uma rede Bayesiana que representa a dependência entre as variáveis do problema. Tanto sua estrutura quanto as TPCs devem ser aprendidas, sendo que para o aprendizado da estrutura da rede é usado um algoritmo guloso com operadores de adição e remoção de arestas entre as variáveis. As TPCs são geralmente atualizadas (por simples contagem) a partir das melhores soluções selecionadas por algum critério tal como roleta, *ranking*, ou torneio.

Como citado, tanto o ECGA quanto o BOA usam algoritmos de busca gulosos na construção de seus modelos e, é sabido, que nem sempre tal tipo de busca é capaz de encontrar um modelo adequado que represente corretamente as dependências entre as variáveis do problema. Atualmente, existem pesquisas focadas em aprendizado de uma boa estrutura do modelo probabilístico, uma vez que este é a parte essencial do funcionamento dos AEDs.

3.5 O Sistema de (OLIPHANT, SHAVLIK, 2007)

Como afirmado anteriormente, o trabalho de (OLIPHANT, SHAVLIK, 2007), chamado de OS, é o sistema que mais se aproxima da aplicação de um AEDs a ILP. Contudo, como será apresentado, tal trabalho usa apenas a motivação dos AEDs, não podendo, portanto, ser considerado um AED propriamente dito. Esta seção apresenta o trabalho OS e aponta as principais diferenças em entre ele e um AED.

O OS pode ser considerado uma modificação do trabalho proposto por (ZELEZNY et al., 2003). Zelezny et al. criaram um método chamado de *Rapid Random Restart* (RRR) e o incorporaram ao sistema Aleph com objetivo de reduzir o tempo de busca por uma cláusula e, consequentemente, por uma teoria. Como visto, o Aleph constrói uma *bottom clause* $\perp$ a partir de um exemplo positivo não coberto e busca por cláusulas em um látice que é limitado superiormente pela cláusula vazia □ e inferiormente pela $\perp$**.** O RRR é utilizado para buscar por cláusulas em tal látice. Basicamente, o RRR seleciona uma cláusula inicial no látice usando uma distribuição uniforme e, posteriormente, realiza uma busca local por uma quantidade de tempo fixa. Tal busca almeja encontrar uma cláusula que satisfaça certas limitações impostas *a priori*. Caso tal cláusula não seja encontrada dentro do limite de tempo estipulado, o RRR abandona toda a sua busca e escolhe, usando distribuição uniforme, outra cláusula

inicial para recomeçar a sua busca. Todo este processo é repetido por um número máximo de tentativas (fornecido como parâmetro de entrada do Aleph). O funcionamento básico do RRR é apresentado no algoritmo 3.7 (ZELEZNY et al., 2003).

Neste algoritmo, o passo 4 executa a busca em largura (RUSSELL, NORVIG, 2003) com a função heurística dada por $h = pos(C) - neg(C)$, em que $pos(C)$ e $neg(C)$ representam, respectivamente, o número de exemplos positivos e negativos provados pela cláusula C. O operador de refinamento (ρ) que gera os vizinhos da cláusula difere dos operadores *top-down* comuns no sentido de que ρ gera todas as cláusulas vizinhas, inclusive as cláusulas que *subsumes* C. Desta forma, a busca no látice não pode ser considerada *top-down* tão pouco *bottom-up*, mas sim radial, uma vez que pode tanto mover-se para cima quanto para baixo. Também no passo 4, uma cláusula C é considerada "boa" caso $pos(C) > 1$ e $Acc(C) > acc$, em que $Acc(C)$ é o valor de acurácia para cláusula C e acc é um dos parâmetros de entrada do RRR.

Como o RRR é apenas um novo método de busca do Aleph, o ciclo básico de cobertura de tal sistema continua o mesmo, ou seja, o RRR é executado repetidas vezes a partir de uma *bottom clause* construída usando um exemplo positivo não provado, até que todos os exemplos positivos sejam provados, sendo que a cada execução do RRR, uma nova cláusula é adicionada à teoria final e removem-se todos os exemplos positivos do conjunto de treinamento que são provados por tal cláusula. Os autores do RRR verificaram que a utilização do RRR em algumas bases de dados obtém uma drástica redução do tempo de execução enquanto que a acurácia sofre apenas uma pequena redução (ZELEZNY et al., 2003).

Algoritmo 3.7 – Rapid Random Restart (RRR)

Entrada:
(E+, E-): conjunto de exemplos de treinamento formado por exemplos positivos E+ e por exemplos negativos E-;
(tempoMax): tempo máximo de busca por uma cláusula.
(tentativasMax): número máximo de tentativas de busca por uma cláusula;
(látice): látice limitado por □ e pela ⊥;
(acc): acurácia imposta para que a cláusula seja considerada "boa";
Saída:
cláusula C encontrada;

Começo-RRR:
01- *tentativas* := 1;
02- Selecione aleatoriamente uma cláusula inicial C_0 no látice;
03- *tempoBusca* := 0 (começa a contar o tempo);
04- Começando em C_0, realize uma busca local até *tempoBusca* > *tempoMax* ou até que uma "boa" cláusula *C* seja encontrada;
05- Se (*C* foi encontrada) então:
06- Retorne *C*;
07- fim-se;
08- Se (*tentativas* < *tentativasMax*) então:
09- *tentativas* := *tentativas* + 1;
10- Retorne ao passo 2;
11- Senão Retorne falha;
12- fim-se;
Fim-RRR.

O trabalho de (OLIPHANT, SHAVLIK, 2007) modifica o RRR em dois pontos principais: (1) as cláusulas não são mais geradas de maneira uniforme, mas sim de maneira adaptativa; (2) após a aplicação do operador de refinamento ρ, nem todas as cláusulas vizinhas são avaliadas, apenas as mais promissoras.

O primeiro ponto de modificação é de certa maneira o mais importante, uma vez que é este ponto que utiliza a idéia de modelar o espaço de busca, que, como visto, é o objetivo principal dos AEDs. Dessa forma, maior ênfase será dada a explicação deste ponto.

Enquanto o RRR usa uma distribuição uniforme sobre todas as cláusulas com objetivo de selecionar o ponto inicial de sua busca local, o OS constrói uma distribuição não uniforme sobre o espaço de busca em direção a áreas mais promissoras de tal espaço. Para mapear tal distribuição, OS usa um par de redes Bayesianas cuja estrutura capta a dependência entre as variáveis dos literais presentes em uma *bottom clause*. A estrutura de tais redes é construída usando o algoritmo 3.8 (OLIPHANT, SHAVLIK, 2007).

Algoritmo 3.8 – Construção de uma Rede Bayesiana

```
Entrada:
      (BC): uma bottom clause consistindo de Cabeça e Corpo;
Variáveis Locais:
      (redeBayes): a rede Bayesiana sendo construída;
      (grupo): contém todos os literais cujas variáveis de entrada
               aparecem como variáveis de entrada na Cabeça ou como
               variáveis de saída em algum outro literal em redeBayes;
      (alcançável): contém todas as variáveis dos literais que podem
                    determinar dependência a outros literais:
Saída:
      Uma rede Bayesiana representada por redeBayes;
Começo-Contruir-Rede-Bayesiana:
01- redeBayes := vazio;
02- alcançável := variáveis de entrada na Cabeça;
03- Enquanto (Corpo não estiver vazio) faça:
04-   grupo := {l | l ∈ Corpo e variáveis de entrada em l estão em
      alcançável};
05-   Para cada literal ∈ grupo faça:
06-         Adicione_nó(literal, redeBayes);
07-   fim-para-cada;
08-   Corpo := Corpo – grupo;
09-   alcançável := alcançável + variáveis de saída de grupo;
10- fim-enquanto;
11- Retorne redeBayes;
12- Fim-Contruir-Rede-Bayesiana.
```

O passo 1 inicializa *redeBayes*, enquanto que o passo 2 armazena em *alcançável* todas as variáveis de entrada de *Cabeça*. O passo 3 controla o ciclo de execução do algoritmo. O passo 4 atualiza a variável *grupo* (que contém todos os literais cujas variáveis de entrada aparecem como variáveis de entrada na *Cabeça* ou como variáveis de

saída em algum outro literal já presente em *redeBayes*). O passo 5 varre todos os literais em *grupo* e adiciona cada um deles à *redeBayes* (passo 6). Assim que um literal é adicionado à *redeBayes*, o algoritmo o conecta a todos os literais que contém alguma variável de entrada que não está contida no literal *Cabeça*. Dessa forma, este passo cria as dependências na *redeBayes*. Para evitar ciclos, removem-se os arcos que poderiam "ligar" um literal que se localiza mais abaixo na *bottom clause* a um mais alto. Os passos 8 e 9 atualizam o *Corpo* e *alcançável* enquanto a *redeBayes* é construída.

Para ilustrar como a estrutura das redes Bayesianas é identificada, considere o exemplo a seguir retirado de (OLIPHANT, SHAVLIK, 2007).

Exemplo 3.19: Suponha que a partir de um exemplo positivo não provado, a seguinte *bottom clause* foi gerada[6]: h(+A,+B) :- p(+A, -C), q(+B, -C), r(+B, -D), p(+C, -D), q(+D, -E), r(+E, -C), p(+E, -F), em que o símbolo '+' significa que a variável é de entrada e '–' significa que a variável é de saída[7]. É importante lembrar que, pela declaração dos modos, qualquer variável de entrada do tipo T em um literal do corpo B_i deve aparecer como uma variável de saída do tipo T em um literal do corpo antes de B_i, ou como uma variável de entrada do tipo T na cabeça. Tal restrição pode ser facilmente capturada por uma rede Bayesiana. A título de ilustração, considere a rede Bayesiana na figura 3.11 cuja estrutura reflete a dependência entre os literais da *bottom clause* apresentada acima.

[6] Recorde-se que a *bottom clause* é gerada a partir de um exemplo positivo não provado usando as restrições impostas pelos modos e pelas *determinations*. Os modos, além de outras funções, especificam quais são as variáveis de entrada e de saída, enquanto que os *determinations* especificam quais literais podem aparecer no corpo da *bottom clause*.

[7] Os símbolos '+' e '-' não aparecem nos literais da bottom clause. No exemplo, exibem-se tais símbolos com o objetivo de identificar quais as variáveis são de entrada e quais são de saídas, contudo, tal informação é estipulada no BK sob a forma de declarações de modos.

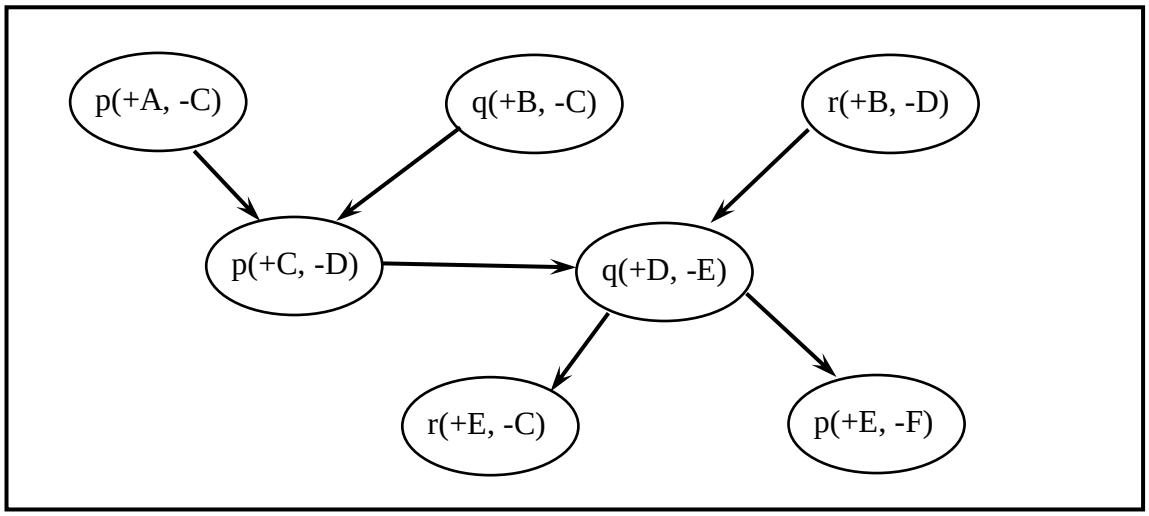

Figura 3.10: Rede Bayesiana construída sobre as dependências da *bottom clause*

A rede Bayesiana capta a dependência entre os literais da *bottom clause*, sendo que tais dependências são ditadas pelas declarações de modos. Pela rede exposta na figura 3.11, fica claro que o aparecimento do literal *p(+C, -D)* em uma cláusula qualquer no látice (limitado por □ e ⊥**)** depende do aparecimento do literal *p(+A, -C)* ou do literal *q(+B, -C)*, uma vez que figura em *p(+C, -D)* uma variável de entrada (*+C*) e pela regra, tal variável deve aparecer em um literal que precede *p(+C, -D)* como sendo uma variável de saída. A mesma interpretação deve ser feita para cada nó da rede, sendo que os literais que não possuem pais não dependem de nenhum literal para figurarem.

Como afirmado, a maior parte das buscas que foram implementadas no látice devem respeitar tais restrições, ou seja, no exemplo 3.19, caso *p(+A, -C)* e *q(+B, -C)* não figurem em um cláusula *C* qualquer, então o literal *p(+C, -D)* não poderá figurar em *C*.

Diferentemente do RRR, o OS usa um par de redes Bayesianas, cuja estrutura reflete a dependência dos literais da *bottom clause*, para gerar a próxima cláusula que será o ponto de partida inicial do algoritmo de busca local. É importante perceber que a estrutura da rede Bayesiana não é alterada durante a busca por uma cláusula no látice, mas suas TPC são atualizadas para que a próxima amostragem da rede Bayesiana gere cláusulas cada vez melhores. Antes de discutir como tais atualizações ocorrem, é importante apresentar como a rede gera uma determinada cláusula (semente) que será explorada pela busca local. Como era de se esperar, a construção de cláusulas passa pela possível geração (aparecimento) de cada um dos nós da rede de acordo com a fórmula a seguir.

$$P\left(N = T \middle| \Pi(N)\right) = \prod_{j=1}^{M} P\left(N = T \middle| S_j(N)\right) = \prod_{j=1}^{M} \left(1 - \prod_{R \in S_j(N)} P(N = F|R)\right) \quad (3.17)$$

em que *N* é um dos nós da rede, $\Pi(N)$ são os pais de *N*, S_j é um subconjunto de $\Pi(N)$ que satisfaz (todos os pais de *N* que possuem a variável *j* como sendo de saída) o argumento de entrada *j*, e *M* é o número de variáveis de entrada presentes no nó *N*.

O produtório de (3.17) varia sobre todas as *M* variáveis de entrada; isto reduz a probabilidade condicional a um produto de probabilidades condicionais simples, um para cada variável de entrada. Tais probabilidades condicionais simples são modeladas como *noisy-ORs*, como apresentado neste capítulo. No trabalho OS, os autores atribuem a $P(N = T \mid R = T)$ o valor 1, ou seja, se alguma variável de entrada não for satisfeita por ao menos um dos seus pais, então sua porção do produto $P(N|S_j(N))$ será igual a 0, fazendo com que todo o produto seja 0. Isso faz com que apenas cláusulas que respeitem estritamente a declaração dos modos sejam geradas. Para geração de uma cláusula, todos os nós da rede de Bayesiana são percorridos, sendo que cada literal aparecerá em uma cláusula obedecendo à probabilidade calculada pela fórmula apresentada.

Esta idéia de semear boas cláusulas (usando redes Bayesianas) no látice e depois realizar uma busca local foi implementada no sistema Gleaner (GOADRICH et al., 2006), diferentemente do RRR, que foi implementado diretamente no Aleph. Dessa forma, para entender com mais detalhes o trabalho OS, faz-se necessária uma breve apresentação do sistema Gleaner.

O objetivo do Gleaner é construir um comitê (*ensemble*) de teorias com o objetivo de produzir rapidamente boas curvas de *recall-precision*[8]. O sistema Gleaner utiliza o sistema Aleph para gerar a *bottom clause* como também seus métodos de busca para encontrar boas cláusulas no látice que é limitado pela cláusula vazia e pela *bottom clause*. Grosso modo, pode-se dizer que o sistema Gleaner é uma capa que envolve o sistema Aleph sendo ele executa uma busca randomizada que coleta boas cláusulas ao longo de sua busca com objetivo de produzir boas curvas de *recall-precision*. O Gleaner é um sistema que executa dois passos principais: (1) aprende um grande espectro de cláusulas e, (2) combina as cláusulas aprendidas em uma teoria objetivando a

[8] Assumindo que VP, FP e FN representam, respectivamente os Verdadeiros Positivos, Falsos Positivos e Falsos negativos, têm-se que, *recall* = VP/(VP + FN) e *precision* = VP/(VP + FP).

maximização da *precision* para um valor particular de *recall*[9]. O Algoritmo 3.9, apresenta o ciclo principal do sistema Gleaner (GOADRICH et al., 2006).

No Gleaner, como afirmado, o sistema Aleph é usado para buscar pelas cláusulas, sendo que o Gleaner utiliza *K* exemplos positivos para a construção de *K bottom clauses* com objetivo de obter diversidade. Para cada *bottom clause* gerada, o sistema considera a exploração de no máximo *N* cláusulas. Assim que estas cláusulas são geradas, o Gleaner computa o *recall* de cada cláusula e determina em qual caixa (intervalo) de *recall* a cláusula cairá. Cada caixa mantém a melhor cláusula que apareceu naquela caixa para a *bottom clause* atual. Para determinar a melhor cláusula em uma caixa, usa-se a função heurística $h = precision \times recall$. No final do processo de busca, existirão *B* cláusulas coletadas para cada *bottom clause* e, uma vez que se usa *K bottom clauses*, ter-se-á um total de $B \times K$ cláusulas (assumindo que ao menos uma das *N* cláusulas exploradas caia em uma caixa de *recall* para uma dada semente). A partir desta grande quantidade de cláusulas, o Gleaner monta finalmente o seu comitê de teorias.

Note que é no passo 4 do algoritmo 3.9 que o trabalho OS é acoplado[10]. Os autores de OS comparam o sistema Gleaner usando o RRR no passo 4 e usando a variação do RRR. Eles reportaram uma melhoria na área sob a curva *recall-precision* quando usam as redes Bayesianas para guiar a busca em vez de usarem o método padrão do RRR.

[9] Como estamos analisando a questão da modelagem do espaço de busca, apenas o passo (1) será revisado, sendo que os leitores interessados em um detalhamento do Gleaner podem procurar (GOADRICH, OLIPHANT, SHAVLIK, 2006).

[10] Os autores não deixam claro como é gerada a primeira cláusula na primeira iteração do sistema, uma vez que as TPCs das redes Bayesianas não foram ainda aprendidas, já que ainda não existem cláusulas, como também não foi apresentado nenhum tipo de inicialização destas tabelas. Dessa forma, supõe-se que a primeira iteração do sistema gera uma *bottom clause* qualquer e a busca começa a partir desta *bottom clause* gerada.

Algoritmo 3.9 - Ciclo Básico do Gleaner

Entrada:
B caixas de recall;
Saída:
Grande conjunto de cláusulas distribuídas nas B caixas de Recall;

Começo-Gleaner:
01- Inicialize as *B* caixas de *recall* que uniformemente dividam o intervalos de *recall* [0, 1];
02- Para *i* de 1 até *K* faça (pode ser feito em paralelo)
03- Selecione um exemplo positivo não coberto e gere a ⊥;
04- Use busca local randomizada para gerar as cláusulas;
05- Após a geração de uma nova cláusula *C* faça:
06- Ache a caixa de *recall r* para *C*;
07- Se (*precision* × *recall* é o melhor para *i* na caixa *r*) então:
08- Armazene *C* em *r* e descarte a antiga melhor cláusula armazenada em *r*;
10- Até *N* cláusulas serem geradas repita passos de 04 a 08;
11- fim-para;
Fim-Gleaner.

Retornando a discussão de OS, como afirmado, tal sistema utiliza duas redes Bayesianas para gerar a cláusula que será explorada pelo mecanismo de busca local modificado. Ambas as redes Bayesianas possuem a mesma estrutura construída sobre a bottom clause *i* ($1 \leq i \leq K$) considerada. O que diferencia ambas as redes Bayesianas é que uma é usada para exploração enquanto que a outra é usada para explotação. As TPCs da rede que é utilizada para exploração são atualizadas considerando todas as cláusulas geradas, enquanto que a as TPCs da rede utilizada para explotação são atualizadas utilizando-se apenas as melhores cláusulas geradas pelo Gleaner. Para a geração de uma cláusula, ambas as redes são combinadas de forma a equilibrar a exploração e a explotação. Diferentemente do RRR, a proposta OS não explora todos os possíveis sucessores de uma cláusula pela aplicação de ρ, mas apenas os sucessores mais promissores, o que, consequentemente, implica em um menor custo de busca.

Uma vez apresentado a proposta OS, pode-se ver que a única semelhança que este sistema possui com um AED é a idéia de modelar o espaço de busca de soluções. De fato, a proposta OS apenas usa o modelo probabilístico aprendido para semear um

bom lugar do espaço de busca e então aplica um mecanismo de busca local para executar a busca por cláusulas. De forma geral, os AEDs utilizam uma abordagem diferente, uma vez que o modelo probabilístico aprendido é responsável tanto por encontrar um lugar promissor do espaço de busca quanto por efetivamente explorá-lo, embora buscas locais possam ser usadas *em conjunto* com o modelo probabilístico para explorar tal local promissor. Outra diferença fundamental em relação aos AEDs é que estes utilizam uma população (conjunto de soluções) que (geralmente) competem entre si para explorar o espaço de busca. Este tipo de exploração não ocorre na proposta OS, uma vez que uma única cláusula é considerada por vez durante a busca local do látice.

3.6 Resumo do Capítulo

Este capítulo apresentou o funcionamento básico de um AG bem como uma simples analogia com o processo de evolução natural. Foram discutidos os principais parâmetros dos AGs bem como o provável motivo pelo qual esta classe de algoritmos funciona bem para a maioria dos problemas. Alguns dos sistemas genéticos para ILP foram apresentados, com destaque para o QG/GA. Adicionalmente, este capítulo apresentou os AEDs, que podem ser considerados uma variação dos AGs tradicionais, em que o principal mecanismo de evolução é um modelo probabilístico que estima e adapta progressivamente a distribuição de boas soluções. Por fim, analisou-se o trabalho de (OLIPHANT, SHAVLIK, 2007) o qual usa redes Bayesianas para localizar boas regiões do espaço de busca, mas que, como visto, não pode ser considerado um AED, uma vez que tal trabalho utiliza apenas o ponto motivador dos AED.

Capítulo 4: Programação em Lógica Indutiva Através de Algoritmo Estimador de Distribuição

Este capítulo apresenta um sistema ILP baseado em um AED, chamado de AED-ILP, bem com suas duas extensões, o RAED-ILP e o HAED-ILP. Os ciclos de execução dos sistemas propostos são apresentados e seus principais parâmetros são discutidos.

4.1 Introdução

Discutiu-se até agora, que o espaço de busca por cláusulas, mesmo quando este é limitado inferiormente pela *bottom clause* e superiormente pela cláusula vazia (formando uma látice), continua sendo muito grande, o que inviabiliza uma busca exaustiva. Dentre alguns dos sistemas cuja busca é realizada em tal látice, podem-se citar: Progol, Aleph, e o QG/GA. Viu-se que o sistema Aleph é muito flexível devido a sua grande quantidade de parâmetros e que o QG/GA, plugado ao Progol, usa o operador QG para semear cláusulas consistentes no látice e a partir daí utiliza um AG convencional para explorar o espaço de busca de cláusulas. Adicionalmente, foi apresentado o trabalho (OLIPHANT, SHAVLIK, 2007), que modela, usando um par de redes Bayesianas, a distribuição de boas cláusulas no espaço de busca e realiza a exploração de cláusulas por meio de um mecanismo de busca local. Ao revisar-se tal trabalho, apresentaram-se os motivos pelo qual a referida proposta não pode ser considerada uma aplicação de um AED a ILP, uma vez que ela só apresenta a motivação dos AED, ou seja, a modelagem do espaço de busca.

A partir do estudo de tais sistemas, decidiu-se propor um sistema baseado em AEDs para gerar teorias constituídas por cláusulas que fazem parte do látice limitado pela cláusula vazia e por uma *bottom clause*, uma vez que, como afirmado, tais algoritmos, embora apresentem resultados bem interessantes, ainda não foram aplicados ao problema de Programação em Lógica Indutiva. Dessa forma, propõe-se o sistema denominado AED-ILP (PITANGUI, ZAVERUCHA, 2010) (PITANGUI, ZAVERUCHA, 2011), bem como suas (duas) extensões, o RAED-ILP (PITANGUI, ZAVERUCHA, 2012) e o HAED-ILP, sendo que todos utilizam a codificação binária

para representar as cláusulas no látice, tal como (ALPHONSE, ROUVEIROL, 2000), (ALPHONSE, ROUVEIROL, 2006), (MUGGLETON, TAMADDONI-NEZHAD, 2007) e usam o sistema Aleph para construção das *bottom clauses*.

De forma geral, os sistemas-AED[1] possuem três objetivos principais: (I) construir teorias simples de alta acurácia usando razoável tempo computacional, sendo capazes de competir ou até mesmo superar alguns sistemas ILP considerados estados da arte; (II) serem aplicáveis a problemas de alta complexidade, tais como (BOTTA et al., 2003) e (ALPHONSE, OSMANI, 2009), uma vez que os sistemas clássicos não obtêm bons resultados quando aplicados a alguns destes problemas; (III) manter certa simplicidade, isto é, os sistemas devem possuir poucos parâmetros a serem configurados, uma vez que os sistemas com grande número de parâmetros, embora sejam flexíveis, são muito difíceis de serem configurados. Com estes objetivos em mente, são apresentadas a seguir algumas diretrizes assumidas para o desenvolvimento do AED-ILP bem como uma explicação que justifica a razão destas serem adotadas. Tais diretrizes são também válidas para os sistemas RAED-ILP e HAED-ILP, uma vez que ambos os sistemas derivaram-se do AED-ILP.

I. Adoção de um AED como mecanismo de exploração padrão: os sistemas-AED propostos utilizam um AED como mecanismo de exploração de hipóteses. Atualmente, nenhuma outra forma de busca é utilizada, ou seja, o AED é responsável tanto por encontrar uma área promissora no espaço de busca quanto por efetivamente explorá-la.
Justificativa: a adoção desta classe de algoritmos se mostra promissora uma vez que, como afirmado, ela já foi aplicada a vários problemas de otimização e obteve melhores resultados quando comparados aos AGs tradicionais. Além disso, existe o fato de que esta "classe" de algoritmos ainda não foi aplicada a ILP, sendo que, caso verificado sua eficiência em tal problema, existe uma grande quantidade de variações da técnica que pode ser utilizada com objetivo de contribuir para o desenvolvimento na área do problema abordado.

II. Uso de *bottom clauses* (cláusulas mais específicas): os sistemas-AED buscam por cláusulas cujos corpos são subconjuntos dos literais das *bottom clauses*.
Justificativa: a implicação inversa através do uso de *bottom clauses* pode ser considerada uma das principais razões do sucesso de importantes sistemas ILP tais

[1] Lembre-se que tal é utilizada em substituição a expressão "AED-ILP, RAED-ILP, e HAED-ILP" na discussão de características que são comuns a todos os sistemas.

como o Progol (MUGGLETON, 1995), Aleph (SRINIVASAN, 2003), QG/GA (MUGGLETON, TAMADDONI-NEZHAD 2006), OS (OLIPHANT, SHAVLIK, 2007) dentre outros.

III. Adoção de Redes Bayesianas como Modelos Probabilísticos: os sistemas-AED propostos utilizam redes Bayesianas para captar as dependências entre os literais da *bottom clause* tal como ocorre no OS. Tais redes são, portanto, os modelos probabilísticos responsáveis por explorar o espaço de busca.

Justificativa: o uso de redes Bayesianas para modelagem do espaço de busca já se mostrou promissor no sistema OS, assim, espera-se que tais estruturas sejam capazes de explorar o espaço de busca de forma efetiva. É importante destacar que as estruturas das redes não se modificam durante a exploração, já que elas apenas capturam as dependências entre os literais da *bottom clause*. Assim, o passo de construção do modelo é efetuado apenas uma vez. Contudo, deve-se ressaltar que as probabilidades armazenadas no modelo são atualizadas a cada geração, i.e., embora as estruturas das redes Bayesianas sejam mantidas fixas durante toda a execução dos sistemas, suas TPCs são atualizadas a cada geração. Assim, pela adoção de um modelo estruturalmente estático, mas ainda assim poderoso, tem-se um bom mecanismo de exploração aliado a um baixo custo de tempo computacional.

IV. Simples Atualização do Modelo Probabilístico: as atuais versões dos sistemas-AED propostos utilizam uma simples variação da forma de atualização do modelo probabilístico do sistema PBIL (BALUJA, 1994).

