PROGRAMAÇÃO LINEAR APLICADA A SISTEMAS DE INFORMAÇÃO

Daniel Segundo Cazalis Salas

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA 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.

Aprovada por:

Clovis Caesar Gonzaga, D.Sc.
Presidente

Paulo Roberto Oliveira, Dr.Ing.

Jayme Luiz Szwarcfiter, Ph.D.

Nair Maria Maia de Abreu, D.Sc.

Suely Bandeira Teixeira Mendes, Ph.D.

Nelson Maculan Filho, D.Sc.

RIO DE JANEIRO, RJ -BRASIL
MAIO DE 1992

SALAS, DANIEL SEGUNDO CAZALIS

Programação Linear Aplicada a Sistemas de Informação [Rio de Janeiro] 1992.

XII, 153 p. 29,7 cm (COPPE/UFRJ, D.Sc., Engenharia de Sistemas e Computação, 1992).

Tese – Universidade Federal do Rio de Janeiro, COPPE.

1.

I. COPPE/UFRJ II. Título (Série)

As Begonhas e as

Yolandas da minha vida.

AGRADECIMENTOS

Ao professor Clovis Gonzaga quem me ensinou os conceitos fundamentais em que se baseia o presente trabalho, ao orientador Gonzaga quem estimulou minha independência e criatividade além dos limites que a prudência recomenda, ao amigo Clovis sempre presente.

Ao professor Julian Araoz quem colocou em minhas mãos o problema que intento resolver.

Ao Reitor Nelson Maculan Filho por sua prolongada colaboração valiosa, acertadíssima seleção bibliográfica; sendo para mim como para tantos outros exemplo de humanidade.

A professora Dina Feigenbaum Cleiman, em cujo seminário foram desenvolvidos conceitos usados neste trabalho, por seu estímulo e colaboração.

Aos colegas Di Novela, Adilson Xavier e Sanchez incansáveis interlocutores.

As secretárias Claudia, Denise e Ana Paula, as melhores que um doutorando pode desejar.

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

PROGRAMAÇÃO LINEAR APLICADA A SISTEMAS DE INFORMAÇÃO

Daniel Segundo Cazalis Salas

Junho de 1991

ORIENTADOR: Clovis Caesar Gonzaga

PROGRAMA: Engenharia de Sistemas e Computação

Provadores em cláusulas de Horn usados extensivamente em sistemas de informação estão baseados no princípio de caminhos de aumento. Propomos um processo diferente para solucionar o problema de dedução automática: a eliminação de cláusulas inúteis.

Na detecção sucessiva de cláusulas inúteis se usam modelos de fluxo em hipergrafos e algoritmos de P.L. por direções viáveis.

Se solucionam por procedimentos semelhantes outros problemas encontrados no manuseio de bases de conhecimento.

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

LINEAR PROGRAMING ON INFORMATION SYSTEMS

Daniel Segundo Cazalis Salas

July 1991

Thesis Supervisor: Clovis Caesar Gonzaga

Department:

Provers on Horn clauses extensivly used in information systems are based on the augmenthed path principle. We propose a different process for solving the automatic deduction problem: Elimination of useless clauses.

In successive detection of useless clauses we use a combinatorial model of flow hipergraphs and feasible directions algorithms for linear programming.

We solve by similar procedures some other problems found in knowedge bases.

ÍNDICE

Pág.

Pág.

Pág.

INTRODUÇÃO

Sistemas especialistas baseados em cláusulas de Horn podem ser modelados como um problema de programação linear. A utilização de Simplex na solução deste modelo reproduz o comportamento combinatório dos algoritmos Bottom–Up e Top–Down.

Desenvolvemos um algoritmo de conjuntos ativos orientado a solução de modelos combinatórios.

Procedimentos desenvolvidos para serem usados em algoritmos de ponto interior nos permitiram construir um modelo combinatório poliédrico suceptível de ser resolvido por este algoritmo.

No presente trabalho pretendemos criar um método para a solução do problema de busca em sistemas clausulares de Horn baseado em modelos combinatórios e algoritmos de conjuntos ativos tal que resulte em um compartimento combinatório diferente do que se obtem usando o algoritmo Simplex, isto é, diferente de Top–Down e Bottom–Up.

No primeiro capítulo apresentamos um algoritmo de programação linear suceptível de resolver problemas com múltiplas degenerações tanto primais como duais, este tipo de problemas degenerados aparece com freqüência em modelos combinatórios poliedricos.

O problema de busca em sistemas de informação e sua solução combinatória é apresentado no segundo capítulo. Os requerimentos da solução ao problema só podem ser examinados no contexto da inteligência artificial, uma

revisão de trabalhos na área de reconhecimento de linguagens nos permite determinar o contexto no que este problema é resolvido.

No terceiro capítulo um modelo de fluxo em hipergrafos é desenvolvido e a equivalência da solução do modelo com a solução do problema de busca é demonstrada.

No capítulo final se usam os algoritmos sobre o modelo para solucionar diferentes problemas encontrados com freqüência no manuseio de sistemas de informação. A interação modelo algoritmo gera um comportamento que permite resolver tais problemas de maneira que reproduza algumas das características geralmente aceitas como humanas resolvendo problemas de informação. A similitude entre o método apresentado e a forma humana de resolver problemas caracteriza uma eventual vocação do método na área de inteligência artificial.

CAPÍTULO 1

PROGRAMAÇÃO LINEAR

No presente capítulo cubriremos o relativo a programação linear.

Inicialmente definiremos as partes com as quais construíremos um algoritmo para programação linear.

Na continuação procederemos a uma descrição geral do algoritmo.

Apresentaremos os elementos teóricos que asseguram sua correção e provaremos que se detem em um número finito de passos em uma solução ótima.

Exporemos brevemente as características de outros algoritmos de programação linear.

Finalmente estudaremos a estrutura combinatória dos algoritmos de programação linear na seqüência de faces do poliedro visitadas no processo de otimização: a interação entre o poliedro e o algoritmo se manisfesta em um determinado comportamento combinatório.

Dado que o algoritmo a ser apresentado se baseia nos conceitos de projeções sobre faces de um poliedro e cálculo de multiplicadores de Lagrange, algumas definições preliminares associadas a estes conceitos são necessárias para proceder com a descrição do algoritmo.

1.1 – PRELIMINARES

1.1.1 – Notação

Os principais elementos da notações são dois:

(i) uma matriz A de n colunas e m filas com $m < n$;

(ii) um conjunto de índices $\beta \subset \{1, ..., n\}$.

As matrizes podem ser indiciadas por conjuntos, por exemplo:

A_β consiste na matriz construída selecionando em A as colunas cujos índices estão em β.

Em forma analoga os vetores, que sempre são considerados colunas, podem também ser indiciados por conjuntos de índices apropriados.

As matrizes super–indiciadas por conjuntos, denotam construções particulares:

Desta forma A^β denota da matriz A aumentada pelas filas da identidade correspondentes aos elementos em β.

$$A^\beta = \begin{bmatrix} A \\ I^t_\beta \end{bmatrix} \tag{1.1}$$

Sendo P a matriz de projeção sobre o nulo de A.

Denominaremos P^β a matriz de projeção sobre o nulo de A^β.

Observe que, P_β não é uma matriz de projeção e sim a submatriz obtida selecionando em P as colunas com índices em β.

1.2 – DEFINIÇÕES

Definição: Problema de Programação Não–Linear (PNL)

Problema de programação não–linear:

$$\begin{aligned} &\min f(x) \\ &\text{s.a.} \quad h_i(x) = 0 \quad i = 1, \ldots, m \\ &\qquad r_i(x) \leq 0 \quad i = 1, \ldots, k \end{aligned}$$

com f, h_i, r_i funções diferenciáveis de $\mathbb{R}^n$ em $\mathbb{R}$.

Definição: Problema de Programação Linear (PL)

Problema de programação linear:

$$\begin{aligned} &\min c^t x \\ &\text{s.a:} \quad Ax = b \\ &\qquad x \leq 0 \end{aligned}$$

com A uma matriz de rank completo de m filas, n colunas, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$.

Definição: Conjunto Viável em PL.

$$S = \{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\}.$$

Definição: Conjunto de Pontos Interiores.

$$S^0 = \{x \in \mathbb{R}^n \mid Ax = b,\ x > 0\}.$$

Consideraremos unicamente conjuntos não vazios de pontos interiores $S^0 \neq \phi$.

Na presente aplicação aparecem com freqüência conjuntos viáveis, tais que existem variáveis cujo valor é zero para todo ponto viável. Nesse caso não se levaram em consideração estas variáveis nas definições de ponto interior e faces que veremos na continuação.

Podemos observar que a programação linear é um caso particular da programação não linear e que portanto são válidos em (PL) todos os resultados obtidos em (PNL). Basta substituir f(x) por $C(x) = c^t x$, $h_i(x)$ por $(a^i)^t\, x - b_i$ com a^i um vetor coluna formado pela i–ésima fila da matriz A y $r_i(x)$ por $-x_i$.

Definição: Face

Dados: Um problema P.L. é um conjunto de índices $\beta \subset \{1, ..., n\}$, com $\beta \neq \phi$.

Dizemos que o conjunto:

$$S_\beta = \{x \in \mathbb{R}^n \mid Ax = b,\ x_\beta = 0,\ x \geq 0\},$$

é uma face se e somente se é diferente de vazio,

$S_\beta \neq \phi$.

Esta definição equivale a mais conhecida definição baseada no conceito de hiperplano de suporte.

Neste caso, dizemos que β é uma caracterização da face S_β.

No nosso caso, face e face própria são conceitos equivalentes.

Definição;: Subespaço Associado a uma Face

A cada face S_β, associamos o subespaço E_β

$$E_\beta = \{x \in \mathbb{R}^n \mid Ax = 0,\ x_\beta = 0\}$$

Podemos observar que E_β está bem definido e não depende da caracterização da face, pois associado a este subespaço existe uma única variedade linear

$$V_\beta = \{x \in \mathbb{R}^n \mid Ax = b,\ x_\beta = 0\}.$$

Que é a envoltória afim dos elementos na face.

Definição: Faceta

São as faces que podem ser caracterizadas por um conjunto de um único elemento $i \in \{1, ..., n\}$

$$S_i \equiv S_{(i)} = \{x \in \mathbb{R}^n \mid Ax = b,\ x_i = 0,\ x \geq 0\}.$$

Em forma equivalente uma faceta é uma face, tal que seu subespaço associado tem dimensão $n - m - 1$, uma a menos que dimensão de N(A), nulo de A.

Observação: Para obter o subespaço E_β associado a face S_β construímos a matriz:

$$A^\beta = \begin{bmatrix} A \\ I^t_\beta \end{bmatrix}$$

donde I_β consiste nas colunas da identidade com índices em β, de acordo com nossas convenções, obtemos uma formulação matricial da definição de E_β:

$$E_\beta = N \begin{bmatrix} A \\ I^t_\beta \end{bmatrix} = N(A^\beta)$$

Respectivamente:

$$E_i = N \begin{bmatrix} A \\ I^t_i \end{bmatrix} = N(A^i)$$

Observação: Os subespaços associados a diferentes faces do poliedro não são necessariamente diferentes.

Definição: Subespaço Complementar

O subespaço complementar a E_i é $\bar{E}_i$

$$\bar{E}_i = R(A^t \; I_i)$$

Esta é a imagem da transposta da matriz cujo nulo é o subespaço associado a faceta:

$$\bar{E}_i = \{d \in \mathbb{R}^n \mid d = (A^t \; I_i)\, y,\; y \in \mathbb{R}^{m+1}\}$$

A matriz de projeção P sobre o nulo de A é dada por:

$$P = I - A^t(AA^t)^{-1} A$$

A matriz $(AA^t)^{-1}$ existe devido que A é de rank completo.

Definição: Vetor Normal

Cada faceta S_i associada a restrição $x_i \geq 0$ tem como vetor normal, a projeção de I_i sobre N(A):

$$P_i = PI_i$$

O vetor P_i é normal a face E_i devido que sendo a projeção de I_i no nulo de A a seguinte igualdade se cumpre:

$$I_i = P_i + (I_i - P_i)$$

Projetando no nulo de R_i, obtenmos:

$$0 = (\text{projeção em } N(E_i) \text{ de } P_i) + 0,$$

devido que I_i é normal ao subespaço $\{x_i = 0\}$ e $(I_i - P^i)$ é normal ao nulo de A.

Observação: Os vetores normais podem ser transformados em vetores de norma unitária, mediante a um pré–escalamento do problema original.

Dois tipos de matrizes se construem a partir de P e a caracterização de uma face $\beta \subset \{1, ..., n\}$.

Definição: Matriz de Vetores Normais

A primeira é a matriz de vetores normais a E_i, $i \in \beta$ é uma submatriz de P que se constrõe tomando as colunas correspondentes aos índices no conjunto β, seguindo a notação usual, denominaremos P_β.

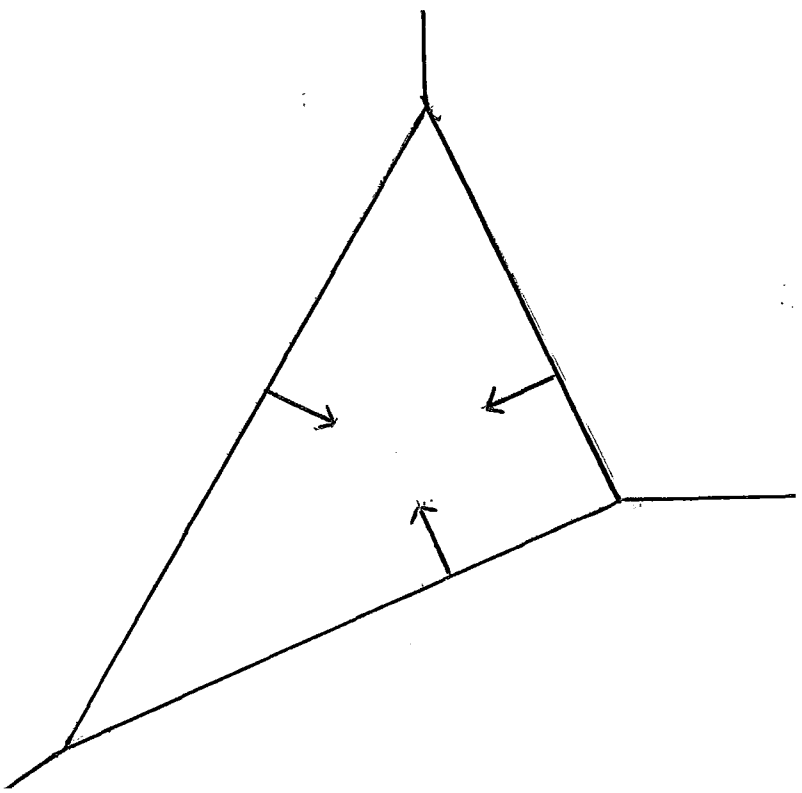


Figura – Vectores P_i

Definição: Matriz de Projeção sobre E_β

A segunda é a matriz de projeção sobre o subespaço E_β e será denominada P^β.

Não deve ser confundida com a matriz P_β que é uma submatriz de P segundo nossa convenção; a matriz P^β é uma matriz de projeção com n filas e n colunas que sempre existe dado que a projeção é um operador linear.

Não obstante a obtenção desta matriz de projeção por meio da formula, a continuação depende de que A^β seja de rank máximo; mostraremos adiante que as caracterizações β das faces geradas por nosso algoritmo resultam em matrizes de rank máximo:

$$P^\beta = I - (A^\beta)^t ((A^\beta)(A^\beta)^t)^{-1} (A^\beta)$$

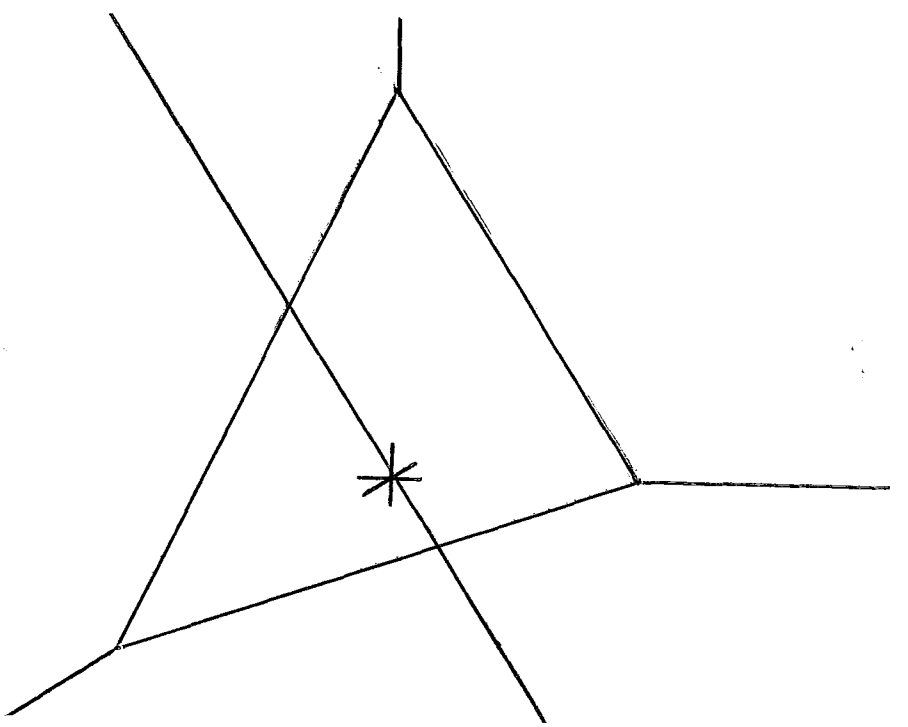


Figura – Subespaço nulo de A^i

Definição: O subespaço complementar de E_i em relação a $N(A)$ é:

$$R(P_i) = \{d \in N(A) | P^i d = 0\}$$

Deste maneira P_i é a única direção em $N(A)$ ortogonal ao subespaço E_i, dado que E_i é maximal em $N(A)$.

Isto nos permite expressar $R(P_i)$ da seguinte forma:

$$R(P_i) = \{\alpha P_i, \alpha \in \mathbb{R}\}$$

A cada face S_β associamos os vetores P_i, $i \in \beta$ e qualquer vetor ortogonal a E_β em $N(A)$ é uma combinação linear dos P_i, $i \in \beta$.

Definição: Face Ativa

Dado um ponto viável x dizemos que uma face S_β é ativa no ponto x se e somente se $x \in S_\beta$. Dizemos que uma face ativa é maximal se o subespaço associado é de dimensão máxima entre as faces ativas em x.

Definição: Custo Projetado

É a projeção do vetor c sobre o nulo de A_i

$$cp = Pc$$

Definição: Direções Projetivas

Dizemos que uma direção é projetiva se é oposta a projeção do custo sobre o subespaço associado a uma face; a cada face S_β associamos a direção projetiva:

$$h(S_\beta) = h_\beta = - P^\beta c.$$

Observação: As direções projetivas dependem unicamente da face S_β serão denominadas h_β mas são independentes da caracterização da face.

Definição: Declive

O declive de uma direção $h \in \mathbb{R}^n$ com $h \neq 0$ vem dado por:

$$\delta(h) = - c^t h / \|h\|$$

1.2.1 – Multiplicadores de Karush–Kunh–Tucker

Apresentaremos inicialmente os teoremas que permitem assegurar a existência de multiplicadores com determinadas características e sua vinculação com a otimalidade.

Em seguida definimos e desenvolvemos o conceito de estimativas de multiplicadores a serem calculados como uma aproximação dos multiplicadores de Lagrange.

Definimos então uma função de perturbação que nos permite justificar a relação existente entre as componentes que serão ativas em um ótimo e as estimativas dos multiplicadores.

1.3 – TEOREMAS DE EXISTÊNCIA

O presente resultado é observado quando se cumprem algumas condições no conjunto viável, como no caso de programação linear, aqui apresentado.

Teorema 1.1 (KKT)

Se $\hat{x}$ é uma solução ótima para o problema (PNL), então existe multiplicadores w_i, $i = 1, ..., m$ e $z_i = 1, ..., k$ tal que:

(i) $$\nabla f(\hat{x}) + \sum_{i=1}^{k} z_i \nabla r_i(\hat{x}) - \sum_{i=1}^{m} w_i \nabla h_i(\hat{x}) = 0$$

(ii) $$z_i r_i(\hat{x}) = 0, \quad i = 1, ..., k$$

(iii) $$z_i \geq 0, \quad i = 1, ..., k$$

(iv) $$\hat{x} \in S$$

A aplicação deste teorema ao problema de programação linear resulta diretamente no seguinte:

Teorema 1.2 (KKT)

Restringindo ao problema linear (PL): $\hat{x}$ é uma solução ótima para o problema (PL) se e somente se existem multiplicadores w_i, $i = 1, ..., m$ e z_i, $i = 1, ..., n$ tal que:

(i) $$c - A^t w - z = 0$$

(ii) $z_i\hat{x}_i = 0,\ i = 1, \ldots, k$

(iii) $z \geq 0$

(iv) $\hat{x} \in S.$

Em alguns problemas de otimização não existem restrições de desigualdade.

Definição: Problema de Programação Não Linear Restrições de Igualdade (PNLRI)

$$\min f(x)$$
$$\text{s.a.} \qquad h_i(x) = 0, \quad i = 1, \ldots, m,$$

com f, h, funções diferenciáveis de $\mathbb{R}^n$ em $\mathbb{R}$.

Teorema 1.3 (KKT)

Se $\hat{x}$ é uma solução ótima para o problema (PNRLI), então existem multiplicadores w_i, $i = 1, \ldots, m$, tais que:

(i) $$\nabla f(\hat{x}) - \sum_{i=1}^{m} w_i \nabla h_i(\hat{x}) = 0$$

(ii) $\hat{x} \in S$

1.3.1 – Estimativas dos Multiplicadores

Os multiplicadores w e z das condições de KKT são chamados "multiplicadores KKT", multiplicadores de Lagrange ou simplesmente multiplicadores.

Os multiplicadores de KKT estão associados a soluções ótimas do problema de otimização e podem ser utilizados para construir regras de parada para algoritmos.

Em pontos que não são soluções ótimas é impossível satisfazer as condições do teorema 1.2 relaxando alguma das restrições ou construindo problemas auxiliares é possível obter multiplicadores que satisfazem "aproximadamente" as condições KKT. Multiplicadores obtidos desta forma são chamados de estimativas de multiplicadores.

Não há critérios precisos para definir o que são estimativas dos multiplicadores. Boas estimativas devem coincidir com multiplicadores ótimos quando associados a soluções ótimas.

Exige–se normalmente que os pontos próximos a uma solução ótima, as estimativas devem aproximar–se de multiplicadores ótimos; a natureza discreta do estudo apresentado não satisfaz esta exigência.

Um estudo profundo das estimativas dos multiplicadores encontra–se em GILL e MURRAY [ ]. Estimativas para problemas de programação linear associados ao algoritmo afim, escala são descritas em GONZAGA e CARLOS [ ].

Em geral se associam estimativas de multiplicadores a problemas com restrições de igualdade. Quando existem restrições de desigualdade; o procedimento é

fixar as restrições de desigualdade que se cumprem como igualdade, restrições ativas, e se despejam as restrições inativas. Gera-se estimativas de multiplicadores para o problema resultante que possue unicamente restrições de igualdade.

As estimativas dos multiplicadores tem duas utilidades:

(i) construir uma função Lagrangeana em algoritmos de tipo programação quadrática sequencial;

(ii) detectar restrições ativas que podem ser liberadas. Isto se tem com o seguinte critério: considerando que as estimativas são boas aproximações de multiplicadores ótimos; a positividade é interpretada como reforço da restrição associada permanecer ativa. Ao contrário se a estimativa associada a fixação de uma variável ativa resulta em um multiplicador negativo, consideramos a desativação dessa restrição.

Em nosso estudo utilizamos unicamente (ii) entre as utilidades das estimativas.

Considera-se $x \in S$ um conjunto $\beta \subset \{1, ..., n\}$ tal que $x \in S_\beta$. Se S_β é uma face ótima, então se satisfaz a condição do Teorema 1.2.

Se S_β não é uma face ótima, poderíamos relaxar de 1.2 a condição (iii) $z \geq 0$ e estimar a solução ao problema resultante.

Procurar $w \in \mathbb{R}^m$ e $z \in \mathbb{R}^n$ tal que satisfaçam aproximadamente:

$$A^t w + z = c$$

$z_i = 0, \ \forall i \notin \beta$

Vamos ver a análise das estimativas por dois métodos distintos, obtendo–se os mesmos resultados.

Primeiro Método: O primeiro método basea–se em uma escolha de estimativas por mínimos quadrados, seguindo o processo mas comum na literatura

$$\text{minimizar } \|A^t w + z - c\|^2$$
$$\text{s.a. } z_i = 0 \ \forall i \notin \beta.$$

Fixando $z_i = 0, \forall i \notin \beta$ obtemos:

$$\text{minimizar } \|A^t w + I_\beta z_\beta - c\|^2$$

Em forma matricial:

$$A^t w + I_\beta z_\beta = c = \begin{bmatrix} A \\ I_\beta^t \end{bmatrix}^t \begin{bmatrix} w \\ z_\beta \end{bmatrix} - c$$

Usando a definição de A^β em (1.1) obtemos:

$$\text{minimizar } \|(A^\beta)^t \begin{bmatrix} w \\ z_\beta \end{bmatrix} - c\|^2$$

equação que define a projeção de c sobre o nulo de A^β. Se $\bar{w}$, $\bar{z}$ são soluções ótimas então:

$$(A^{\beta})^t \begin{bmatrix} \bar{w} \\ \bar{z}_{\beta} \end{bmatrix} - c = P^{\beta} c$$

$$A^t \bar{w} + I_{\beta} \bar{z}_{\beta} = c - P^{\beta} c$$

Finalmente, projetando sobre o nulo de (A) obtemos:

$$P_{\beta} z_{\beta} = c_p - P^{\beta} c$$

dado que $A^t \bar{w}$ está na imagem de A, $PI_{\beta} \bar{z}_{\beta} = P_{\beta} z_{\beta}$ e $P^{\beta} c$ está no nulo de A.

Multiplicando–se ambos lados por P_{β}^t obtemos:

$$P_{\beta}^t P_{\beta} z_{\beta} = P_{\beta}^t (c_p - P^{\beta} c)$$

Resolvendo–se $P_{\beta}^t P_{\beta}$ é invertível (o que sempre é verdade é nosso caso)

$$z_{\beta} = (P_{\beta}^t P_{\beta})^{-1} P_{\beta}^t (c_p - P^{\beta} c)$$

Segundo Método: O método baseia–se na obtenção de multiplicadores para o seguinte problema auxiliar:

Dado um problema (PL) e uma face S_{β}, chamamos estimativa dos multiplicadores, aos multiplicados de Lagrange associados a um problema auxiliar definido da seguinte maneira:

Problema Auxiliar

$$\begin{aligned} &\min (cp - P^{\beta} c)^t x \\ &\text{s.a.} \qquad Ax = b \\ &\qquad\quad x_i = 0, \ \forall i \in \beta, \end{aligned}$$

ou em formulação matricial:

Problema Auxiliar

$$\begin{aligned} &\min (cp - P^{\beta} c)^t x \\ &\text{s.a.} \qquad \begin{bmatrix} A \\ I^t_{\beta} \end{bmatrix} x = \begin{bmatrix} b \\ 0 \end{bmatrix} \end{aligned}$$

Devido a que $(cp - P^{\beta} c)$ é ortogonal a face S_{β}, x é ótimo, se $x \in S_{\beta}$. Desta forma este problema é trivial no sentido de que todo ponto viável é ótimo.

Observamos que este problema é um problema que possue unicamente restrições de igualdade sendo ademais um problema de programação linear, desta forma usando os teoremas 1.2 e 1.3 com $w \in \mathbb{R}^m$ e $z_{\beta} \in \mathbb{R}^{|\beta|}$ obtemos:

$$(cp - P^{\beta}) - (A^t \quad I_{\beta}) (w \quad z_{\beta}) = 0,$$

$$A^t w + I_{\beta} z_{\beta} = (cp - P^{\beta} c).$$

Projetando ambos os lados no nulo de A, obtemos:

$$Pz = (cp - P^{\beta} c).$$

o equivalente:

$$P_{\beta}{}^{z}{}_{\beta} = (cp - P^{\beta}c).$$

Desta forma, os multiplicadores z_{β} do problema auxiliar são as estimativas dos multiplicadores z_{β} do problema original.

1.4 – FUNÇÃO PERTURBAÇÃO

Uma função de perturbação associada ao problema auxiliar que relaciona cada $y \in \mathbb{R}^{\|\beta\|}$ com um valor $v(y) \in \mathbb{R}$. Esta função nos indica como varia a componente ortogonal do custo $(cp - P^{\beta}c)$ em relação a cada uma das variáveis em β.

Função Perturbação

$$v(y) = \min\ (cp - P^{\beta}c)^t\, x$$
$$\text{s.a.} \quad Ax = b$$
$$x_{\beta} = y$$

Este problema é também trivial num mesmo sentido que o problema auxiliar, isto é, todo ponto viável é ótimo e realiza o valor mínimo.

Esta função está bem definida para conjuntos β que são caracterizações mínimas de faces de S.

Podemos observar que a função de perturbações no caso de problemas degenerados depende da caracterização da face S_{β}; pois diferentes caracterizações

mínimas da face resultam em diferentes funções de perturbações.

$$\frac{\delta v(0)}{\delta y_i} = z_i, \quad i \in \beta$$

Os valores z_β são os multiplicadores de Lagrange associados ao problema auxiliar.

Se o multiplicador associado ao elemento i do conjunto ativo é positivo significa que toda direção viável contida em $E_{\beta-\{i\}}$ é de menor declive que d_β e que $d_{\beta-\{i\}}$ não é viável

$z_i > 0$ então $(d_{\beta-\{i\}})_i < 0$.

Desta maneira se todos os multiplicadores associados aos elementos do conjunto ativo β são positivos a direção projetiva sobre qualquer subconjunto próprio de β não é viável

$z_\beta > 0$ então $\forall \gamma \subset \beta \exists i \in \beta | (d_t)_i < 0$

Pelo contrário se um multiplicador associado a um elemento $i \in \beta$ é negativo, a direção projetiva $d_{\beta-\{i\}}$ é viável e de maior declive que d_β

$z_i < 0$ então $(\delta_{\beta-\{i\}})_i > 0$

Mas assim um multiplicador negativo é o mais negativo entre todos, a direção projetiva $d_{\beta-\{i\}}$ é la de maior declive entre todas as direções viáveis $d_{\beta-\{j\}}$ com $j \in \{1, ..., n\}$.

Desta maneira concluímos que as estimativas dos multiplicadores nos permitem gerar direções viáveis com um declive máximo entre as direções projetivas em subespaços de dimensões $\|\beta\| - 1$ adjacentes a S_β.

1.5 – ALGORITMO DE DIREÇÕES PROJETIVAS

Apresentaremos na continuação um algoritmo baseado no cálculo em cada iteração das estimativas dos multiplicadores.

Inicialmente apresentaremos um algoritmo de direções viáveis que começando num ponto interior procede sucessivamente anulando restrições e incorporando estas a um conjunto ativo que caracteriza minimalmente uma face. Projeção e estimativas são usadas em cada iteração.

O algoritmo também é apresentado em duas formas: como descrição detalhada dos passos dados e por meio de um diagrama de fluxo.

Na continuação, se analisa formalmente a correção do algoritmo assegurando a capacidade para realizar os diferentes passos da seqüência da iteração e finalmente se prova que o processo termina com a obtenção de um ótimo em um número finito de iterações.