Justificativa: a forma de atualização do modelo probabilístico presente no PBIL é bem simples e por isso demanda baixo custo de tempo computacional quando comparado a atualização de outros sistemas baseados em AED, como, por exemplo, o *Bayesian Optimization Algorithm* (PELIKAN, 2005). Como se almeja que os sistemas propostos sejam competitivos em relação à demanda por tempo, uma atualização simples do modelo probabilístico se faz necessária.

V. Evolução de Teorias: os sistemas propostos consideram uma abordagem diferente da maioria dos sistemas ILP, uma vez que eles evoluem teorias, ou seja, cada indivíduo da população representa um conjunto de cláusulas. Esta abordagem contrasta com a maioria dos sistemas ILP atuais, uma vez que tais sistemas, tanto os tradicionais como FOIL, Progol e Aleph, quanto os sistemas ILP baseados em AGs, como REGAL, G-Net, SIA01, ECL e QG/GA, buscam por cláusulas isoladas em vez de teorias. Algumas

exceções a esta regra são o HYPER (BRATKO, 1999) e FORTE-MBC (DUBOC et al., 2009). Este último, embora seja um sistema de revisão de teorias, pode ser usado para aprendê-las "do zero" também.

Justificativa: quando a busca é realizada por cláusulas isoladas, geralmente emprega-se o algoritmo de cobertura para que este progressivamente monte a teoria. Um dos problemas já verificados com a utilização deste tipo de algoritmo é que ele gera teorias desnecessariamente grandes, i.e., teorias com um grande número de cláusulas (BRATKO, 1999). Com a evolução de teorias, espera-se que estas possuam um menor número de cláusulas quando comparadas a teorias desenvolvidas por sistemas que usam um algoritmo de cobertura.

Uma vez discutidos os pontos em comum que nortearam o desenvolvimento do AED-ILP e de suas extensões, é importante destacar os pontos principais que os diferenciam.

De forma geral, o RAED-ILP pode ser entendido como o AED-ILP que utiliza o algoritmo Reduce (MUGGLETON, FENG, 1990) com objetivo de reduzir o seu espaço de busca. De forma geral, como visto no capítulo 3 (seção 3.3.1), pode-se dizer que o algoritmo Reduce retira alguns literais da *bottom clause* fazendo com que a mesma se torne menor. Em consequencia, o espaço de busca dos sistemas que utilizam *bottom clauses* "reduzidas" se torna menor em relação ao espaço de busca delimitado por *bottom clauses* não reduzidas.

Por sua vez, o HAED-ILP pode ser entendido como o AED-ILP com seu ciclo de busca completamente remodelado. Esta nova versão do AED-ILP é chamada AED-ILP Híbrido (por isso o nome de HAED-ILP) uma vez que ele continua evoluindo teorias (tal como motivado pela 5ª diretriz discutida), mas estas, em tal sistema, são construídas de maneira iterativa. Tal como será apresentado posteriormente, esta nova abordagem de evoluir teorias é completamente diferente da abordagem utilizada pelos sistemas AED-ILP e RAED-ILP, uma vez que em tais sistemas as teorias possuem um tamanho fixo (fornecido *a priori* pelo usuário) durante toda a execução do sistema.

Os pontos principais que, em conjunto, diferenciam os sistemas AED-ILP, RAED-ILP, e HAED-ILP de todos os outros sistemas ILP são:

- Utilização de um AED responsável por encontrar um lugar promissor no espaço de busca e por explorá-lo;
- Busca por teorias em vez de busca por cláusulas isoladas;

- Utilização de uma variação da regra de atualização do sistema PBIL para modificar as TPCs das redes Bayesianas;
- Utilização do algoritmo Reduce para reduzir o espaço de busca por teorias (apenas para o RAED-ILP);
- Utilização da nova abordagem híbrida para evolução de teorias (apenas para o HAED-ILP).

4.2 ILP através de AEDs

Primeiramente, esta seção apresenta os ciclos de execução dos sistemas AED-ILP e RAED-ILP, uma vez que eles apenas se diferenciam pela aplicação do algoritmo Reduce por parte do RAED-ILP. A seguir, discutem-se os tópicos principais (e comuns) relacionados a todos os sistemas-AED, tal como a codificação do espaço de busca (seção 4.2.2), e de como o modelo probabilístico é construído (seção 4.2.2), amostrado (seção 4.2.3), e atualizado (seção 4.2.4) com intuito de explorar o espaço de busca. Com a apresentação do AED-ILP e do RAED-ILP, bem como com a discussão dos principais pontos relacionados a todos os sistemas propostos, pode-se então entender no novo ciclo de busca do AED-ILP desenvolvido sob o nome de HAED-ILP, que será explicado ao final desta seção. Como já mencionado, este novo ciclo de busca é uma forma híbrida que considera tanto o algoritmo de cobertura quanto a evolução de teorias completas em detrimento a cláusulas isoladas.

4.2.1 Os Ciclos de Execução dos Sistemas AED-ILP e RAED-ILP

Os ciclos de execução dos sistemas AED-ILP e RAED-ILP podem ser resumidos como apresentado no algoritmo 4.1.

Algoritmo 4.1 – O Ciclo de Execução dos Sistemas AED-ILP e RAED-ILP

Entrada:
(E+, E-, BK): conjunto de exemplos positivos E+; conjunto de exemplos negativos E-, e o conhecimento preliminar BK; **(numCls)**: número de cláusulas na teoria; **(p_1, p_2, p_3)**: probabilidades iniciais para as TPCs; **(λ_1)**: taxa de aprendizado para a regra de atualização do PBIL; **($numPop_1$, $numGen_1$)**: número de indivíduos na população e número de gerações do algoritmo;

Variáveis Locais:
(conj[BC]): um conjunto de *bottom clauses* construídas usando-se exemplos positivos; **(modProb):** modelo probabilístico que armazena um conjunto de redes Bayesianas; **(popPrimária):** população de teorias; **($cont_1$):** contador usado para controlar o número de gerações executadas;

Saída:
Teoria Aprendida;

```
Começo-AED-ILP-e-RAED-ILP:
01- conj[BC] := Crie_bottom_clauses(numCls exemplos positivos
    aleatórios de E+, BK);
02- conj[BC]:= Reduce(conj[BC]);
03- modProb := Crie_redes_bayesianas(conj[BC]);
04- Para( (cont1 := 1) até numGen1) faça:
05-   popPrimária := Gere_população(modProb, numPop1);
06-   Avalie_população(popPrimária, numPop1);
07-   Atualize_modelo_probabilístico(modProb, o indivíduo mais apto
      em popPrimária, λ1);
08- fim-para;
09- popPrimária := Gere_população(modProb, numPop1);
10- Avalie_população(popPrimária, numPop1);
11- Retorne o melhor indivíduo em popPrimária;
Fim-AED-ILP-e-RAED-ILP.
```

O passo 1 cria *numCls bottom clauses* selecionando-se aleatoriamente *numCls* exemplos positivos (a seção 4.2.2 descreve como as *bottom clauses* são codificadas usando *strings* binárias) e as atribui a variável *conj[BC]*. O passo 2 é apenas aplicado ao RAED-ILP: o algoritmo Reduce (revisado no capítulo 3, seção 3.3.1) é aplicado a todas as *bottom clauses* no *conj[BC]* e retorna um conjunto de *numCls bottom clauses* reduzidas. O passo 3 executa a função *Crie_redes_bayesianas*/1 que recebe como entrada o conjunto de *bottom clauses conj[BC]* (reduzidas ou não) e retorna um

conjunto de redes Bayesianas atribuindo-o à variável *modProb* (veja o algoritmo 4.2 da seção 4.2.2). O passo 5 gera $numPop_1$ indivíduos, amostrando (veja seção 4.2.3) cada rede Bayesiana $numPop_1$ vezes e atribui os indivíduos gerados a variável *popPrimária*. Este passo é executado como segue: para gerar um indivíduo que contém *numCls* cláusulas, a primeira RB do *modProb* é amostrada (gerando a primeira cláusula do indivíduo), em seguida, a segunda RB do *modProb* é amostrada (gerando a segunda cláusula do indivíduo) e assim por diante, até ser amostrada a *numCls*-ésima RB do *modProb* para gerar a *numCls*-ésima cláusula do indivíduo. Para gerar uma população de $numPop_1$ indivíduos, este processo é repetido $numPop_1$ vezes (veja algoritmo 4.3 na seção 4.2.2). O passo 6 avalia a população com uma função de aptidão pré-definida (acurácia, por exemplo). O passo 7 atualiza (veja seção 4.2.4) todas as RBs no *modProb* usando o indivíduo mais apto na população e a taxa de aprendizado λ_1. Para atualizar cada uma das *numCls* RBs, todas *numCls* cláusulas do indivíduo mais apto são sequencialmente usadas. Assim, a primeira cláusula do melhor indivíduo é usada para atualizar a primeira cláusula da RB, a segunda cláusula do melhor indivíduo é usada para atualizar a segunda RB, e assim por diante, até usar-se a *numCls*-ésima cláusula do melhor indivíduo para atualizar a *numCls*-ésima RB (veja algoritmo 4.4 da seção 4.2.4). O passo 9 gera uma nova população de indivíduos (como explicado acima) e a atribui à variável *popPrimária*. O passo 10 avalia todos os indivíduos em *popPrimária*, e, finalmente, o passo 11 retorna o melhor indivíduo em *popPrimária* como a teoria aprendida.

A diferença fundamental entre o RAED-ILP e o AED-ILP está em destaque no passo 2. Enquanto o AED-ILP gera diretamente *numCls* redes Bayesianas das *numCls bottom clauses* criadas, o RAED-ILP, primeiro aplica o algoritmo Reduce a cada uma das *numCls bottom clauses* geradas, para só depois construir suas RBs. Dessa forma, pode-se notar que o espaço de busca do RAED-ILP é menor quando comparado ao do AED-ILP, uma vez que as *bottom clauses* reduzidas possuem menos literais quando comparadas a *bottom clauses* que não foram reduzidas.

Com objetivo de avaliar os nossos sistemas-AED, criou-se também o sistema chamado AG-ILP pela substituição o AED no AED-ILP por um AG "convencional" que usa seleção por torneio, recombinação de dois pontos e mutação de um bit.

4.2.2 Sistemas-AED: Busca por Teorias e Codificação do Espaço de Busca

O sistemas-AED buscam por teorias em vez de cláusulas isoladas, e, como uma teoria é composta por várias cláusulas, é necessário que se mantenha um modelo probabilístico responsável pela geração de uma determinada cláusula na teoria. Conforme afirmado, nossos sistemas utilizam redes Bayesianas como modelos probabilísticos, sendo que tais redes, de forma equivalente ao OS, captam as dependências entre os literais da *bottom clause*. É importante destacar que os sistemas-AED utilizam o mesmo algoritmo apresentado em (OLIPHANT, SHAVLIK, 2007) (e revisado no capítulo 3, seção 3.8) para construir suas redes Bayesianas. De forma geral, o número de redes Bayesianas que compõem o modelo probabilístico dos sistemas-AED é igual ao número de cláusulas que compõe as suas teorias.

Para o AED-ILP e o RAED-ILP, o número de cláusulas que compõe uma teoria é um parâmetro de entrada, assim, logo no inicio do ciclo de execução destes sistemas, tem-se o número fixo de redes Bayesianas que compõem o modelo. Já para o HAED-ILP, como será apresentado posteriormente, as redes Bayesianas são adicionadas iterativamente ao modelo probabilístico. Apesar desta diferença, é importante perceber que para todos os sistemas-AED, cada rede Bayesiana será construída para captar as dependências entre os literais de uma *bottom clause* específica, assim, caso deseje-se (tanto para o AED-ILP e RAED-ILP, quanto para o HEAD-ILP) evoluir teorias compostas por três cláusulas, será necessária a construção de três *bottom clauses* e, consequentemente, a construção de três redes Bayesianas, sendo que cada rede representa a dependência dos literais de uma determinada *bottom clause.* Sob este aspecto, é importante destacar uma das diferenças entre o HAED-ILP e os outros dois sistemas (o AED-ILP e RAED-ILP): para estes dois últimos, a construção das redes Bayesianas se faz logo no começo do algoritmo (como pôde ser verificado no algoritmo 4.1), enquanto que para o HAED-ILP, a construção das redes Bayesianas é realizada ao decorrer do seu ciclo de execução (como será posteriormente apresentado).

O processo de construção do modelo probabilístico dos sistemas AED-ILP e RAED-ILP pode ser mais bem entendido pelo algoritmo 4.2. Já o processo construção do modelo probabilístico para o HEAD-ILP poderá ser mais bem visualizado posteriormente durante a apresentação de seu ciclo de execução.

O algoritmo 4.2 recebe como entrada um conjunto de *bottom clauses conj[BC]* construído sobre exemplos positivos selecionados aleatoriamente. Seu ciclo principal

constrói tantas redes Bayesianas quantos são o número de *bottom clauses* em *conj[BC]* (tal número é representado por |*conj[BC]*|**)**. Para isso, o passo 3 seleciona a *i*-ésima *bottom clause* em *conj[BC]*, enquanto o passo 4 constrói uma rede Bayesiana (usando o algoritmo 3.5 - revisado no capítulo 3, seção 3.8) que capta as dependências entre os literais da *bottom clause* selecionada. O passo 5 adiciona a rede Bayesiana construída ao modelo probabilístico que, em nossa abordagem, é representado por um conjunto de redes Bayesianas.

Algoritmo 4.2 – Construção do Modelo Probabilístico

Entrada:
(conj[BC]): um conjunto de *bottom clauses* construídas sobre exemplos positivos selecionados aleatoriamente;
Variáveis Locais:
(modProb): modelo probabilístico constituído por redes Bayesianas;
(bc): *bottom clause* construída a partir de um exemplo positivo;
(rb): rede Bayesiana construída a partir de uma *bottom clause*;
Saída:
(modProb modificado): composto por numCls redes Bayesianas;

Começo-Construir-Modelo-Probabilístico:
01- *modProb* := ∅;
02- Para *i* = 1 até |*conj[BC]*| faça:
03- *bc* := a *i*-ésima *bottom clause* de *conj[BC]*;
04- Construa *rb* a partir de *bc* e da declaração dos modos usando o algoritmo 3.5 (capítulo 3, seção 3.8);
05- *modProb* := *modProb* ∪ *rb*;
06- fim-para;
07- Retorne *modProb*;
Fim-Construir-Modelo-Probabilístico.

Independentemente de o modelo probabilístico ser construído no início do algoritmo (para o AED-ILP e RAED-ILP), ou no decorrer de seu ciclo de execução (para o HAED-ILP), uma vez que se possui o modelo probabilístico representando um conjunto de redes Bayesianas construídas sobre diferentes *bottom clauses*, os sistemas podem então construir uma população de teorias para explorar o espaço de busca. Como afirmado, cada rede Bayesiana no modelo probabilístico é responsável por gerar uma cláusula para uma teoria. O algoritmo 4.3 ilustra de que forma o modelo probabilístico é usado para constituir uma população de teorias.

Algoritmo 4.3 – Construção de uma População de Teorias

Entrada:
(modProb): modelo probabilístico constituído por redes Bayesianas;
(numPop): número de indivíduos (teorias) na população;
Variáveis Locais:
(C_j): j-ésima cláusula construída ($1 \leq j \leq |modProb|$); **(T_i)**: i-ésima teoria construída ($1 \leq i \leq numPop$);
Saída:
(pop): população de numPop teorias;

Começo-Construção-População-Teorias:
01- $pop := \varnothing$;
02- Para i = 1 até *numPop* faça:
03- $T_i := \varnothing$;
04- Para j = 1 até $|modProb|$ faça:
05- Selecione a j-ésima rede Bayesiana de *modProb* e a amostre para gerar a cláusula c_j;
06- $T_i := T_i \cup c_j$;
07- fim-para;
08- $pop := pop \cup T_i$;
09- fim-para;
10- Retorne *pop*;
Fim-Construção-População-Teorias.

O algoritmo 4.3 recebe como entrada um modelo probabilístico constituído por |*modProb*| redes Bayesianas e retorna como saída uma população de teorias sendo que cada teoria é composta por |*modProb*| cláusulas. O passo 1 inicializa a população de teorias enquanto o passo 2 assegura que todas as teorias da população serão construídas. O laço no passo 4 é o responsável por criar, sequencialmente, as |*modProb*| cláusulas de uma teoria *i*. Para isso, ele seleciona no passo 5 a *j*-ésima ($1 \leq j \leq |modProb|$) rede Bayesiana de *modProb* e a amostra para gerar a *j*-ésima cláusula (representada por c_j) da teoria *i*. O passo 6 do laço adiciona a *j*-ésima cláusula gerada a teoria *i*. Como pode ser visto, o laço no passo 4 criará uma teoria composta por |*modProb*| cláusulas. O passo 7 é o responsável por adicionar à população de teorias, a teoria recentemente criada. As etapas de construção de uma teoria (passos 5 e 6) juntamente com a sua adição à população (passo 7), são repetidos *numPop* vezes (passo 2). Tal repetição dá origem a uma população de *numPop* teorias, sendo que cada teoria é composta por |*modProb*| cláusulas.

Pelo algoritmo 4.3, é possível perceber que a ordem das cláusulas em uma teoria é determinada pela ordem em que as redes Bayesianas são construídas que, por sua vez, seguem a ordem em que as *bottom clauses* foram construídas. Desta forma, tem-se que a *i*-ésima cláusula de uma teoria pode ser visualizada como um subconjunto dos literais da *i*-ésima *bottom clause* construída. Tal ponto ficará mais claro com o exemplo a seguir.

Exemplo 4.1: Assuma que objetiva-se aprender a relação definida por q(X,Y) e que qualquer um dos sistemas-AED está trabalhando com teorias que possuem 2 cláusulas e uma população de 2 indivíduos (teorias). Assim, considere que duas *bottom clauses* foram construídas, a saber:

bc_1: q(X,Y) :- r(Z,Y), w(X,Y).

bc_2: q(X,Y) :- t(X,Z), h(Z), s(X,Y,Z), p(X,Y).

Observe a tabela 4.1 que exibe a codificação de duas possíveis teorias baseadas em bc_1 e bc_2.

Tabela 4.1: Representação de Teorias por Strings Binárias

1	**bc_1**	**q(X,Y) :-**	**r(Z,X)**	**w(X,Y)**	**-**	**-**
2	ind.1.bin.1	1	0	1	-	-
3	ind.2.bin.1	1	1	0	-	-
4	ind.1.cls.1	q(X,Y) :-		w(X,Y)	-	-
5	ind.2.cls.1	q(X,Y) :-	r(Z, X)		-	-
6	**bc_2**	**q(X,Y) :-**	**t(X,Z)**	**h(Z)**	**s(X,Y,Z)**	**p(X,Y)**
7	ind.1.bin.2	1	1	0	0	1
8	ind.2.bin.2	1	1	1	0	0
9	ind.1.cls.2	q(X,Y) :-	t(X,Z)			p(X,Y)
10	ind.2.cls.2	q(X,Y) :-	t(X,Z)	h(Z)		

As linhas 1 e 6 da tabela ilustram as *bottom clauses* construídas. Como afirmado, as teorias geradas pelo sistema serão constituídas por um subconjunto dos literais destas *bottom clauses*. Assim, para se construir o primeiro indivíduo da teoria, deve-se amostrar a primeira rede Bayesiana (que capta a dependência entre os literais de bc_1) para gerar uma *string* binária, representada por ind.1.bin1 (indivíduo 1 string binária 1) que representa os literais da bc_1 que farão parte da primeira cláusula do indivíduo 1. Deve-se amostrar também a segunda rede Bayesiana (que capta a dependência entre os literais de bc_2) para gerar uma *string* binária representada por ind.1.bin.2 que representa os literais da bc_2 que farão parte da segunda cláusula do indivíduo 1. Após estas amostragens, têm-se duas *strings* binárias (linha 2 e 7) que juntas codificam, de forma binária, a teoria do primeiro indivíduo da população. Para se saber qual cláusula cada

uma das *strings* binárias representa, basta realizar um mapeamento entre as suas posições e a *bottom clause* que deu origem a rede Bayesiana usada para amostrar tal *string*. Tal mapeamento é idêntico ao realizado em (MUGGLETON, TAMADDONI-NEZHAD, 2007): caso a *i*-ésima posição da *string* amostrada pela *j*-ésima rede Bayesiana (construída sobre a *bottom clause j*), seja igual a 1, o *i*-ésimo literal da *bottom clause j* figurará na cláusula; caso este valor seja 0, tal literal não figurará. No exemplo considerado, as *strings* binárias que codificam a teoria representada pelo primeiro individuo são (101) para bc_1, e (11001) para bc_2. Assim, como pode ser visto nas linhas 4 e 9 da tabela 4.1, a teoria representada pelo primeiro indivíduo é dada pelas seguintes cláusulas:

1. q(X,Y) :- w(X,Y).
2. q(X,Y) :- t(X,Z), p(X,Y).

Este mesmo processo deve ser feito para se obter a teoria representada pelo segundo indivíduo que, neste exemplo, é dada por (linhas 5 e 10 da tabela 4.1):

1. q(X,Y) :- r(Z,X)
2. q(X,Y) :- t(X,Z), h(Z).

Pelo exemplo exposto, fica claro que a construção de teorias se faz pela amostragem de todas as redes Bayesianas que juntas compõem o modelo probabilístico dos sistemas-AED. Obviamente, a amostragem de cada rede Bayesiana gera uma cláusula cujos literais são um subconjunto dos literais da *bottom clause* sobre a qual a rede amostrada foi construída.

4.2.3 Sistemas AEDs: A Amostragem do Modelo Probabilístico

Como visto, os sistemas-AED utilizam um modelo probabilístico formado por um conjunto de redes Bayesianas construídas sobre *bottom clauses* distintas para realizar a busca por teorias. Dessa forma, para que as teorias sejam geradas, é necessário que tais redes sejam amostradas. Além disso, para que elas gerem teorias cada vez melhores, se faz necessária a atualização de suas TPCs. Esta seção discute de que maneira as redes são amostradas enquanto que a seção 4.2.4 discute como tais redes são atualizadas.

Tal como apresentado no capítulo anterior, o sistema OS utiliza a fórmula 4.1, replicada a seguir, que calcula a probabilidade de um nó *N* assumir o valor T (verdadeiro); tais probabilidades são então usadas para amostrar cada nó da rede Bayesiana com objetivo de se gerar uma cláusula.

$$P\big(N = T \big| \Pi(N)\big) = \prod_{j=1}^{M} P\left(N = T \middle| S_j(N)\right) = \prod_{j=1}^{M}\left(1 - \prod_{R \in S_j(N)} P(N = F|R)\right) \quad (4.1)$$

Na referida fórmula, *N* é um dos nós da rede, $\Pi(N)$ são os pais de *N*, S_j é um subconjunto de $\Pi(N)$ que satisfaz (todos os pais de *N* que possuem a variável *j* como sendo de saída) o argumento de entrada *j*, e *M* é o número de variáveis de entrada presentes no nó *N*.

Esta abordagem realiza um produtório de *M* probabilidades condicionais sobre um literal da *bottom clause* representado como um nó da rede Bayesiana. Tais probabilidades condicionais são modeladas usando o *noisy-OR*. Lembre-se que na proposta de tais autores, foi atribuído o valor 1 à *P(N = F | R = F)*, o que faz com que as cláusulas geradas respeitem estritamente a declaração dos modos.

Em nossos sistemas, também se utiliza o *noisy-OR*. Entretanto, a maneira adotada para se calcular a probabilidade de um nó *N* ser T (verdadeiro) é diferente da forma usada pelo sistema OS. Esta diferença foi motivada pelo fato de que as probabilidades iniciais das TPCs das redes Bayesianas, nas versões atuais dos sistemas propostos, devem ser genericamente inicializadas (pelo usuário[2]) para que estas sejam refinadas (atualizadas) sucessivamente de acordo com a evolução da busca. Sendo assim, necessita-se de um método de amostragem da rede que, a partir do fornecimento das probabilidades iniciais, proveja certa intuição ao usuário no sentido de que a especificação de uma determinada probabilidade ditaria, de forma aproximada, quantos literais figurarão na cláusula após a amostragem de uma rede Bayesiana. Com este objetivo em mente adotou-se a fórmula 4.2 que nada mais é que a aplicação direta do *noisy-OR* para calcular a probabilidade de que um nó *N* assuma o valor T (note que, caso não utilizássemos o *noisy-OR*, a aplicação da fórmula 4.2 não seria necessária, uma vez que todas as probabilidades estariam armazenadas nas TPCs):

[2] Ver-se-á mais adiante que tais probabilidades iniciais das TPCs, em todos os experimentos realizados, foram genericamente inicializadas de uma forma bastante simples.

$$p\big(N = T \big| \Pi\,(N)\big) = 1 - \prod_{j \in S(\Pi(N))} p(N = F|\pi_j) \qquad (4.2)$$

em que $\Pi\,(N)$ são os pais de *N*, S($\Pi\,(N)$) é o subconjunto dos pais de *N* que assumem o valor T (verdadeiro), e p($N = F|\ \pi_j$) é a probabilidade de *N* assumir o valor F (falso) dado que o *j*-ésimo pai de *N* é T (verdadeiro) e todos os outros pais de *N* assumem o valor F (falso) - estas probabilidades estão armazenadas na TPCs.

É importante lembrar que a fórmula (4.2) é utilizada pela técnica *Forward Sampling* (revisada no capítulo 3, seção 3.4.1.1) para amostrar as redes Bayesianas. Adicionalmente, algumas considerações importantes devem ser realizadas em relação à fórmula adotada, são elas:

1) Diferentemente da fórmula (4.1) adotada pelo OS, a fórmula (4.2) não respeita a restrição dos modos, i.e., ao se amostrar uma rede Bayesiana usando (4.2), pode-se gerar um cláusula cujas restrições dos modos não sejam respeitadas. Contudo, como experimentos realizados demonstram que melhores resultados são obtidos quando tais restrições são respeitadas, decidiu-se fazer com que os sistemas-AED, verificassem, para a amostragem de cada nó da rede, se o literal que representa este nó pode figurar na cláusula. Em outras palavras, os sistemas-AED verificam, após amostrarem um nó da rede, se o literal que representa o nó amostrado pode figurar na cláusula devido às restrições impostas pelos modos[3]. Assim, apesar dos sistemas-AED apresentarem um componente externo a fórmula (4.2) para restringir a geração de cláusulas que respeitem os modos, tanto o OS quanto os sistemas-AED respeitam de forma idêntica as restrições impostas para geração de cláusulas. Para ilustrar o que foi explicado acima, considere o exemplo 4.2.

   Exemplo 4.2: Considere uma *bottom clause* bc_1, dada por h :- l_1, l_2, l_3, l_4, em que h é a cabeça de bc_1, e l_1, l_2, l_3, e l_4 são os literais do corpo de bc_1. Considere que o literal l_4 possua três variáveis de entrada, a saber, X, Y, Z. Assuma que os três literais l_1, l_2, e l_3 são os pais de l_4, uma vez que cada um deles possui exatamente uma variável de l_4 como sendo de saída. Tais literais são representados por: l_1 = p(-X, H), l_2 = q(-Y, K), l_3 = r(-Z, W), e l_4 = s(+X, +Y,

[3] Tal verificação é bastante rápida de ser feita, pois devem-se considerar apenas as variáveis de entrada do literal que foi amostrado e as variáveis de saída de seus pais para saber se as restrições impostas pelos modos são respeitadas.

+Z, A). A figura 4.1 apresenta a rede Bayesiana que representa a dependência entre estes literais.

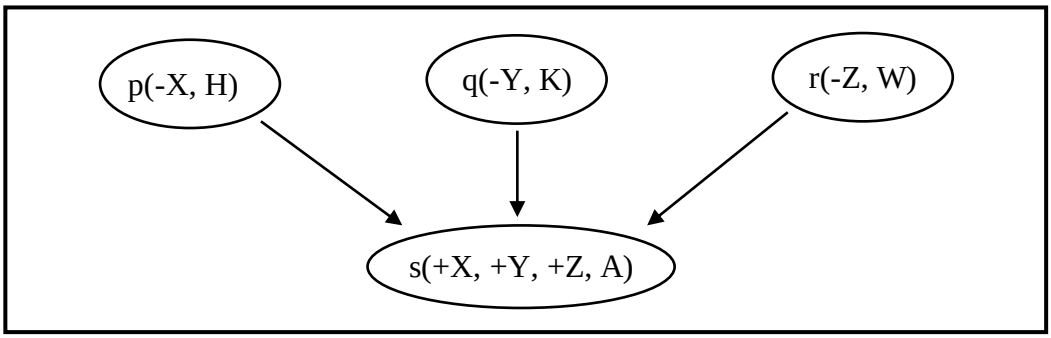

Figura 4.1: Dependências entre literais representadas por uma rede Bayesiana

Assuma que após a amostragem da rede (da figura 4.1) pelo *Forward Sampling*, os literais l_1 = p(-X, H), l_3 = r(-Z, Y), e l_4 = s(+X, +Y, +Z, A) devam figurar na cláusula gerada. Contudo, verifica-se que l_4 não pode figurar. Isto porque ele possui uma variável de entrada +Y, que depende da variável de saída -Y presente em l_2, mas l_2 não figurou na cláusula. Portanto, os únicos literais a figurarem na cláusula serão l_1 = p(-X, H), l_3 = r(-Z, Y), i.e., cláusula representada por *h* :- p(-X, H), r(-Z, Y).