ALGORITMO BEGO (BE GOING)

Dados um problema PL e um ponto interior x^0:

1. $k := 0$
2. $d := -cp$
3. SE $d = 0$ SAÍDA
4. $\beta := \phi$
5. REPITA
6. SE $d \geq 0$ problema ilimitado SAÍDA
7. $\lambda := \min_{i=1,\ldots,n} \{x_i/-d_i \mid d_i < 0\}$
8. ESCOLHA $\operatorname{argmin}_{i=1,\ldots,n} \{x_i/-d_i \mid d_i < 0\}$
9. $x^{k+1} := x^k + \lambda d$
10. $k := k+1$
11. $\beta := \beta \cup \{j\}$
12. $d := -P^B c$
13. $z_\beta = (P_\beta^t P_\beta)^{-1} P_\beta^t (cp - d)^*$
14. SE $(\{j \in \beta \mid z_j < 0\} \neq \phi)$
15. ESCOLHA $j \in \operatorname{argmin}_{i \in \beta} z_i$
16. $\beta := \beta - \{j\}$
17. $d := -P^\beta c$
18. FIM SE
19. ATÉ $(\|\beta\| = n - m)$ OU $(d = 0)$

*Observação: Será provado que os conjuntos β gerados, permitem efetuar os cálculos especificados no passo 13.

Na continuação apresentaremos uma descrição detalhada dos passos do algoritmo, os cálculos que devem ser efetuados, alguns mecanismos para facilitar os cálculos e um diagrama de fluxo.

Passo 1: Se inicializa o contador k.

Passo 2: A direção inicial é oposta ao custo projetado no nulo de A.

Passo 3: Se a direção é nula todos os pontos do conjunto viável são ótimos em particular x_0.

Passo 4: O conjunto ativo se inicializa vazio.

Passo 5: Este é o laço principal formado por dois blocos; 6–13, neste bloco chamado de restrição se calcula o passo na direção d, se ativa uma nova restrição, se atualiza a direção e se estimam os multiplicadores. 14–16 Este bloco é condicional, se elimina do conjunto ativo um elemento correspondente a um multiplicador mais negativo e se calcula a direção para a nova iteração.

Passo 6: Se a direção não possue componentes negativos, nenhuma restrição de desigualdade se anulará e por isso, o problema é ilimitado.

Passo 7: Se calcula o passo máximo viável na direção, mediante o teste de razão.

Passo 8: Se escolhe uma restrição j que se anula no passo $x^{k+1} = x^k + \lambda d$, em geral é única, mas se são várias, o algoritmo escolhe entre elas, aquela

com d_j, mais negativo.

Passo 9: Se calcula x^{k+1} o qual é viável, pois $d \in N(A)$ e λ é máximo passo na direção d completamente contido em S.

Passo 10: Se incrementa o contador.

Passo 11: Se incorpora ao conjunto ativo, o índice da restrição escolhida j.

Passo 12: Se calcula uma nova direção projetiva, que será usada na seguinte iteração se os multiplicadores estimados não tem componentes negativos. Este passo pode ser realizado em ordem $O(n^2)$ se reinvertemos a matriz da interação anterior.

Passo 13: Se estimam os multiplicadores do problema auxiliar. Não é necessário neste caso calcular $(P_{\beta}^{T} P_{\beta})^{-1}$, pois a inversão pode se obter mediante reinversão. Este cálculo também se realiza em ordem $O(n^2)$.

Passo 14: Se pelo menos um multiplicador resulta em negativo, se realiza o bloco de relaxação.

Passo 15: Se escolhe $j \in \beta$, tal que z_j é um valor mais negativo entre as estimativas.

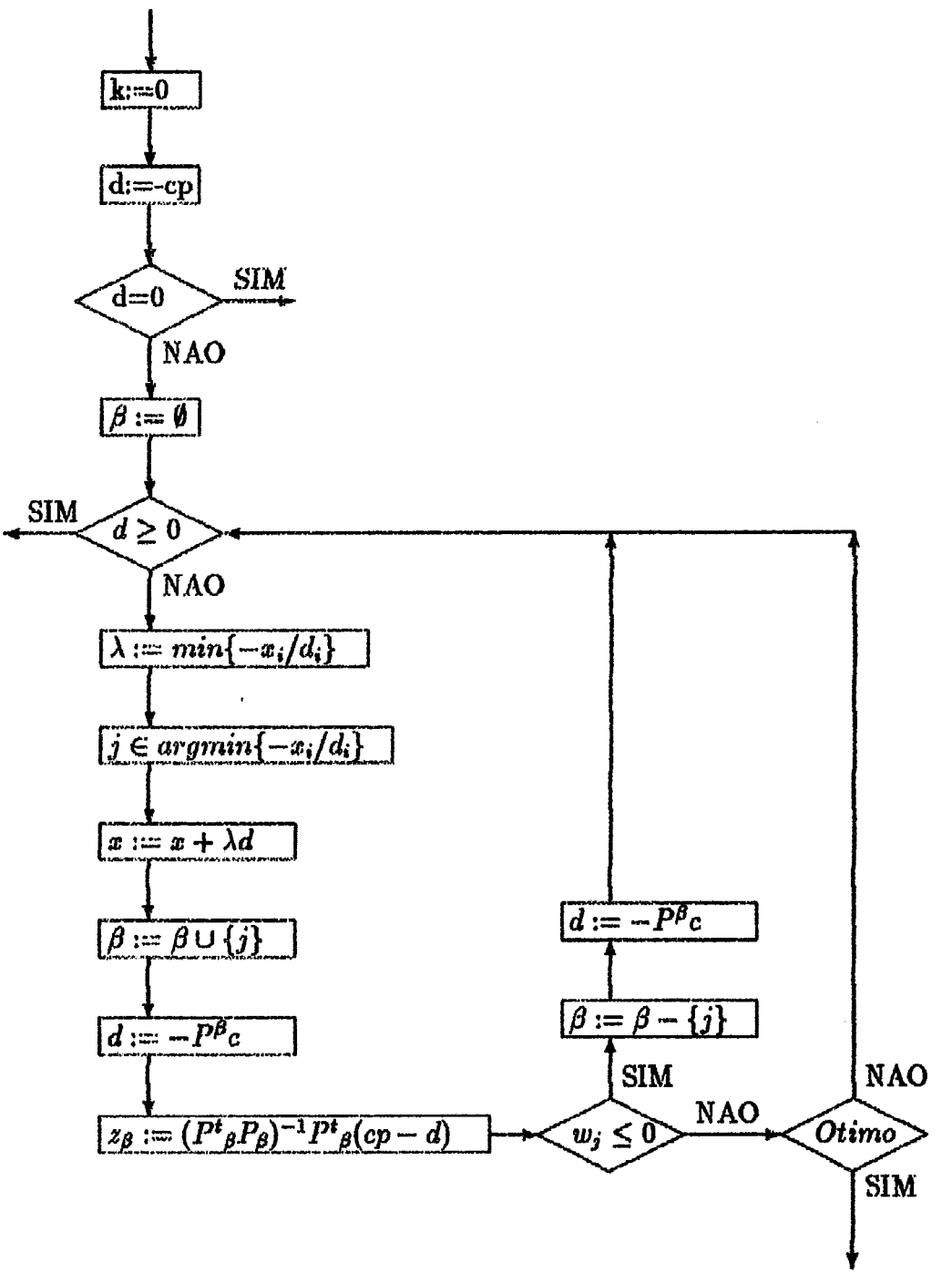
Passo 16: Se elimina j do conjunto ativo.

Passo 17: Se recalcula a direção usando reinversão na ordem $O(n^2)$.

Passo 18: Fim do bloco de relaxação. Poderia realizar–se um curto–circuito devido a que se o algoritmo entra no bloco de relaxação o critério de parada não pode ser cumprido.

Passo 19: Critério de parada; se obtemos n–m restrições ativas, implica que estamos em um vértice e que o multiplicador z_β é positivo, desta forma não se realiza a projeção sobre um vértice; outra possibilidade é que a direção projetiva é nula nesse, caso obtemos um ótimo, pois multiplicadores z_β do problema auxiliar, todos positivos coincidem com multiplicadores do problema original.

Diagrama de Fluxo do Algoritmo Bego



1.5.1 – Algoritmo de Pseudo–Purificação

Este algoritmo se chama de pseudo–purificação devido a que não necessariamente obtemos um vértice. A utilidade desta característica será explicada no capítulo final.

Em cada iteração o algoritmo incrementa em um conjunto ativo.

Pelo lema 5, a ser provado, podemos observar que no caso degenerado a matriz de projeção sobre subespaço ativo e sempre calculável pela formula devido a independência linear das restrições ativas.

Na degeneração dual o algoritmo gera uma direção nula antes de conseguir um vértice.

ALGORITMO PURIF (PURIFICAÇÃO)

Dado um problema PL, um ponto interior x^0 e um custo cp

1. $k := 0$
2. $d := -cp$
3. SI $D = 0$ SAÍDA
4. $\beta := \phi$
5. REPITA
6. SI $d \geq 0$ problema ilimitado SAÍDA
7. $\lambda := \min_{j=1,\ldots,n} \{x_i/-d_i \mid d_j < 0\}$
8. ESCOLHA $i \in \operatorname{argmin}_{j=1,\ldots,n}\{x_i/-d_j\}$
9. $x^{k+1} := x^k + \lambda d$
10. $k := k+1$
11. $\beta := \beta \cup \{j\}$
12. $d := -P^{\beta}c$
13. ATÉ $(d = 0)$ OU $(k = n - m)$

Passos 1, 2, 3, 4: Se inicializam contadores, direções e conjunto ativo

Passo 5: Um laço de máximo $n-m$ iterações. Não é necessário hipóteses de não–degeneração.

Passo 6: O algoritmo não soluciona problemas ilimitados.

Passos 7,8,9,10: Se calcula mediante o teste de razão a seguinte variável a ser anulada e o tamanho do passo λ, e o ponto da seguinte iteração.

Passo 11: Se atualiza o conjunto ativo

Passo 12: Se calcula a direção.

Passo 13: Se a direção é nula e não estamos em um vértice o algoritmo para.

1.5.2 – Algoritmo de Centro

Apresentaremos, a seguir, um algoritmo que em forma conseqüente com os anteriores se move por direções que são projeções nas faces do poliedro com a diferença que o conjunto ativo diminue em cada iteração até deter–se em um ponto interior.

Desta forma pretendemos resolver o problema de dado um ponto na fronteira conseguir um ponto interior.

Para isso procederemos em cada iteração da seguinte forma: se fixa um custo

unitário para todas as componentes i tais que $x_i = 0$, sendo o custo zero para as outras componentes, se calculam os multiplicadores para a projeção desse custo em relação ao conjunto β, (a desativação de uma componente negativa produz uma direção viável) dá-se um passo de longitude $\frac{\lambda}{2}$ e repete-se a operação até não existir componentes negativas nos multiplicadores.

Os lema 1-5 asseguram a correção dos passos do algoritmo

ALGORITMO ANTIPURIF (BE GOING)

Dado um problema PL, um ponto x^0 e um conjunto ativo β

1. $k := 0$
2. $c := (c_i = 1\ \forall i | x_i = 0,\ c_i = 0\ \forall i | x_i > 0)$.
3. $z_\beta = (P_\beta^t P_\beta)^{-1} P_\beta^t (cp)^*$.
4. SI $(\{j \in \beta | x_j < -\} = \phi)$ SAÍDA
5. ESCOLHA $j \in \text{argmin}_{i \in \beta} z_i$.
6. $\beta := \beta - \{j\}$
7. $d := -P^\beta c$.
8. $x^{k+1} := x^k + (\lambda/2)\, d$
9. $k := k+1$
10. VÁ PARA 2.

* Observação: Será provado que os conjuntos β gerados, permitem efetuar os cálculos especificados no passo 3.

Apresentaremos uma descrição dos passos do algoritmo de centro.

Passo 1: Se inicializa o contador de iterações.

Passo 2: Se assinala um custo tal que é unitário nas componentes no conjunto ativo.

Passo 3: Se calcula a estimativa para esse custo.

Passo 4: Se a estimativa é positiva, não é possível aumentar o valor das componentes i tal que $x_i = 0$, obtendo-se um β tal que $x_\beta = 0$ para todo $x \in S$.

Passos 5,6,7: Nos permitem calcular uma direção viável estritamente positiva em pelo menos uma componente i tal que $x_i = 0$.

Passo 8: Nos permite calcular um ponto donde β será ativa.

Passo 9,10: Se incrementa o contador e se itera de novo.

1.6 – BASE TEÓRICA

Apresentaremos vários lemas que asseguram o comportamento adequado dos algoritmos. A notação coincide com as definições e procedimentos especificados previamente.

Lema 1

O cálculo das estimativas dos multiplicadores de Lagrange do problema auxiliar sempre tem solução, dadas as condições em que o algoritmo apresenta o problema.

Prova

Devido a que $(cp - d)$ pertence ao nulo de A, é ortogonal ao subespaço E_β então pertence ao subespaço complementar $R(P_\beta)$. No caso de P_β ser de rank máximo, existe uma única solução. Em outras palavras x^k é um ótimo do problema auxiliar dado que S_β é uma face ativa.

Lema 2

Considerando um problema PL e uma face S_β tal que P_β é de rank completo observamos que a eliminação de um índice associado a uma estimativa não nula produz uma direção projetiva cuja componente associada a esse índice é de sinal contrário ao da estimativa.

Consideremos $x_\beta \in \mathbb{R}^{\|\beta\|}$ tal que:

$$z_\beta = (P_\beta^t P_\beta)^{-1} P_\beta^t (c - d)$$

$z \in \mathbb{R}^n$ tal que:

$$z = I_\beta z_\beta$$

$\bar{d}$ tal que:

$$\bar{d} = -P^{\beta - \{i\}} c$$

então

SI $z_i < 0$ então $\bar{d}_i > 0$

SI $z_i > 0$ então $\bar{d}_i < 0$.

Prova

$$d - \bar{d} = (cp + d) - (cp + \bar{d})$$

Observando que tanto d como $\bar{d}$ pertencem ao subespaço associado a face $S_{\beta-\{i\}}$ concluímos que $d - \bar{d} \in E_{\beta-\{i\}}$.

Pela definição de estimativas $(cp + d) = P^{\beta} z_{\beta}$:

$$d - \bar{d} = (cp + d) - (cp + \bar{d}) = P_{\beta} z_{\beta} - P_{\beta-\{i\}} \bar{z}_{\beta-\{i\}}$$

$$d - \bar{d} = z_i P_i + (P_{\beta-\{i\}} z_{\beta-\{i\}} - P_{\beta-\{i\}} \bar{z}_{\beta-\{i\}})$$

Projetando em $E_{\beta-\{i\}}$:

$$d - d = P^{\beta-\{i\}}(z_i P_i),$$

devido a que $(P_{\beta-\{i\}} z_{\beta-\{i\}} - P_{\beta-\{i\}} z_{\beta-\{i\}})$ pertence ao subespaço complementar de $E_{\beta-\{i\}}$

$$d - \bar{d} = z_i P^{\beta-\{i\}} P_i$$

$$d - \bar{d} = z_i P^{\beta-\{i\}} PI_i.$$

Como $E_{\beta-\{i\}} \subset N(A)$:

$$d - \bar{d} = z_i \, P^{\beta-\{i\}} \, I_i.$$

Como $d_i = 0$:

$$\bar{d}_i = - z_i (P^{\beta-\{i\}} \, I_i)_i.$$

Observando que $(P^{\beta-\{i\}} I_i)_i = (P^{\beta-\{i\}} I_i)^t \, I_i > 0$ concluímos: que as componentes na direção e nas estimativas calculadas da forma indicada tem sinais opostos.

Lema 3

Dado um problema PL, um ponto viável x, uma face ativa S_β e uma direção projetiva $d = P^\beta c$: se existem duas componentes negativas i, j nas estimativas dos multiplicadores, então a projeção $\bar{d} = P^{\beta-\{i,j\}} c$ pode ser tal que: $d_i < 0$ ou $d_j < 0$.