2) A fórmula (4.2) utiliza diretamente o *noisy-OR*, sem se preocupar com a quantidade de variáveis de entrada do nó. Tal fórmula pode ser considerada uma simplificação da fórmula (4.1), contudo, como pode ser visto no exemplo 4.3, tal simplificação fornece uma maior intuição ao usuário que poderá fornecer certo valor de probabilidade almejando controlar, de certa forma, o número de literais em uma cláusula.

Exemplo 4.3: Considere os literais e a rede Bayesiana definidos no exemplo 4.2. Como afirmado anteriormente, algumas probabilidades devem ser inicializadas pelo usuário, sendo que, uma vez que se utiliza o *noisy-OR*, as outras probabilidades poderão ser facilmente calculadas a partir das probabilidades inicialmente fornecidas. Assim, uma vez que os l_1, l_2, e l_3 não têm pais, existem apenas probabilidades marginais associadas a estes nós. Portanto, considere que as seguintes probabilidades marginais, $P(l_1 = T) = P(l_2 = T) = P(l_3 = T) = 0{,}5$ foram fornecidas pelo usuário para l_1, l_2, e l_3. Já a tabela 4.2 exibe as probabilidades condicionais para o nó da rede Bayesiana que representa o literal l_4. Destacam-se, em tom de cinza nesta tabela, as probabilidades inicializadas pelo usuário para l_4. Note que, com exceção da linha 8, todas as outras

probabilidades que foram fornecidas apresentam o mesmo valor, ou seja, todas elas foram inicializadas de forma bem geral (como será apresentado posteriormente, todos os experimentos foram realizados usando-se formas simples de inicializações como esta). A probabilidade da linha 8 é igual a 1 uma vez que um literal não deve figurar quando nenhum de seus pais figuram. As probabilidades não especificadas foram calculadas utilizando-se o *noisy-OR*.

Tabela 4.2: TPC para o nó da Rede Bayesiana representando o literal l_4

-	l_1	l_2	l_3	$P(l_4 = F \mid l_1, l_2, l_3)$
1	T	T	T	$0,5 \times 0,5 \times 0,5 = 0,125$
2	T	T	F	$0,5 \times 0,5 = 0,25$
3	T	F	T	$0,5 \times 0,5 = 0,25$
4	T	F	F	0,5
5	F	T	T	$0,5 \times 0,5 = 0,25$
6	F	T	F	0,5
7	F	F	T	0,5
8	F	F	F	1

Note que pela fórmula proposta, $P(l_4 = T \mid l_1 = T, l_2 = T, l_3 = T) = 0.875 = (1 - 0,125)$ é maior que: $P(l_4 = T \mid l_1 = T, l_2 = F, l_3 = F) = 0,5 = (1 - 0,5)$, *e*, $P(l_4 = T \mid l_1 = F, l_2 = T, l_3 = F) = 0,5 = (1 - 0,5)$, e, $P(l_4 = T \mid l_1 = F, l_2 = F, l_3 = T) = 0,5 = (1 - 0,5)$, o que é bastante intuitivo, uma vez que estas três últimas probabilidades refletem o fato de o nó figurar dado que um (e apenas um) de seus pais figurou. Assim, quanto mais pais de um nó figurarem, maior será a probabilidade de este nó figurar. Esta intuição é facilmente captada pela fórmula proposta e é de suma importância, uma vez que é de responsabilidade do usuário inicializar genericamente as probabilidades em destaque para os sistemas-EDA.

Antes de finalizar o exemplo 4.3, é importante apresentar o motivo pelo qual a fórmula (4.1) não seria intuitiva caso fosse usada nos sistemas-AED, uma vez que se faz necessária a inicialização, por parte do usuário, das probabilidades iniciais das redes Bayesianas. Considere as probabilidades destacadas em tom de cinza na tabela 4.2 e suponha que se deseja calcular $P(l_4 = T \mid l_1 = T, l_2 = T, l_3 = T)$. Pela fórmula (3.17), $P(l_4 = T \mid l_1 = T, l_2 = T, l_3 = T) = (1 - 0,5) \times (1 - 0,5) \times (1 - 0,5) = 0,125$, que é menor que $P(l_4 = T \mid l_1 = T, l_2 = F, l_3 = F) = 0,5$, e, $P(l_4 = T \mid l_1 = F, l_2 = T, l_3 = F) = 0,5$, e $P(l_4 = T \mid l_1 = F, l_2 = F, l_3 = T) = 0,5$. Este fato se deve a existência de um produtório que varia sobre todas

as variáveis de entrada do literal, o que acaba por reduzir a probabilidade calculada.

Nos sistemas-AED, caso seja atribuído o valor 0,5 à $P(l_4 = F| l_1 = T, l_2 = F, l_3 = F)$ e, por consequência, $P(l_4 = T| l_1 = T, l_2 = F, l_3 = F) = 0,5$, sabe-se que caso apenas l_1 figure, a probabilidade de l_4 figurar é de 0,5, mas que caso outros pais de l_4 figurem, a probabilidade de l_4 figurar tende a aumentar. Este fato ajuda o usuário a inicializar, mesmo que genericamente, as probabilidades iniciais das redes Bayesianas de acordo um número aproximado de literais desejados na cláusula.

Para os sistemas-AED, as probabilidades a serem inicializadas se denominam *P_1, P_2, e P_3*. *P_1* é a probabilidade marginal de que um nó que não possui pais assuma o valor T (tem-se uma *P_1* para cada nó da rede que não possui pai). *P_2* é a probabilidade condicional de que um nó assuma o valor T dado que um de seus pais assuma o valor T e todos os outros assumam o valor F. Para um dado nó da rede, tem-se um *P_2* para cada um de seus pais. *P_3* é a probabilidade condicional de que um nó assuma o valor T dado que todos os seus pais assumam o valor F[4]. Note que, a partir de *P_2*, todas as outras possíveis probabilidades (para todas as possíveis configurações dos pais de um nó) podem ser calculadas usando-se o *noisy-OR*. Mais uma vez é importante frisar que para a realização de todos dos experimentos deste trabalho, *P_1*, *P_2*, e *P_3* foram inicializadas de maneira simples e bastante genérica, sendo que se verificou que o valor inicial de tais probabilidades, desde que não muito próximos de 0 ou de 1, não determinam o comportamento dos sistemas, uma vez que as probabilidades são refinadas durante o aprendizado das teorias.

Antes de discutir como as TPCs das redes Bayesianas são atualizadas, é importante frisar que a fórmula (4.2) é usada, através da técnica *Forward Sampling*, sobre cada um dos nós de uma rede Bayesiana. Dessa forma, gera-se uma *string* binária que representa, como explicado na seção anterior, um subconjunto dos literais da *bottom clause* sobre a qual a rede Bayesiana amostrada foi construída.

[4] Esta probabilidade foi inicializada como *0* para todas as redes em todos os experimentos realizados, uma vez que não faz sentido um literal figurar na cláusula sendo que nenhum outro literal da qual ele dependa figurou.

4.2.4 Sistemas AEDs: A Atualização do Modelo Probabilístico

Uma vez que os Sistemas-AED trabalham com uma população de teorias, a cada ciclo de execução é selecionada a melhor teoria da população, e então esta teoria, mais precisamente o conjunto de vetores binários que a representa, é usado para atualizar as redes Bayesianas que compõem o modelo probabilístico. Como afirmado, o modo de atualização do modelo é baseado na forma de atualização do sistema PBIL. Como já apresentado, tal sistema atualiza o seu modelo probabilístico de acordo com a regra (apresentada na seção 3.4.1.2), replicada a seguir:

$$p_i := p_i + \lambda \times (x_i - p_i) \tag{4.3}$$

onde p_i é a probabilidade de figurar o valor 1 na posição i, $\lambda \in (0, 1)$ é a taxa de aprendizado, e x_i é o valor da i-ésima posição da *string* binária que representa o melhor indivíduo da geração atual.

Como revisado, o sistema PBIL assume que as variáveis do problema são independentes, o que não ocorre no sistemas-AED, uma vez que admitem-se que os literais das cláusulas guardam dependências captadas por redes Bayesianas. Além desta diferença, é importante lembrar que os sistemas-AED utilizam o *noisy-OR* para modelar suas probabilidades, assim, as únicas probabilidades que podem ser atualizadas são as que de fato são armazenadas na memória. De forma geral, como visto, pode ser dito que a regra de atualização do sistema PBIL faz com que as probabilidades movam-se em direção a gerar o melhor indivíduo da população. Em nossos sistemas-AED, a intuição descrita acima funciona exatamente da mesma maneira, contudo, é importante notar que nossos sistemas trabalham com dois tipos de probabilidades, a saber: probabilidades marginais (para nós que não possuem pais) e probabilidades condicionais (para nós que possuem um ou mais pais). Dessa forma, devem ser considerados os dois casos a seguir para a atualização do modelo probabilístico.

- **Caso 1 (atualizando probabilidades de um nó *n* que não possui pais):** neste caso, aplica-se diretamente a regra (3.16), sendo p_i a probabilidade marginal associada ao nó que terá sua probabilidade atualizada.
- **Caso 2 (atualizando a probabilidade de um nó *n* que possui pais):** neste caso, primeiro seleciona-se o subconjunto de pais *sp* de *n* que possuem valor igual a 1 (ou verdadeiro) na string binária que representa a melhor solução da população atual. Em seguida, devem-se fazer |*sp*| atualizações na TPC de *n*, onde cada uma atualiza uma linha da TPC que corresponde à probabilidade P_2 (veja penúltimo

parágrafo da seção 4.2.3) associada a cada um dos pais de *n* que assumiu o valor 1 (ou verdadeiro) na melhor solução da população atual. Estas atualizações são realizadas pela aplicação direta da regra (3.16), sendo que p_i representa uma linha da TPC do nó *n*.

A atualização de uma rede Bayesiana é feita percorrendo-se todos os seus nós. Para cada nó que não possui pai, aplica-se o caso 1, enquanto que para os nós que possuem pais, é aplicado o caso 2. Ambos os passos de atualização podem ser mais bem compreendidos pelo exemplo 4.4.

Exemplo 4.4: Considere a descrição do exemplo 4.2, repetida aqui por conveniência, i.e., considere uma *bottom clause* bc_1, dada por h :- l_1, l_2, l_3, l_4, em que h é a cabeça de bc_1, e l_1, l_2, l_3, e l_4 são os literais do corpo de bc_1. Considere que o literal l_4 possua três variáveis de entrada, a saber, X, Y, Z. Assuma que os três literais l_1, l_2, e l_3 são os pais de l_4, uma vez que cada um deles possui exatamente uma variável de l_4 como sendo de saída. Tais literais são representados por: l_1 = p(-X, H), l_2 = q(-Y, K), l_3 = r(-Z, W), e l_4 = s(+X, +Y, +Z, A). A figura 4.1 apresenta a rede Bayesiana que representa a dependência entre estes literais.

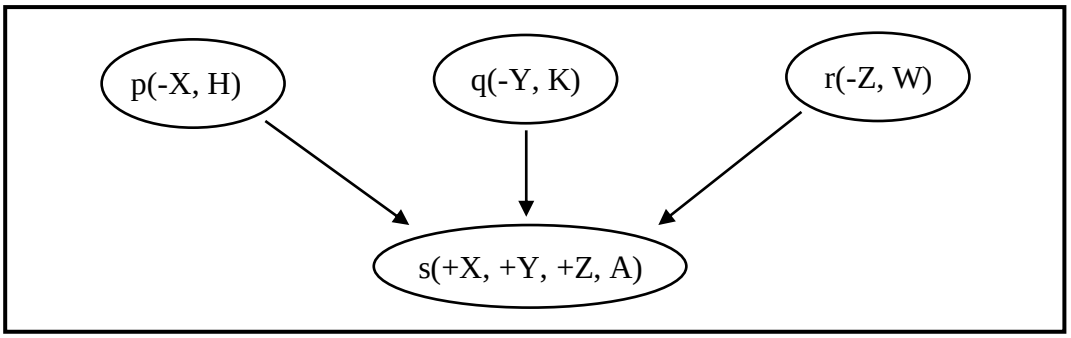

Figura 4.1: Dependências entre os literais representadas por uma rede Bayesiana

Considere também as probabilidades associadas a esta rede que foram descritas no exemplo 4.3, i.e., as probabilidades marginais $P(l_1 = T) = P(l_2 = T) = P(l_3 = T) = 0{,}5$, e as probabilidades condicionais para l_4 = s(+X, +Y, +Z, A) (replicadas) na tabela 4.2 a seguir (lembre-se que foram destacadas em tom de cinza as probabilidades de fato armazenadas na memória).

Considere que a teoria é composta por uma única cláusula. Adicionalmente, assuma que a taxa de atualização $\lambda = 0.5$, e que a *string* binária que representa o melhor indivíduo da população na geração atual é dada por {0, 1, 1, 1}. Esta *string* codifica a cláusula h :- q(-Y, K), r(-Z, W), s(+X, +Y, +Z, A) - apenas o literal l_1 não aparece no corpo da cláusula, uma vez que o valor da primeira posição da *string* é 0.

Tabela 4.2: TPC para o nó da Rede Bayesiana representando o literal l_4

-	l_1	l_2	l_3	$P(l_4 = F \mid l_1, l_2, l_3)$
1	T	T	T	$0,5 \times 0,5 \times 0,5 = 0,125$
2	T	T	F	$0,5 \times 0,5 = 0,25$
3	T	F	T	$0,5 \times 0,5 = 0,25$
4	T	F	F	0,5
5	F	T	T	$0,5 \times 0,5 = 0,25$
6	F	T	F	0,5
7	F	F	T	0,5
8	F	F	F	1

Dando início à atualização das probabilidades da rede Bayesiana, como os nós que representam os literais l_1, l_2, e l_3 não possuem pais, a aplicação da regra de atualização do sistema PBIL é direta. Assim (note que se atualizam as probabilidades do literal figurar em vez de não figurar, e que para isso se usa o vetor {0, 1, 1, 1} que representa o melhor indivíduo (teoria) da população atual):

1) $P(l_1 = T) := 0,5 + 0,5 \times (0 - 0,5) = 0,25$, assim, $P(l_1 = F) = 0,75$ (1 - 0,25)
2) $P(l_2 = T) := 0,5 + 0,5 \times (1 - 0,5) = 0,75$, assim, $P(l_2 = F) = 0,25$ (1 - 0,75)
3) $P(l_3 = T) := 0,5 + 0,5 \times (1 - 0,5) = 0,75$, assim, $P(l_1 = F) = 0,25$ (1 - 0,75)

As novas probabilidades refletem o aparecimento ou o não aparecimento do literal na cláusula. Assim, como o literal l_1 não apareceu no melhor indivíduo (representado pelo valor 0 na primeira posição da *string* que representa o melhor indivíduo), a probabilidade de não aparecimento deste literal foi então atualizada de 0,5 para 0,75. O mesmo raciocínio pode ser feito para o literal l_2, sendo que ele aparece no melhor indivíduo, e, desta forma, a probabilidade de seu não aparecimento foi atualizada de 0,5 para 0,25, ou seja, aumentou-se a probabilidade deste literal figurar em futuras cláusulas. O mesmo raciocínio feito para l_2 deve ser realizado para l_3.

Para a atualização das probabilidades do literal l_4, deve-se, como descrito no caso 2, verificar quais os pais do literal l_4 figuraram no melhor indivíduo. Isto pode ser verificado na *string* binária, ou seja, verificam-se quais são os valores das posições da *string* relativas aos literais que são pais de l_4. Como as posições dos pais de l_4 possuem os valores 0, 1, e 1, sabe-se que l_1 não figurou na cláusula enquanto que l_2 e l_3 figuraram. Dessa forma, de acordo com o caso 2, as probabilidades $P(l_4 = T \mid l_1 = F, l_2 = T, l_3 = F)$, e $P(l_4 = F \mid l_1 = F, l_2 = F, l_3 = T)$ serão atualizadas usando a regra do PBIL. Assim:

4) $P(l_4 = T| l_1 = F, l_2 = T, l_3 = F)$:= $0,5 + 0,5 \times (1 - 0,5) = 0,75$, assim $P(l_4 = F| l_1 = F, l_2 = T, l_3 = F) = 0,25$

5) $P(l_4 = T| l_1 = F, l_2 = F, l_3 = T)$:= $0,5 + 0,5 \times (1 - 0,5) = 0,75$, assim $P(l_4 = F| l_1 = F, l_2 = F, l_3 = T) = 0,25$.

Uma vez que os literais l_2, l_3, e l_4 figuraram na melhor cláusula, a TPC da rede Bayesiana será atualizada no sentido de que a cláusula que possua os literais l_2, l_3 e l_4 seja construída na próxima geração com maior probabilidade. Viu-se, anteriormente, que as probabilidades de l_2 e l_3 figurarem em um cláusula foram aumentadas, enquanto que a probabilidade l_1 figurar foi diminuída (atualizações 1, 2 e 3). Veja a tabela 4.4, que ilustra a TPC atualizada do nó que representa o literal l_4.

Tabela 4.3: TPC atualizada para o nó da Rede Bayesiana que representa o literal l_4

-	l_1	l_2	l_3	$P(l_4 = F\| l_1, l_2, l_3)$
1	T	T	T	$0,5 \times 0,25 \times 0,25 = 0,03125$
2	T	T	F	$0,5 \times 0.25 = 0,125$
3	T	F	T	$0,5 \times 0.25 = 0,125$
4	T	F	F	0,5
5	F	T	T	$0,25 \times 0,25 = 0,0625$
6	F	T	F	0,25
7	F	F	T	0,25
8	F	F	F	1

Repare na linha 5 que exibe a probabilidade do literal l_4 não figurar em uma cláusula dado que o literal l_1 não figurou e l_2 e l_3 figuraram. Em relação à tabela anterior, a referida probabilidade agora é 0,0625, em vez de 0,25, o que representa um aumento na probabilidade de l_4 figurar em uma cláusula dado que l_2 e l_3 já figuram nela. Repare na tabela 4.4, que a atualização de duas probabilidades alterou indiretamente quatro probabilidades.

Com a regra de atualização apresentada, fica claro que a rede Bayesiana, gerará, a cada iteração, indivíduos mais próximos ao melhor indivíduo da geração anterior, mas que devido à intrínseca natureza probabilística de tais redes, existirá uma grande diversidade de indivíduos gerados, e, por isso, ocorrerá uma ampla exploração do espaço de busca. É importante perceber que o modelo probabilístico dos sistemas-AED são compostos por várias redes Bayesianas e, desta forma, todas as redes são atualizadas em direção ao melhor indivíduo, sendo que cada rede é individualmente atualizada de acordo com a cláusula que ela gerou e que compõe o melhor indivíduo da população. Este fato pode ser mais bem ilustrado pelo algoritmo 4.4.

Algoritmo 4.4 – Atualização do Modelo Probabilístico

Entrada:
(modProb): modelo probabilístico constituído por redes Bayesianas;
(**melhorInd**): melhor indivíduo (teoria) da população atual; (**λ**): taxa de atualização do modelo probabilístico;
Variáveis Locais:
(rb_i): i-ésima rede Bayesiana em modProb ($1 \leq i \leq |modProb|$);
(C_i): i-ésima cláusula ($1 \leq i \leq |modProb|$) em melhorInd;
Saída:
modProb atualizado;

Começo-Atualização-Modelo-Probabilístico:
01- Para i = 1 até $|modProb|$ faça:
02- rb_i := i-ésima rede Bayesiana de *modProb*;
03- C_i := *i-ésima* cláusula de *melhorInd*;
04- Atualize rb_i usando Binário(C_i) e λ;
05- fim-para;
06- Retorne *modProb*;
Fim-Atualização-Modelo-Probabilístico.

O algoritmo 4.4 recebe como entrada *modProb* (que representa um conjunto de redes Bayesianas), *melhorInd* (que representa a melhor teoria da população atual), e λ (que representa a taxa de atualização da regra do sistema PBIL). Como saída, o algoritmo retorna *modProb* com todas as TPCs das redes Bayesianas atualizadas no sentido de gerar com maior probabilidade o *melhorInd*. Dessa forma, o passo 1 é responsável por "passar" por todas as redes Bayesianas em *modProb*. O passo 2 seleciona a *i*-ésima rede Bayesiana em *modProb* enquanto que o passo 3 seleciona a *i*-ésima cláusula do melhor indivíduo (que foi gerada pela amostragem da *i*-ésima rede Bayesiana rb_i). O passo 4 atualiza a *i*-ésima rede Bayesiana utilizando a *string* binária que foi mapeada na *i*-ésima cláusula, e a taxa de atualização λ. Tal atualização é realizada conforme explicado nesta seção. Por fim, o passo 6 retorna o *modProb* com todas TPCs das redes Bayesianas atualizadas no sentido de gerar com maior probabilidade o *melhorInd*.

Repare que em relação ao BOA, ou até mesmo em relação ao sistema OS, que utilizam, de forma geral, uma contagem sobre as melhores soluções geradas (na geração atual) para atualização das redes Bayesianas, os sistemas-AED usam apenas o melhor indivíduo para atualizar suas redes, sendo, dessa forma, sistemas que usam uma atualização mais simples considerando o tempo computacional.

4.2.5 O Ciclo de Execução do HAED-ILP

Antes de apresentar o ciclo de execução do HAED-ILP, é importante destacar que as discussões anteriores sobre a codificação do espaço de busca, e sobre o modelo probabilístico, são também válidas para o HAED-ILP. Em outras palavras, o que diferencia o HAED-ILP dos outros dois sistemas propostos é o seu novo ciclo de busca.

O HAED-ILP é uma extensão do AED-ILP que usa um ciclo de evolução completamente novo para explorar o espaço de busca. Esta nova forma de evoluir teorias é uma abordagem híbrida que considera tanto as potencialidades do método de cobertura quanto às vantagens em se evoluir teorias em vez de cláusulas isoladas. Para se desenvolver o HAED-ILP, levaram-se em consideração duas motivações fundamentais, a saber:

1) Nossos experimentos, comparando o AED-ILP e o RAED-ILP ao Aleph, confirmaram o argumento em (BRATKO, 1999) de que o uso do método de cobertura para construir teorias pode fazer com que elas se tornem desnecessariamente grandes.
2) Apesar do fato da abordagem de cobertura gerar, algumas vezes, teorias desnecessariamente grandes, este método é muito robusto e alcança bons resultados em uma grande quantidade de problemas; não é por acaso que o método de cobertura é usando em alguns dos principais sistemas ILP, tais como o FOIL, o Progol, o QG/GA, o OS, dentre outros.

Baseando-se nestes dois pontos, desenvolveu-se uma nova abordagem híbrida para evoluir teorias. Esta nova abordagem apresenta dois ciclos de evolução distintos, a saber, o ciclo primário e o ciclo secundário. O primeiro ciclo tem o objetivo de evoluir teorias completas, enquanto que o secundário evolui cláusulas isoladas. Tecnicamente falando, é importante notar que, assim como ocorreu com o AED-ILP e RAED-ILP, o HAED-ILP evolui modelos probabilísticos e os amostra para construir suas teorias. Assim, é mais apropriado dizer que o ciclo primário evolui um conjunto de redes Bayesianas que quando amostradas gerarão um conjunto de cláusulas, i.e., uma teoria, enquanto que o ciclo secundário evolui uma única rede Bayesiana que quando amostrada gerará apenas cláusulas isoladas.

O ciclo de evolução primário começa evoluindo uma única rede Bayesiana, contudo, outras redes Bayesianas podem ser adicionas, uma a uma, ao decorrer deste ciclo, às outras redes Bayesianas já presentes e que estão sendo evoluídas neste ciclo.

Assim, a analogia de evoluir teorias é feita no ciclo primário, uma vez que ele inicia com uma única rede Bayesiana (evoluindo primeiramente cláusulas isoladas), mas assim que novas redes são adicionadas, ele passar a evoluir teorias completas.

A analogia com o método de cobertura ocorre no ciclo de evolução secundário, uma vez que é nele que uma nova rede Bayesiana é criada e evoluída sobre todos os exemplos negativos da base de dados, mas apenas sobre o subconjunto de exemplos positivos não cobertos (provados) pela melhor teoria atual criada pela amostragem das redes Bayesianas presentes no ciclo de evolução primário. Após ser evoluída por um determinado número de gerações, esta nova rede (criada na evolução secundária) é adicionada ao conjunto de redes Bayesianas do ciclo de evolução primário. Assim, após a adição uma nova rede Bayesiana ao ciclo primário, este ciclo passa agora a evoluir teorias que têm uma cláusula a mais do tinham antes da adição desta nova rede.

Como pode ser notado, têm-se dois ciclos de evolução que juntos formam o ciclo de evolução completo do sistema HAED-ILP: o ciclo primário evolui um conjunto de redes Bayesianas, enquanto que o ciclo secundário evolui uma única rede que será adicionada às redes evoluídas no ciclo primário. Uma vez que as teorias sendo evoluídas começam com uma única cláusula, em última análise, pode ser dito que novas cláusulas são iterativamente adicionadas às teorias, uma vez que existe apenas uma rede Bayesiana no começo do ciclo primário, mas que, devido à adição de redes Bayesianas (realizada pelo ciclo secundário), novas cláusulas são adicionadas às teorias geradas pelas redes no ciclo primário.

Dada a intuição de como o HAED-ILP funciona, o algoritmo 4.5 apresenta o ciclo de execução do sistema. Para facilitar o entendimento de tal algoritmo, é importante identificar os dois ciclos de evolução do HAED-ILP. Assim, os passos de 1 a 6 determinam o ciclo de evolução primário, enquanto que os passos de 7 a 25 fazem parte do ciclo de evolução secundário. Por razões didáticas, os ciclos serão apresentados separadamente.

Algoritmo 4.5: O Ciclo de execução do sistema HAED-ILP

Entrada:

(E+, E-, BK): o conjunto de exemplos positivos E+, negativos E-, e o *background* knowledge BK; **(p_1, p_2, p_3)**: probabilidades iniciais das TPCs tanto para o ciclo primário quanto para o secundário; **(λ_1, λ_2)**: taxa de aprendizado do sistema PBIL para o ciclo primário e secundário; **($numPop_1$, $numPop_2$, $numGen_1$ $numGen_2$)**: número de indivíduos no ciclo primário e secundário; número de gerações do ciclo primário e secundário; **(numVezesGerSec)**: número máximo de tentativas de se executar o ciclo secundário; **(k)**: numero natural que determina quando o ciclo secundário ocorrerá;

Variáveis Locais:

(BC): uma *bottom clause* construída sobre um exemplo positivo; **(modProb)**: o modelo probabilístico que armazena um conjunto de redes Bayesianas; **(novoModProb)**: rede Bayesiana construída no ciclo secundário; **(subConj[E+])**: subconjunto de exemplos positivos não provados pela melhor teoria atual do ciclo primário; **(popPrimária, popSecundária)**: populações do ciclo primário e secundário; (**novaTeoria**): teoria composta pelos melhores indivíduos atuais do ciclo primário e secundário; **(melhorAptCicloPrim, aptNovaTeoria)**: aptidão da melhor teoria do ciclo primário e aptidão de novaTeoria; **($coun_1$, $cout_2$)**: contadores dos ciclos primário e secundário;

Saída:

uma teoria aprendida pelo HAED-ILP;

```
Começo-HAED-ILP:
01- BC := Crie_bottom_clause(um exemplo aleatório de E+, BK);
02- modProb := Crie_rede_bayesiana(BC);
03- Para( (cont1 := 0) até numGen1) faça:
04-    popPrimária := Gere_população(modProb, numPop1);
05-    melhorAptCicloPrim := Avalie_população(popPrimária, numPop1);
06-    Atualize_modelo_probabilístico(modProb, indivíduo mais apto em popPrimária, λ1);
07-    Se( (count1 % k) == 0 ) então:
08-       count2 := 1;
09-       Faça:
10-          novoModProb := Evolução_secundária(subConj[E+], E-, BK, numPop2, numGen2, λ2);
11-          popSecundária := Gere_população(novoModProb, numPop2);
12-          Avalie_população(popSecundária, numPop2);
13-          novaTeoria := (teoria mais apta em popPrimária) U (cláusula mais apta em
             popSecundária);
14-          aptNovaTeoria := Avalie_população(novaTeoria, 1);
15-          Se(aptNovaTeoria > melhorAptCicloPrim) então:
16-             modProb := modProb U novoModProb;
17-          senão:
18-             novaTeoria := ∅;
19-             cont2 := cont2 + 1;
20-          fim-se;
21-          Se(cont2 == numVezesGerSec) então:
22-             vá para o passo 27;
23-          fim-se;
24-       Enquanto(melhorAptCicloPrim > aptNovaTeoria);
25-    fim-se;
26- fim-para;
27- popPrimária := Gere_população(modProb, numPop1);
28- Avalie_população(popPrimária, numPop1);
29- Retorne o indivíduo mais apto em popPrimária;
Fim-HAED-ILP.
```

O Ciclo de Evolução Primário: como esperado, já que tal ciclo evolui teorias, os seus passos principais são muito similares aos do sistema AED-ILP. Dessa forma, o primeiro passo deste ciclo gera uma *bottom clause BC* usando um exemplo positivo *e* selecionado aleatoriamente, enquanto que o passo 2 constrói uma rede Bayesiana *RB* sobre *BC* e a atribui a variável chamada *modProb* (esta variável irá armazenar um conjunto de redes Bayesianas durante o ciclo de execução do HAED-ILP). O passo 3 controla o ciclo de execução do HAED-ILP. Aqui, é importante notar que, diferentemente do AED-ILP e do RAED-ILP, o HAED-ILP não será necessariamente executado por $numGen_1$ vezes. Este ponto será explicado quando discutir-se o ciclo de evolução secundário, mas, por enquanto, pode-se dizer que o HAED-ILP realizará, no máximo, $numGen_1$ ciclos primários. O passo 4 amostra o *modProb* para gerar uma população de teorias (algoritmo 4.3) que é atribuída à variável *popPrimária*. O passo seguinte avalia (usando uma função de aptidão, acurácia, por exemplo) a *popPrimária* atribuindo a cada teoria um valor de aptidão. Note que, no passo 5, a função que avalia a população retorna a melhor aptidão encontrada na população e a atribui a variável *melhorAptCicloPrim*. O passo 6 atualiza todas as redes Bayesianas em *modProb* usando o melhor indivíduo da atual geração e a taxa de atualização λ_1 (algoritmo 4.4). Note que o ciclo primário de evolução é muito parecido ao processo de evolução usado pelo sistema AED-ILP. De fato, usando apenas o ciclo primário de evolução, o HAED-ILP iria retornar uma teoria composta por uma única cláusula, uma vez que é de responsabilidade do ciclo de evolução secundário adicionar novas redes Bayesianas à *modProb*, e, portanto, adicionar novas cláusulas as teorias evoluídas no ciclo de evolução primário.

O Ciclo de Evolução Secundário: como afirmado, este ciclo tem o objetivo de incorporar novas redes Bayesianas ao *modProb* e assim adicionar novas cláusulas as teorias sob evolução. Assim, o primeiro passo deste ciclo (passo 7) controla o número máximo de redes Bayesianas a serem adicionadas a *modProb*, e, consequentemente, controla o número máximo de cláusulas em uma teoria. Este controle é realizado pelo parâmetro *k* que dita que caso o valor atual da variável $cont_1$ (usado para contar o número de gerações) seja múltiplo de *k*, então a geração secundária deve ocorrer. Assim, pelo uso dos parâmetros de entrada $numGen_1$ e *k*, o usuário pode fornecer um limite superior para o número de cláusulas nas teorias desenvolvidas pelo HAED-ILP. Assim, por exemplo, caso o usuário atribua 150 à $numGen_1$ (as gerações variam de 0 a 149), e 50 à *k*, então as teorias do HAED-ILP possuirão no máximo 3 cláusulas, uma

vez que quando $cont_1$ chegar aos valores 50 e 100, o ciclo secundário será executado e adicionará, cada vez executado, uma nova rede Bayesiana às redes previamente presentes em *modProb*. Note que se fala em um número máximo de redes Bayesianas em *modProb* e não em um número exato delas, uma vez que a o ciclo secundário pode ocorrer, mas não necessariamente adicionará uma nova rede a *modProb*. Para entender como isso pode ocorrer, note que o algoritmo possui outro contador, chamado $cont_2$, que é inicializado com o valor 1 no passo 8. Esta variável, em uma situação específica, é usada para quebrar (sair) tanto o ciclo de evolução primário quanto o secundário. Este processo será explicado mais tarde. Contudo, para entender o motivo pelo qual esta saída forçada pode ocorrer, é importante entender os primeiros passos do ciclo de evolução secundário.

Assim, o passo 10 executa a função *Evolução_secundária/5* que recebe 5 parâmetros de entrada, a saber: o subconjunto de exemplos positivos não cobertos pela melhor teoria atual do ciclo de evolução primário (*subConj[E+]*), os exemplos negativos (*E*-), o conhecimento preliminar (*BK*), o número de indivíduos na população do ciclo secundário ($numPop_2$), e o número de vezes que este ciclo será executado ($numGen_2$).

O procedimento *Evolução_secundária/5* trabalha exatamente da mesma forma como descrito nas primeiras linhas (1 a 6) do ciclo de evolução primário. Assim, ele primeiro cria uma *bottom clause BC* de um exemplo positivo selecionado aleatoriamente em *subConj[E+]*, então gera uma rede Bayesiana *RB* usando *BC*, e realiza $numGen_2$ ciclos de evolução que compreendem: a geração de $numPop_2$ cláusulas[5], a avaliação de todas as $numPop_2$ cláusulas geradas, e a atualização da rede Bayesiana *RB* usando a melhor cláusula gerada atualmente. Após $numGen_2$ ciclos, a *Evolução_secundária/5* retorna a *RB* atualizada e a atribui a variável *novoModProb*. Assim, *novoModProb* armazena a rede Bayesiana *RB* que foi especificamente devolvida para se "comportar" bem no subconjunto de exemplos positivos que foi negligenciado pela melhor teoria atual do ciclo primário. O passo 11 gera uma nova população de $numPop_2$ cláusulas pela amostragem de *novoModProb* $numPop_2$ vezes, e as atribui a uma variável chamada *popSecundária*. O passo 12 avalia todas as cláusulas em *popSecundária* atribuindo a cada uma um valor de aptidão. O passo 13 constrói uma teoria combinando a melhor teoria atualmente encontrada no ciclo primário, com a

[5] Lembre-se que o ciclo secundário evolui cláusulas em vez de teorias.

melhor cláusula encontrada no ciclo secundário. Esta nova teoria é atribuída à variável *novaTeoria*. A linha 14 avalia a *novaTeoria* e atribui sua aptidão à variável *aptNovaTeoria*.

O próximo passo é crucial para o correto funcionamento do HAED-ILP e dita que, caso *aptNovaTeoria*, que armazena a aptidão da nova teoria *(novaTeoria)*, seja melhor que *melhorAptCicloPrim*, que armazena a aptidão da melhor teoria atual do ciclo primário, então a nova rede Bayesiana gerada, armazenada em *novoModProb*, *será* adicionada à *modProb* (passo 16). Assim, a nova rede Bayesiana *RB* é adicionada a *modProb* apenas se ela for promissora no sentido de ajudar o *modProb* na busca por melhores (mais aptas) teorias. Caso *aptNovaTeoria* não seja melhor que *melhorAptCicloPrim*, então o *novoModProb* não parece promissor e, portanto, não é adicionado a *modProb*. Note que o passo 18 reinicializa a teoria. Este passo é necessário uma vez que todo este procedimento será executado novamente (este é o motivo de se atribuir 1 a variável $cont_2$ no passo 8) até encontrar uma rede Bayesiana promissora a ser adicionada em *modProb*. Note que, no passo 24, uma das maneiras de se finalizar o ciclo de evolução secundário é encontrando uma *novaTeoria* (composta pela melhor teoria atual encontrada no ciclo primário, juntamente com a melhor cláusula encontrada em *popSecundária)* que possui aptidão *aptNovaTeoria* melhor que *melhorAptCicloPrim*. Caso isto ocorra, pode-se dizer que o ciclo secundário de evolução logrou sucesso em encontrar uma rede Bayesiana promissora. Contudo, pode ocorrer que o ciclo de evolução secundário nunca encontre uma rede Bayesiana promissora para ser adicionada a *modProb*, i.e., para todas as redes Bayesianas construídas e atualizadas por $numGen_2$ vezes durante a execução de *Evolução_secundária/5*, e para todas as *novaTeoria*(s) construídas no ciclo de evolução secundário, a *aptNovaTeoria* é sempre pior que *melhorAptCicloPrim*. Caso isto ocorra, se faz necessário quebrar tal ciclo. Para fazer isso, a variável $cont_2$ é incrementada em uma unidade, no passo 19, para cada vez que o ciclo secundário falha em produzir uma rede Bayesiana promissora. Caso esta falha ocorra durante *numVezesGerSec* vezes, então todo o processo de evolução (tanto o ciclo primário quanto o secundário) é parado devido à condição estipulada no passo 21. Assim, caso isso ocorra, as linhas 27, 28, e 29, respectivamente, constrói a *popPrimária* pela amostragem de *modProb*, avalia os indivíduos de *popPrimária*, e retorna o melhor indivíduo em *popPrimária* como a teoria aprendida.

Optou-se por parar ambos os ciclos de evolução, caso as tentativas de encontrar uma rede Bayesiana cheguem à *numVezesGerSec*, uma vez que se a teoria gerada pelo ciclo primário de evolução não pôde ser melhorada por várias tentativas de adição de novas cláusulas evoluídas em todos os exemplos negativos, mas exclusivamente nos exemplos positivos não cobertos pela melhor teoria atual do primeiro ciclo, então deve ser o caso de que o sistema já encontrou a melhor teoria que poderia ser descoberta para esta execução em particular. Assim, neste caso, o sistema deve retornar a melhor teoria atual gerada pela amostragem de *modProb*. Isto, como explicado, é realizado pelos passos 27, 28, e 29.

Note que, neste caso, a teoria final produzida pelo sistema não irá possuir o número máximo de cláusulas estipulado pelos parâmetros $numGen_1$ e *k*, uma vez que todo o ciclo de execução do sistema foi parado uma vez que ele foi incapaz de encontrar uma rede Bayesiana promissora em *numVezesGerSec*. Em contrapartida, caso o sistema "sempre" encontre uma rede Bayesiana promissora em *numVezesGerSec* tentativas, então o número final de cláusulas na teoria pode ser calculado usando-se a fórmula (4.2). Tal fórmula soma o número de múltiplos de *k* em ($numGen_1$ - *1*) (lembre-se que o número da geração varia de 0 a $numGen_1$ - *1*) a 1, que representa a primeira cláusula na teoria, uma vez que o sistema começa com o *modProb* contendo uma única rede Bayesiana. Assim:

$$numCláusulasTeoria = piso\left(\frac{numGen_1 - 1}{k}\right) + 1 \qquad (4.2)$$

onde *numCláusulasTeoria* é o número de cláusulas na teoria final; *piso(X)* converte um número real *X* no maior número inteiro menor ou igual a *X*; $numGen_1$ é o número máximo de vezes que o ciclo primário deve ser executado, e *k* é um valor inteiro que determina quando o ciclo secundário deve ocorrer.

Como apresentado, o ciclo de execução do HAED-ILP pode ser resumido como segue: o ciclo primário tem o objetivo de evoluir teorias, enquanto que o ciclo secundário, cada vez em que é executado, tenta encontrar uma rede Bayesiana promissora para ser adicionada às redes Bayesianas do ciclo primário. Para decidir se uma rede Bayesiana *RB* é promissora ou não, simplesmente constrói-se uma teoria combinando a melhor teoria atual do ciclo primário com a melhor cláusula atual do ciclo secundário. Caso a aptidão desta nova teoria seja melhor que a aptidão da melhor teoria atual do ciclo primário, então *RB* é considerada promissora e é, portanto, adicionada ao conjunto de redes Bayesianas do ciclo primário. Uma vez que o objetivo

da evolução secundária é encontrar uma rede Bayesiana promissora, este é ciclo é parado assim que tal rede é encontrada. Contudo, existe o caso em que uma rede Bayesiana promissora não pode ser encontrada, assim, a busca por uma boa rede Bayesiana é realizada por certa quantidade de iterações dada pelo parâmetro *numVezesGerSec*. Dessa forma, caso a evolução secundária não encontre uma rede promissora após *numVezesGerSec* tentativas, então toda a busca realizada pelo HAED-ILP é parada e o sistema retorna a melhor teoria atual encontrada amostrando-se, uma última vez, o conjunto de redes Bayesianas da em *modProb*.

Uma análise detalhada da busca realizada pelo HAED-ILP em relação ao AED-ILP e RAED-ILP revela as principais diferenças e as vantagens no novo sistema. Em seguida, discutem-se os principais benefícios trazidos por esta nova abordagem de evoluir teorias.

1) Teoria sem tamanho fixo: não é necessário definir *a priori* o número de cláusulas em uma teoria.

   Para usar os sistemas AED-ILP e RAED-ILP, o usuário deve fornecer o número de cláusulas nas teorias que serão desenvolvidas - este número é controlado pelo parâmetro *numCls*. As teorias de tamanho fixo possuem uma desvantagem substancial: caso o *numCls* seja subestimado, o sistema não irá produzir teorias de qualidade, uma vez que mais cláusulas são necessárias para se obter um bom valor de aptidão. Em contrapartida, caso o *numCls* seja superestimado, o sistema trabalhará com mais cláusulas que necessário, e, assim, gastará mais tempo do que necessário e ainda desenvolverá teorias desnecessariamente grandes.

   No HAED-ILP, o usuário deve fornecer apenas um limite superior de cláusulas na teoria, não um número exato delas. Lembre-se que o ciclo de execução do HAED-ILP (pela adição iterativa de redes Bayesianas ao *modProb*), adiciona, iterativamente, cláusulas às teorias. Dessa forma, em um dado momento, o sistema pode parar de adicionar cláusulas com o objetivo de produzir teorias de qualidade e de tamanho razoável. Assim, a nova abordagem que iterativamente adiciona cláusulas às teorias alivia os problemas citados acima como também retira a difícil tarefa, antes delegada ao usuário do sistema, de suprir um tamanho exato à teoria.

2) Teorias que crescem iterativamente: as teorias do HAED-ILP começam contendo uma única cláusula, mas iterativamente crescem durante o curso da execução.

   Em última análise, o HEAD-ILP, tal como no AED-ILP e RAED-ILP, continua evoluindo teorias em vez de cláusulas isoladas. Contudo, diferentemente dos outros dois sistemas, o HAED-ILP adiciona, de forma iterativa, novas cláusulas às suas teorias pela adição de novas redes Bayesianas ao *modProb*. É importante lembrar que uma nova rede Bayesiana será adicionada ao *modProb* apenas se ela for promissora, i.e., uma rede será adicionada apenas se ela pode melhorar as teorias que já podem ser geradas pelas redes Bayesianas previamente presentes em *modProb*. Dessa forma, o sistema não desperdiça tempo adicionando uma rede Bayesiana que gerará cláusulas que não melhoram a aptidão das teorias ou, que, algumas vezes, podem até torná-las piores.

4.3 Principais Parâmetros dos Sistemas-AED

Os parâmetros dos sistemas controlam sua capacidade de busca e consequentemente a qualidade das teorias encontradas. Conforme afirmado anteriormente, almeja-se manter os sistemas tão simples quanto possível, o que implica que eles detenham um pequeno número de parâmetros, uma vez que, embora um grande número de parâmetros permita certa flexibilidade, eles acabam por fazer com que a configuração do sistema se torne complexa.

A seguir discutem-se todos os atuais parâmetros dos sistemas, sendo que o número de gerações (***numGer*$_1$** para o AED-ILP e RAED-ILP, e ***numGer*$_1$**, ***numGer*$_2$** para o HAED-ILP) e o tamanho da população (***numPop*$_1$** para o AED-ILP e RAED-ILP, e ***numPop*$_1$**, ***numPop*$_2$** para o HAED-ILP), como, de modo geral, são relativos aos AGs, foram discutidos no capítulo anterior. É importante notar que tal discussão apresenta apenas uma visão geral de como os parâmetros afetam o funcionamento do sistema, uma vez que, indiscutivelmente, a influência mais precisa de tais parâmetros depende invariavelmente do problema tratado.

***numCls* (número de cláusulas que compõe a teoria – para o AED-ILP e RAED-ILP):** de forma geral, quanto maior o valor de *numCls*, maior será o tempo computacional gasto, uma vez que a prova de exemplos se tornará mais custosa. Dessa

forma, um número excessivamente grande *numCls* implica em uma lentidão de busca e em longas teorias que provam muitos exemplos positivos e negativos, o que não é interessante. Um valor muito baixo para este parâmetro acarretará, de forma geral, em teorias muito simples que provam poucos exemplos positivos e negativos, o que também não é interessante. Desta forma, *numCls* deve ser determinado para que o sistema evolua teorias simples (com poucas cláusulas), mas ainda assim acuradas.

***P_1, P_2, e P_3* (probabilidades iniciais das redes bayesianas – para todos os sistemas-AED):** com visto, é necessário que tais probabilidades sejam inicializadas de forma genérica, sendo que estas probabilidades serão atualizadas durante as sucessivas gerações e usadas para o cálculo das outras probabilidades das TPCs usando-se o *noisy-OR*. As probabilidades influenciam diretamente o número de literais que figurarão em uma cláusula, assim, altos valores de probabilidades acarretam em um grande número de literais na cláusula, enquanto que probabilidades baixas acarretam no aparecimento de uma pequena quantidade de literais. No primeiro caso, as cláusulas serão mais específicas e necessitarão de um maior tempo computacional para verificação de prova de exemplos quando comparado ao segundo caso, em que as cláusulas são mais gerais e por isso usam menos tempo computacional para este fim.

λ_1 e λ_2 (taxa de aprendizado do modelo probabilístico - λ_1 para AED-ILP e READ-ILP, e λ_1 e λ_2 para o HEAD-ILP): os valores de λ_1 e λ_2 devem pertencer ao intervalo (0, 1), sendo que quando eles assumem valores altos, os sistemas tendem a realizar uma convergência prematura da população para o melhor indivíduo atual. Já com baixos valores, a busca se torna desnecessariamente ampla e não converge para uma "boa" teoria. Assim, devem-se atribuir valores para que a busca não convirja prematuramente, mas que também não seja demasiadamente ampla.

***k* (número natural que determina quando o ciclo de evolução secundário deve ocorrer – para o HAED-ILP):** este parâmetro, juntamente com o $numGer_1$, determina o limite superior de cláusulas nas teorias do HAED-ILP. Assim, para um dado valor fixo de $numGer_1$, um valor baixo de *k* implica em teorias que podem ter muitas cláusulas (problema já abordado na discussão do parâmetro *numCls*) e também faz também com que novas cláusulas sejam adicionadas prematuramente a teoria, uma vez que a evolução primária não teve tempo necessário para desenvolver boas cláusulas. Já um valor muito alto, além de implicar em teorias que podem ter poucas cláusulas (problema já abordado na discussão do parâmetro *numCls*) faz com que a adição de

cláusulas ao ciclo de evolução primário seja feito tardiamente, uma vez que as redes Bayesianas de tal ciclo já podem ter convergido para a geração de algum indivíduo. Caso isto ocorra, a adição de uma nova rede Bayesiana não gerará grande impacto, uma vez que o restante da teoria já estará praticamente formado.

***numVezesGerSec* (número máximo de vezes que o ciclo de evolução secundário tentará encontrar uma rede Bayesiana promissora – para o RAED-ILP):** caso este número seja muito grande, pode-se aumentar em muito o tempo computacional demandado pelo sistema, caso uma rede Bayesiana promissora não possa de fato ser encontrada**.** Com um valor muito baixo, poucas redes Bayesianas serão geradas no ciclo secundário e, portanto, corre-se o risco de não se encontrar uma rede Bayesiana que de fato melhoraria as teorias geradas pelo ciclo de evolução primário.

4.4 Resumo do Capítulo

Este capítulo apresentou uma proposta de aplicação de um AED ao problema de Programação em Lógica Indutiva. Como resultado, criou-se o sistema AED-ILP, bem como suas duas extensões, o RAED-ILP, e o HAED-ILP. O RAED-ILP pode ser entendido como o AED-ILP que usa o algoritmo Reduce para reduzir seu espaço de busca. Já o HAED-ILP pode ser visto como o AED-ILP com seu ciclo de busca completamente remodelado, uma vez que tal sistema apresenta dois "tipos" de evolução; um que trata do desenvolvimento de um conjunto de redes Bayesianas (para gerar teorias) enquanto que o outro lida com o desenvolvimento de uma única rede Bayesiana (para gerar cláusulas isoladas) a ser, caso seja promissora, adicionada ao conjunto de redes do primeiro ciclo.

De forma geral, todos os sistemas-AED utilizam redes Bayesianas para buscar por teorias cujos literais são subconjuntos dos literais de *bottom clauses*, sendo que a cada ciclo, tais redes Bayesianas são atualizadas no sentido de gerar teorias mais parecidas com a melhor teoria do ciclo atual. Com objetivo de fornecer certa velocidade aos sistemas, a regra de atualização do sistema PBIL foi adaptada para trabalhar com redes Bayesianas, não se manteve populações entre as gerações, bem como não foram utilizados operadores de seleção, tais como roleta, *ranking* ou torneio.

Capítulo 5: Resultados e Discussão

Este capítulo apresenta os resultados obtidos com os sistemas-AED. Tais resultados foram comparados aos resultados obtidos pelo sistema Aleph, pelo Progol (e algumas de suas variantes) e pelo AG-ILP, um sistema criado pela substituição do AED no AED-ILP, por um AG "padrão".

5.1 Introdução

Com objetivo de verificar as potencialidades dos sistemas propostos, estes foram comparados ao sistema Aleph, ao sistema Progol (e algumas de suas variantes) e ao sistema denominado AG-ILP, criado pela substituição do AED no AED-ILP por um AG "padrão" [6]. O Aleph foi escolhido para os testes devido a sua grande flexibilidade (MUGGLETON, 2008) e popularidade, como também devido ao fato de que ele é capaz de obter bons resultados em uma grande gama de problemas. O Progol foi selecionado para os testes uma vez que foi neste sistema que o QG/GA foi implementado. Isto resultou em variantes bem interessantes do sistema tais como o Progol-GA, Progol-QG, e o Progol-QG/GA. Já o sistema AG-ILP, foi concebido com objetivo de fornecer uma visão geral sobre a capacidade de busca utilizando um AED frente à busca usando-se um AG "padrão" em ILP. Com objetivo de avaliar os sistemas, foram utilizadas 8 bases de dados do mundo real e 7 bases de dados artificiais do problema de Transição de Fase (BOTTA et al., 2003).

Este capítulo se organiza como segue. A próxima seção apresenta as bases de dados usadas neste trabalho. A seção 5.3 apresenta os experimentos realizados sobre as bases de dados do mundo real, enquanto que a seção 5.4 apresenta os experimentos realizados sobre os problemas de Transição de Fase. Por sua vez, a seção 5.5 apresenta um resumo deste capítulo.

[6] O AG do sistema AG-ILP usa mutação de um ponto, recombinação de um ponto, e torneio como mecanismo de seleção.

5.2 Bases de Dados

Primeiramente, os sistemas foram avaliados em problemas ILP (bem conhecidos) do mundo real, tais como carcinogenesis (Carc) (SRINIVASAN et al., 1997), alzheimers-amine (Amine), alzheimers-toxic (Toxic), alzheimers-acetyl (Acetyl), alzheimers-memory (Memory) (KING et al., 1995), pyrimidines (Pyrim) (MUGGLETON, 2005) (KING et al., 1992), proteins (Prot) (MUGGLETON et al. 1992), e yeast_sensitivity (Yeast) (SPELLMAN et al. 1998) (KADUPITIGE et al., 2009). Todas estas bases podem ser encontradas em (http://www.doc.ic.ac.uk/~jcs06/). Em seguida, no segundo lote de experimentos, foram usadas 7 bases de dados artificiais retiradas do problema de Transição de Fase (BOTTA et al. 2003) (usadas também no QG/GA (MUGGLETON, TAMADDONI-NEZHAD, 2006)) com tamanho dos conceitos variando de 6 a 16. Estas bases de dados podem ser encontradas em (http://www.doc.ic.ac.uk/~shm/progol.html) Todas as bases de dados usadas são brevemente descritas a seguir.

Carcinogenesis: este é o problema de predição carcinogênica em bioensaios de roedores. Estes ensaios são conduzidos pelo Programa Nacional de Toxicologia dos Estados Unidos no projeto de Avaliação de Predição Toxicológica (BRISTOL et al. 1996). A base de dados usada consta de 192 exemplos positivos e 196 exemplos negativos.

Amine, Toxic, Acetyl, e Memory: estes problemas tratam da predição da atividade bioquímica das variantes da Tacrina (uma droga para o tratamento do Alzheimer) (KING et al., 1995). Esta base de dados contém conhecimento preliminar sobre as propriedades físicas e químicas de vários compostos. O objetivo é comparar várias propriedades bioquímicas, tais como: baixa toxicidade (**Toxic**), alta inibição da acetilcolinesterase (**Acetyl**), boa reversão da perda de memória induzida pela escopolamina (**Memory**), e a inibição recaptação da amina (**Amine**). Para cada propriedade, os exemplos positivos e negativos são comparados par a par. Por exemplo, *less_toxic(d_1, d_2)* significa que a droga d_1 é menos tóxica que a d_2. A base de dados Amine possui 343 exemplos positivos e 343 negativos; a base de dados Toxic possui 443 exemplos positivos e 443 exemplos negativos; a base de dados Acetyl possui 663 exemplos positivos e 663 negativos, e a base de dados Memory possui 321 exemplos positivos e 321 exemplos negativos.

Pyrimidines: é um problema de Relações Quantitativas de Estrutura-atividade sobre a inibição da *E. Coli Dihydrofolate Reductase*, pela Pyrimidines, que são antibióticos

que atuam inibindo a *Dihydrolate Reductase*, uma enzima utilizada para a formação do DNA (KING et al., 1992). A base de dados utilizada possui 2361 exemplos positivos e 2361 exemplos negativos.

Proteins: este é um problema de predição de estrutura secundária de proteínas. O objetivo é aprender regras que identificam quando a posição de uma proteína é uma alfa-hélice (MUGGLETON et al., 1992). A base de dados é composta por 1070 exemplos positivos e 970 exemplos negativos.

Yeast-sensitivity: esta base de dados aborda o problema de interação entre genes da levedura *Saccharomyces cerevisiae* (SPELLMAN et al. 1998). Ela é composta de 430 exemplos positivos, e 580 exemplos negativos e possui um conhecimento preliminar enorme, com aproximadamente 170.000 fatos.

Phase Transition: foram usados 7 problemas artificiais com tamanho de conceitos variando de 6 a 16. Estes problemas foram retirados de (BOTTA et al., 2003) e correspondem as bases de dados *m*6.*l*12, *m*7.*l*12, *m*8.*l*12, *m*10.*l*12, *m*11.*l*12, *m*14.*l*12, e *m*16.*l*12. Cada um destes problemas possui 100 exemplos positivos e 100 exemplos negativos.

5.3 Os Problemas do Mundo Real

Metodologia Experimental. Neste lote de experimentos foram utilizados os sistemas Aleph, GA-ILP, AED-ILP, RAED-ILP, e o HAED-ILP. Todos os sistemas foram avaliados em relação à acurácia obtida (usando t-testes (corrigidos) de Student (NADEAU, BENGIO, 2003) com $p < 0,05$), em relação à complexidade (número de cláusulas) da teoria, e em relação ao tempo computacional gasto (medido em segundos). Todos os experimentos foram realizados usando-se validação cruzada com 10 *folds*, sendo que os resultados apresentados para cada sistema (acurácia, número de cláusulas na teoria, e tempo de execução) são a média dos resultados destes *folds*. Para os sistemas-AED e GA-ILP, devido aos seus comportamentos estocásticos, o resultado em cada um dos 10 *folds* foi obtido calculando-se a média de 10 execuções em cada *fold*.

Utilizou-se validação cruzada interna com 10 *folds* para se determinar os valores dos parâmetros para o AED-ILP e para o HAED-ILP nas bases de dados Amine, Carc, Prot, Pyrim, e Yeast. Uma vez que Amine, Toxic, Acetyl e Memory são problemas "similares", os mesmos valores de parâmetros usados no Amine, para o AED-ILP e HEAD-ILP, foram também usados nos problemas Toxic, Acetyl, e Memory.

Em relação aos valores de parâmetros do RAED-ILP, uma vez que o seu espaço de busca é consideravelmente menor em relação ao do AED-ILP, simplesmente reduziu-se o número de gerações à metade e dobrou-se a taxa de aprendizado do sistema. Essa estratégia usada para parametrizar o RAED-ILP se mostrou bastante interessante em todos os experimentos, uma vez o RAED-ILP tende a se superadaptar (sofrer *overfitting*) ao conjunto de treinamento caso os mesmos parâmetros do AED-ILP sejam usados. Aqui, é importante notar que, infelizmente, o RAED-ILP não pôde ser executado nas bases Yeast e Prot, uma vez que quando o algoritmo Reduce (implementado no Aleph) é aplicado a tais problemas, uma recursão infinita ocorre no sistema. Assim, o Aleph não retorna a cláusula reduzida após horas de execução. Este é um problema particular do Aleph e não se conseguiu resolver tal questão.