Seja:

$$z_\beta = (P_\beta^t \, P_\beta)^{-1} \, P_\beta^t (c + d),$$

$$\bar{d} = -P^{\beta-\{i,j\}} \, c$$

então:

$$z_i, \, z_j < 0 \rightarrow \bar{d}_i > 0.$$

Prova

Por contra–exemplo:

$A = (10, 10, 1)$

$b = 1$

$c = (0, -1, -10)$

$c_p = 200/201, -1/201, -1990/201$

$x = (0, 0, 1)$

$\beta = \{2,1\}$

$z = (-99, -100)$.

A eliminação dos índices $\{1,2\}$ do conjunto ativo produz a direção $d = -c_p$ com $d_3 = -1/201 < 0$.

O seguinte lema garantia o cálculo das estimativas dos multiplicadores nas faces ativas tal e como são geradas pelo algoritmo.

Lema 4

Dado um problema PL, um ponto viável x, uma face ativa S_β, uma direção projetiva $d = -P^\beta c$: Si P_β é de rank completo RC e x_i se anula no teste de Razão

então $P_{\beta\cup\{i\}}$ é LI.

SI P_β RC, $i \in \text{argmin}_{i+1,\ldots,m}\{x_i/-d_i | d_i < 0$ então $P_{\beta\cup\{i\}}$ RC

Prova

Por contradição:

Suponha-se que $P_{\beta\cup\{i\}}$ não é de rank completo.

Como P_β é de rank completo então P_i pode ser expressado como combinação linear $y \in \mathbb{R}^{\|\beta\|}$ com $y \neq 0$:

$$P_\beta y = P_i,$$

com $i \notin \beta$.

Equivalentemente:

$$R(Pi) \subseteq R(P_\beta)$$

Tomando o subespaço complementar a cada lado da contenção obtemos:

$$E_\beta \subseteq E_i,$$

como $d \in E_\beta$ a seguinte implicação é válida:

$$d \in E_\beta \rightarrow d \in E_i$$

$$d \in E_i \rightarrow d_i = 0,$$

estabelecendo-se uma contradição com $i \in \text{argmin}_{i=1,\ldots,n}\{x_i/-d_i \mid d_i < 0\}$.

Este lema garante que as componentes ativadas pelo algoritmo não são desativadas na seguinte iteração.

Lema 5

Dado um problema PL, um ponto viável x, uma face ativa S_β, uma direção projetiva $d = -P_\beta c$: se a componente x_i se anula como resultado do teste de Razão então os multiplicadores $\bar{z}$ calculados no ponto $\bar{x} = x + \lambda d$ com $S_{\beta \cup \{i\}}$ como face ativa resultam em z_i positivo

$$i \in \text{argmin}_{i=1,\ldots}\{x_i/-d_i \mid d_i < 0\} \rightarrow \bar{z}_i < 0$$

com

$$\bar{z}\ w\ \cup\{i\} = (P^t_{\beta \cup \{i\}}\ P_{\beta \cup \{i\}})^{-1}\ P^t_{\beta \cup \{i\}}(c-d)$$

Observemos que a inversa $(P^t_{\beta \cup \{i\}}\ P_{\beta \cup \{i\}})^{-1}$ existe pelo lema anterior.

Prova

Suponha por absurdo que o multiplicador associado a i fosse negativo sua eliminação produziria a direção d tal que $d_i > 0$ pelo lema 2, então não poderia pertencer a $\text{argmin}\{x_i/-d_i,\ i \mid d_i < 0)$ estabelecendo-se uma contradição.

Observações

Estes lemas garantam um funcionamento adequado do algoritmo BEGO: Lema 1 garante a existência das estimatuvas dos multiplicadores o Lema 2 garante que a eliminação de um índice negativo gera uma direção na qual este índice não se ativa imediatamente e Lema 3 explica, ao menos parcialmente, a escolha da desativação de uma única restrição ativa. Já o Lema 5, sendo complementar ao Lema 2, garante o índice recém ativado que não se desativará imediatamente enquanto o Lema 4 garante que a matriz $P_{\beta}^{t} P_{\beta}$ é não singular.

Podemos assegurar que a desativação única usada pelo algoritmo gera direções que ativam restrições não consideradas na iteração imediatamente anterior; que a ativação única de variáveis usadas pelo algoritmo, garante a solução única do sistema de estimativas dos multiplicadores, bem como a correção do algoritmo em quaisquer das iterações sucessivas. Falta somente provar que o algoritmo não repite conjuntos ativos de maneira sucessiva seja por meio de ciclos repetidos ou por seqüências complexas.

1.7 – CONVERGÊNCIA

Na continuação provaremos a convergência exponencial do algoritmo BEGO no caso não–degenerado.

Lema 6

Dado um conjunto ordenado $K = \{1, ..., i, ..., k\}$: uma seqüência de elementos desse conjunto $Q = i^1, ..., i^t$ com $t > 2^k - 1$ possue um ciclo $i^1, ..., i^p$ que começa com o menor elemento do ciclo.

Prova

Esse tipo de ciclo em que todos os elementos j no ciclo, são tais que $j \geq i$ serão chamados ciclo de mínimo.

Usando a indução:

1) se cumpre para $k = 1$ onde a seqüência i, i de longitude dois tem um ciclo de mínimo;
2) assumimos certo para $k - 1$;
3) por contradição:

Se existe uma seqüência sobre k elementos de longitude maior que $2^k - 1$ sem ciclos de mínimos, então existe uma de longitude exatamente igual a 2^k, basta eliminar os elementos que sobraram ao final da seqüência, seja C uma seqüência de longitude 2^k sem ciclos de mínimo sobre k elementos.

Procede-se a eliminar de C todos os elementos máximos, podem existir como máximo k elementos máximos, pois dois elementos consecutivos constituem um ciclo de mínimo.

Como os elementos eliminados eram máximos não aparecem ciclos de mínimos, obtendo-se uma seqüência $\bar{C}$ de longitude 2^{k-1} maior que $2^{k-1} - 1$ sem ciclos de mínimo sobre o conjunto sobre o conjunto $\{1, ..., k - 1\}$ o que contradiz 2).

Observação

As direções geradas pelo algoritmo dependem exclusivamente do conjunto ativo β. Desta forma poderemos denominá–las d_β.

Podemos estabelecer uma ordem nos conjuntos ativos em relação com o declive das direções que geraram, isto é, associando a cada conjunto ativo β um valor real $\delta(\beta)$

$$\delta(\beta) = -c^t\left(\frac{d_\beta}{|d_\beta|}\right).$$

Observação

Existe um número finito de conjuntos ativos β e por conseguinte um número finito de faces S_β e de direções projetivas d_β.

Estes conjuntos ativos podem ser ordenados por declive em uma lista finita $\beta^1, \ldots, \beta^k$.

Teorema 1

O algoritmo, em problemas não degenerados, não gera ciclos de conjuntos ativos que começe pelo conjunto ativo cuja direção é a de menor ou igual declive no ciclo.

Prova por contradição

Suponha–se que o algoritmo gera um ciclo de conjuntos ativos tal o primeiro elemento é de menor ou igual declive no ciclo

$$C = \beta^i, \ldots, \beta^j, \ldots, \beta^i$$

com $i < j$ para todo elemento j no ciclo diferente de i.

Seja p o número de elementos no ciclo, consideremos a seqüência de pontos viáveis não co–lineares associada ao ciclo:

$$x^i, \ldots, x^{i+p}$$

Donde todos são pontos viáveis diferentes, que pertencem as faces ativas correspondentes S_β

$$x_{i+1} = x_i + \lambda_i d_i,$$

define a seqüência de pontos nas iterações do algoritmo.

Seja $\hat{d}$ a diferença entre o primeiro e o último ponto na seqüência de pontos viáveis

$$\hat{d} = x^{i+p} - x^i.$$

O declive mínimo no ciclo $\delta(\beta^i)$ é menor que o declive pro–médio no ciclo que é menor que o declive de $\hat{d}$,

$$\delta(\beta^i) \leq -\frac{c^t\ \hat{d}}{\sum_{j=i}^{i+p} |x^j\ \ d^j|} < -\frac{c^t \hat{d}}{|\hat{d}|}$$

Como o problema é não degenerado, os pontos gerados pelo algoritmo não são co–lineares, portanto a propriedade triangular se cumpre estritamente

$$\delta(\hat{d}) > \delta(d^i)$$

Contradição, pois d^i é uma direção projetiva a de maior declive no subespaço β^i.

Teorema 2

O algoritmo BEGO é ótimo em um número finito de iterações. Seja $\ell = 2^n$ o número máximo de faces no poliedro S tomemos:

$$K = \{\beta_1, \ldots, \beta_\ell\}$$

o conjunto de todas as faces ordenadas por declive.

O algoritmo BEGO gera uma seqüência de conjuntos ativos

$$Q = \beta_j^1, \ldots, \beta_j^k, \ldots$$

De longitude menor que 2^ℓ, pois ele não gera ciclos de mínimos. Portanto para um número menor que 2^ℓ iterações.

Por outra parte, o algoritmo só pára se se cumprem as condições de otimalidade.

Concluímos que o algoritmo detem-se em uma solução ótima, depois de um número finito de iterações.

Observação

Esta demonstração é geral, no sentido de que qualquer algoritmo que gera direções projetivas independentemente do critério de manejar o conjunto ativo não produz ciclos de mínimo.

1.8 – ALGORITMOS DIFERENTES A DIREÇÕES VIÁVEIS

1.8.1 – Preliminares

Existem várias classificações dos algoritmos usados para resolver o problema de programação linear:

É amplamente conhecida a equivalência entre o problema PL e o problema de viabilidade:

Definição: Problema de Viabilidade:

Achar x

s.a.: $Ax = b$

$x >= 0$

O conjunto viável S pode ser expressado:

$$S = \{x \in \mathbb{R}^n \mid Ax = b,\ x \geq 0\},$$

desta forma, o problema de viabilidade pode ser expresso como:

Achar $x \in S$

Não levaremos em consideração para nossa classificação, dada a natureza imediata da equivalência, o fato do algoritmo obter um ponto ótimo ou um ponto viável.

Consideraremos em nossa classificação, a natureza do princípio utilizado no processo da iteração.

1.8.2 – Algoritmos Volumétricos

São aqueles que definem uma seqüência de conjuntos $E^0, E^1, \ldots, E^k, \ldots, \hat{E} \subset \mathbb{R}^n$ tal que $S \subseteq E^k$. Uma redução do volume do conjunto E^k se efetua em cada iteração: $E^{k+1} \subset E^k$ preservando–se a característica de conter o conjunto viável.

O mais conhecido exemplo da classe é o algoritmo elipsoide [3] por ter sido o primeiro a provar convergência polinominal na solução do problema PL. Outras figuras geométricas tem sido usadas preservando o mecanismo original, conseguindo–se em alguns casos reduzir a ordem de convergência.

Nesta classe, de grande importância teórica, não se tem conseguido até agora resultados satisfatórios na solução de problemas práticos. Algoritmos deste tipo tem

sido usados na solução de modelos combinatórios.

1.8.3 – Algoritmos de Pontos Interiores

São os que geram uma seqüência de pontos $x^0, x^1, ..., x^k, ..., \hat{x} \in S$ tal que $x^k > 0$, i.é., x^k está contido no interior relativo deS.

Os algoritmos conhecidos nesta classe são: Algoritmo Karmarkar [4], Algoritmo Afim Escalamento, Algoritmos de Trajetória Central, etc. Tais algoritmos são grande importância teórica, pois são conseguem provar ordem de convergência $O(n^3L)$ operações aritméticas, e os melhores até agora obtidos. Diversas implementações têm obtido, resultados satisfatórios na solução de problemas práticos, especialmente problemas de grande porte.

1.8.4 – Algoritmos Discretos

Geram em forma ordenada combinações de restrições. Dado que, na prática, apresentam um comportamento exponencial, são aplicáveis somente a problemas de dimensões reduzidas. Possuem importância histórica, pois foram originalmente propostos por Fourier para a solução de desigualdades simultâneas.

1.9 – ALGORITMOS DE DIREÇÕES VIÁVEIS

1.9.1 – Preliminares

São os que geram uma seqüência de pontos $x^i, ..., x^k, ..., \hat{x} \in S$ na fronteira de S, salvo para instâncias de natureza trivial.

Nesta seção apresentaremos uma definição operativa da classe de algoritmos de direções viáveis, apresentaremos os exemplos mais conhecidos da classe, outras metodologias semelhantes e explicaremos as vantagens que seu uso pode apresentar quando aplicados a modelos combinatórios.

O algoritmo Simplex, amplamente conhecido pertence a esta classe. Possue a particularidade de gerar unicamente vértices. O algoritmo Simplex é na realidade uma classe de algoritmos, pois seu comportamento varia consideravelmente com a variação dos critérios de pivoteamento e escalamento. O conhecimento prévio das características do problema permite melhorar o desempenho do algoritmo em problemas de grande porte.

Historicamente o termo direções viáveis aparece associado a variações do algoritmo Simplex [5] (Multiplex) agrupando também algoritmos projetivos para problemas não lineares [6].

Na literatura o termo é usado em referência a algoritmos cercadamente relacionados ao Simplex [7] Kortanek, em ocasiões modificações dos mais conhecidos códigos de Simplex [8].

Os algoritmos de direções viáveis pertencem a classe de algoritmos de conjuntos ativos.

Os mais conhecidos algoritmos da classe de direções viáveis são o Simplex em programação linear e o algoritmo projetivo de Rosen [6] em programação não-linear. Variações destes esquemas constituem o grosso da literatura estudada. Nos referiremos mais adiante a direções viáveis exclusivamente no contexto da programação linear, quando o algoritmo genérico apresentado permite a não

linearidade da função objetivo.

É interessante observar a ausência de algoritmos de direções viáveis orientados à solução de modelos combinatórios e de programação inteira, problemas para os quais a metodologia parece apropriada.

Nenhum algoritmo na classe possue uma demonstração de complexidade polinomial no número de operações aritméticas, existem provas de complexidade de ordem exponencial para vários algoritmos de direções viáveis incluindo Simplex; outros sem impedimento não apresentam provas de convergência.

Usaremos uma definição operativa da classe direções viáveis usando um patrão de algoritmo, isto é, uma descrição propositadamente vaga dos passos fundamentais que todo algoritmo de direções viáveis deverá seguir.

1.9.2 – Descrição Genérica

Definição

Dizemos que um algoritmo pertence a classe de direções viáveis se é conformável com o seguinte módulo de algoritmo, isto é, se pode ser obtido por meio de substituições apropriadas na descrição genérica sem deformar o esquema geral da mesma.

ALGORITMO GENÉRICO CONCEITUAL DE DIREÇÕES VIÁVEIS

1) Obtenção de um ponto inicial viável x^0.

Se não se conhece um ponto inicial viável se obtém por meio de variáveis artificiais; obviamente este ponto inicial deverá cumprir requerimentos próprios do algoritmo, como o de ser um vértice, no caso de Simplex.

2) Seleção do conjunto ativo inicial

Escolhas comuns são: conjuntos básicos, ou o conjunto vazio. Em certos casos não existe relação com o ponto inicial e o conjunto ativo nas iterações.

3) REPITA

4) Cálculo de uma direção d_i viável em x_i.

É neste ponto onde os algoritmos de direções viáveis estabelecem diferenças notáveis: alguns escolhem direções que podem ser calculadas rapidamente mais que não são necessariamente as melhores, outros empenham consideráveis esforços na obtenção de uma direção.

5) Obtenção do *passo* $x_{i+1} := x_i + \lambda d_i$.

O cálculo do tamanho do passo é com freqüência obtido mediante o Teste de Razão mais em alguns casos, passos de menor tamanho podem ser usados.

6) Atualização do conjunto ativo

Em geral a restrição obtida mediante o Teste de Razão passa a ser ativa, no caso degenerado várias restrições se obtem do Teste de Razão e cada algoritmo resolve este problema com alguma política anticiclo específica.

7) Condição para estimar os multiplicadores

A estimativa dos multiplicadores pode gastar muito tempo, pois envolve uma inversão, alguns algoritmos evitam efetuá–la em todas as iterações.

8) Estimativa de multiplicadores

Em geral a positividade da estimativa é tomada como reafirmação da atividade da restrição.

9) Reatualização do conjunto ativo

Algumas das restrições com estimativas negativas poderiam sair do conjunto ativo, a política de manejo dos elementos ativos é elemento principal no desempenho de algoritmos de direções viáveis.

10) Condições de otimalidade.

Os algoritmos de direções viáveis assim com os de conjunto ativo param quando as condições de otimalidade são cumpridas pelas restrições ativas.

Apresentaremos algumas das características de vários algoritmos de direções viáveis diferentes de Simplex.

1.9.3 – Método Transversal

Proposto por BROWN e KOOPMANS [9] em 1951 é um método que alterna dois tipos de direções: uma é a direção de custo projetado e outra uma direção de centralização.

Partindo–se de um ponto inicial interior $x^0 \in S$, $x^0 > 0$, o algoritmo dá um passo na direção oposta ao custo projetado, até anular uma variável (excepcionalmente várias variáveis se anulam simultaneamente).

A direção $h = -cp$ com $cp = P_c$ o custo projetado no nulo de A permanece inalterada durante todo o processo.

Em cada iteração partindo–se de um ponto interior obteremos um ponto interior, com um custo maior alternando a direção h com direções de centralização

$$\tilde{x}^{k+1} = x^k + \lambda h$$

Esta direção nos permite melhorar o custo $c^t \tilde{x}^{k+1} < c^t x^k$ devido a que x^k é um ponto interior $\lambda > 0$ pode ser calculado mediante o Teste de Razão

$$\lambda = \min_{i \mid h_i < 0} \{-x_i/h_i\}$$

Pelo menos uma restrição de desigualdade se cumpre como igualdade, depois de dado o passo e não se pode continuar na direção h sem sair do conjunto viável. Se faz necessário gerar uma direção viável a partir de $\tilde{x}^{k+1}$, para isso se gera uma direção $\tilde{h}$ que será positiva na variável da restrição (restrições) que se anula no teste de

razão.

Para isso, o método obtem um ponto w^{k+1} pertencente a fronteira da interseção da variedade paralela ao nulo de cp que passa por $\tilde{x}^{k+1}$ com o politopo S; este ponto deve ser positivo na restrição anulada.

Uma combinação convexa destes dois pontos pertencem ao interior S, isto é:

$$x^{k+1} = \gamma \tilde{x}^{k+1} + (1 - \gamma)\, w^{k+1}$$

O algoritmo usa $\gamma = 1/2$, no entanto numa escolha mais adequada poderia ser proporcional ao ângulo que forma a linha $(\tilde{x}^{k+1}, w^{k+1})$ com as faces do politopo.

Encontrando-se novamente o interior S o método, usa novamente a direção $h = -cp$, e assim sucessivamente até obter um ótimo. Este método pode não converger em um número finito de passos.

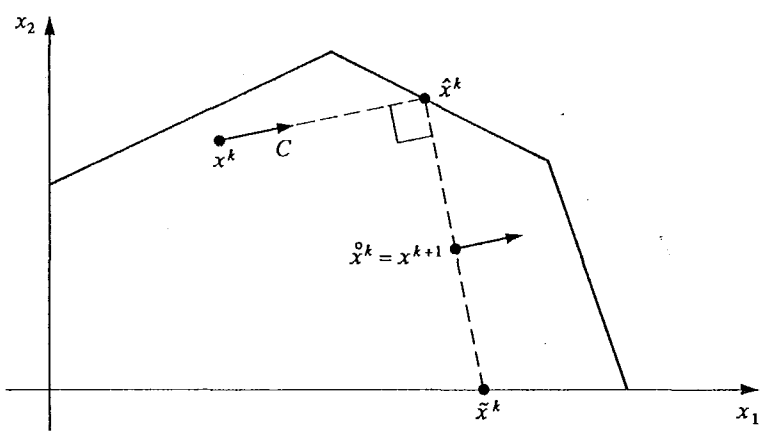


Fig. I.1

Apresentaremos várias considerações gerais em relação com as características do algoritmo transversal.

As duas técnicas que se alternam no desenvolvimento do algoritmo: melhorar o custo, se afastar da fronteira; ilustram uma dicotomia intrínseca ao problema PL.

A obtenção de um ponto interior de mesmo custo a um ponto na fronteira do politopo não é um problema trivial.

1.9.4 – Algoritmos de Direções de Busca

Apresentado originalmente por FRISCH [5] em 1957, estes algoritmos pertencem a uma classe de algoritmos de direções viáveis que podem ser consideradas como uma extensão do método Simplex.

O algoritmo arranca com um ponto interior x^0 mais na seguintes os pontos x^k gerados, pertencem pelo menos a uma face do poliedro.

Toma–se uma base A_β qualquer; não precisa ser viável, esta base sempre existe, pois a matriz A é de rank máximo. Desta forma dividimos o conjunto de variáveis em básicas e não básicas, particionando os índices nos conjuntos B e $NB = \{1, ..., n\} - B$.

Construa–se o vetor π das variáveis duais básicas:

$$\pi^t = c_B^t \, B^{-1}$$

e o custo associado à base da seguinte maneira:

$$\bar{c}^t_{NB} = c^t_{NB} - \pi^t A_{NB}$$

Sendo $\bar{c}_B = 0$ o que permitiria assegurar que a direção de busca, a ser gerada, melhora o custo.

Existem várias formas de gerar um conjunto de variáveis $P \subseteq NB$ chamadas promissórias. A mais freqüente na literatura consiste em escolher todas as não básicas que formam uma direção viável que decresce o custo no ponto x^k:

$$i \in P \mapsto (\bar{c}_j < 0, x_j \geq 0), (\bar{c}_j > 0, x_j > 0)$$

Agora determinamos a direção h de maneira que seja uma direção de descanso

$$h = h_P + h_{\{NB-P\}} + h_B$$

$$h_P = \bar{c}_P$$

$$h_{NB-P} = 0$$

$$h_B = - B^{-1} A_P \bar{c}_P$$

Dois lemas permitem assegurar a propriedade desta escolha: no primeiro se prova que esta direção é de descanso para qualquer base e no segundo se prova que se $P = \phi$ então o ponto x^k é ótimo.

Esta técnica permite anular várias restrições simultaneamente; diferentes

formas de determinar o conjunto promissor P e o quociente π geram os diferentes algoritmos de direções de busca.

Este tipo de algoritmos tem sido revisado recentemente à luz da metodologia de pontos interiores KLOTZ [8], NAZARETH [10].

1.9.5 – Algoritmos por Purificações Sucessivas

O problema de purificação:

Dados:

$A \in \mathbb{R}^{m \times n}$, $c, x^0 \in \mathbb{R}^n \mid ax^0 = b$

achar:

$\bar{x} \mid c^T\bar{x} \geq c^T x^0$. $A\bar{x} = b$, $\bar{x}$ o vértice.

Este problema que sobre um problema limitado constitue um exercício elementar da otimização pode ser resolvido em ordem $O(n^3)$ usando direções tipo Simplex. Pode também ser resolvido usando direções projetivas sobre as faces do poliedro.

Obviamente se alternarmos a solução de um problema de purificação com um problema de centralização, solucionaremos o problema PL por direções viáveis.

KORTANEC [7] apresenta um algoritmo usando direções Simplex para resolver o problema de purificação e logo usa um processo semelhante ao de direções

de busca para conseguir uma direção de centralização.

Partindo–se de um ponto viável x^0 e uma base não necessariamente viável B escolhe–se um $i \in NB$ qualquer para gerar uma direção no nulo da matriz B aumentada com $A_{\{i\}}$ como sabemos esta direção pode corresponder com uma aresta do politopo $Ax = b,\ x \geq 0$

$$h = h_i + h_{NB-\{i\}} + h_B$$

$$h_i = 1$$

$$h_{\{NB-\{i\}} = 0$$

$$h_B = -B^{-1} A_P B$$

Existem duas possibilidades se $c^t h \leq 0$ se tem uma direção não crescente a qual é apropriada para a resolução do problema de purificação caso contrário se toma $h = -h$ a qual é uma direção de descida.

Se realiza o teste de razão que pode resultar em um $\lambda = 0$ pois a direção não é necessariamente viável em x^0. Obtendo–se uma variável $i \in N$ tal que $x^1 = 0$; se $i \in B$ ela deve sair da base, calculando–se uma nova base e se repete o processo até anular $n - m$ variáveis. Finalmente nos encontramos em um vértice x.

Agora devemos gerar uma direção de centralização usando uma direção de busca das geradas pelo algoritmo: teremos:

$$h^c = x - x_0 - \lambda h^j$$

$$\lambda = \frac{c^t x - c^t x^0}{c_j - z_j}$$

com $c_j - z_j > 0$.

Pode se observar que esta direção está no nulo de A, é nula no custo e efetivamente tem componentes positivos em todas as variáveis não básicas.

1.9.6 – Algoritmos de Custo Projetado

O algoritmo PROJECT foi apresentado por RODDER e BLAUTH [11] em 1981, sendo uma adaptação a programação linear do algoritmo de gradiente projetado de ROSEN [6] originalmente apresentado em 1960.

Descrição do algoritmo:

1) Obter uma solução inicial viável $x^0 \in S$.

2) Calcular a projeção de c no nulo de A, $h = -cp = -Pc$.

3) Dar um passo nesta direção até anular uma ou mais variáveis $x^{k+1} = x^k + \lambda h$ as variáveis recém anuladas se tornam ativas.

4) Projetar o custo sobre a face ativa $h = -\bar{P}c$ com $\bar{P}$ a matriz de projeção no subespaço $Ax = 0$, $x_i = 0$ para todo i ativo.

5) Se o gradiente projetado é nulo fazer um teste de otimalidade e parar se a solução é ótima.

6) Caso contrário, eliminar do conjunto ativo as variáveis assinaladas pelo teste de otimalidade.

A diferença dos algoritmos de direção de busca que procuram a eliminação múltipla de variáveis neste tipo de algoritmos somente se consegue eliminar uma variável em cada iteração; em compensação o declive das direções projetivas é em geral maior.

1.9.7 – Conclusões

A diferença fundamental entre os algoritmos apresentados neste trabalho e o algoritmo PROJECT, é que PROJECT somente efetua o teste de otimalidade quando o gradiente projetado se anula; nos algoritmos apresentados efetua o teste de otimalidade para um problema associado em cada iteração.

Pode se observar: os algoritmos de direções de busca, o algoritmo de Gradiente projetado de Rosen, e os algoritmos de direções projetivas são muito semelhantes entre eles e com o Simplex. O fato de usar um conjunto de variáveis básicas que parece a diferença fundamental, não é devido a que várias características da direção gerada pelos algoritmos de direções de busca são independentes da base.

Assim como no Simplex as regras de pivoteamento são definitivas no comportamento do algoritmo; o manejo do conjunto ativo é o elemento fundamental no comportamento dos algoritmos de direções viáveis, sejam eles projetivos ou de direções de busca.

1.10 – CARACTERÍSTICAS COMBINATÓRIAS DOS ALGORITMOS DE DIREÇÕES VIÁVEIS (ADV)

1.10.1 – Estrutura Combinatória do Simplex

A natureza combinatória do algoritmo Simplex tem sido ilustrada usando um grafo.

É um fato bem conhecido que o Simplex se move pelos vértices do poliedro usando as arestas como direções viáveis, até conseguir um vértice ótimo. Esta característica permite reduzir o tempo empregado por iteração, mais ao não considerar algumas faces do poliedro não consegue refletir integralmente a estrutura combinatória do poliedro S.

Esta característica é determinante no comportamento do Simplex quando aplicado a modelos combinatórios.

1.10.2 – Estrutura Combinatória dos ADV

1.10.2.1 – Generalidades

Uma característica comum a um grande número de algoritmos é a de descrever uma trajetória poligonal através da generação sucessiva de pontos viáveis $x^0, x^1, \ldots, x^i \in S$.

Quando essa seqüência de pontos pertencem ao interior do poliedro dizemos que o algoritmo é de ponto interior. Ao contrário, quando essa seqüência de pontos pertencem a fronteira do poliedro, a diferentes faces do poliedro, então denominamos o algoritmo como de direções viáveis.

A seqüência de pontos corresponde a uma seqüência de faces as quais os pontos pertencem. Estas faces são uma expressão combinatória das iterações do algoritmo.

A parte combinatória dos algoritmos de Programação Linear pode ser vista em termos de grafos como um caminho pelas faces, até uma face ótima.

1.10.2.2 – Uma Partição do Conjunto Viável S

Uma partição dos pontos de S seria a de pontos interiores e pontos na fronteira.

Uma partição mais fina pode ser definida da seguinte forma:

$$x \equiv y \leftrightarrow \{i \,|\, x_i = 0\} = \{i \,|\, y_i = 0\}$$

Esta é uma relação de equivalência por ser definida como igualdade numa função e portanto gera uma partição do conjunto S.

As classes de equivalência são o interior relativo das faces do politopo. Serão denominadas faces abertas por ser abertos na topologia relativa [12].

É natural definir as classes desta maneira por ser partições das faces do poliedro.

1.10.2.3 – Direções Viáveis

Os algoritmos de direções viáveis geram uma seqüência de pontos $x^0, \ldots, x^i \in S$

e conseqüentemente uma seqüência de direções

$$d^k = \frac{x^{i+1} - x^i}{\lambda} \in N(A).$$

A escolha da direção junto com o manejo do conjunto ativo são os elementos fundamentais dos algoritmos de direções viáveis. Estes são dois aspectos que estão fortemente relacionados.

Enumeramos algumas condições que as direções deverão cumprir:

1) A direção d^k deve ser afim, isto é, $Ad^i = 0$.

2) A direção d^k deve ser viável, isto é, positiva ou nula nas componentes i em que $x_i = 0$.

3) A direção não deve piorar o custo $c^t d^k \geq 0$, melhor ainda a direção deveria melhorar o custo $c^t d^k > 0$.

Uma pequena troca na escolha da direção basta para produzir uma troca considerável no comportamento combinatório do algoritmo.

Desta forma numerosos comportamentos combinatórios podem ser associados aos algoritmos de direções viáveis.

Neste trabalho procuramos utilizar a estrutura combinatória do algoritmo apresentado, na solução de um problema combinatório, (Busca em Bases de Conhecimento).

1.11 – CONCLUSÕES

1) Os algoritmos de direções viáveis tem uma forte componente combinatória expressada nos conjuntos ativos e em sua evolução durante a solução do problema de programação linear.

2) O algoritmo Simplex trabalha unicamente nos vértices reduzindo o número de conjuntos ativos possíveis.

3) Os algoritmos de direções viáveis em geral aceitam todas as faces do poliedro como conjuntos ativos, esta característica assinala a eventual vocação destes algoritmos na solução de modelos combinatórios.

4) Na prática, os conjuntos ativos variam pouco em iterações sucessivas, ou bem acrescentando componentes ou bem eliminando componentes. Esta variação, quando associada a um modelo combinatório, corresponde com uma forma metódica de gerar a combinação que resolve o problema original.

CAPÍTULO 2

COMBINATÓRIA

Neste capítulo apresentaremos um problema combinatório e sua solução combinatória mais conhecida.

Inicialmente apresentaremos mediante exemplos, alguns tipos de sistemas de informação.

Posteriormente definiremos suas funções e as partes em que convencionalmente pode ser dividido. Os problemas que se apresentam com freqüência no manuseio de sistemas de informação são caracterizados.

Definiremos formalmente um tipo de sistemas de informação baseados em regras simples chamados de sistemas de Horn.

A matriz associada a um sistema de Horn é definida e ilustrada mediante um exemplo.

A solução do problema de busca mediante ao algoritmo Top–Down é apresentada.