Uma vez que o objetivo do AG-ILP é testar a capacidade de busca do AED-ILP, os valores dos parâmetros $numGen_1$, $numPop_1$, e *numCls* para o AG-ILP foram os mesmos utilizados no AED-ILP. Os valores dos outros parâmetros, tais como *probRec* (taxa de recombinação), *probMut* (taxa de mutação), e *tamTor* (tamanho do torneiro) foram encontrados usando o mesmo procedimento adotado para parametrizar os sistemas AED-ILP e HAED-ILP. Para executar o AG-ILP, testaram-se duas formas de se gerar a população inicial. Na primeira, adotou-se o método clássico, i.e., a população inicial foi gerada de forma aleatória. Já o segundo método gerou a população inicial da mesma forma que no AED-ILP, i.e., usando-se redes Bayesianas construídas sobre *bottom clauses*. Esta segunda forma de inicialização foi escolhida uma vez que ela levou o sistema a obter melhores resultados. As configurações para os sistemas AG-ILP, AED-ILP, RAED-ILP, e HAED-ILP para as bases Amine, Toxic, Acetyl, Memory, e Carcinogenesis são apresentadas na tabela 5.1. As configurações de tais sistemas para as bases Proteins, Pyrimidines, e Yeast são mostradas na tabela 5.2. Todos os nossos sistemas usaram acurácia como função de aptidão (função de avaliação).

Para o Aleph, sua configuração foi retirada dos trabalhos (HUYNH, MOONEY 2008), (MUGGLETON et al. 2010-a), e (MUGGLETON et al. 2010-b), uma vez que tais trabalhos sugerem bons valores de parâmetros para o Aleph para as bases de dados consideradas. Optou-se por adotar os valores de parâmetros sugeridos nestes trabalhos em vez de usar a validação cruzada para determiná-los, pois é muito difícil adotar tal procedimento devido à grande quantidade de parâmetros deste sistema. Assim, adotaram-se os valores de parâmetros sugeridos já que o Aleph obtém bons resultados quando parametrizado como indicado.

Tabela 5.1: Configuração dos sistemas AG-ILP, AED-ILP, RAED-ILP e HAED-ILP para as bases de dados Amine, Toxic, Acetyl, Memory, e Carcinogenesis.

-	**Amine, Toxic, Acetyl, e Memory**				**Carcinogenesis**			
	AG-ILP	AED-ILP	RAED-ILP	HAED-ILP	AG-ILP	AED-ILP	RAED-ILP	HAED-ILP
$numGen_1$	500	500	250	500	100	100	50	100
$numGen_2$	NA*	NA	NA	100	NA	NA	NA	50
$numPop_1$	20	20	20	20	10	10	10	10
$numPop_2$	NA	NA	NA	10	NA	NA	NA	10
numCls	3	3	3	NA	3	3	3	NA
λ_1	NA	0,005	0,01	0,005	NA	0,01	0,02	0,01
λ_2	NA	NA	NA	0,01	NA	NA	NA	0,02
k	NA	NA	NA	100	NA	NA	NA	30
numVezesGerSec	NA	NA	NA	10	NA	NA	NA	10
$P_1\backslash P_2\backslash P_3$	NA	0,5\0,5\0**	0,5\0,5\0**	0,5\0,5\0**	NA	0,1\0,1\0**	0,1\0,1\0**	0,1\0,1\0**
probRec	0,75	NA	NA	NA	0,6	NA	NA	NA
probMut	0,05	NA	NA	NA	0,05	NA	NA	NA
tamTor	2	NA	NA	NA	2	NA	NA	NA

* NA: não aplicável.
** P_1, P_2 e P_3 para todas as redes Bayesianas foram inicializadas com estes valores.

Tabela 5.2: Configuração dos sistemas AG-ILP, AED-ILP, RAED-ILP e HAED-ILP para as bases de dados Proteins, Yeast, e Pyrimidines.

-	**Proteins e Yeast**				**Pyrimidines**			
	AG-ILP	AED-ILP	RAED-ILP	HAED-ILP	AG-ILP	AED-ILP	RAED-ILP	HAED-ILP
$numGen_1$	500	500	-	500	800	800	400	800
$numGen_2$	NA*	NA	-	100	NA	NA	NA	100
$numPop_1$	20	20	-	20	20	20	20	20
$numPop_2$	NA	NA	-	10	NA	NA	NA	10
numCls	3	3	-	NA	10	10	10	NA
λ_1	NA	0.005	-	0.005	NA	0.005	0.01	0.005
λ_2	NA	NA	-	0.002	NA	NA	NA	0.002
k	NA	NA	-	100	NA	NA	NA	40
numVezesGerSec	NA	NA	-	20	NA	NA	NA	20
$P_1\backslash P_2\backslash P_3$	NA	0.6\0.6\0**	-	0.6\0.6\0**	NA	0.5\0.5\0**	0.5\0.5\0**	0.5\0.5\0**
probRec	0.8	NA	-	NA	0.8	NA	NA	NA
probMut	0.05	NA	-	NA	0.05	NA	NA	NA
tamTor	2	NA	-	NA	2	NA	NA	NA

* NA: não aplicável.
** P_1, P_2 e P_3 para todas as redes Bayesianas foram inicializadas com estes valores.

Para avaliar os experimentos realizados nas bases de dados do mundo real, primeiramente apresenta-se uma discussão geral dos resultados obtidos, seguido de uma análise específica do resultado para cada base de dados.

Resultados e Discussão Geral. A tabela 5.3 apresenta os resultados obtidos para os problemas do mundo real, sendo que (Acc) representa a acurácia obtida, (Cls) representa o número de cláusulas na teoria, (Lit) representa o número de literais na teoria e (T) é o tempo (em segundos) usado para executar os sistemas. Observe que os melhores resultados para cada critério avaliado foram destacados.

Tabela 5.3: Resultados Experimentais para as bases de dados Carc, Acetyl, Amine, Memory, Toxic, Pyrim, Prot, e Yeast

	ALEPH				**HAED-ILP**				**RAED-ILP**				**AED-ILP**				**AG-ILP**			
	Acc	**Cls**	**Lit**	**T(s)**	**Acc**	**Cls**	**Lit**	**T(s)**	**Acc**	**Cls**	**Lit**	**T(s)**	**Acc**	**Cls**	**Lit**	**T(s)**	**Acc**	**Cls**	**Lit**	**T(s)**
Carc	62.7	4.7	1.8	5.8	71.2	2.9	2.5	16.8	67.3	3	1.32	5.1	68.7	3	2.1	11.8	63.8	3	6	42.6
Acetyl	63.6	5.5	2.9	116.2	69.8	4.4	4.8	40.2	66.4	3	4.2	21.5	67.0	3	6.5	47.8	63.4	3	11	70.8
Amine	69.7	7	3.4	56.6	79.5	4.5	4.4	31.9	72.2	3	4	14.4	74.3	3	6.7	41.3	71.3	3	9.8	50
Memory	63.8	5.3	2.8	31.2	69.9	4.8	4.2	33.4	66.1	3	3.2	10.1	64.8	3	4.7	23.2	61	3	9.0	35.3
Toxic	78.3	4.4	3	27.4	81.5	4.9	3.7	36.7	74	3	4.1	14.8	74.7	3	6.2	37.4	71.8	3	10.2	60.7
Pyrim	74.4	32.1	3	555.3	76.7	13.2	3.6	751.8	69.8	10	3.24	625.6	71.6	10	6.2	950.8	68.7	10	7.5	1151
Prot	55.7	12.3	3.9	5237	60.2	4.8	6.7	428.5	NE*	NE	NE	NE	56.2	3	7.8	460.7	53.1	3	9.8	630.8
Yeast	62.2	11.6	2.4	128.3	65.5	4.8	3.4	132.9	NE	NE	NE	NE	61.9	3	4.6	140.3	57.8	3	5.9	175.6

Em relação aos resultados de acurácia, os experimentos mostram que o novo método para evoluir teorias é uma estratégia muito poderosa, uma vez que, para todas as bases de dados avaliadas, a acurácia obtida pelo HAED-ILP é estatisticamente significante em relação ao Aleph, RAED-ILP, AED-ILP, e AG-ILP. Considerando AED-ILP, pode-se dizer este sistema é bastante competitivo quando comparado ao Aleph, já que ele obteve acurácias estatisticamente significantes para o Carc e o Amine, equivalentes para o Acetyl, Memory, Prot e Yeast, e piores em Toxic e Pyrim. O RAED-ILP é, também, bastante competitivo quando comparado tanto ao AED-ILP quanto ao Aleph. Em relação ao AED-ILP, o RAED-ILP obtém acurácias equivalentes para todas as bases de dados, e, em relação ao Aleph, este sistema obtém acurácias significantes para todas as bases de dados com exceção de Toxic e Pyrim; casos em que o Aleph obtém acurácias significativamente melhores em relação ao RAED-ILP. O sistema mais fraco, o AG-ILP, obtém resultados equivalentes em relação ao Aleph para as bases de dados Carc, Acetyl, e Amine; para todos os outros problemas, o Aleph obtém acurácias estatisticamente significantes. Quando comparado ao RAED-ILP, o sistema AG-ILP obtém acurácia equivalente apenas nas bases Amine e Toxic; para todas as outras bases de dados, os resultados obtidos pelo RAED-ILP são estatisticamente significantes em relação ao AG-ILP. De maneira similar, todos os resultados de acurácia obtidos pelo AED-ILP são estatisticamente significantes em relação ao AG-ILP.

Considerando tempo de execução, os experimentos mostram que o HAED-ILP é bastante competitivo quando comparado aos outros sistemas, uma vez que, em relação ao Aleph, por exemplo, pode-se notar que para as bases de dados Acetyl, Amine, e Prot, o referido sistema usa menos tempo para obter teorias mais acuradas. Para outras bases de dados, tais como Memory e Yeast, os resultados de tempo de execução para ambos os sistemas é praticamente o mesmo. Contudo, como já explicado, o HAED-ILP obtém teorias mais acuradas. Como esperado, os tempos de execução do HAED-ILP são piores em relação aos tempos demandados pelo RAED-ILP. Esta diferença se explica uma vez que o RAED-ILP usa o algoritmo Reduce para reduzir o seu espaço de busca. Assim, o RAED-ILP necessita de uma configuração mais humilde (considerando o parâmetro $numGen_1$) para obter seus resultados. Esta redução no espaço de busca implica diretamente numa redução do tempo computacional demandado pelo sistema. Ainda em relação aos tempos de execução, o HAED-ILP é mais rápido que o AED-ILP para todas as bases de dados com exceção da base de dados Carc. Em relação ao AG-ILP, pode-se notar que o HAED-ILP usa consideravelmente menos tempo para obter resultados bem melhores. Em resumo, o RAED-ILP obteve 4 melhores resultados de tempo (para as bases Acetyl, Amine, Memory, e Toxic), seguido pelo Aleph que obteve 3 melhores resultados (para as bases Carc, Pyrim, e Yeast), e finalmente pelo HAED-ILP, que obteve um único melhor resultado para a base de dados Prot.

Em relação ao tamanho das teorias, pode-se dizer que o Aleph desenvolve teorias com o menor número de literais. Contudo, considerando o número de cláusulas na teoria, o Aleph é claramente superado pelos sistemas-AED e o AG-ILP, uma vez que todos estes sistemas, para todas as bases de dados avaliadas (com exceção do HAED-ILP na base de dados Toxic), desenvolveram teorias com um número menor de cláusulas que o Aleph. Sob este aspecto, pode-se notar que o RAED-ILP, o AED-ILP, e o AG-ILP desenvolvem teorias com o menor número de cláusulas, seguidos, nesta ordem, pelos sistemas HAED-ILP, e o Aleph. Uma vez que é trabalhoso se analisar o tamanho de uma teoria considerando o número de cláusulas e o número de literais independentemente, a seguir, durante a discussão detalhada dos resultados experimentais, o tamanho de uma teoria será dado pelo produto do número de cláusulas pelo número de literais que a compõe. Apesar disto ser uma simplificação “grosseira”, este número poderá ser usado para se comparar diretamente o tamanho de uma teoria como se ela fosse composta por uma única cláusula que contém vários literais.

Resultado para o Carcinogenesis. A figura 5.1 analisa graficamente o resultado para o Carcinogenesis para o Aleph, o RAED-ILP, o AED-ILP, o HAED-ILP e o GA-ILP. As barras em destaque na análise de acurácia exibem o desvio padrão.

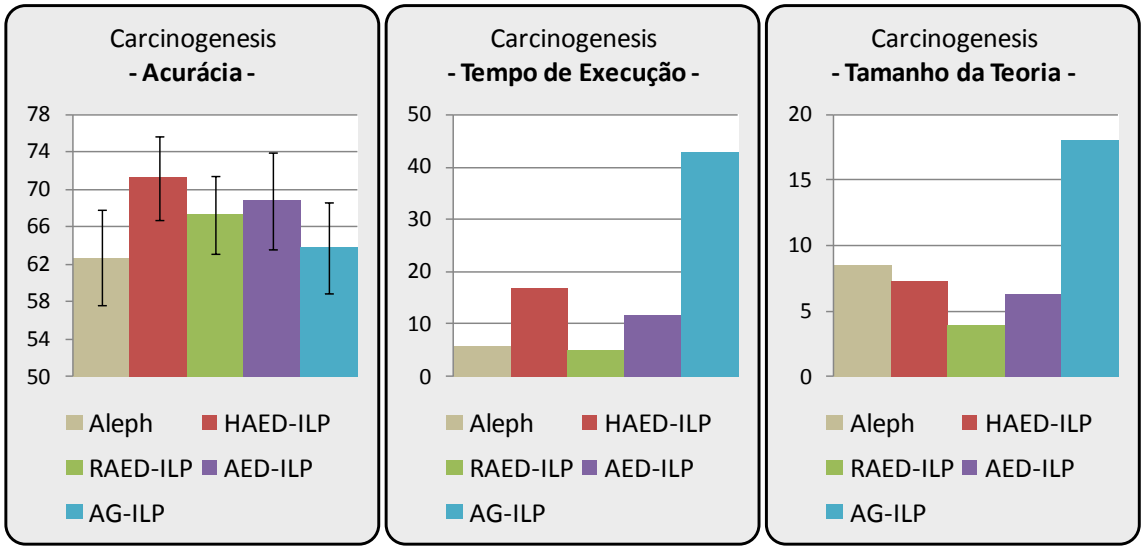

Figura 5.1: Análise do resultado para a base de dados Carcinogenesis

Para esta base de dados, todos os sistemas-AED obtiveram acurácia estatisticamente significante (e com teorias mais simples) em relação ao Aleph. Pode-se notar que as diferenças entre os resultados de acurácia para o Aleph e os sistemas-AED são consideráveis, uma vez que o Aleph, o HAED-ILP, o RAED-ILP, e o AED-ILP, obtiveram, respectivamente, 62,7%, 71,2%, 67,3%, e 68,3%. Contudo, o Aleph usa bem menos tempo em relação ao HAED-ILP e ao AED-ILP para obter seus resultados; ele precisa de apenas 5,8s contra 16,8s usados pelo HAED-ILP e 11,8s usados pelo AED-ILP. Por sua vez, pode ser dito que o RAED-ILP se comporta muito bem nesta base de dados; ele encontra as teoria menos complexas, com uma boa acurácia, usando menos tempo que todos os outros sistemas (apenas 5,1s). O AG-ILP obteve acurácia equivalente em relação ao Aleph, mas precisa de muito mais tempo para obter uma teoria muito mais complexa.

Embora o HAED-ILP tenha obtido o melhor resultado de acurácia (71,2%), pode-se dizer que o melhor resultado geral para o Carcinogenesis foi obtido pelo RAED-ILP, que, como mostrado, alcançou uma boa acurácia, com teorias menos complexas, usando menos tempo que todos os outros sistemas. Este resultado sugere que o Carcinogenesis tende a ser "resolvido" pelo uso de poucos literais em suas teorias. Sob este aspecto, tentou-se executar o Aleph usando o algoritmo Reduce. Contudo, para todos os experimentos realizados, o Aleph foi incapaz de melhorar seus resultados.

Tentou-se, também, executar o HAED-ILP usando o algoritmo Reduce com objetivo de reduzir seu espaço de busca, entretanto, tal como ocorreu com o Aleph, o HAED-ILP não foi capaz de melhorar seus resultados. Dessa forma, pode-se concluir que o algoritmo Reduce claramente ajudou o RAED-ILP a encontrar resultados bastante interessantes nesta base de dados, contudo, a busca do AED por teorias também desempenhou um papel fundamental para que este resultado pudesse ser obtido.

Resultado do Acetyl. A figura 5.2 analisa graficamente o resultado para o Acetyl para o Aleph, o RAED-ILP, o AED-ILP, o HAED-ILP e o GA-ILP. As barras em destaque na análise de acurácia exibem o desvio padrão

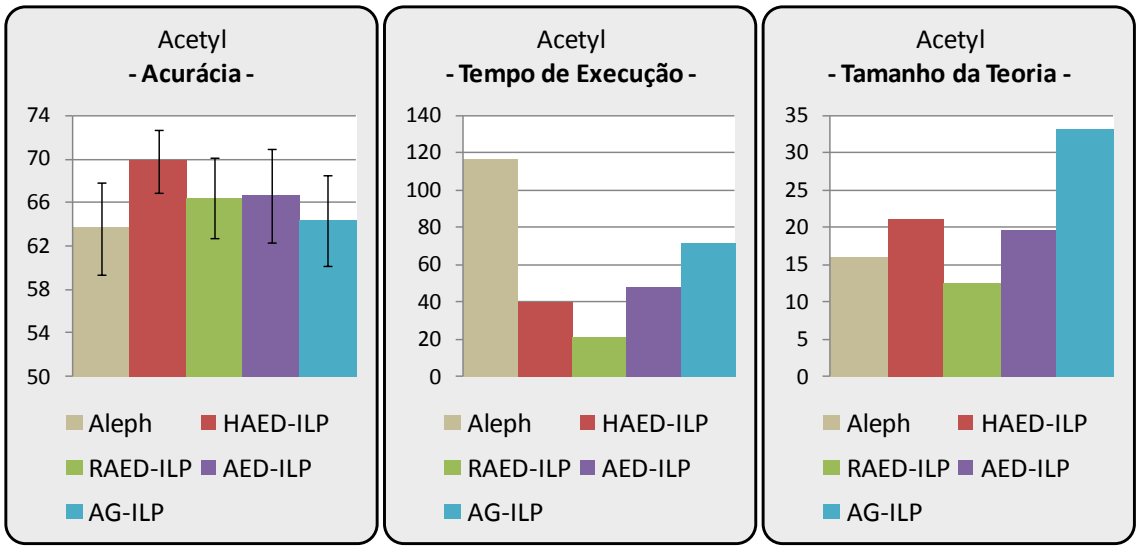

Figura 5.2: Análise do resultado para a base de dados Acetyl

Pode-se notar que o Aleph, usando o algoritmo de cobertura, encontra dificuldades para aprender suas teorias, uma vez que tal sistema usa muito mais tempo para obter teorias de baixa acurácia em relação a todos os outros sistemas. Como afirmado, o HAED-ILP obteve acurácia estatisticamente significante em relação a todos os outros sistemas. Já os resultados de acurácia para o AED-ILP, RAED-ILP, e AG-ILP são equivalentes em relação ao Aleph. Considerando os tempos de execução, o HAED-ILP, o RAED-ILP, o AED-ILP, e o AG-ILP, usaram, respectivamente 65%, 81%, 59%, e 39% menos tempo para obterem teorias mais acuradas em relação ao Aleph. Adicionalmente, é importante notar que mesmo usando mais tempo, o Aleph desenvolveu teorias menos acuradas e mais complexas que as teorias desenvolvidas pelo RAED-ILP, ou desenvolveu teorias tão complexas quanto às teorias desenvolvidas pelo AED-ILP.

Os experimentos realizados no Acetyl sugerem que melhores resultados podem ser obtidos usando estratégias que de busca que desenvolvem teorias, em vez do método de cobertura usado pelo Aleph. Além disto, o novo método híbrido para evoluir teorias

parece ter funcionado muito bem nesta base de dados, se considerarmos que o HAED-ILP obteve um valor bem mais alto de acurácia em relação aos seus competidores.

Resultado do Amine. A figura 5.3 analisa graficamente o resultado para o Amine para o Aleph, o RAED-ILP, o AED-ILP, o HAED-ILP e o GA-ILP. As barras em destaque na análise de acurácia exibem o desvio padrão.

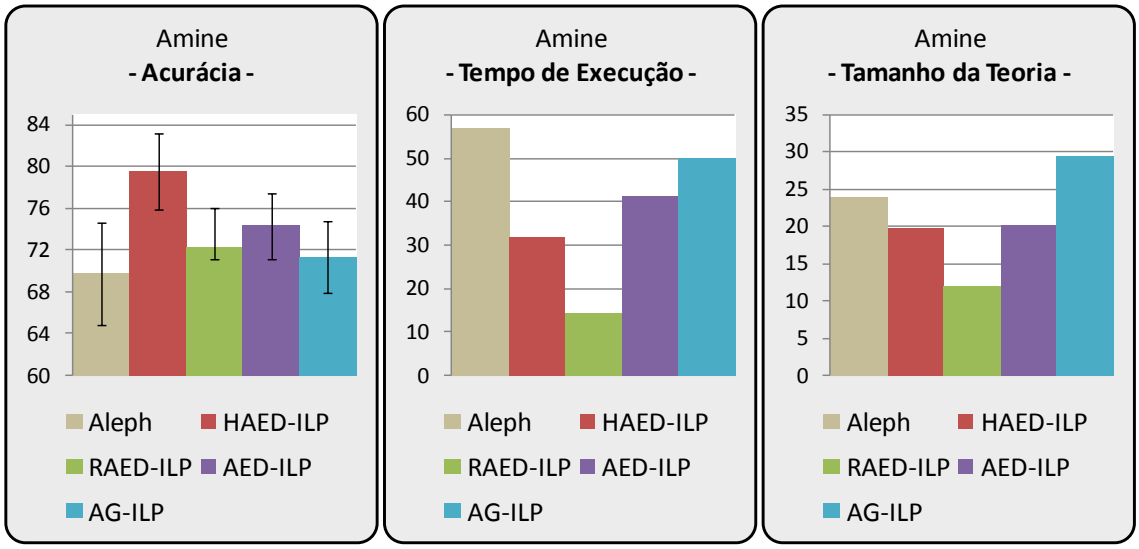

Figura 5.3: Análise do resultado para a base de dados Amine

Para a base de dados Amine, o Aleph obteve o pior resultado de acurácia usando mais tempo para obter teorias mais complexas em relação a todos os sistemas-AED. Os resultados de acurácia para o Aleph e para o AG-ILP foram muito próximos, contudo, este último sistema usa um pouco menos de tempo para obter teorias maiores. Tanto o HAED-ILP quanto o AED-ILP obtiveram acurácia estatisticamente significante em relação ao Aleph. Considerando o RAED-ILP, tal sistema obteve uma acurácia equivalente em relação ao Aleph. Em relação ao AED-ILP, o RAED-ILP usa 65% menos tempo para obter um resultado de acurácia equivalente com uma teoria muito mais simples.

Uma vez que o HAED-ILP e o AED-ILP obtiveram acurácias estatisticamente significantes, em menos tempo, e com teorias mais simples em relação ao Aleph, pode-se dizer que ambos os sistemas se comportaram bem na base de dados Amine. Contudo, esta base de dados demonstrou claramente o potencial do método híbrido de busca: a diferença entre os resultados de acurácia para o HAED-ILP e o AED-ILP é mais de 5%, sendo que o HAED-ILP desenvolve teorias de mesma complexidade que o AED-ILP, mas usa 24% menos tempo. Além disto, pode-se notar que o método híbrido de busca claramente superou o procedimento de cobertura, uma vez que a diferença entre o resultado de acurácia para o HAED-ILP e o Aleph chega a quase 10%.

Resultado do Memory. A figura 5.4 analisa graficamente o resultado para o Memory para o Aleph, o RAED-ILP, o AED-ILP, o HAED-ILP e o GA-ILP. As barras em destaque na análise de acurácia exibem o desvio padrão.

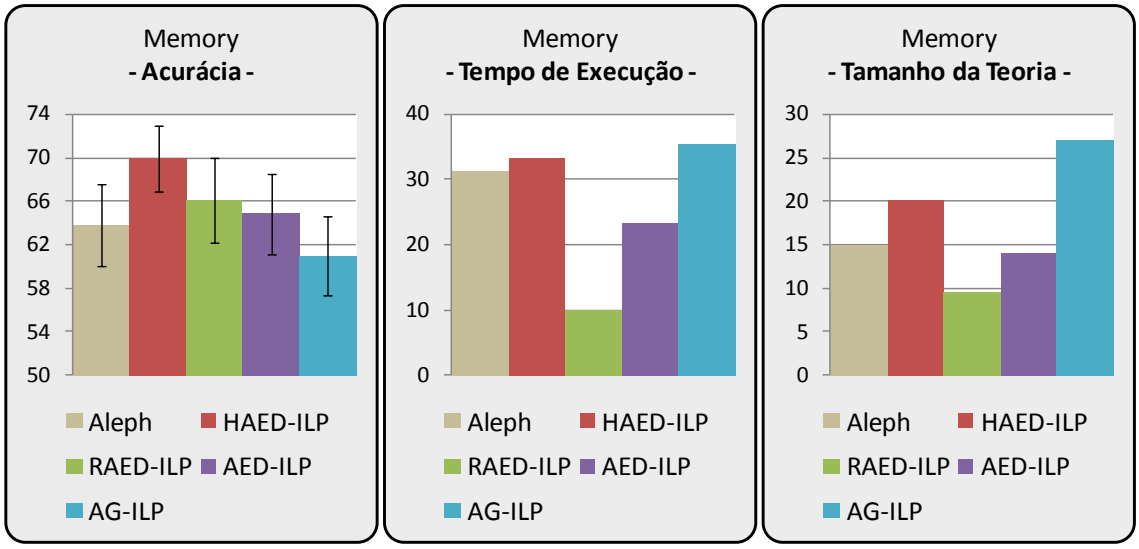

Figura 5.4: Análise do resultado para a base de dados Memory

Para a base de dados Memory, os resultados de acurácia para o Aleph, RAED-ILP, e AED-ILP são equivalentes. Em relação ao AG-ILP, o Aleph obtém acurácia estatisticamente significante. Note, contudo, que o tempo de execução usado pelo Aleph é maior que o do RAED-ILP e do AED-ILP, mas um pouco menor que o do AG-ILP. Considerando a complexidade das teorias, as do Aleph são quase de mesmo tamanho que as teorias do AED-ILP, maiores que as do RAED-ILP, e menores que as do AG-ILP. Nota-se que, para esta base de dados, que o Aleph é um sistema competente em relação ao AED-ILP e ao RAED-ILP, mas claramente melhor que o AG-ILP. Já em relação ao HAED-ILP, apesar deste sistema necessitar de mais tempo computacional, ele encontra teorias mais acuradas e menos complexas em relação ao Aleph. Em relação aos outros sistemas-AED, o HAED-ILP usa mais tempo para encontrar teorias mais complexas, mas estatisticamente significantes em relação à acurácia.

Para a base de dados Memory, mais uma vez o HAED-ILP mostra a importância do processo de evolução híbrido, uma vez que a despeito de encontrar teorias mais complexas usando mais tempo que o Aleph, o RAED-ILP e o AED-ILP, tal sistema encontra teorias estatisticamente significantes em relação a todos os seus competidores. Claramente, para esta base de dados, o método de busca híbrido foi responsável pelo desenvolvimento destas teorias acuradas, uma vez que o método de cobertura e o método busca usado pelos sistemas AED-ILP e RAED-ILP obtêm resultados bastante similares nesta base de dados.

Resultado do Toxic. A figura 5.5 analisa graficamente o resultado para o Toxic para o Aleph, o RAED-ILP, o AED-ILP, o HAED-ILP e o GA-ILP. As barras em destaque na análise de acurácia exibem o desvio padrão.