Um inventário de vários trabalhos importantes na área de linguagens naturais ilustra o contexto em que os problemas associados ao manuseio de sistemas de informação são solucionados.

Finalmente se apresenta um princípio diferente para a solução deste tipo de

problemas, procurando resolver os requerimentos específicos do processamento de informação em sistemas interativos com humanos.

2.1 – BUSCA EM SISTEMAS CLAUSULARES DE HORN

Apresentaremos os sistemas de informação em geral, os sistemas clausulares e o caso particular que nos interessa mais os sistemas de Horn.

2.1.1 – Sistemas de Informação

De início, apresentaremos alguns exemplos de sistema de informação para logo estudar as partes que os compõem, em seguida mostraremos uma lista de problemas semelhantes que se apresentam freqüentemente no manuseio de sistemas clausulares.

2.1.1.1 – Generalidades

Os sistemas de informação podem ter as mais variadas características:

Numa biblioteca consultamos um fichário em ordem alfabética por autores ou título, isto constitue um sistema de informação em sua forma mais simples.

Um exemplo mas complexo seria um sistema de diagnóstico médico no qual depois de introduzir alguns sintomas e características do paciente, o sistema solicitaria alguma informação adicional, estabelecendo–se um diálogo ao redor dessa consulta, que deve terminar em um relatório de informações pertinentes.

Outro exemplo ainda mais complexo seria um sistema para ensinar matemática moderna a crianças pequenas, com ênfase na comunicação falada, a complexidade reside em que o diálogo não está enquadrado em um protocolo estrito e o sistema necessita processar as respostas do usuário em vários níveis tanto sintáticos como semânticos.

Um diálogo hipotético em um ambiente deste tipo pode ilustrar algumas das dificuldades associadas a processar a informação em ambientes interativos.

1) o sistema: quantos são dois mais dois ?
2) a criança: quatro
3) o sistema: E três mais um ?
4) a criança: o mesmo...
5) o sistema: muito bem. Vejamos agora se sabes quanto é três mais três
6) a criança: quero ir ao banheiro
7) o sistema: volte antes de 5 minutos.

Vários problemas característicos dos sistemas de informação são ilustrados no exemplo:

Elipsis, fala indireta, reconhecimento de planos conflitivos, referências ao contexto anterior.

Na linha 3) o sistema evita repetir "quantos são" para dar naturalidade ao diálogo, aos olhos do usuário este comportamento é completamente transparente.

Na linha 4) a criança dá uma resposta difícil de processar:

a) não é um número;

b) deve ser processada a um nível mais geral, é uma resposta indireta ou um comentário paralelo;

c) "o mesmo" é uma referência a resposta anterior, comparar com a resposta correta e observar que a pergunta foi corretamente respondida.

Na linha 5) observamos que o sistema cumprimenta o usuário e procede realizando outra pergunta de uma forma indireta, apresentando um conhecimento motivacional.

Na linha 6) oferece outras dificuldades associadas ao contexto.

Estas dificuldades associadas a comunicação se estabelecem como uma barreira no desenvolvimento dos sistemas de informação. Considerável esforço está sendo dirigido para melhorar nossa compreensão dos mecanismos de processamento de informação.

Este tipo de problemas característicos dos sistemas de informação se resolvem em muitos casos mediante processamento de listas. Linguagens de programação como Lisp e Prolog se desenvolveram com o específico objetivo de programar sistemas desta natureza mediante o processamento de listas.

Um sistema de informação pode ser dividido conceitualmente em duas partes: a base de conhecimento e o manejador do sistema.

A seguir apresentaremos uma descrição menos informal dos sistemas de informação em geral.

2.1.1.2 – Partes de um Sistema de Informação

Um sistema de informação possue dois módulos fundamentais: base de conhecimento e sistema de controle.

i) a base de conhecimento consiste:

1. conjunto de elementos $N = \{1, 2, ..., m\}$
2. lista de regras $A = \{a_1, a_2, ..., a_n\}$
3. axiomas para serem utilizados como argumentos das regras
4. regras de domínio ou critério de parada

ii) o sistema de controle é um programa que combina as regras de forma metódica respeitando os axiomas até conseguir uma distribuição apropriada. Isto é, conseguir uma combinação de regras que preservem os axiomas e cumpram com o critério de parada. Ele também altera a base de conhecimento preservando a integridade desta.

A parte do manejador do sistema que revisa e combina as regras até conseguir uma distribuição apropriada se chama de algoritmo provador (prover).

O manuseio de sistemas de informação é um dos problemas mais importantes da computação. Apresentaremos um mecanismo novo para solucionar alguns problemas inerentes ao processo de consulta e manutenção de sistemas de informação. Pretendemos construir um provador usando modelos combinatórios e algoritmos de programação linear.

2.1.1.3 – Problemas Específicos

Vários problemas encontrados com freqüência no manuseio de sistemas de

informação são semelhantes, sua solução é o objetivo principal do presente trabalho:

Um sistema de informação é um conjunto de elementos agrupados e associados mediante regras explícitas.

Na prática um conjunto de propriedades está associado aos elementos do sistema, sua natureza e utilidade são improcedentes no contexto do presente trabalho.

Em qualquer sistema de informação é necessário desenvolver mecanismos que nos permitam acelerar e aumentar nossa capacidade para obter propriedades dos elementos.

O que é mais importante, será necessário criar mecanismos que nos permitam obter propriedades dos elementos do sistema em um ambiente de interação com humanos não treinados.

Um caso particular é estabelecer a validez de predicados mediante a combinação apropriada de regras. Isto é o que se conhece como estabelecer um "matching" ou casamento. Por exemplo um sistema de informação é consultado com uma entrada em forma de pergunta: é Rex um cão ? se é possível associar mediante a combinação apropriada de regras, os elementos na frase, a resposta é: (certo); do contrário a resposta é: (certo ou falso). De qualquer forma a resposta é um predicado verdadeiro que se refere a configuração atual do sistema de informação.

Esta combinação apropriada de regras é conhecida como "estado de conhecimento" e permite ao sistema de informação gerar mensagens que eventualmente tenham sentido para o usuário.

Estes sistemas de informação podem ter as mais variadas características com esquemas altamente flexíveis como "scripts"e "frames" [12] ou pelo contrário esquemas rígidos como "block word"e os usados no jogo de xadrez; sistemas mixtos com módulos independentes são usados na solução de problemas de compressão de diálogos e de histórias.

Existem alguns problemas específicos que se apresentam com freqüência no manuseio de sistemas de informação:

1) Problema de dedução ou de busca:

Consiste em verificar a validez de um predicado no sistema de informação.

2) Demonstração generalizada:

Consiste em conseguir as regras no sistema de informação que podem ser usadas na verificação de um predicado.

3) Problema de demonstração:

Conseguir uma seqüência de regras que permitam verificar um predicado.

4) Problema perturbado:

Verificar um predicado em um sistema levemente alterado.

Estes problemas serão definidos formalmente na medida em que os sistemas de informação a serem estudados sejam detalhados, pois obviamente o significado da

verificação de um predicado é dependente da natureza do sistema de informação.

2.2 – SISTEMAS CLAUSULARES

Apresentaremos uma definição de cláusula em geral, definição de cláusulas de Horn como um caso particular e vários conceitos associados a demonstrações em sistemas de Horn.

2.2.1 – Cláusulas Lógicas

Existem muitos esquemas de representação de conhecimento, alguns deles, como vimos, possuem uma grande flexibilidade, mais a lógica simbólica é possivelmente o modelo mais geral e exato na representação do conhecimento.

Dado um conjunto $N = \{p_1, p_2, ..., p_m\}$ de predicados, variáveis lógicas ou proposições atômicas, qualquer proposição lógica complexa sobre esse conjunto pode ser expressada em forma clausular, como conjunção finita de cláusulas lógicas. Esta conjunção se conhece como um sistema de cláusulas:

$$A = \{a_1, a_2, ..., a_n\}$$

Donde cada cláusula é uma expressão da forma seguinte:

$$a = \bigwedge_{k \in K} p_k \rightarrow \bigvee_{i \in I} p_i;$$

donde: $I, K \subseteq \{1, 2, ..., m\}$ subconjuntos disjuntos de índices $I \cap K = \phi$.

2.2.2 – Cláusulas de Horn

Um tipo particular de cláusulas são as chamadas cláusulas de Horn com um único elemento no conjunto implicado:

Definição: Cláusula de Horn

Dado um conjunto $N = \{1, ..., m\}$ uma cláusula de Horn é uma expressão do seguinte tipo:

$$a = \bigwedge_{k \in I} p_k \rightarrow p_i$$

com $I \subset N$, $i \in N$, $i \notin I$.

Uma cláusula pode ser representada por um par:

$$a = (I, i),$$

com $I \subset N$, $i \in N$, $i \notin I$, ou de maneira equivalente

$$a = (A(a), S(a))$$

com $A(a)$ o antecessor de a e $S(a)$ o sucessor de a.

Definição: Sistema de cláusulas de Horn

$$A = \{a_1, a_2, ..., a_n\},$$

um sistema de Horn é uma lista de cláusulas de Horn.

Este tipo de cláusula não sendo completamente geral apresenta a vantagem de sua simplicidade e a capacidade para representar muitos problemas de interesse prático. Como conseqüência os sistemas de Horn são usados extensivamente em diversas áreas de aplicação assim como em linguagens baseadas em cláusulas de Horn, como Lisp e Prolog. Algoritmos provadores em sistemas deste tipo são de importância central na solução eficiente de numerosos problemas de computação.

As cláusulas de Horn podem ser usadas em reconhecimento de planos, pois um plano complexo pode ser visto como uma seqüência de planos simples, cada um representado por uma cláusula Horn, como um conjunto de pré–requisitos para obter um objetivo.

Um sistema de Horn A sobre N é um conjunto de cláusulas de Horn que se referem a proposições atômicas em N.

2.2.3 – Definições em Sistemas de Horn

Definição: Estado de conhecimento

É um conjunto das cláusulas do sistema.

Uma proposição atômica p_i é chamada de proposição domínio se no sistema de cláusulas é referenciada em uma cláusula com o lado esquerdo vazio:

$$\phi \rightarrow p_i$$

O domínio $0(A)$ de um sistema é o conjunto dos elementos referenciados em proposições de domínio em A:

$$0(A) = \{p \in N \mid (\phi \rightarrow p) \in A\}.$$

Definição: Demonstração restrita

Uma demonstração de ℓ em um sistema de Horn $A = \{a_1, ..., a_n\}$ é uma seqüência de cláusulas $D = (d_1, ..., d_q)$ tais que:

1) $\forall d_k = (I_k, i_k) \in D: I_k \subseteq 0(A) \bigcup_{j<k} \{i_j\}$

2) $\forall d_k = (I_k, i_k) \in D: i_k \notin 0(A) \bigcup_{j<k} \{i_j\}$

3) $\forall d_k = (I_k, i_k) \in D: i_k \in \bigcup_{j>k} I_j$

4) $d_q = (I_q, \ell)$.

Isto é toda cláusula na seqüência usa somente proposições previamente demonstradas, toda cláusula na seqüência demonstra proposições não demonstradas antes de serem utilizadas posteriormente e a cláusula final demonstra ℓ.

Definição: Demonstração irrestrita

O conceito é análogo ao de demonstração somente que aceita ciclos, restando da definição anterior somente os ítem 1), 3) e 4).

Definição: Demonstração generalizada

É a união de todas as cláusulas que aparecem em alguma demonstração irrestrita de ℓ em A.

Definição: Demonstração total

É a união de todas as cláusulas que aparecem em alguma demonstração restrita de ℓ em A.

Definição: *Combinação de demonstrações*

É a união das cláusulas em várias demonstrações de ℓ em A.

Estes conceitos poderiam gerar estados de conhecimentos importantes por sua associação a princípios fundamentais da compreensão de diálogos e histórias.

Definição: Busca ou dedução

O problema de busca ou dedução é: dado um sistema de Horn A sobre N, provar a existência de uma demonstração de $\ell \in N$ em A.

Estados de conhecimentos significativos resultam da manipulação da lista de cláusulas em um processo algoritmico para solucionar buscas particulares. Uma demonstração é um estado de conhecimento na medida em que suas cláusulas definem um subconjunto do sistema, uma combinação de demonstrações também é um estado de conhecimento.

Quando falamos de estados de conhecimentos relacionados nos referimos a subconjuntos significativos no contexto de um sistema clausular particular, que estão

relacionados desde o ponto de vista da estrutura do sistema. Um exemplo de estados de conhecimentos relacionados vem dado pelas demonstrações diferentes da mesma proposição.

2.3 – MATRIZ DO SISTEMA DE HORN

Um sistema $A = \{a_1, ..., a_n\}$ de cláusulas lógicas não tautológicas pode ser expressado em forma matricial com [m x n] entradas 1, 0, −1 da seguinte maneira:

$$A = \begin{cases} a_{i,j} = 1 & \text{se } i = S(a_j) \\ a_{i,j} = -1 & \text{se } i \in A(a_j) \\ a_{i,j} = 0 & \text{se outro} \end{cases}$$

Isto é, uma matriz cujas colunas expressam as cláusulas do sistema com entradas 1 para as variáveis implicantes, entradas −1 para as variáveis implicadas e entradas nulas para as variáveis não expressadas na cláusula.

Exemplo:

Sistema de cláusulas de Horn.

Seja N = {a, b, c, d, e, f, g, h, i, j, k}.

A =

1) $b \rightarrow c$,

2) $b \wedge c \rightarrow h$,

3) $b \rightarrow \ell$,

4) $\phi \rightarrow c$,

5) $c \to d$,

6) $d \to e$,

7) $d \wedge e \to i$,

8) $e \wedge f \wedge g \to j$,

9) $\phi \to f$,

10) $f \to g$

11) $h \wedge i \to k$

12) $h \to i$,

13) $i \cap j \to \ell$,

14) $j \to i$,

15) $k \to d$,

16) $k \cap \ell \to i$.

Matriz associada ao sistema de Horn:

$$A = \begin{array}{c|cccccccccccccccc} & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 & a_7 & a_8 & a_9 & a_{10} & a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} \\ \hline b. & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ c. & -1 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ d. & 0 & 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ e. & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ f. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ g. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ h. & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ i. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & -1 & 0 & -1 \\ j. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ k. & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 1 \\ \ell. & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{array}$$

2.4 – ALGORITMOS COMBINATÓRIOS

Apresentaremos nesta seção um dos métodos mais usados para resolver o problema de busca em sistemas de Horn.

Estudaremos as características e limitações deste método e do princípio em que se baseia.

Posteriormente, revisaremos a área de inteligência artificial particularmente no relativo a sistemas de informação, reconhecimento de planos e linguagens naturais para deduzir algumas características desejáveis em um algoritmo provador a ser usado em um ambiente deste tipo.

Finalmente especularemos ao redor da idéia de processar informação mediante descarte sucessivo de cláusulas inúteis, que não pertencem a nenhuma demonstração.

2.4.1 – Bottom–up

Via de regra os provadores se baseiam em um único princípio, o de caminhos de aumento, isto é, em caso de Bottom–up escolher sucessivamente regras que associem elementos previamente deduzidos a elementos não deduzidos, aumentando a cada passo o volume de conhecimento.

Fundamentalmente dois algoritmos denominados Top–down e Bottom–up se desenvolvem a partir do princípio dos caminhos de aumento; estes algoritmos complementados com estruturas de dados e heurísticas adequadas constituem o núcleo dos provadores mais conhecidos.

O algoritmo Bottom–up é baseado numa estratégia de caminhos de aumento (Augmented Path). Ele parte de um estado de conhecimento vazio, revisa a lista até conseguir uma cláusula que aumente o número de verdades conhecidas, agregando essa cláusula ao estado de conhecimento. Como o sistema tem m predicados, depois de efetuar no máximo iterações conheceremos todo o demonstrável no sistema A.

ALGORITMO BOTTOM–UP

$B = \phi$

REPEAT

flag = 0

FOR i = 1 to n

IF $a_i = (I, i)$: $I \subseteq B$ e $i \notin B$ THEN

$B = B \cup \{i\}$

flag = 1

END IF

END FOR

UNTIL flag = 0

IF $\ell \in B$ THEN busca êxito ELSE não êxito.

A complexidade do algoritmo é facilmente calculável: para checar o IF é necessário no máximo m comparações, o FOR se realiza n vezes e o REPEAT não pode ser realizado, mais de m vezes dado que em cada passada uma nova proposição é agregada a B e só há m proposições. O número máximo de comparações é (m x n x m). No entanto, na prática este número é muito menor, pois as cláusulas em geral possuem um pequeno número de entradas não nulas.

2.4.2 – Algumas Limitações de Caminhos de Aumento

O algoritmo Bottom–up é eficiente, simples e fácil de implementar; no entanto deve–se assinalar algumas características não congruentes com a forma humana de processar informação, tais como:

i) o estado de conhecimento resultante da busca é influenciado pelo algoritmo;

ii) quando várias combinações estão disponíveis para a verificação, só a primeira delas é usada, bloqueando a incorporação de outras cláusulas reiterativas. Tal comportamento contraria um dos princípios da compreensão humana, conhecido como mínimas conclusões;

iii) o estado de conhecimento é parcialmente aleatório, pois está determinado pela ordem em que as cláusulas são apresentadas na lista do sistema clausular;

iv) não estabelece relações entre estados de conhecimento semelhantes;

v) a busca exaustiva não é um procedimento usado pelos seres humanos, na resolução de seus problemas;

vi) a interação com usuários implica buscas em sistemas ligeiramente alterados associados com o estado de conhecimento previamente gerado, coisa que os algoritmos combinatórios não conseguem implementar, pois por sua natureza são muito suceptíveis as perturbações;

vii) em sistemas interativos o usuário se perde com facilidade, dado que a natureza combinatória dos algoritmos provadores procede mediante saltos bruscos nos estados de conhecimento. Características humanas expressadas em termos usados na linguagem comum como: "pouco a pouco fui trocando de opinião" ou "estou quase convencido" são alheias aos processos da combinatória pura.

Podemos, em termos gerais, assegurar que algumas destas características são intrínsecas do princípio algoritmico de caminhos de aumento e não particularidade do algoritmo "Bottom–up"; a estratégia de caminhos de aumento influe determinantemente tanto na simplicidade e eficiência como nas limitações assinaladas.

Uma conclusão inevitável: se desejamos construir um provador que manipule a informação a de forma mais global e natural, que esteja bem condicionado ante pequenas perturbações, que seja mais flexível e conseqüente, sem dúvida nos basear em princípios diferentes aos de caminhos de aumento.

2.5 – INTELIGÊNCIA ARTIFICIAL

Os algoritmos provadores são usados principalmente em várias áreas da inteligência artificial, seus prós e contras só podem ser analisados no contexto dos requerimentos específicos desta área de aplicação.

É fundamental que os algoritmos provadores processem a informação de forma que suas conclusões sejam compatíveis com a forma humana de processá–la. Este objetivo primordial na opinião dos profissionais da área da inteligência artificial, não tem sido alcançado. As causas são diversas, na minha opinião, a pouca flexibilidade dos algoritmos combinatrórios é uma das razões deste fracasso.

NEWELL e SIMON em seu trabalho de "Humanos resolvendo problemas" [14] sugerem algumas idéias acerca de como poderia ser o processo de dedução no ser humano e as respectivas diferenças dos processos de dedução automática. O referido trabalho se concentra em análises do processo de dedução, em formalizações fechadas como os jogos de xadrez e aritméticas especiais e conclue que o ser humano só processa um número insignificante de possibilidades em relação com a máquina, descartando eficientemente um conjunto muito grande de ramificações pouco promissoras.

Provavelmente o problema central em inteligência artificial seja desenvolver métodos para representar e raciocinar sobre planos e ações em situações reais.

Sistemas baseados em regras são usados extensivamente na solução de sistemas específicos que interagem com o usuário por meio de linguagem "falada".

Dificuldades para resolver elipsis e fala indireta própria da linguagem informal, assim como no reconhecimento de planos, espaço de crença e até lógica proposicional tem impedido a implementação de um sistema de informação que interage naturalmente com usuários não treinados.

É em geral aceito que a origem da dificuldade em se desenvolver um sistema amigável está associada a que os computadores e os seres humanos processam

informações de forma basicamente diferente. Esta dificuldade se acentua devido a pouca compreensão que possuímos sobre a forma como os humanos resolvem problemas.

Esta lacuna tem sido preenchida mediante o treinamento de pessoal para servir de enlace entre o usuário e o computador, o tão solicitado analista.

Por outra parte, na área da inteligência artificial tem-se tratado, com relativo êxito, de "treinar" os computadores para que pensem como seres humanos.

Um resumo breve dos trabalhos associados ao problema de reconhecimento de linguagens naturais nos permitirá compreender a importância dos provadores em sistemas baseados em regras.

No entanto, a síntese de planos tem sido analizada formalmente por McCARTY, HAYES [16] no entanto o problema de reconhecimento de planos por sua natureza ambigua tem sido objeto de estudos de natureza empírica referindo-se a um domínio de aplicações fortemente restrito.

A frase reconhecimento de planos foi usada pela primeira vez por SCHNIDT, SRIDHARAM, GOODSON [16] no pioneiro sistema BELIEVER, no qual a entrada de sistema (simples descrições de seqüências de ações por um agente único) era casada com elementos de uma biblioteca de planos usando uma técnica Top-Down.

Depois da aparição de BELIEVER cresce o interesse em desenvolver sistemas que pudessem compreender histórias com diferentes personagens com objetivos conflitivos e planos que interagem. O trabalho de BRUCE [17] com histórias infantis como Hansel e Grettel não produz um modelo combinatório.

A teoria de Roger Schank dos "scripts" levava em consideração todas as regularidades presentes no mundo físico, social, psicológico SCHANK [18]. Seu grupo desenvolveu vários programas de partes desta teoria, mais o problema dos princípios da compreensão: consistência, coerência, parcimonia, mínimas conclusões só foi desenvolvido recentemente.

Por outro lado, a teoria dos atos de fala de AUSTIN [19], SEARLE [20] foi formalizada em termos de "planing" por COHEN [21] permitindo a ALLEN, PERRAULT [22] extender tal análise ao reconhecimento de planos, abordagem que tem resultado frutífera, pelo menos no número de trabalhos publicados.

O sistema de Allen tratava de encontrar a mais provável explicação através de um sistema de busca que calculava numericamente a cadeia de inferências mediante a aplicação de penalidades a configurações particulares explicitamente definidas.

Trabalhos posteriores ampliaram o sistema de Allen, permitindo o manuseio de seqüências de vocalizações; LITMAN [23] produz um algoritmo de pilha para resolver as referências indiretas a vocalizações prévias , POLLACK [24] analisa planos incorretos, CHARNIAC [25] trata o reconhecimento de planos como um problema de análise motivacional.

Consultores automáticos em sistemas operativos MACSYMA e de diagnóstico médico INTERNIST CADUCEUS assim com outras aplicações usam reconhecimento de planos como uma ferramenta central.

Sistemas de informação como os anteriormente mencionados fazem uso constante de algoritmo provadores que constroem cadeias de inferências de maneira

recursiva sobre uma lista de regras ou cláusulas de Horn. Desenvolver algoritmos provadores que ademais de eficientes processem a informação de uma forma diferente e eventualmente mais humana pode contribuir para a solução deste problema central na inteligência artificial.

É dentro desse objetivo geral que apresentamos os seguintes algoritmos combinatórios baseados em um princípio provador diferente ao de caminhos de aumento.

Desejamos, em síntese, construir algoritmos provadores destinados a servir nas diferentes áreas da inteligência artificial que processem a informação preservando o princípio da compreensão das mínimas conclusões, que descartem sucessivamente regras, à maneira que os seres humanos procedem e que possuam mecanismos flexíveis que permitam controlar o processo de interação em diálogos.

A forte associação que estes sistemas tem com a teoria de grafos sugere que esta área também pode ser um campo propício para a aplicação de modelos combinatórios.

2.6 – NOVOS ALGORITMOS COMBINATÓRIOS

Neste ítem desenvolveremos uma idéia para resolver o problema de busca em sistemas de Horn, especulando ao redor da possibilidade de detectar cláusulas inúteis para a demonstração de ℓ em A.

Cláusulas que não aparecem em nenhuma demonstração de ℓ em A poderiam ser detectadas e eliminadas do sistema em forma recursiva até esvaziar o sistema ou efetuar a dedução.

2.6.1 – Fundamentos

Nesta seção apresentaremos um novo princípio para a solução de problemas de busca ou dedução em bases de conhecimento.

O problema de dedução em bases de conhecimento sendo um problema de importância capital em diversas áreas da computação tem sido estudado de forma exaustiva sob o ponto de vista combinatório.

Via de regra as técnicas de busca exaustiva Top–Down, Botton–Up são na atualidade o elemento central dos esquemas de acesso a sistemas de informação baseados em cláusulas de Horn.

Estruturas de dados complexas como tabelas de Hashing e redes de apontadores são usadas para facilitar e acelerar o acesso à informação mas sempre sobre um esquema Top–Down, Bottom–Up ou uma combinação de ambos.

Alguns autores na área a inteligência artificial, em particular NEWEL e SIMON [14] contrapõem a estes esquemas dedutivos de natureza construtiva, a forma em que o ser humano resolve problemas, enfatizando o fato de que o homem intuitivamente descarta uma grande parte do sistema de informação para concentrar sua busca em um conjunto extremamente pequeno de regras no sistema; por exemplo, os programas que jogam xadrez revisam um grande número de jogadas até escolher a que consideram adequada, no entanto um grande mestro internacional concentra seu pensamento em um número relativamente pequeno de jogadas, conseguindo superar a máquina.

Esta forma de proceder descartando o inútil, parece constituir o selo particular da forma humana de processar informação.

Na continuação apresentaremos um algoritmo "impossível", baseado em um oráculo capaz de detectar cláusulas inúteis, isto é, descartando cláusulas de Horn que não aparecem em nenhuma demonstração.

2.6.2 – Método Baseado em Oráculo

Assumimos que possuímos um programa que detecta uma cláusula inútil (não aparece em nenhuma demonstração) cada vez que é chamado, e desta forma propomos um novo princípio de dedução automática: eliminação sucessiva de cláusulas inúteis preservando o conceito de mínimas conclusões.

Obviamente a dificuldade fundamental na implementação deste princípio algorítmico consiste na seleção da cláusula a ser eliminada, por agora passaremos por alto, mecanismos que eventualmente permitiram superar esta dificuldade.

ALGORITMO FACHO

1) Se existem cláusulas inúteis no sistema o oráculo retorna uma delas como resposta ORACULO(A, ℓ) a_i = se não existem GOTO 4.

2) $A = A - \{a_i\}$. Se elimina a cláusula inútil do sistema.

3) GOTO 1.

4) Se restam cláusulas no sistema, busca êxito, se não busca infrutífera.

Este algoritmo é de difícil (talvez impossível) implementação devido a dificuldade em reconhecer a inutilidade de uma cláusula.

A dificuldade principal nesta abordagem do problema reside na impossibilidade até agora em assegurar a inutilidade de uma cláusula, isto é, o que sucederia se o "oráculo" erradamente eliminasse uma cláusula útil. Se faz necessário reconsiderar algumas cláusulas erradamente eliminadas pelo algoritmo.

2.7 – MÉTODO BASEADO EM HEURÍSTICA

Apresentaremos uma versão de FACHO na qual o oráculo é substituído por uma heurística que detecta cláusulas pressumivelmente inúteis e em caso de errar reincorpora essas cláusulas posteriormente.

ALGORITMO DEMO

1) Revisar o sistema de cláusulas tentando descobrir uma possivelmente inútil.

2) Eliminar a cláusula suspeitosa do sistema, quando for o caso.

3) Revisar as cláusulas eliminadas intentando descobrir uma útil.

4) Reincorporar ao sistema, a cláusula pressumivelmente útil.

5) Se existem cláusulas inúteis no sistema, ou existem cláusulas úteis fora do

sistema GOTO 1.

Um processo com estas características pode não ser interessante se o número de vezes em que a heurística falha é elevado.

Obviamente a complexidade deste algoritmo está associada ao número de vezes que uma cláusula sair e entrar no sistema. Se este número pode ser cotado por um valor constante, a complexidade linear no número de iterações seria provada.

Se fosse possível construir um oráculo então o algoritmo FACHO poderia ser usado, caso contrário, se só for possível conceber um mecanismo de seleção de cláusulas pressumivelmente inúteis com um determinado grau de eficiência que nos permita assegurar a finitude do processo, poderíamos utilizar o algoritmo DEMO.

De agora em diante dirigiremos nossos esforços a explicar como construir uma heurística com estas características.

CAPÍTULO 3

MODELO EM HIPERGRAFOS

3.1 – INTRODUÇÃO

A associação entre pensamento e fluxo é antiga e natural; expressões como: "interromper o fluxo das idéias" são comuns a todas as culturas.

As definições primitivas de máquinas pensantes de natureza analógica se baseiam em controle de diferentes fluidos, tanto reais como virtuais. Por outra parte na implementação das linguagens de computação são múltiplas as referências a mecanismos de controle de fluxo particularmente nos operadores lógicos e nos mecanismos de repetição.

Tem–se desenvolvido em forma intensiva modelos combinatórios associados a uma ampla variedade de problemas: modelos econômicos, de localização, transporte, planilhas de projetos e construção, e os mais variados problemas combinatórios e de otimização em grafos (caixeiro viajante, ciclo Hamiltoniano, Matching Perfect, etc.). Seria natural esperar que a solução a problemas como os apresentados no capítulo anterior (demonstração e busca em sistemas clausulares) tenha sido estudada amplamente desde o ponto de vista de modelos combinatórios; com a exceção de alguns trabalhos pioneiros, entre os que se destacam, JERSOLOW, WANG [26] e ARAOZ, TORRES [27] o problema de resolver sistemas clausulares mediante modelos combinatórios poliedricos é um campo relativamente novo à investigação.

3.2 – PRELIMINARES

Como vimos no capítulo anterior, uma forma de expressar um sistema de Horn $A = \{a_1, ..., a_n\}$ é através de uma matriz de [m x n] com valores 1, 0, −1

$$A = \begin{cases} a_{ij} = 1 \text{ se } a_j = (I, i) \in A \\ a_{ij} = -1 \text{ se } a_j = (I, k), \ i \in I \\ a_{ij} = 0 \text{ se outro} \end{cases}$$

Esta representação nos leva a considerar alguns aspectos relacionados.

Em primeiro lugar desde o ponto de vista computacional essa matriz representa uma estrutura de dados que permite arquivar o sistema de cláusulas de uma maneira eficiente; permitindo acessar a informação em uma forma rápida, introduzir novas cláusulas, eliminar, etc. Utilizando a esparcidade da matriz, esta estrutura de dados ocupa na memória um espaço proporcional ao número de entradas não nulas.

Observamos também que estruturas de dados deste tipo são usadas tradicionalmente por algoritmos combinatórios na solução de problemas de grafos. Mais ainda, a matriz A é muito semelhante à matriz de incidência de um grafo dirigido, com a diferença de que a matriz de incidência de um grafo só aceita uma entrada 1 e uma −1 em cada coluna.

Finalmente, desde o ponto de vista de modelos combinatórios e de programação linear devemos recordar que matrizes de incidência em grafos são usadas na solução de numerosos problemas combinatórios mediante modelos de fluxo nos quais a matriz de incidência representa o conjunto de equações de fluxo em grafo.

Uma generalização do conceito de grafos nos permitirá modelar problemas em sistemas clausulares do tipo Horn. Esta generalização é conhecida como hipergrafo e o modelo associado é um modelo de fluxo em hipergrafos.