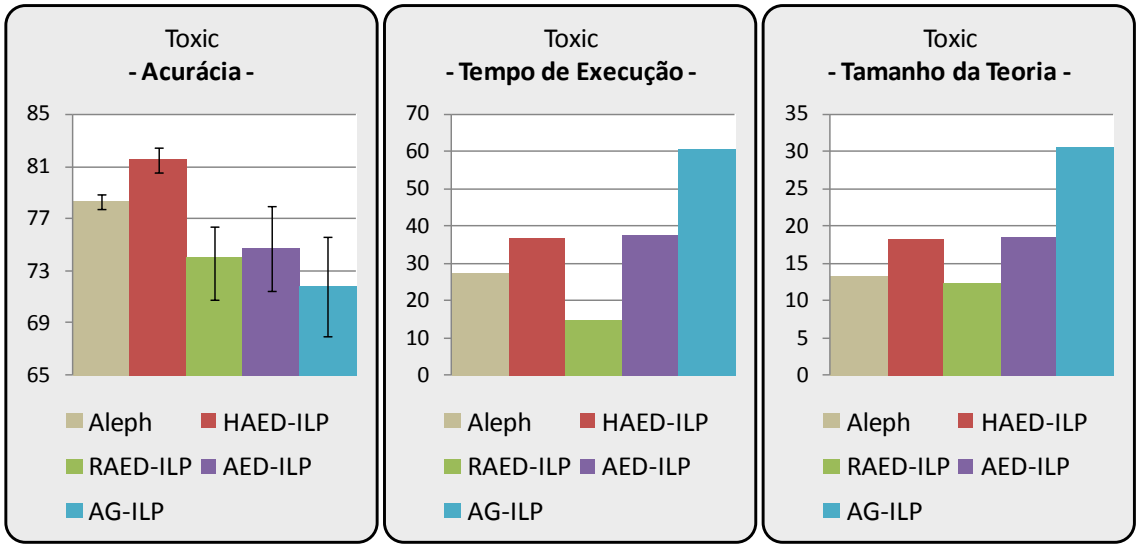

Figura 5.5: Análise do resultado para a base de dados Toxic

O Aleph se comporta muito bem na base de dados Toxic. Considerando os resultados de acurácia, as teorias do Aleph são estatisticamente significantes em relação ao RAED-ILP, ao AED-ILP, e ao AG-ILP. Além disto, o Aleph usa menos tempo que o HAED-ILP, AED-ILP, e o AG-ILP para desenvolver teorias mais simples. Em contrapartida, o RAED-ILP usa consideravelmente menos tempo para desenvolver teorias mais simples que o Aleph. Contudo, tais teorias são substancialmente menos acuradas em relação às do Aleph. O sistema AG-ILP é claramente superado pelo Aleph, uma vez que ele usa muito mais tempo para desenvolver teorias complexas e pouco acuradas.

Para a base de dados Toxic, fica claro que o método de cobertura adotado pelo Aleph supera os métodos de busca por teorias adotados pelos sistemas RAED-ILP, AED-ILP, e AG-ILP. Neste sentido, parece que métodos de busca que iterativamente constroem teorias podem melhor solucionar o problema Toxic. Esta suposição pode ser confirmada pelos resultados obtidos pelo HAED-ILP, uma vez que ele foi o único sistema capaz de superar significativamente o excelente resultado de acurácia obtido pelo Aleph. Contudo, é importante notar que este resultado foi obtido através do custo de se usar mais tempo (e teorias mais complexas) em relação aos resultados do Aleph. Apesar disto, o tempo gasto pelo HAED-ILP é bem competitivo em relação ao Aleph e ao AED-ILP, mas claramente melhor que o tempo do AG-ILP.

Resultado do Pyrimidines. A figura 5.6 analisa graficamente o resultado para o Pyrimidines para o Aleph, o RAED-ILP, o HAED-ILP, e o GA-ILP. As barras em destaque na análise de acurácia exibem o desvio padrão.

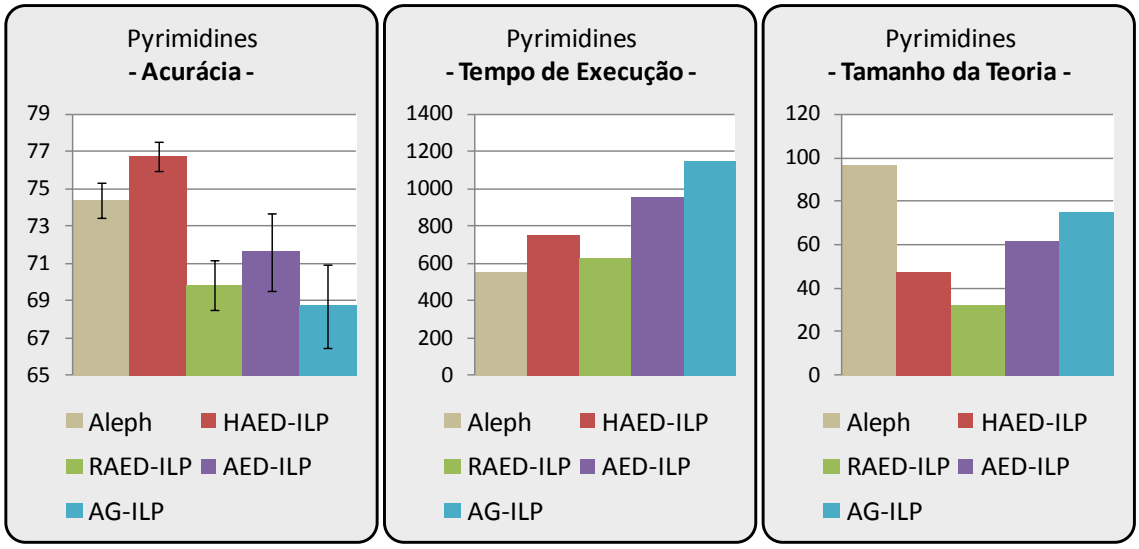

Figura 5.6: Análise do resultado para a base de dados Pyrimidines

Para a base de dados Pyrimidines, caso considerem-se apenas os resultados de acurácia e tempo, pode-se dizer que o Aleph se comporta muito bem, uma vez que ele obtém acurácia estatisticamente significante em relação ao RAED-ILP, AED-ILP, e AG-ILP, usando menos tempo que todos os seus sistemas competidores (incluindo o HAED-ILP). Contudo, nota-se que as teorias desenvolvidas pelo Aleph são muito mais complexas que as teorias desenvolvidas pelos outros sistemas. De fato, existe uma diferença considerável na complexidade das teorias desenvolvidas. Precisamente, o Aleph usa 96,3 literais contra 47,2, 32,4, 62, e 75 literais usados, respectivamente, pelo HAED-ILP, RAED-ILP, AED-ILP e AG-ILP. Esta diferença se deve ao fato de o Aleph usar 32,1 cláusulas em suas teorias, contra 13,2 cláusulas usadas pelo HAED-ILP, e 10 cláusulas usadas pelos outros sistemas. Assim, caso consideremos apenas os valores de acurácia e tempo, o Aleph será o melhor sistema entre o AED-ILP, RAED-ILP, e AG-ILP. Todavia, o sistema que claramente pode balancear os três objetivos (obter teorias simples e de alta acurácia em tempo computacional factível) é o HAED-ILP, uma vez que tal sistema desenvolve teorias significativamente mais acuradas, mais simples (perdendo apenas para o RAED-ILP), usando tempo computacional razoável em relação aos seus competidores.

Tal como ocorreu para outras bases de dados, mais uma vez o método híbrido de evolução mostra seu potencial ao encontrar teorias mais acuradas (perdendo em simplicidade apenas para o RAED-ILP) e em tempo computacional factível.

Resultado do Proteins. A figura 5.7 analisa graficamente o resultado para o Proteins para o Aleph, o RAED-ILP, o AED-ILP, o HAED-ILP e o GA-ILP. As barras em destaque na análise de acurácia exibem o desvio padrão.

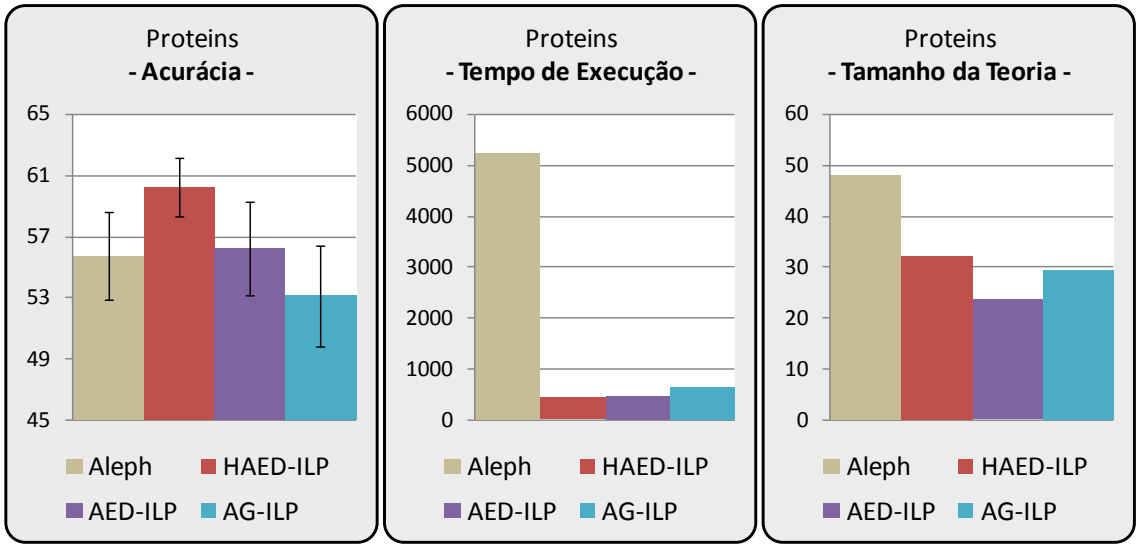

Figura 5.7: Análise do resultado para a base de dados Proteins

Como mostra a figura 5.7, o Aleph apresenta grande dificuldade para lidar com a base de dados Proteins: ele encontra acurácia equivalente em relação ao AED-ILP e ao AG-ILP usando tempo computacional muito maior e teorias muito mais complexas. Tal como ocorreu na base de dado Pyrimidines, o Aleph precisa de muitas cláusulas em suas teorias, precisamente, 12,3 cláusulas, contra 4,8 cláusulas do HAED-ILP e 3 cláusulas do AED-ILP e AG-ILP. Durante os experimentos, percebeu-se a estratégia de busca do Aleph, especificamente para este problema, parece ser incapaz de produzir boas cláusulas. Assim, o Aleph deve explorar uma enorme quantidade de cláusulas para construir sua teoria final, e, dessa forma, o sistema necessita de grande tempo computacional para construir sua teoria. Embora o HAED-ILP se comporte muito bem em relação ao Aleph, os resultados do HAED-ILP claramente chamam atenção: tal sistema obtém acurácias estatisticamente significantes em relação a todos os outros sistemas, usando a menor quantidade de tempo para produzir teorias mais simples que as do Aleph, e praticamente tão complexas quanto às teorias encontradas pelo AED-ILP e AG-ILP.

Como visto, na base de dados Proteins, o método de cobertura demonstra ser bastante ineficaz. Em contrapartida, a abordagem híbrida usada pelo HAED-ILP, embora parcialmente inspirada pelo método de cobertura, é capaz de obter bons resultados.

Resultado do Yeast. A figura 5.8 analisa graficamente o resultado para o Yeast para o Aleph, o RAED-ILP, o AED-ILP, o HAED-ILP e o GA-ILP. As barras em destaque na análise de acurácia exibem o desvio padrão.

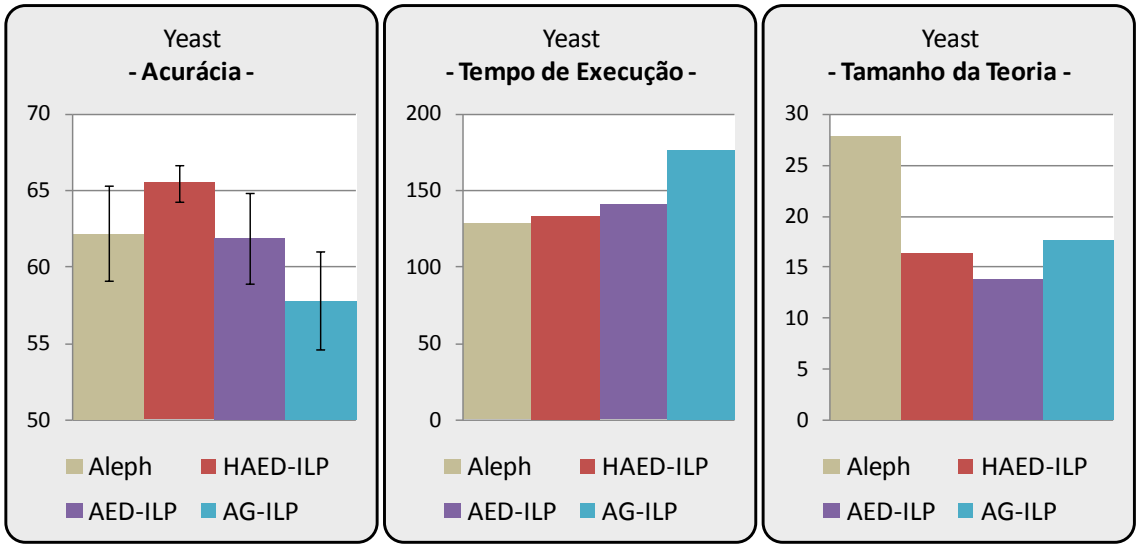

Figura 5.8: Análise do resultado para a base de dados Yeast

Na base de dados Yeast, o Aleph obtém acurácia equivalente a do AED-ILP, usando menos tempo, mas com uma teoria consideravelmente mais complexa. Em relação ao AG-ILP, o Aleph obtém acurácia estatisticamente significante usando menos tempo, mas ainda com uma teoria mais complexa. Novamente, como ocorrido para as bases Pyrimidines e Proteins, a complexidade das teorias desenvolvidas pelo Aleph se deve a este sistema usar muito mais cláusulas em suas teorias do que os outros sistemas. Contudo, diferentemente do que ocorreu na base de dados Proteins, o tempo de execução do Aleph é bem competitivo em relação aos seus competidores; de fato, o Aleph usa o menor tempo, dentre todos os sistemas, para aprender suas teorias. Isto mostra que o método de busca usado pelo Aleph, apesar de construir teorias complexas, funciona bem nesta base de dados, uma vez que ele foi capaz de encontrar teorias acuradas usando o menor tempo de execução. Por sua vez, o HAED-ILP obtém acurácias estatisticamente significantes em relação a todos os outros sistemas, usando, praticamente, o mesmo tempo computacional usado pelo Aleph. Contudo, o HAED-ILP desenvolve teorias muito mais simples que as do Aleph.

Na base de dados Yeast, o método de cobertura usado pelo Aleph é competitivo quando comparado à busca realizada pelo AED-ILP. Contudo, fica claro que o método de busca híbrido é capaz de obter melhores resultados quando comparado a estas duas outra "formas" de busca.

5.4 Os Problemas de Transição de Fase

Metodologia Experimental. Em todos os problemas de Transição de Fase, foram usados exatamente os mesmos conjuntos de treinamento/tese e declarações de modos usados no QG/GA (MUGGLETON, TAMADDONI-NEZHAD, 2006) (que usaram a estratégia de teste *hold-out*). Tal adoção teve como objetivo avaliar diretamente os sistemas AED-ILP e AG-ILP em relação ao Progol-A*, Progol-QG, Progol-GA, e Progol-QG/GA. Aqui, é importante esclarecer alguns pontos:

1) **Por que o RAED-ILP e o HAED-ILP não foram avaliados nos problemas de Transição de Fase?**

   O RAED-ILP não foi avaliado devido ao mesmo problema ocorrido nas bases de dados Proteins e Yeast, i.e., o algoritmo Reduce implementado no Aleph, realiza uma recursão infinita quando executado nestes problemas. Este mesmo problema também ocorreu em todas as bases de dados de Transição de Fase consideradas neste trabalho.

   Por sua vez, o HAED-ILP não foi usado nestes experimentos uma vez que os melhores resultados para o AED-ILP e o AG-ILP foram obtidos usando-se uma única cláusula em suas teorias. De fato, estes problemas foram construídos para serem classificados com 100% de acurácia por uma teoria composta por uma única cláusula. Assim, uma vez que o HAED-ILP se diferencia do AED-ILP justamente por adicionar cláusulas iterativamente à suas teorias, e, considerando que uma única cláusula pode classificar corretamente os exemplos deste problema, não faz sentido usar o HAED-ILP, que, neste caso, adicionará cláusulas desnecessárias às teorias.

2) **Por que optamos por usar diretamente os resultados em (MUGGLETON, TAMADDONI-NEZHAD, 2006) em vez de replicá-los?**

   A versão do Progol que implementa todas estas variações do sistema (Progol-QG, Progol-GA, e Progol-QG/GA) apresenta um erro de segmentação de memória que causa o término da execução do sistema logo no começo de sua execução. Este erro aparenta ser devido à incompatibilidade do código original com o gerenciamento de memória de novas versões do compilador *gcc* e do sistema operacional. Os autores de tal sistema foram informados do problema e apesar de tentativas junto a eles, não se conseguiu resolver o problema.

Os resultados do AED-ILP e AG-ILP para todos os problemas de Transição de Fase foram obtidos calculando-se a média de 10 execuções sobre cada conjunto de teste. Tal como nos experimentos anteriores, todos os nossos sistemas usaram acurácia como função de aptidão (função de avaliação). Para parametrizar o AED-ILP, criou-se um conjunto de validação do conjunto de treinamento do problema *m*16.*l*12. Para todos os problemas, exceto para o *m*6.*l*12 e *m*7.*l*12, usaram-se os valores de parâmetros determinados para o problema *m*6.*l*12. Estas exceções vêm do fato de que ambos os problemas, *m*6.*l*12 e *m*7.*l*12, são fáceis de serem resolvidos, e, portanto, eles não necessitam dos mesmos parâmetros que foram determinados para o *m*16.*l*12. Para parametrizar os nossos sistemas para estes dois problemas, criou-se um conjunto de validação do conjunto de treinamento do *m*7.*l*12. Assim, os mesmos parâmetros usados para o *m*7.*l*12 também foram usados para o *m*6.*l*12. Tal como nos experimentos anteriores, aos parâmetros $numGen_1$, $tamPop_1$, e *numCls* para o AG-ILP, foram atribuídos os mesmos valores que no AED-ILP. Os outros parâmetros, tais como *probMut*, *probRec*, e *tamTor*, tiveram seus valores determinados utilizando o mesmo procedimento usado para o AED-ILP, i.e., usando-se o conjunto de validação utilizado para se parametrizar o AED-ILP. A população inicial do AG-ILP foi gerada tal como discutido nos experimentos anteriores, i.e., usando-se redes Bayesianas. A tabela 5.4 apresenta os valores dos parâmetros para o AED-ILP e AG-ILP para os problemas de Transição de Fase.

Tabela 5.4: Configuração dos sistemas AG-ILP e AED-ILP, para os problemas de Transição de Fase *m*6.*l*12, *m*7.*l*12, *m*8.*l*12, *m*10.*l*12, *m*11.*l*12, *m*14.*l*12, e *m*16.*l*12.

-	***m*6.*l*12 e *m*7.*l*12**		***m*8.*l*12, *m*10.*l*12, *m*11.*l*12, *m*14.*l*12, e *m*16.*l*12**	
	AG-ILP	AED-ILP	AG-ILP	AED-ILP
$numGen_1$	20	20	200	200
$numGen_2$	NA*	NA	NA	NA
$numPop_1$	10	10	20	20
$numPop_2$	NA	NA	NA	NA
numCls	1	1	1	1
λ_1	NA	0,07	NA	0,05
λ_2	NA	NA	NA	NA
k	NA	NA	NA	NA
numVezesGerSec	NA	NA	NA	NA
$P_1\backslash P_2\backslash P_3$	NA	0,5\0,5\0**	NA	0,5\0,5\0**
probRec	0,6	NA	0,75	NA
probMut	0,01	NA	0,05	NA
tamTor	2	NA	2	NA

* NA: não aplicável.

** P_1, P_2 e P_3 para todas as redes Bayesianas foram inicializadas com estes valores.

Resultados e Discussão. A tabela 5.5 apresenta os resultados obtidos para os problemas de Transição de Fase, sendo que (Acc) representa a acurácia obtida e (T) é o tempo (em segundos) usado para executar os sistemas. Observe que os melhores resultados para cada critério avaliado foram destacados.

Conforme explicado anteriormente, como não se foi capaz de reproduzir os resultados para o Progol-QG, Progol-GA, e Progol-QG/GA, não se pôde fazer uma comparação direta dos tempos de execução entre o AG-ILP e AED-ILP, e estas variantes do Progol. Contudo, uma vez que os resultados do Progol-A* puderam ser reproduzidos, executou-se tal sistema em todos os problemas de Transição de Fase considerados, com objetivo de se visualizar a diferença de tempo de execução entre a máquina usada para realizar os experimentos em (MUGGLETON, TAMADDONI-NEZHAD, 2006) e a que foi utilizada neste trabalho. Dessa forma, pôde-se visualizar que a máquina usada em (MUGGLETON, TAMADDONI-NEZHAD, 2006) é, em média, 1,97 vezes mais rápida que máquina utilizada em nossos testes. Assim, embora a tabela 5.4 apresente os resultados de tempo retirados de (MUGGLETON, TAMADDONI-NEZHAD, 2006), juntamente com os resultados de tempo de nossos sistemas, pode-se ter, ao menos, uma noção do tempo gasto pelo AED-ILP e AG-ILP em relação às variantes do Progol. Dito isso, é importante destacar que os resultados de tempo apresentados para o AED-ILP e AG-ILP não consideram que a máquina de (MUGGLETON, TAMADDONI-NEZHAD, 2006) é, em média, ~ 2x mais rápida que a nossa. Em outras palavras, a tabela 5.4 apresenta os resultados de tempos reais para os sistemas AED-ILP e AG-ILP.

Tabela 5.5: Resultados experimentais dos sistemas Progol-A*, Progol-QG, Progol-GA, Progol-QG/GA, AG-ILP e AED-ILP, para os problemas de Transição de Fase *m*6.*l*12, *m*7.*l*12, *m*8.*l*12, *m*10.*l*12, *m*11.*l*12, *m*14.*l*12, e *m*16.*l*12.

m	**Progol-A***		**Progol-QG**		**Progol-GA**		**Progol-QG/GA**		**AG-ILP**		**AED-ILP**	
	Acc	**T(s)**	**Acc**	**T(s)**	**Acc**	**T(s)**	**Acc**	**T(s)**	**Acc**	**T(s)**	**Acc**	**T(s)**
6	98	3,22	99,5	3,89	99,5	5,83	99,5	10,32	99,3	5,8	**100**	2,2
7	99,5	633,16	99,5	45,1	99,5	12,99	99,5	86,51	99	27,5	**100**	23,1
8	**100**	1416,55	**100**	175	**100**	13,92	**100**	169,55	88,5	86	92,5	72,4
10	97,5	25852,80	99	242,29	95,5	74,68	99	1064,22	97	205	**99,5**	129,9
11	80	37593,20	91	774	94	30,37	**99,5**	110,15	99	211	99	160,2
14	50	128314,00	69	4583,2	79,5	529,67	88,5	1184,76	92	115,2	**96**	55,8
16	59	55687,44	77,5	4793,0	74	297,93	89,5	4945,20	88,5	369,8	**94,5**	234,9

Pela tabela 5.5, nota-se que diferentes bases de dados, mesmo oriundas do mesmo tipo de problema, apresentam diferentes peculiaridades. Por exemplo, as instâncias *m*6.*l*12 e *m*7.*l*12 são facilmente resolvidas por todos os sistemas, com destaque para o AED-ILP,

que obteve o melhor resultado de acurácia usando menos tempo que os seus competidores. Por sua vez, para a instância *m*8.*l*12, todas as versões do Progol superaram tanto o AG-ILP quanto o AED-ILP em relação à acurácia. Para esta instância, destaca-se o Progol-GA, que obteve melhor acurácia e tempo mais baixo em relação a todos os outros sistemas. Para o *m*10.*l*12, o AED-ILP obteve o melhor resultado de acurácia perdendo em tempo de execução (desconsiderando a diferença de velocidades entre as máquinas) apenas para o Progol-GA. Contudo, a acurácia obtida por este último sistema é consideravelmente mais baixa que a acurácia do AED-ILP (o Progol-GA obteve 95,5% de acurácia contra 99,5% obtida pelo AED-ILP). Para a instância *m*11.*l*12, o Progol-QG/GA foi o melhor sistema em relação à acurácia obtida. Contudo, seus resultados são bastante próximos aos resultados obtidos pelo AED-ILP e AG-ILP. Considerando os dois últimos problemas (os mais difíceis), fica claro que nenhum outro sistema pode competir de igual para igual com o AED-ILP, uma vez tal sistema obtém acurácias consideravelmente mais altas usando substancialmente menos tempo que todos os outros sistemas.

Em resumo, pode-se dizer que o AED-ILP nitidamente supera todos os outros sistemas: com exceção dos problemas *m*8.*l*12 e *m*11.*l*2, pode-se notar que o AED-ILP obtém acurácias melhores (ou iguais) a todos os seus competidores. Claramente, o sistema que melhor compete com o AED-ILP é o Progol-QG/GA. Contudo, para os problemas mais difíceis (*m*14.*l*12 e *m*16.*l*12) o AED-ILP obtém diferenças notáveis em relação à acurácia. Apesar de não se poder realizar uma comparação direta entre os tempos de execução dos sistemas, fica claro que o AED-ILP é muito mais eficiente que o Progol-QG/GA: 19x mais rápido no melhor caso (*m*16.*l*12), e, em média, 10x mais rápido sem se considerar que o computador de (MUGGLETON, TAMADDONI-NEZHAD, 2006) é, ao menos executando o Progol-A*, em média ~2x mais rápido que o nosso. Surpreendentemente, o AG-ILP se comporta muito bem nos problemas de Transição de Fase. Contudo, o Progol-QG/GA geralmente obtém melhores acurácias que o AG-ILP, e o AED-ILP é sempre melhor (tanto em acurácia quanto em tempo) em relação a este sistema.

5.5 Resumo do Capítulo

Este capítulo apresentou os resultados experimentais dos sistemas-AED em bases de dados do mundo real e em problemas de Transição de Fase. Os sistemas propostos foram comparados aos sistemas Aleph, Progol (com algumas de suas variantes), e ao AG-ILP. De forma geral, mostrou-se que, em problemas do mundo real, os sistemas-AED são bem competitivos e até mesmo superam o sistema Aleph. Já em relação aos problemas de Transição de Fase, foi mostrado que o sistema AED-ILP é de longe o melhor sistema quando avaliado em relação ao AG-ILP, Progol-A*, Progol-QG, Progol-GA, e Progol-QG/GA.

Capítulo 6: Conclusões e Trabalhos Futuros

Este capítulo apresenta algumas conclusões que puderam ser obtidas a partir dos experimentos realizados, juntamente com propostas de alguns trabalhos futuros.

6.1 Introdução

Este trabalho apresentou uma proposta de aplicação de um AED ao problema de Programação em Lógica Indutiva. Como resultado, criou-se o sistema AED-ILP, bem como suas duas extensões, o RAED-ILP, e o HAED-ILP. O RAED-ILP pode ser entendido como o AED-ILP que usa o algoritmo Reduce para reduzir seu espaço de busca. Já o HAED-ILP pode ser visto como o AED-ILP com seu ciclo de busca completamente remodelado, uma vez que tal sistema apresenta dois "tipos" de evolução; uma que trata do desenvolvimento de um conjunto de redes Bayesianas (para gerar teorias) enquanto que a outra lida com o desenvolvimento de uma única rede Bayesiana (para gerar cláusulas isoladas).

Este capítulo se estrutura como segue. A próxima seção apresenta algumas conclusões relativas a cada componente principal dos sistemas. A seção 6.2 apresenta uma lista de trabalhos futuros. A seção 6.3 apresenta um resumo deste capítulo.

6.2 Conclusões

Como apresentado no capítulo 4, o sistemas-AED foram desenvolvidos assumindo-se certas diretrizes. A seguir, apresentam-se conclusões sobre cada diretriz assumida.

I. Adoção de um AED como mecanismo de exploração padrão: da mesma forma que ocorreu quando esta classe de algoritmos foi aplicada a problemas de otmização, o AED, quando aplicado a ILP, se mostrou eficiente tanto em encontrar um local promissor no espaço de busca, quanto por efetivamente explorá-lo.

II. Adoção de Redes Bayesianas como Modelos Probabilísticos: embora as estruturas das redes Bayesianas não sejam modificadas durante as sucessivas gerações dos algoritmos, elas são poderosas o suficiente para gerar teorias acuradas sem, no entanto, comprometer o tempo de execução dos sistemas.

III. Simples Atualização do Modelo Probabilístico: a atualização das redes Bayesianas em direção a melhor teoria da população se mostrou bastante eficiente tanto em tempo de execução quanto na complexidade e acurácia das teorias evoluídas.

IV. Evolução de Teorias: de forma geral, as teorias desenvolvidas pelos sistemas-AED são mais simples, enquanto mantém e até mesmo superam a acurácia das teorias desenvolvidas pelo Aleph e Progol (com algumas de suas variantes).

O sistema **AED-ILP** se mostrou bastante competitivo em relação ao Aleph; ele obtém acurácias estatisticamente significantes em duas bases de dados (Carc e Amine), perde em duas (Toxic e Pyrimidines) e obtém acurácias equivalentes em todas as outras. O potencial da nova abordagem híbrida de evolução pôde ser claramente visualizado, uma vez que em relação ao HAED-ILP, o AED-ILP geralmente usa mais tempo para desenvolver teorias pouco mais simples, mas bem menos acuradas. Já em relação aos problemas de Transição de Fase, fica clara a força do AED-ILP frente aos sistemas AG-ILP, Progol-A*, Progol-QG, Progol-GA, e Progol-QG/GA, já que tal sistema claramente supera estes últimos.

O **RAED-ILP** é o sistema que, de forma geral, obtém teorias mais simples usando o menor tempo de execução. Seus resultados de acurácia são consideravelmente piores em relação ao HAED-ILP, mas bem competitivos em relação ao AED-ILP, uma vez que ele obtém acurácias equivalentes em relação a este último sistema. Em relação ao Aleph, as acurácias do RAED-ILP são também bem competitivas: tal sistema obtém um resultado estatisticamente significante (para a base Carc), perde em duas (Toxic e Pyrimidines), e para todas as outras, as acurácias são equivalentes. Note, contudo, que o RAED-ILP usa menos tempo (com exceção do Pyrimidines) para obter teorias mais simples que o Aleph.

Os resultados experimentais para o **HAED-ILP** provaram que o novo método híbrido de evolução é um mecanismo de busca muito competente, uma vez que o HAED-ILP obteve acurácias estatisticamente significantes em todas as bases de dados em relação a todos os sistemas avaliados (Aleph, AED-ILP, RAED-ILP, e AG-ILP). Mesmos para os casos Pyrimidines e Toxic, onde o método de cobertura se comportou melhor que a busca por teorias dos sistemas AED-ILP e RAED-ILP, o método de busca híbrido se mostrou eficiente, uma vez que superou os bons resultados de acurácia obtidos pelo Aleph. É interessante notar que estes melhores valores de acurácia não foram obtidos à custa de um aumento significativo no tempo de execução e na

complexidade das teorias; pelo contrário, examinando os resultados obtidos pelo HAED-ILP, pode-se notar que a complexidade das teorias deste sistema é bem competitiva em relação aos resultados do Aleph e do AED-ILP, mas bem menor às teorias desenvolvidas pelo AG-ILP. De fato, verificando-se os resultados para as bases de dados Pyrimidines, Proteins, e Yeast, nota-se que as teorias do HAED-ILP são bem mais simples que as teorias desenvolvidas pelo Aleph. De forma similar, considerando os tempos de execução, o HAED-ILP é muito competitivo em relação aos outros sistemas: em relação ao Aleph, o HAED-ILP é mais rápido em três bases de dados do mundo real, e em relação ao AED-ILP, o referido sistema é mais rápido em seis problemas do mundo real. Já em relação ao AG-ILP, o HAED-ILP é mais rápido para todas as bases de dados consideradas.

6.3 Trabalhos Futuros

A seguir são apresentados alguns trabalhos futuros a serem realizados.

- **Adaptação das Redes Bayesianas:** como visto, o trabalho utiliza uma variação da regra do PBIL para atualizar as TPCs das redes Bayesianas. O próximo passo é utilizar modos de atualização mais complexos, tal como o usado no BOA, e comparar os resultados obtidos por diferentes técnicas de atualização. Pode-se, também, permitir que as estruturas das redes Bayesianas sejam modificadas durante o ciclo de evolução.
- **Usar algoritmo de busca local**: pode-se usar um algoritmo de busca local estocástica para auxiliar os sistemas-AED a encontrar melhores teorias. Tal método será inspirado em (PITANGUI, ZAVERUCHA, 2008).
- **Realizar otimização multiobjetiva:** podem-se utilizar mecanismos para otmização multiobjetiva, uma vez que em ILP têm-se vários objetivos conflitantes, tais como, por exemplo, o tamanho da teoria e sua acurácia.
- **Aproximação de aptidão**: podem-se criar mecanismos para que a aptidão dos indivíduos seja aproximada em vez de realmente calculá-las, uma vez que tal cálculo é responsável pela maior quantidade de tempo gasto pelo sistema. A aproximação de aptidão pode ser realizada de certas em certas gerações.
- **Construção de comitê (ensemble)**: pode-se criar um comitê de teorias selecionadas durantes as sucessivas gerações dos sistemas-AED com objetivo de melhorar a acurácia preditiva de tais sistemas. Em consequencia, poder-se-á

comparar tais comitês com os comitês desenvolvidos pelo Gleaner usando o OS. Pretende-se inspirar em (GAGNÉ et al., 2007) para se realizar tal trabalho.

6.4 Resumo do Capítulo

Este capítulo apresentou algumas conclusões sobre os sistemas propostos. De forma geral, os experimentos mostram que o AED-ILP é um sistema bem competitivo em relação ao Aleph e ao Progol (e suas variantes) em problemas do mundo real e de Transição de Fase. Já o RAED-ILP chama atenção devido à simplicidade das teorias que ele desenvolve usando baixo tempo computacional. Contudo, o sistema que claramente se destaca é o HAED-ILP, uma vez que ele obtém as teorias mais acuradas dentre todos os sistemas avaliados, sem, no entanto, que estas sejam muito complexas, e sem necessitar de um grande tempo computacional para desenvolvê-las.

Referências Bibliográficas

AGUILAR-RUIZ, J. S. GIRALDEZ, R. RIQUELME, J. C., 2007, "*Natural Encoding for Evolutionary Supervised Learning*", Evolutionary Computation**,** Vol. 11, Issue: 4 Aug. 2007, pp. 466-479.

ALPHONSE, E., OSMANI, A., 2009, Empirical Study of Relational Learning Algorithms in the Phase Transition Framework. ECML/PKDD (1) 2009: 51-66

ALPHONSE, E., ROUVEIROL, C., 2000, Lazy propositionalisation for Relational Learning, 14th European Conference on Artificial Intelligence 2000 (ECAI'00), pages 256-260, IOS Press

ALPHONSE, E., ROUVEIROL, C., 2006, Extension of the Top-Down Data-Driven Strategy to ILP. Int. Conference on 16th International Conference on Inductive Logic Programming (ILP 2006), LNAI 4455, p. 49-63, 2006, Springer

AMABIS J. M., MARTHO G., R., 1994, Biologia das Populações: Genética, Evolução e Ecologia, São Paulo, 3ª edição, Editora Moderna.

ANGLANO, C., GIORDANA, A., BELLO, G. L., SAITTA, L., 1998 "*An experimental evaluation of co-evolutive concept learning"*. In Proc. 15th International Conf. on Machine Learning, pages 19.27. Morgan Kaufmann, San Francisco, CA.

AUGIER, S., VENTURINI, G., KODRATOFF, Y., 1995 "*Learning First order logic rules with a genetic algorithm".* In Fayyad, U. M. and Uthurusamy, R., editors, The First International Conference on Knowledge Discovery and Data Mining, pages 21.26, Montreal, Canada. AAAI Press.

BACARDIT J., GARRELL J. M., 2003, "*Bloat control and generalization pressure using the minimum description length principle for a pittsburgh approach learning classifier system*", Sixth International Workshop on Learning Classifier Systems (IWLCS-2003), Chicago, July 2003.

BACK, T., 1995, *Evolutionary algorithms in theory and practice, USA,* Oxford University Press, 1995.

BAKER J., E., 1985, "*Adaptive Selection Methods for Genetic Algorithms*", First International Conference on Genetic Algorithms and their Applications, San Mateo: Morgan Kaufmann, pp. 101-111, 1985.

BALUJA, S., 1994, Population-based incremental learning: A method for integrating genetic search based function optimization and competitive learning. Tech. Rep. No. CMU-CS-94-163, Carnegie Mellon University, Pittsburgh, PA.

BALUJA, S., DAVIES, S., 1997, Using optimal dependency-trees for combinatorial optimization: Learning the structure of the search space. In Proceedings of the 14th International Conference on Machine Learning (pp. 30--38). Morgan Kaufmann.

BLOCKEEL, H., DE RAEDT, L., 1998, Top-down induction of first order logical decision trees. Artificial Intelligence, 101(1-2):285-297.

BOTTA, M., GIORDANA, A., SAITTA, L., & SEBAG, M., 2003, Relational learning as search in a critical region. Journal of Machine Learning Research, 4, 431–463.

BRATKO, I., 1999, Refining complete hypotheses in ILP. Proc. *ILP'99 (9th Int. Workshop on Inductive logic programming)*, Bled, Slovenia, June 1999 (Lecture notes in computer science, Lecture notes in artificial inteligence, 1634). Berlin: Springer, 1999, pp. 44-55.

BRINDLE, A., 1981, *Genetic Algorithms for Function Optimization*, PhD thesis, University of Alberta, Edmonton, Canada, 1981.

CASANOVA, M. A., GIORNO, F., FURTADO, A., 1987, Programação em Lógica e a Linguagem Prolog, Edgard Blücher.

BRISTOL D.W., WACHSMAN J.T., AND GREENWELL A. (1996) The NIEHS Predictive-Toxicology Evaluation Project. *Environmental Health Perspectives*, pages 1001-1010, 1996. Supplement 3.

CHIH-HSUN C.; JOU-NAN C., 2000, "*Genetic algorithms: initialization schemes and genes extraction***",** The Ninth IEEE International Conference on Fuzzy Systems, FUZZ IEEE. Volume 2, Page(s):965 - 968 vol.2, 2000

COHEN W., MCCALLUM A., ROWEIS S., 2008, Machine Learning, Proceedings of the Twenty-Fifth International Conference (ICML 2008), Helsinki, Finland.

DE BONET, J. S., ISBELL, C. L., & VIOLA, P., 1997, MIMC: Finding optima by estimating probability densities. In Mozer, M. C., Jordan, M. I., & Petsche, T. (editors), Advances in Neural Information Processing Systems, Volume 9 (pp. 424). The MIT Press, Cambridge.

DIVINA, F., 2004, Hybrid Genetic Relational Search for Inductive Learning. Ph.D. Thesis.

DIVINA, F., MARCHIORI, E., 2002 "*Evolutionary concept learning"*. In GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, pages 343.350, New York. Morgan Kaufmann Publishers.

DUBOC, A, L, 2008, "Utilizando a Cláusula Mais Específica e Declaração de Modos na Revisão de Teorias de Primeira-Ordem a Partir de Exemplos". Dissertação de Mestrado, COPPE – UFRJ, Rio de Janeiro, RJ.

DUBOC, A, L., PAES, A., ZAVERUCHA, G., 2009, "Using the Bottom Clause and Modes Declarations on FOL Theory Revision from Examples." *in* Machine Learning Journal, v. 76, p. 73-109.

FERRO, M., PAIVA, F., MONARD, M.,C.,"O Sistema de Programação em Lógica Indutiva Aleph – Características e Funcionamento.", Technical Report 300, Universidade de São Paulo.

FOGEL, D. B., 2000, *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence* IEEE Press, New York, 2000.

Gagné, C., Sebag, M., Schoenauer, M., Tomassini, M., 2007, Ensemble Learning for Free with Evolutionary Algorithms ? Dirk Thierens et al., editor, *Proc. of Genetic and Evolutionary Conference*, pages 1782-1789. ACM SIGEVO, ACM.

GALLIER, H., 1986, *Logic for Computer Science: Foundations of Automatic Theorem Proving*. Harper & Row Publishers.

GIORDANA, A., NERI, F., 1996, "*Search-intensive concept induction"*. Evolutionary Computation, 3(4):375.416.

GIORDANA, A., SAITTA, L., 2000, Phase Transitions in Relational Learning. *Machine Learning*, 41(2):217.251.

GOADRICH, M., 2009, Adaptively Finding and Combining First-Order Rules for Large, Skewed Data Sets. PhD thesis, Department of Computer Sciences, University of Wisconsin-Madison.

GOLDBERG, D., 1989, Genetic Algorithms in search, optimization and machine learning. Addison-Wesley, 1989.

GOADRICH, M., OLIPHANT, L., SHAVLIK, J., 2006 Gleaner: Creating Ensembles of First-Order Clauses to Improve Recall-Precision Curves. Machine Learning, 64(1-3):231–261.

GHOSH A, TSUTSUI S., TANAKA H. E CORNE D, 2000, "*Genetic Algorithms with Substitution and Re-entry of Individuals*", International Journal of Knowledge-Based Intelligent Engineering Systems, Vol. 4, No. 1, pp. 64-71, 2000.

HARIK, G., 1999, Linkage learning via probabilistic modeling in the ECGA. IlliGAL Report No. 99010, University of Illinois at Urbana-Champaign, Illinois Genetic Algorithms Laboratory, Urbana, IL.

HARIK, G. R., LOBO, F. G., GOLDBERG, D. E., 1998, The compact genetic algorithm. In Proceedings of the IEEE Conference on Evolutionary Computation 1998 (ICEG'98) (pp. 523-528). Piscataway, NJ: IEEE Service Centre.

HEKANAHO, J., 1996, "*Background knowledge in GA-based concept learning"*. In International Conference on Machine Learning, pages 234.242.

HENRION, M., 1988, Propagating Uncertainty in Bayesian Networks by Probabilistic Logic Sampling. In Lemmer, J.F. and Kanal, L.N. (Eds.) Uncertainty in Artificial Intelligence, 2. North Holland. 149-163.

HOLLAND, J., 1975, Adaptation in natural and artificial systems. MIT Press, Cambridge.

HUYNH T., MOONEY R., 2008, Discriminative Structure and Parameter Learning for Markov Logic Networks. In Proceedings of the 25th International Conference on Machine Learning (ICML-2008), Helsinki, Finland.

KADUPITIGE, S.R., JULIA, K.C.L., SELLMEIER, SIVIENG, J., CATCHPOOLE, D.R., BAIN, M., GAETA, B.A., 2009, MINER: Exploratory Analysis of Gene Interaction Networks by Machine Learning from Expression Data. BMC Genomics 10(Suppl 3).

KING, R.D., MUGGLETON, S., STERNBERG, M., 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. Proceedings of the National Academy of Sciences 89(23), 11,322–11,326.

KING R.D., SRINIVASAN A., STERNBERG M. J. E., 1995, Relating chemical activity to structure: an examination of ILP successes. New Gen. Comp., 13:411–433, 1995.

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

LLOYD, J., 1987, Foundations of Logic Programming, 2 ed., Springer Verlag.

MICHALEWICZ, Z., 1999, Genetic Algorithms + Data Structures = Evolution Programs, 3rd edition, Springer-Verlag.

MITCHELL, T., 1997, Machine learning. New York: McGraw-Hill.

MUGGLETON, S., 1991, "Inductive Logic Programming", New Generation Computing, v. 13, n. 4, pp. 245-286.

MUGGLETON, S., 1995, "Inverse Entailment and Progol", New Generation Computing, v. 13, n. 4, pp. 245-286.

MUGGLETON, S., 1995, Inverse entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming, 13(3-4):245–286.

MUGGLETON, S., 1996, Learning from positive data. In Muggleton, S., editor, Proceedings of the 6th International Workshop on Inductive Logic Programming, pages 225–244. Stockholm University, Royal Institute of Technology.

MUGGLENTON, S., 2001, http://www.doc.ic.ac.uk/~shm/applications.html, último acesso realizado em 18/07/2010.

MUGGLETON, S., 2005, Machine Learning for Systems Biology. In: Proceedings of the 15th International Conference on Inductive Logic Programming (ILP-05), Lecture Notes in Computer Science, vol. 3625, pp. 416–423. Springer

MUGGLETON, S. H., FENG, C., 1990. Efficient induction of logic programs. In Proceedings of the first conference on algorithmic learning theory (pp. 368–381). Tokyo: Ohmsha.

MUGGLETON, S.., KING, R.D., STERNBERG, M.J.E., 1992, Protein Secondary Structure Prediction Using Logic-Based Machine Learning. Protein Engineering 5(7), 647–657.

MUGGLETON, S., SANTOS, J., TAMADDONI-NEZHAD, A., 2010-a, TopLog: ILP using a logic program declarative bias. In Proceedings of the International Conference on Logic Programming 2008, LNCS 5366, pages 687-692. Springer-Verlag.

MUGGLETON, S., SANTOS, J.C.A., TAMADDONI-NEZHAD, A., 2010-b, ProGolem: 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., DE RAEDT, L., 1994, "Inductive Logic Programming: Theory and Methods", Journal of Logic Programming, v. 19, n. 20.

MUGGLETON, S., TAMADDONI-NEZHAD A., 2006, QG/GA: A stochastic search approach for Progol. In Proceedings of the 16th International Conference on Inductive Logic Programming, LNAI 4455, pages 37-39. Springer-Verlag.

MUGGLETON, S., TAMADDONI-NEZHAD A., 2007, QG/GA: A stochastic search for Progol. *Machine Learning*, 70(2-3):123-133, 2007.

MÜHLENBEIN, H., & PAAß, G., 1996, From recombination of genes to the estimation of distributions I. Binary parameters. Parallel Problem Solving from Nature, eds. Voigt, H.-M and Ebeling, W. and Rechenberg, I. and Schwefel, H.-P., LNCS 1141, Springer:Berlin, (pp. 178-187).

NADEAU C., BENGIO Y., 2003, "Inference for the Generalization Error", Machine Learning 52(3) pp. 239-281.

NIENHUYS-CHENG, S., WOLF, R., SIEKMANN, J., CARBONELL, J., 1997 Foundations of Inductive Logic Programming, Springer-Verlag New York, Inc., Secaucus, NJ, 1997

OCHOA G., HARVEY I., BUXTON, H., "*Optimal Mutation Rates and Selection Pressure in Genetic Algorithms*", Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2000), Morgan Kaufmann, pp. 93-101 San Francisco, CA, 2000.

OLIPHANT, L., SHAVLIK, J. 2007, Using Bayesian Networks to Direct Stochastic Search in Inductive Logic Programming. Proceedings of the 17th International Conference on Inductive Logic Programming.

PELIKAN, M, 2005, Hierarchical Bayesian Optimization Algorithm Toward a New Generation of Evolutionary Algorithms, Series: Studies in Fuzziness and Soft Computing, Vol. 170, Springer; 1 edition.

PELIKAN, M., GOLDBERG, D. E., AND CANTU-PAZ, E., 1999, BOA: The Bayesian optimization algorithm. Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99), I:525–532.

PELIKAN, M., GOLDBERG, D. E., LOBO, F. G. 1999, "A *survey of optimization by building and using probabilistic models"*. Urbana, IL: University of Illinois Genetic Algorithms Laboratory (IlliGAL Report No. 99018).

PELIKAN, M., MÜHLENBEIN, H., 1999, The bivariate marginal distribution algorithm. In Roy, R., Furuhashi, T., & Chawdhry, P. K. (Eds.), Advances in Soft Computing - Engineering Design and Manufacturing (pp. 521--535). London: Springer-Verlag.

PEARL, J., Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

PITANGUI, C., G., ZAVERUCHA, G., 2006, Genetic Based Machine Learning: Merging Pittsburgh and Michigan, an Implicit Feature Selection Mechanism e a New Crossover Operator. 6th International Conference on Hybrid Intelligent Systems. Auckland, New Zealand, 2006.

PITANGUI, C., G., ZAVERUCHA, G., 2007, Improved Natural Crossover Operators in GBML. In: IEEE Congress on Evolutionary Computation (CEC), Cingapura. IEEE Congress on Evolutionary Computation (CEC 2007), p. 2157-2164

PITANGUI, C., G., ZAVERUCHA, G., 2008, Genetic Local Search for Rule Learning. In: ACM Genetic and Evolutionary Computation Conference (GECCO-2008), 2008, Atlanta, Georgia. ACM Genetic and Evolutionary Computation Conference (GECCO-2008). p. 1427-1428.

PITANGUI, C., G., ZAVERUCHA, G., 2010, Algoritmo Genético Construtor de Modelo Probabilístico Aplicado à Programação em Lógica Indutiva. In: Joint Conference SBIA/SBRN/JRI 2010 / III Workshop on Computational Intelligence, 2010, São Bernardo do Campo. Joint Conference SBIA/SBRN/JRI 2010 / III Workshop on Computational Intelligence, 2010. p. 1-6.

PITANGUI, C., G., ZAVERUCHA, G., 2011, Inductive Logic Programming through Estimation of Distribution Algorithm. In: IEEE Congress on Evolutionary Computation (CEC), 2011, New Orleans. IEEE Congress on Evolutionary Computation (CEC 2011), 2011. p. 54-61.

PITANGUI, C., G., ZAVERUCHA, G., 2012, Learning Theories Using Estimation Distribution Algorithms and (Reduced) Bottom Clauses. In: 21st International Conference on Inductive Logic Programming (ILP 2011), 2012, Windsor. Leccture Notes in Artificial Intelligence. Heidelberg: Springer, 2012. v. 7207. p. 286-301.

QUINLAN, J., 1990, Learning logical definition from relations. *Machine Learning*, 5:239.266.

ROBINSON, A., 1965, A Machine-Oriented Logic Based on the Resolution Principle, Journal of the ACM, vol.12, pp. 23-41.

RUSSELL, S., NORVIG, P., 2003, Artificial Intelligence: A Modern Approach, Prentice-Hall, Englewood Cliffs, NJ.

RÜCKERT, U., KRAMER, S., 2003, Stochastic local search in k-term dnf learning. In Proc. of the 20th ICML, p. 648-655.

SEBESTA, R., 1996, Concepts of programming languages (3rd ed.), Addison Wesley Longman Publishing Co., Inc., Redwood City, CA.

SRINIVASAN. A., 2003, http://www.comlab.ox.ac.uk/activities/machinelearning/Aleph/ último acesso realizado em 18/07/2010.

SRINIVASAN, A., KING R.D. S.H. MUGGLETON S, STERNBERG M., 1997, Carcinogenesis predictions using ILP. In Proceedings of the Seventh International Workshop on ILP, pages 273–287. Springer-Verlag, Berlin, LNAI 1297.

SPELLMAN, P., SHERLOCK, G., ZHANG, M., IYER, V., ANDERS, K., EISEN, M., BROWN, P., D.BOTSTEIN, B.FUTCHER , 1998, Comprehensive Identification of Cell Cycle-Regulated Genes of the Yeast Saccharomyces Cerevisiae by Microarray Hybridization. Molecular Biology of the cell 9(12), 3273–3297.

SYSWERDA, G., 1989, "Uniform Crossover in Genetic Algorithms", Third International Conference on Genetic Algorithms and their Applications, San Mateo, Morgan Kaufmann, pp. pages 2-9, 1989.

WIELEMAKER, J., 2003, An overview of the SWI-Prolog Programming Environment. Proceedings of the 13th International Workshop on Logic Programming Environments, pp. 1-16, Fred Mesnard and Alexander Serebenik, Katholieke Universiteit Leuven.

YAP PROLOG: http://www.dcc.fc.up.pt/~vsc/Yap/, ultimo acesso realizado em 18/07/2010.

ZAVERUCHA, G., 2008, "Apostila de Lógica – Notas de Aula" – material distribuído nos cursos cos230 e cos705 do programa de pós-graduação em Engenharia de Sistemas e Computação na COPPE/UFRJ.

ZELEZNY, F., SRINIVASAN, A., PAGE, D., Lattice-Search Runtime Distributions may be Heavy-Tailed. In Proceedings of the 12th International Conference on Inductive Logic Programming 2002, volume 2583 of Lecture Notes in Artificial Intelligence, pages 333-345, Sydney, Australia, 2003.

Apêndice A: Algoritmo de Busca Local Estocástica para Aprendizado de Regras Proposicionais

Algoritmo A.1: Busca Local Estocástica para Regras Proposicionais

Entrada:
(teoria): teoria composta por uma ou mais regras proposicionais; **(p_1, p_2, p_3)**: três valores de probabilidades; **(aptMáxima)**: valor de aptidão que controla o ciclo de execução do algoritmo; ***(maxPassos)**: número de passos que controla o ciclo de execução do algoritmo;* **(BD)**: base de dados composta por exemplos positivos e negativos;

Variáveis Locais:
(regra): uma regra proposicional pertencente à teoria; **(atributo)**: um atributo pertencente a uma regra de uma teoria; **(numPassos)**: contador de número de ciclos executados; (**exemplo**): um exemplo de BD classificado incorretamente;

Saída:
teoria modificada;

Começo-Busca-Local-Estocástica:
01- *numPassos* := 0;
02- Enquanto ((Aptidão(*teoria*) < *aptMáxima*) E (*numPassos* < *maxPassos*)) faça:
03- *exemplo* := um exemplo de *BD* classificado incorretamente por *teoria*;
04- Se *exemplo* é um exemplo positivo então:
05- *regra :=* (com probabilidade p_1) uma regra aleatória de *teoria; caso* contrário, a regra que difere a menor quantidade de atributos de *exemplo*;
06- *atributo* := (com probabilidade p_2) um atributo aleatório utilizado em *regra;* caso contrário, o atributo utilizado em regra, que, quando não utilizado, melhora a aptidão de *teoria* ao máximo;
07- *teoria* := *teoria* com *atributo* removido (não mais utilizado);
08- fim-se;
09- Se *exemplo* é um exemplo negativo então:
10- *regra :=* uma regra aleatória de *teoria* que cobre *exemplo*;
11- *atributo* := (com probabilidade p_3) um atributo aleatório não utilizado em *regra*, que, quando utilizado, faz com que *regra* não cubra mais *exemplo;* caso contrário, o atributo não utilizado em *regra*, que, quando utilizado, melhora a aptidão de *teoria* ao máximo;
12- *teoria* := *teoria* com *atributo* adicionado (utilizado);
13- fim-se;
14- *numPassos* := *numPassos* + 1;
15- fim-enquanto;
16- Retorne *teoria*;
11- Fim-Busca-Local-Estocástica.

O algoritmo A.1 é bastante simples de ser entendido. Basicamente, ele generaliza a *teoria* ou a especializa, dependendo do *exemplo* incorretamente classificado que é selecionado aleatoriamente no passo 3. Assim, têm-se dois blocos principais em A.1, a saber: o bloco de generalização (passos 4 a 8) que generalizará *teoria* caso *exemplo* seja positivo, e o bloco de especialização (passos 9 a 13) que especializará *teoria* caso *exemplo* seja negativo. Tais blocos serão explicados separadamente a seguir.

Bloco de generalização: o primeiro passo deste bloco, passo 5, atribui à *regra*, com probabilidade p_1, uma regra aleatória em *teoria*, ou com probabilidade 1- p_1, a regra mais próxima a classificar corretamente *exemplo*. Uma vez escolhida *regra*, o passo 6 seleciona um atributo de *regra* que atualmente é utilizado e fará com que ele não seja mais utilizado (generalização da regra). Para esta escolha, têm-se duas opções: (1) com probabilidade p_2 escolhe-se aleatoriamente um atributo que é utilizado atualmente, ou, (2) com probabilidade 1 - p_2 escolhe-se o atributo utilizado atualmente, que, quando não utilizado, faz com que a aptidão de *teoria* melhore ao máximo. Neste último caso, todos os atributos utilizados em *regra* devem ser testados, um a um, para se determinar o atributo que quando não utilizado, melhora da melhor forma possível a aptidão de *teoria*. Ao fim deste bloco, a *teoria* é então generalizada (passa a não utilizar um atributo que antes era utilizado).

Bloco de especialização: o primeiro passo deste bloco (passo 10) atribui à *regra* uma regra em *teoria* que cobre o exemplo negativo classificado incorretamente. Uma vez escolhida *regra*, o passo 10 escolhe um atributo de *regra* que atualmente não é utilizado e fará com que ele seja utilizado (especialização da regra). Para esta escolha, têm-se duas opções: (1) com probabilidade p_3 escolhe-se aleatoriamente um atributo não utilizado atualmente, que, quando utilizado, fará com que *regra* não cubra mais *exemplo*, ou, (2) com probabilidade 1 - p_3 escolhe-se o atributo não utilizado atualmente, que, quando utilizado, faz com que a aptidão de *teoria* melhore ao máximo. Neste último caso, todos os atributos não utilizados em *regra* devem ser testados, um a um, para se determinar o atributo que quando utilizado, melhora da melhor forma possível a aptidão de *teoria*. Ao fim deste bloco, a *teoria* é então especializada (passa a utilizar um atributo que antes não era utilizado).

O número de tentativas de se generalizar ou de se especializar uma regra é controlado, no passo 2, pelos parâmetros de entrada *aptMáxima* e *maxPassos*. Claramente, o algoritmo irá executar no máximo *maxPassos* tentativas de se melhorar

teoria (note o incremento de *numPassos* no passo 14 para cada ciclo do algoritmo), sendo que caso o algoritmo encontre uma teoria cuja aptidão seja melhor ou tão boa quanto *aptMáxima,* o procedimento é finalizado antes de se executar *maxPassos* ciclos.

O método apresentado foi aplicado ao nosso sistema genético de aprendizado de regras proposicionais com objetivo de auxiliá-lo a explorar de maneira mais efetiva o espaço de busca. Resumidamente, os resultados mostraram que o sistema, utilizado o método proposto, desenvolve teorias mais acuradas e mais simples usando menos tempo em relação ao sistema sem o uso da busca local estocástica (vide (PITANGUI, ZAVERUCHA, 2008)).

Apêndice B: AGs como Algoritmos de Busca e Justificativa de Funcionamento

B.1 Algoritmos Genéticos como um Algoritmo de Busca[29]

Esta seção dedica-se a apresentar de que forma um AG pode ser considerado um algoritmo de busca e desta forma ser aplicado a este tipo de problema. Uma vez que será apresentada tal forma de visualização de um AG, se faz necessário a apresentação do conceito de problema de busca. Desta forma, esta seção apresenta, primeiramente, problema de busca, para posteriormente apresentar uma breve revisão de alguns algoritmos de busca objetivando situar os AGs entre eles.

I. Problemas de Busca

Como ilustração de um problema de busca, o clássico problema de 8 rainhas será apresentado. Antes, contudo, se faz necessária a apresentação dos principais componentes de um problema de busca:

- Estado em que um agente[30] começa - denominado de estado inicial.
- Uma descrição das ações possíveis que estão disponíveis para o agente. A formulação mais comum utiliza a função sucessor. Dado um estado particular *x*, Sucessor(*x*) retorna um conjunto de pares ordenados (ação, sucessor) em que cada ação é uma das ações válidas no estado *x* e cada sucessor é um estado que pode ser alcançado a partir de *x* aplicando-se a ação.
- O espaço de estados que é definido implicitamente pelo estado inicial e a função Sucessor. O espaço de estado nada mais é que o conjunto de estados acessíveis a partir do estado inicial.
- Caminho no estado de espaços: sequência de estados conectados por uma seqüência de ações.
- Teste de Objetivo: determina se um dado estado é o objetivo a ser alcançado.
- Custo de Caminho: atribui um custo numérico a cada caminho.

[29] Todo o conteúdo desta seção, incluindo definições, exemplos e algoritmos foram retirados de (RUSSEL, NORVIG, 2003).

[30] Um agente é tudo que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por meio de atuadores.

O objetivo do problema das 8 rainhas é posicionar 8 rainhas em um tabuleiro de xadrez de tal forma que nenhuma rainha ataque qualquer outra. Uma rainha ataca qualquer peça situada na mesma linha coluna ou diagonal. A figura B.1 ilustra uma tentativa em que a solução falhou: a rainha da coluna mais a direita é atacada pela rainha do canto superior esquerdo.

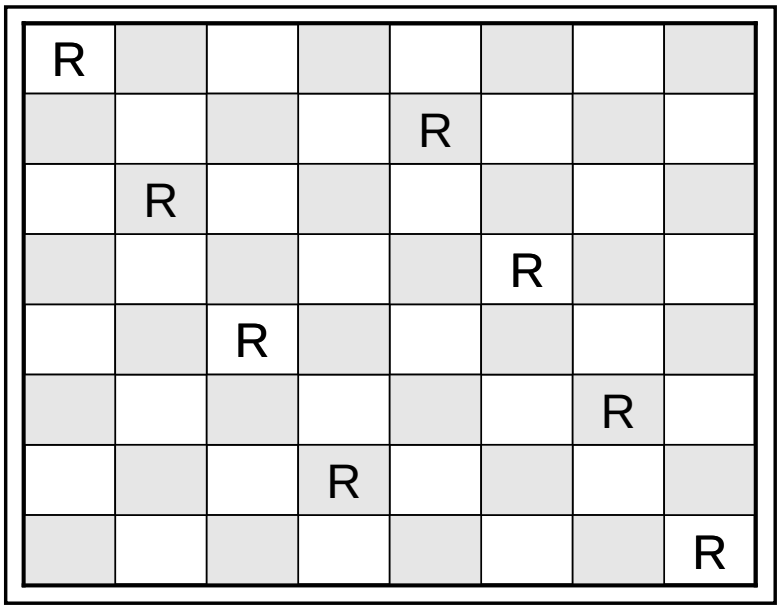

Figura B.1: Resolução falha do problema das 8 rainhas

É interessante perceber que existem dois tipos de formulações para o problema de 8 rainhas, e, de modo geral, para qualquer outro problema de busca. As formulações são conhecidas como formulação incremental e formulação de estados completos:

Formulação Incremental: envolve operadores que ampliam a descrição dos estados, iniciando com um estado vazio. Para o problema de 8 rainhas, isso significa que cada ação acrescenta uma rainha ao estado.

Formulação de Estados Completos: começa com todas as rainhas e as desloca pelo tabuleiro.

A título de esclarecimento, a seguir exibem-se uma formulação incremental e uma formulação de estados completos para o problema de 8 rainhas.

Formulação Incremental: uma formulação incremental poderia ser dada por:

- Estados: qualquer disposição de 0 a 8 rainhas no tabuleiro.
- Estado Inicial: nenhuma rainha no tabuleiro.
- Função Sucessor: posicionar a rainha em qualquer quadrado vazio.
- Teste de Objetivo: as oito rainhas estão no tabuleiro e nenhuma é atacada.

Formulação de Estados Completos: uma formulação de estados completos poderia ser dada por:

- Estados: qualquer disposição de 8 rainhas no tabuleiro.
- Estado Inicial: todas as rainhas no tabuleiro.
- Função Sucessor: retorna todos os estados possíveis gerados pela movimentação de uma única rainha para outro quadrado na mesma coluna que a rainha ocupa.
- Teste de Objetivo: as oito rainhas estão no tabuleiro e nenhuma é atacada.

II. Uma breve revisão de alguns algoritmos de Busca

Uma vez entendido o conceito de problema de busca, esta seção apresenta uma breve revisão dos algoritmos de busca mais importantes. Os algoritmos revisados são: busca de Subida de Encosta (*Hill Climbing*), busca de Subida de Encosta Estocástica (*Stochastic Hill Climbing*), busca de Subida de Encosta pela Primeira Escolha (*First Choice Hill Climbing*), busca Subida de Encosta com Reinício Aleatório (*Random-restart Hill Climbing*), e busca de Têmpera Simulada (*Simulated Annealing*).

A. Busca de Subida de Encosta (*Hill Climbing*)

Este algoritmo é constituído de apenas um laço repetitivo que se move de forma contínua no sentido do valor crescente – isto é, encosta acima. O algoritmo termina quando alcança um "pico" em que nenhum vizinho tem valor mais alto. A busca subida da encosta não examina antecipadamente valores de estados além dos vizinhos imediatos do estado corrente. O algoritmo B.1 exibe o processo de subida de encosta.

Algoritmo B.1 - Algoritmo Subida de Encosta

```
Entrada:
    Problema de busca;
Variáveis Locais:
    nóCorrente, nóVizinho;
Saída:
    estado que é um máximo local;
Começo-Subida-de-Encosta:
01- nóCorrente := Criar_nó(Estado-Inicial(problema));
02- Repita:
03-   nóVizinho := um sucessor corrente com valor mais alto;
04-   Se (Valor(nóVizinho) ≤ Valor(nóCorrente)) então:
05-     Retorne estado(nóCorrente);
06-     Finalize Algoritmo;
07-   fim-se;
08- nóCorrente ← nóVizinho;
08- fim-repita;
Fim-Subida-de-Encosta.
```

A aplicação deste algoritmo para o problema de 8 rainhas, segue a formulação de estados completos tal como discutido anteriormente. O algoritmo tem que minimizar a função de custo que retorna a quantidade de pares de rainhas que estão atacando umas às outras, direta ou indiretamente. O mínimo global desta função é 0, que ocorre quando uma das soluções perfeitas é encontrada, ou seja, quando o teste de objetivo é tido como verdadeiro.

Frequentemente, a Subida de Encosta funciona muito bem, já que ela progride com grande rapidez em direção a uma solução, porque normalmente é bem fácil melhorar um estado ruim. Infelizmente, a Subida de Encosta fica paralisada em máximos locais, picos e platôs. Na tentativa de solucionar o problema de paralisação do algoritmo de subida de encosta, algumas variações deste algoritmo foram propostas. Estas variações são:

Busca de Subida de Encosta Estocástica (*Stochastic Hill Climbing*): a Subida de Encosta Estocástica escolhe ao acaso entre os movimentos encosta acima sendo que a probabilidade de seleção pode variar com a declividade do movimento encosta acima.

Em geral, a Subida de Encosta Estocástica converge mais lentamente que a Subida de Encosta, mas em algumas topologias de estados encontra soluções melhores.

Busca de Subida de Encosta pela Primeira Escolha (*First Choice Hill Climbing*): a Subida de Encosta pela Primeira Escolha implementa a subida de Encosta Estocástica gerando sucessores ao acaso, até ser gerado um sucessor melhor que o estado corrente. Essa é uma estratégia muito utilizada quando um estado possui muitos sucessores (milhares).

Busca de Subida de Encosta com Reinício Aleatório (*Random-restart Hill Climbing*): os algoritmos de subida de encosta até agora citados são incompletos, i.e., deixam de encontrar um objetivo que existe porque podem ficar paralisados em máximos locais. A Subida de Encosta com Reinicio Aleatório conduz uma série de buscas a partir de estados iniciais gerados ao acaso, parando ao encontrar um objetivo. Ela é completa com probabilidade próxima de 1, já que eventualmente irá gerar um estado objetivo como estado inicial.

O sucesso dos algoritmos de subida de encosta depende muito da forma da topologia do espaço de estados. Se houver poucos máximos locais e platôs, a Subida de Encosta com Reinicio Aleatório encontrará uma solução com muita rapidez. Contudo, em geral, para os problemas que apresentem um grande número de máximos locais, esta gama de algoritmos fica paralisada. Apesar disso, um máximo local razoavelmente bom pode ser encontrado depois de um pequeno número de reinícios.

B. Busca de Têmpera[31] Simulada (*Simulated Annealing*)

O algoritmo de Têmpera Simulada pode ser visto como uma versão de Subida de Encosta Estocástica em que alguns movimentos encosta abaixo são permitidos. Movimentos encosta abaixo são aceitos com maior frequência no inicio do escalonamento da têmpera e depois com menos frequência com o passar do tempo. Dessa forma, este algoritmo está menos sujeito a ficar paralisado em máximos locais quando comparado aos algoritmos de Subida de Encosta. Seu funcionamento é apresentado no algoritmo B.2.

[31] Em metalurgia, a têmpera é o processo usado para temperar ou endurecer metais e vidros aquecendo-os em alta temperatura e depois os esfriando gradualmente, permitindo, assim, que o material seja misturado em um estado cristalino e de baixa energia.

Algoritmo B.2 - Algoritmo de Têmpera Simulada

```
Entrada:
    problema;
    escalonamento (um mapeamento de tempo para temperatura);
Variáveis Locais:
    nóCorrente, nóPróximo;
    T (temperatura);
Saída:
    estado solução;
```

```
Começo-Têmpera-Simulada:
01- Para t := 1 até ∞ faça:
02-   T := Escalonamento(t);
03-   Se (T == 0) então:
04-     Retorne nóCorrente;
05-     Finalize Algoritmo;
06-   fim-se;
07-   nóPróximo := um sucessor de nóCorrente selecionado;
      aleatoriamente;
08-   ΔE := Valor(nóPróximo) - Valor(nóCorrente);
09-   Se (ΔE > 0) então:
10-     nóCorrente := nóPróximo;
11-     Senão: nóCorrente := nóPróximo com probabilidade e^(ΔE/T);
12-   fim-se;
13- fim-para;
Fim-Têmpera-Simulada.
```

O laço de repetição do algoritmo apresentado é muito semelhante à subida da encosta. Contudo, em vez de escolher o melhor movimento, ele escolhe um movimento aleatório. Se o movimento melhorar a situação, ele será sempre aceito. Caso contrário o algoritmo aceitará este movimento com uma probabilidade menor que 1. Essa probabilidade diminui exponencialmente com a má qualidade do movimento (o valor ΔE segundo o qual a avaliação piora). A probabilidade também diminui à medida que a temperatura T se reduz. Pode-se provar que se o escalonamento diminuir T com lentidão suficiente, o algoritmo encontrará um valor ótimo global com probabilidade próxima de 1.

C. Busca em Feixe Local (*Beam Search*)

A Busca em Feixe Local mantém *k* estados na memória em vez de somente um como ocorreu nos algoritmos de busca descritos. Ela começa com *k* estados gerados aleatoriamente. Em cada passo são gerados todos os sucessores de todos os *k* estados. Se qualquer um deles for o objetivo, o algoritmo é finalizado. Caso contrário, ele selecionará os *k* melhores sucessores a partir da lista completa e repetirá a ação. A Busca em Feixe Local pode sofrer de falta de diversidade entre os *k* estados - estes podem ficar rapidamente concentrados em uma pequena região do espaço de estados, tornando a busca um pouco melhor que uma versão dispendiosa da Subida de Encosta. Uma variante chamada de Busca em Feixe Estocástica, análoga à Subida de Encosta Estocástica, ajuda a atenuar este problema. Em vez de escolher o melhor *k* a partir do conjunto de sucessores candidatos, a Busca em Feixe Estocástica escolhe *k* sucessores ao acaso, com a probabilidade proporcional ao valor de adaptação ou aptidão.

III. Busca com Algoritmos Genéticos

Um AG pode ser visto como uma variante de Busca em Feixe Estocástica, no qual os estados sucessores são gerados pela aplicação dos operadores de seleção, mutação e recombinação. Como ocorre na Busca em Feixe Estocástica, os AGs começam com um conjunto de *k* estados gerados aleatoriamente (população).

Uma das principais diferenças que deve ser notadas entre as duas técnicas se relaciona à geração dos sucessores. A geração de sucessores na Busca em Feixe Estocástica se faz de forma assexuada, o que difere do AG, já que este apresenta o operador de recombinação que produz novos indivíduos pela troca de seus materiais genéticos, assemelhando-se, portanto, a uma reprodução sexuada. Intuitivamente, a recombinação tem a capacidade de recombinar grandes blocos de genes que evoluem de forma independente para executar funções úteis, elevando assim o nível de granularidade em que a busca opera. Entretanto, pode ser demonstrado matematicamente, que se as posições do código genético forem permutadas inicialmente em ordem aleatória, a recombinação não terá nenhuma vantagem. Assim, sob certas circunstancias, a recombinação é uma das vantagens dos AGs sobre a Busca em Feixe Estocástica, já que tal operador “manipula” uma grande quantidade de genes de uma só vez fazendo com que a busca seja mais “global” quando comparada à Busca em Feixe Estocástica, cujos sucessores são gerados por operadores mais “conservadores”, i.e., operadores que não realizam grandes mudanças nas soluções atuais. Contudo, podem-se

construir operadores de exploração “mais gerais” na para qualquer tipo de busca, e, sob este aspecto, caso tais operadores sejam adotados na Busca em Feixe Local Estocástica, este algoritmo deve exibir um comportamento semelhante ao dos AGs.

B.2 Justificativa de Funcionamento dos AGs[32]

A justificativa teórica do funcionamento dos AGs se baseia a representação dos indivíduos por *strings* binárias e na noção de esquema, que pode ser definido como segue.

Definição B.1 (Esquema): Um esquema é construído pela introdução de símbolos ‘não importa’ (*) no alfabeto de genes.

De forma geral, um esquema representa todas as *strings* (um subconjunto do espaço de busca) que casam (*match*) com ele em todas as posições diferentes de *. Assim, pode-se dizer que um esquema representa um conjunto de *strings* que guardam semelhanças entre si.

Exemplo B.1: Considere *strings* e um esquema de tamanho 10. O esquema S = (* 1 1 1 0 0 1 0 0) casa com duas strings s_1 = (0 1 1 1 1 0 0 1 0 0) e s_2 = (1 1 1 1 1 0 0 1 0 0).

O esquema (1 0 0 1 1 1 0 0 0 1) representa apenas uma *string*, enquanto que o esquema (* * * * * * * * * *) representa todas as possíveis *strings* de tamanho 10. Dessa forma, pode-se notar que todo esquema casa com exatamente 2^r strings, onde *r* é o número de símbolos ‘*’ no *template* do esquema. Em contrapartida, cada *string* de tamanho *m* será casada por 2^m esquemas.

Exemplo B.2: A string (1 0 0 1 1 1 0 0 0 1) é casada pelos 2^{10} esquemas {(1 0 0 1 1 1 0 0 0 1), (* 0 0 1 1 1 0 0 0 1), (1 * 0 1 1 1 0 0 0 1), ..., (* * 0 1 1 1 0 0 0 1), (* 0 * 1 1 1 0 0 0 1), ... (* * * * * * * * * *)}.

Denota-se ξ(S, t) o número de strings na geração *t* que casam com o esquema *S*. Existem varias definições relativas a esquemas. Elas serão apresentadas a seguir:

Definição B.2 (Aptidão de um Esquema): Define-se aptidão de um esquema *S* em uma determinada geração *t*, denotado por apt(S,t), como sendo a média da aptidão das

[32] Todo o conteúdo desta seção se baseou em (MICHALEWICZ, 1999).

strings que são casadas por *S*. Formalmente, assuma que existam *p* strings (s_1, s_2, ..., s_p) na população que casam com o esquema *S* em uma dada geração *t*, desta forma:

$$apt(S,t) = \sum_{j=1}^{p} apt(s_j)/p \qquad \text{(B. 1)}$$

onde s_j representa a *j*-ésima string que foi casada por *S*, e apt(s_j) representa o valor da aptidão da j-ésima string.

Definição B.3 (Ordem de um Esquema): A ordem de um esquema *S*, denotado por o(S) é o número de posições 0 e 1 presentes no esquema.

Em outras palavras, a ordem do esquema é simplesmente seu número de posições fixas, ou ainda, a ordem do esquema é dada pelo tamanho do esquema menos o número de símbolos * que figuram no esquema.

Exemplo B.3: Os esquemas S_1 = (* * * 1 1 0 0 1 0 1), S_2 = (* 1 1 1 1 * * 1 * *), S_3 = (1 1 1 1 * 0 1 * 1 1), possuem, respectivamente, o(S_1) = 7, o(S_2) = 5, o(S_3) = 8.

Definição B.4 (Comprimento de um Esquema): O comprimento de um esquema *S*, denotado por δ(S), é a distância entre a última e a primeira posição fixa de *S*.

Exemplo B.4: Os esquemas S_1 = (* * * 1 1 0 0 1 0 1), S_2 = (* 1 1 1 1 * * 1 * *), S_3 = (1 1 1 1 * 0 1 * 1 1), possuem, respectivamente, δ (S_1) = 6 (10 – 4), δ (S_2) = 6 (8 - 2), δ (S_3) = 9 (10 - 1).

Será mostrado posteriormente que a ordem do esquema servirá para calcular a probabilidade de sobrevivência de um esquema quando a mutação é aplicada, enquanto que o conceito de comprimento servirá para calcular a probabilidade de sobrevivência de um esquema quando a recombinação é aplicada.

Como apresentado anteriormente, viu-se que os passos responsáveis pela evolução da população são (I) seleção, (II) recombinação, e (III) mutação dos indivíduos. Será apresentado como estes três passos fundamentais afetam o número esperado de esquemas na população.

Seleção: durante o passo de seleção, uma população intermediária é criada e, desta forma, cada *string* (ou indivíduo) é copiada zero, uma ou mais vezes de acordo com sua aptidão. Como visto anteriormente, caso o critério de seleção por roleta seja utilizado, a probabilidade (p_i) do indivíduo *i* ser selecionado, é dada por:

$$p_i = apt(s_i)/F(t) \quad \text{(B. 2)}$$

onde, como já apresentado, apt(s) é o valor de aptidão para a string s e F(t) é dado por:

$$F(t) = \sum_{j=1}^{N} apt(s_j) \quad \text{(B. 3)}$$

em que **N** é o tamanho da população.

Após o passo de seleção, espera-se ter ξ(S, t + 1) strings *casadas* pelo esquema *S*. Uma vez que: (1) para uma dada *string* s casada pelo esquema *S*, a sua probabilidade de seleção é igual a apt(S,t)/F(t); (2) o número de strings casadas pelo esquema *S* é ξ(S, t); (3) o número de seleções efetuadas é igual ao tamanho da população (*N*), pode-se mostrar que:

$$\xi(S, t+1) = \xi(S, t) \times N \times apt(S, t)/F \quad \text{(B. 4)}$$

Considerando-se que a média da aptidão da população pode ser expressa por $\overline{F(t)} = F(t)/N$, pode-se reescrever a fórmula acima como:

$$\xi(S, t+1) = \xi(S, t) \times apt(S, t)/\overline{F(t)} \quad \text{(B. 5)}$$

O que, em outras palavras, significa que o número de strings na população cresce de acordo com a razão da aptidão do esquema pela aptidão média da população. Isso significa que esquemas "acima da média" (com aptidão acima da média) recebem um número crescente de strings na próxima geração, e que, esquemas "abaixo da média" (com aptidão abaixo da média) recebem um número decrescente de strings ao longo das gerações, enquanto que esquemas "na média" mantêm a mesma quantidade de strings.

O efeito em longo prazo da regra acima pode ser analisado como segue. Caso o esquema *S* se mantenha sempre como "acima da média" por ∈% (ou seja, $apt(S,t) = \overline{F(t)} + \in \times \overline{F(t)}$) então

$$\xi(S, t) = \xi(S, 0) \times (1 + \in)^t \quad \text{(B. 6)}$$

e $\in = (apt(S,t) - \overline{F(t)})/\overline{F(t)}$, sendo $\in > 0$ e $\in < 0$, respectivamente para esquemas a cima e a baixo da média.

Pela equação B.6, nota-se, agora, que esquemas acima da média recebem um número exponencial de *strings* com o passar das gerações. A equação B.5 é chamada equação de reprodução do crescimento dos esquemas.

Como o operador de seleção não introduz nenhuma variação na população, deve-se considerar como a progressão dos esquemas é afetada pelo operador de recombinação. Esta análise será realizada em seguida.

Recombinação: para ilustrar como o operador de recombinação altera o aparecimento dos esquemas, considere o exemplo B.5.

Exemplo B.5: Suponha duas strings, s_1 = (1 1 1 1 1 1 0 1 0 1) e s_2 = (0 0 1 0 0 0 1 1 1 1), bem como dois esquemas s_1 = (* * * 1 1 * * * * *) e s_2 = (1 1 1 * * * * * 0 1). Nota-se, que tanto s_1 quanto s_2 casam com s_1. Agora, suponha que s_1 e s_2 sejam selecionados para recombinação, em que o ponto de corte seja destacado por | abaixo:

s_1 = (1 1 1 1 1 | 1 0 1 0 1)

s_2 = (0 0 1 0 0 | 0 1 1 1 1)

dando origem a

s_1' = (1 1 1 1 1 0 1 1 1 1)

s_2'= (0 0 1 0 0 1 0 1 0 1).

Pode-se verificar que s_1 sobrevive à recombinação, uma vez que ao menos um dos filhos (s_1') ainda pode ser casado com ele. O mesmo não ocorre com s_2. Diz-se que s_2 foi destruído.

Pelo exemplo B.5, pode ser observado que a possibilidade de um esquema sobreviver está ligado ao seu comprimento, uma vez que, $\delta(s_1) = 1$, e $\delta(s_2) = 9$. De forma geral, para uma *string* de tamanho *m*, existem *m* - 1 possíveis pontos para a recombinação de um ponto. Assim, a probabilidade de destruição de um esquema *S* é dada por:

$$p_d(S) = \frac{\delta(S)}{m-1} \tag{B.7}$$

e, consequentemente, a possibilidade de não destruição é dada por 1 - $p_d(S)$. É importante perceber que apenas alguns cromossomos são selecionados para recombinação, assim, a probabilidade de um esquema *S* sobreviver à recombinação é dada por:

$$p_s(S) = 1 - p_r \times \frac{\delta(S)}{m-1} \tag{B.8}$$

em que p_r é a probabilidade de ocorrer uma recombinação. Pode-se mostrar que mesmo se o ponto de corte do operador de recombinação estiver entre posições fixas do esquema, este ainda pode sobreviver, embora isso ocorra raríssimas vezes. Contudo, a

fórmula B.8 deve ser modificada para acomodar este acontecimento e, portanto, pode ser rescrita como:

$$p_s(S) \geq 1 - p_r \times \frac{\delta(S)}{m-1} \qquad \text{(B. 9)}$$

Considerando a influência do operador de recombinação na progressão dos esquemas, tem-se uma nova fórmula de reprodução do crescimento dos esquemas, dada por:

$$\xi(S, t+1) \geq \ \xi(S, t) \times \frac{apt(S, t)}{\overline{F(t)} * \left[1 - p_r \times \frac{\delta(S)}{m-1}\right]} \qquad \text{(B. 10)}$$

A equação B.10 diz que o número de *strings* esperadas que casem com um esquema *S* na próxima geração pode ser visto como uma função do atual número de *strings* que casam com *S*, a aptidão relativa de *S*, e o seu comprimento. Fica claro que esquemas "acima da média" com comprimento reduzido receberão um número crescente de *strings* com o passar das gerações. Será analisado em seguida como a mutação influencia a progressão dos esquemas.

Mutação: A mutação troca o valor do bit de uma dada posição. Pode-se mostrar que todas as posições fixas de um esquema devem se manter inalteradas para que o esquema sobreviva à mutação. Para visualizar tal afirmação, considere o exemplo a seguir.

Exemplo B.6: Suponha a string, s_1 = (1 1 1 1 1 1 0 1 0 1) bem como o esquema S_1 = (* * * 1 1 * * * * *), que casa com s_1. Caso s_1 sofra mutação e ela ocorra em qualquer posição que em s_1 figure um *, fica claro que s_1 continuará casando com s_1. Contudo, caso a mutação ocorra em alguma posição em s_1 em que no esquema figure uma posição fixa, s_1 será destruído.

Pelo exemplo B.6, fica claro que a probabilidade de destruição de um esquema *S* pelo operador de mutação é uma função da ordem do esquema. Seja p_m a probabilidade de se alterar um bit, assim, a probabilidade de um único bit não ser alterado é 1 - p_m. Uma mutação é independente das outras, assim, a probabilidade de um esquema *S* sobreviver dada uma sequencia de mutações de um bit é:

$$p_s(S) = (1 - p_m)^{o(S)} \qquad \text{(B. 11)}$$

Como normalmente $p_m << 1$, pode-se aproximar B.11, por:

$$p_s(S) \approx 1 - o(S) \times p_m \qquad \text{(B. 12)}$$

Finalmente, chega-se a uma nova fórmula de reprodução do crescimento dos esquemas que agora é combinada com a influência da seleção, da recombinação e da mutação. Esta fórmula é dada por:

$$\xi(S, t+1) \geq \xi(S, t) \times \frac{apt(S, t)}{\overline{F(t)} * \left[1 - p_r \times \frac{\delta(S)}{m-1} - o(S) \times p_m\right]} \qquad \text{(B. 13)}$$

A equação B.13 diz que o número esperado de *strings* que casem com um esquema *S* na próxima geração pode ser visto como uma função do atual número de *strings* que casam com *S*, da aptidão relativa de *S*, do comprimento de *S* e de sua ordem. Fica claro que esquemas "acima da média" com comprimento e ordem reduzidos receberão um número crescente de strings com o passar das gerações. O resultado final de B.13 pode ser anunciado como (HOLLAND, 1975):

Teorema 1: Esquemas "acima da média", de pequeno comprimento e de baixa ordem são propagados de forma exponencial para toda a população em gerações subsequentes do Algoritmo Genético.

A idéia de esquemas com aptidão acima da média da população, com um pequeno comprimento e baixa ordem, é tão importante que Goldberg (GOLDBERG, 1989) chamou essas estruturas de blocos construtores e, propôs a hipótese de blocos construtores. Essa hipótese tenta explicar como um AG é capaz de gerar soluções de alta aptidão e, consequentemente, de obter bons resultados nos mais diversos problemas. A seguir apresenta-se a definição de blocos construtores para posteriormente discuti-la.

Definição B.5 (Blocos Construtores): Blocos construtores são esquemas "acima da média", de pequeno comprimento e de baixa ordem.

A hipótese dos blocos construtores é constituída de (GOLDBERG, 1989):

1. Uma descrição do mecanismo de adaptação abstrato que realiza a adaptação pela combinação de blocos construtores.
2. A hipótese de que um Algoritmo Genético realiza sua adaptação pela implementação implícita e eficiente deste mecanismo de adaptação.

Goldberg descreve este mecanismo abstrato de adaptação como: “Esquemas com pequeno comprimento, baixa ordem e alta aptidão, são instanciados, recombinados e re-instanciados para formarem *strings* potencialmente mais adaptadas. Até certo ponto, trabalhar com este tipo específico de esquemas (os blocos construtores), reduz a complexidade do problema. Em vez de construir *strings* de alta aptidão pela tentativa de combinação de cada *string* concebível, os AGs constroem *strings* cada vez melhores pela recombinação das melhores cadeias *strings* de gerações passadas”.

Assim, uma clássica analogia realizada sobre o procedimento de busca dos AGs é que estes constroem soluções próximas a ótima pela combinação de blocos construtores, assim como uma criança constrói belíssimas fortalezas através da combinação de simples blocos de madeira (GOLDBERG, 1989). Dessa forma, os blocos construtores podem ser vistos como pedaços da solução ótima que, quando combinados, são capazes de remontá-la.

A Hipótese dos Blocos Construtores têm sido fortemente criticada no sentido que ela carece de uma justificação teórica (FOGEL 2000). De qualquer forma, tal hipótese “grosseiramente” descreve o comportamento dos AGs no sentido de que, hoje, eles são usados nas mais diversas formas e, portanto, com várias diferenças em relação ao AG padrão sem, no entanto, perder a sua característica de alcançar bons resultados nos mais variados problemas. De forma mais geral, os blocos construtores estão ligados à estrutura da solução do problema. Este conceito será o principal motivador para os AEDs revisados no capítulo 3.