3.3 – HEURÍSTICAS

Dizemos que um algoritmo é heurístico se não soluciona corretamente todas as instâncias de um determinado problema. Desta forma um algoritmo é heurístico se não garante a solução do problema para o que foi projetado. Da mesma forma na vida real dizemos que um processo é heurístico, ou mais radicalmente aleatório, quando seu resultado depende de componentes que escapam a nossa análise.

De maneira semelhante, dizemos que um algoritmo usa heurísticas ou dá passos heurísticos, quando ainda garantindo a solução do problema, sua análise não permite uma previsão clara do comportamento futuro ou não garante a solução do problema em um número mínimo de passos. Mais ainda, quando existindo várias soluções ao problema, o algoritmo não é específico em relação a qual entre essas soluções ele vá obter, dizemos que o algoritmo escolhe heuristicamente entre as várias soluções corretas.

No contexto do presente trabalho os algoritmos apresentados garantem a solução correta de todas as instâncias do problema de busca. Passos heurísticos podem influenciar no número de passos dados pelo algoritmo, até conseguir uma solução ou determinar a preferência por uma solução entre várias. Desta forma, quando nos referimos a heurísticas falamos de uma particular abordagem a solução do problema, que não garante a solução no número mínimo de iterações para todas as instâncias ou não específica uma particular solução entre várias.

Desta forma nos referimos a Bottom–Up e Top–Down como heurísticas apropriadas a solução dos problemas de busca e demonstração em bases de conhecimento, pois não garantem a solução de todas as instâncias do problema em um número mínimo de passos.

Mas ainda existem instâncias do problemas em que várias alternativas de escolha de cláusulas são válidas no contexto Top–Down, respectivamente Bottom–Up, nesses casos, passos de natureza heurística são dados influenciando o desenvolvimento das iterações posteriores.

Os modelos combinatórios associados aos algoritmos de otimização podem ser vistos como um método inteligente de determinar um passo no processo de solução de um problema combinatório.

Por exemplo, o método de cortes quando aplicado a um determinado modelo combinatório constitue a geração heurística de uma possível solução para a qual se checa se é inteira, caso contrário se gera uma nova solução heurística e assim até conseguir uma solução.

Um exemplo de algoritmos que não dá passados heurísticos é o que soluciona o problema das torres de Hanoi, pois o algoritmo garante a solução do problema em número mínimo de iterações.

Estas considerações nos conduzem a expressar por meio de uma "fórmula" toda a generalidade dos modelos combinatórios interagindo com algoritmos de programação linear

modelos "+" PL "=" heurística

Basta introduzir nesta formula, particulares modelos e algoritmos para obter métodos para a solução de problemas combinatórios.

Esta formula representa a interação entre modelo e algoritmo na solução de um problema combinatório; desta forma não é suficiente modelar um problema para ser resolvido por um algoritmo de programação linear qualquer, pois, no mínimo, corremos o risco de reproduzir um algoritmo combinatório puro, facilmente implementável em forma direta.

A princípio construiremos um modelo do problema de busca que quando solucionado pelo algoritmo Simplex reproduz um comportamento conhecido, em seguida apresentaremos um modelo que quando solucionado pelo algoritmo BEGO reproduz um comportamento novo.

3.4 – HIPERGRAFOS

Definição: *Hipergrafo dirigido:*

Um hipergrafo é um par $G = (N, A)$, com N, um conjunto de vértices $N = \{v_1, ..., v_m\}$ e A uma coleção de pares de subconjuntos de vértices $A = \{a_1, ..., a_n\} \subseteq 2^N \times 2^N$, onde cada $a_i \in A$ é um par $a_i = (T, H)$, $T, H \subseteq N$ e $T \cap H = \phi$.

O conceito de hipergrafo é uma generalização do conceito de grafo, na qual os arcos (hiperarcos) dirigidos vão de um conjunto de vértices (T = cauda) a outro conjunto de vértices (H = cabeça) tal que a interseção é vazia

$$H = (N,A) \mapsto \forall a_i = (T_i, H_i) \in A,\ T_i \cap H_i = \phi$$

Dado um hipergrafo dirigido $H = (N, A)$, com $N = \{v_1, v_2, ..., v_m\}$ e $A = \{a_1, ..., a_n\}$ a matriz de incidência $A \in \mathbb{R}^{mxn}$ será definida da seguinte forma:

$$A = \begin{cases} a_{ij} = 1 \text{ se } v_i \in T_j \\ a_{ij} = -1 \text{ se } v_i \in H_j \\ a_{ij} = 0 \text{ se outro} \end{cases}$$

Cada linha da matriz está associada a um nó e cada coluna a um arco.

Por motivos práticos, convencionamos o conjunto vazio de nós ϕ como um nó por motivos práticos. Este nodo está associado as variáveis de domínio e não está representado na matriz de incidência.

Uma conseqüência interessante desta generalização é que qualquer matriz de 0, 1, −1 é matriz de incidência de algum hipergrafo dirigido.

Definição: Hipergrafo induzido

Dado um hipergrafo $H = (N, A)$ um sub–hipergrafo induzido por $N' \subseteq N$ e é hipergrafo $H' = (N', A')$ com $A' \subseteq A$ definido da seguinte maneira $a_j = (J, j) \in A' \mapsto \{J \cup \{j\}\} \subseteq N'$.

Definição: Hipergrafo unitário

Um hipergrafo é unitário se para todo hiperarco $a_j = (T_j, H_j)$ o conjunto H_j é

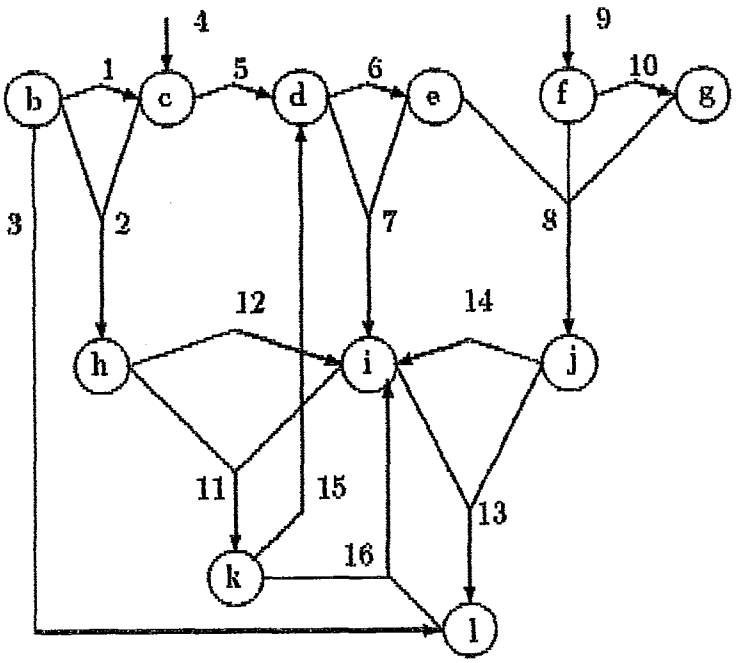
unitário.

A matriz de incidência de um hipergrafo unitário possue em cada coluna uma única entrada −1.

Em hipergrafos unitários é possível definir árvores e caminhos.

Os sistemas de Horn podem ser representados usando hipergrafos unitários: associando à cada proposição atômica um nó do hipergrafo e a cada cláusula de Horn um hiperarco. Desta forma a matriz associada ao sistema e a matriz de incidência do hipergrafo resultam idênticas.

Hipergrafo associado ao sistema do Capítulo 2:



Três representações equivalentes de um sistema clausular nos permitiram referenciar seus elementos de três formas diferentes.

Um elemento $I \in \{1, ..., n\}$ pode ser denominado de três maneiras equivalentes dependendo do contexto:

i) como uma proposição atômica do sistema clausular;

ii) como um nó do hipergrafo;

iii) como uma linha da matriz associada.

Da mesma forma um par $a = (I, i)$ pertencente ao sistema clausular pode ser denominado:

i) como uma cláusula da lista de cláusulas do sistema clausular;

ii) como um hiperarco do hipergrafo;

iii) como uma coluna da matriz associada.

O sistema como um todo também pode ser visto em suas três representações:

i) como uma lista de cláusulas;

ii) como um hipergrafo;

iii) como uma matriz.

Partes do sistema podem também ser referenciadas:

i) como um estado de conhecimento;

ii) como um hipergrafo induzido;

iii) como uma submatriz.

Estas três representações sendo estruturalmente idênticas nos permitem analisar o problema em diferentes pontos de vista.

Definições e propriedades próprias de uma particular representação podem ser herdadas pelas outras representações. Por exemplo, podemos referir ao domínio de um hipergrafo como o conjunto de hiperarcos associados a cláusulas de domínio.

As demonstrações constituem estados de conhecimento de particular interesse neste estudo, sua representação como hipergrafo nos permitirá deduzir um modelo combinatório para resolver o problema de busca.

Duas definições em hipergrafos: hiperárvore e hiperciclo, que são generalizações de conceitos equivalentes em grafos, nos permitem estabelecer a caracterização dos hipergrafos associados a demonstrações.

Estas definições estão relacionadas com o conceito de fluxo em hipergrafos.

3.4.1 – Fluxo em Hipergrafos

Definição: Fluxo

Dado um hipergrafo dirigido H = (N, A) um fluxo x sobre esse hipergrafo é uma função dos arcos nos reais positivos:

$$x : A \rightarrow \mathbb{R}^{+}$$

Como o hipergrafo H possue n hiperarcos, a função de fluxo pode ser representada por um vetor em $x \in \mathbb{R}^{m}$.

Chamaremos a componente x_i de um fluxo x "fluxo no hiperarco a_i".

Definição: Divergência

Dado um fluxo $x \in \mathbb{R}^{n}$ em um hipergrafo definimos a divergência b_i em um nó i como a diferença entre o fluxo que entra e o fluxo que sai:

$$b_i \overset{def}{=} \sum_{i \in T_j} x_j - \sum_{i \in H_j} x_j$$

assim associamos a um fluxo $x \in \mathbb{R}^{n}$ uma divergência $b \in \mathbb{R}^{m}$.

Na notação matricial com A a matriz de incidência do hipergrafo:

$$b \overset{def}{=} Ax$$

Definição: Fluxo unitário

Dada uma partição de N em três conjuntos N_u, N_s, N_i, dizemos que um fluxo $x \in \mathbb{R}^n$ é unitário de N_i a N_u se e somente se induz uma divergência unitária negativa no conjunto N_u uma divergência zero no conjunto N sendo a divergência irrestrita no conjunto restante N_i.

Observação: Uma característica que diferencia grafos e hipergrafos está associada ao valor da soma da divergência em todos os nós, que chamaremos divergência total.

Em um grafo, a divergência total é zero devido a que a matriz de incidência possue unicamente duas entradas não nulas 1, −1, respectivamente.

Em um hipergrafo a divergência total pode tomar qualquer valor, pois as entradas das colunas são múltiplos valores de 1, −1.

Em um hipergrafo unitário a divergência total é positiva, pois nas colunas da matriz de incidência existe uma única entrada −1 e múltiplas entradas 1. Esta característica não se faz presente nas equações aqui apresentadas devido a divergência no "nó vazio", sendo irrestrita, não se inclue na matriz de incidência do hipergrafo.

Esta característica dos hipergrafos unitários é fundamental em nosso caso, pois se construímos um fluxo unitário de ϕ a ℓ, dado que a divergência é nula em todos os outros hiperarcos, a divergência em ϕ tem que ser estritamente positiva e existe uma seqüência de arcos entre ϕ e ℓ que translada fluxo entre os dois nós. Esta seqüência é a demonstração que buscamos.

Definição: Vetor de propriedade

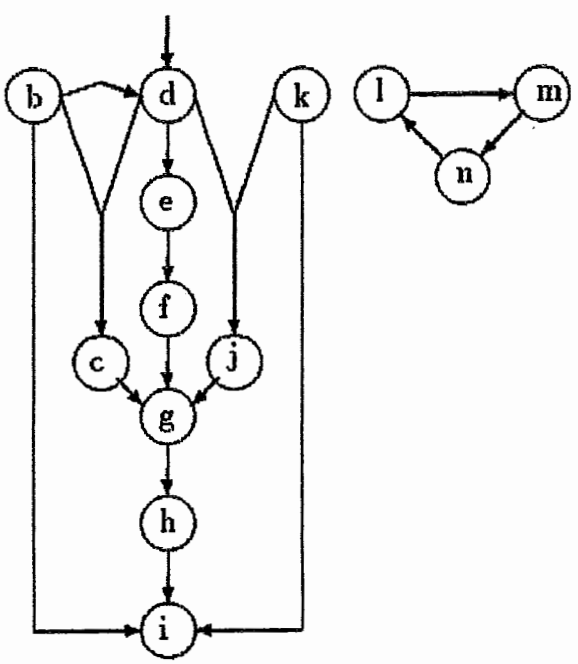
Chamamos vetor de propriedade de um conjunto N' sobre um conjunto N, com $N' \subseteq N = 1, ..., m$ a um vetor $b \in \{0, 1\}^m$ tal que:

$b_i = 1$ se $i \in N'$;

$b_i = 0$ se $i \notin N'$.

Definição: Hiperárvore com raiz $\ell \in N$ e folhas $K \subset N$, $\ell \notin K$

Um hipergrafo unitário é chamado de hiperárvore com raiz ℓ e folhas K se existe um fluxo unitário de K a ℓ estritamente positivo em todos os nodos.



Hiperárvore com raiz i e folhas ϕ, b, k.

Dizemos que o hipearco a = (I, i) é incidente em i.

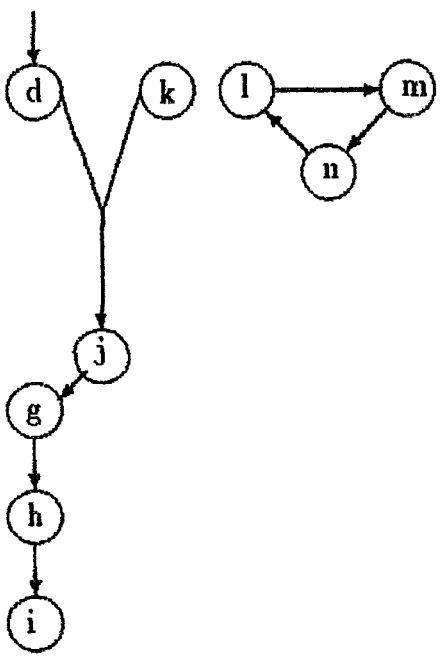
Definição: Hipercaminho entre $K \subset N$ e $\ell \in N$

Um hipercaminho P = (N, E) entre K, ℓ é uma hiperárvore com raiz ℓ e folhas K tal que suas folhas não possuem arcos incidentes e seus outros nós só possuem um arco incidente.

Um hipercaminho é portanto uma hiperárvore que cumpre com as seguintes características:

i) $\forall k \in K$ não existe um hiperarco da forma (I, k);

ii) $\forall i \in N$ existe um único hiperarco (I, i).



Hipercaminho de ϕ, k a i.

Definição: Hiperciclo

Dizemos que um hipergrafo $H = (N, A)$ é um hiperciclo se existe um fluxo $x \neq 0$ não nulo que induz uma divergência $b = 0$ zero.

Observaremos que as definições de hiperárvore e hipercaminho não descartam os hiperciclos.

Podemos observar que estas definições estão relacionadas com as definições de demonstração generalizada e demonstração.

Recordemos a definição da demonstração com $0(A)$ as cláusulas de domínio.

Definição: Demonstração restrita

Uma demonstração de ℓ em um sistema de Horn $A = \{a_1, ..., a_n\}$ é uma seqüência de cláusulas $D = (d_1, ..., d_q)$ tais que:

1) $\forall d_k = (I_k, i_k) \in D$: $I_k \subseteq 0(A) \cup_{j<k} \{i_j\}$;

2) $\forall d_k = (I_k, i_k) \in D$: $i_k \notin 0(A) \cup_{j<k} \{i_j\}$;

3) $\forall d_k = (I_k, i_k) \in D$: $i_k \in \cup_{j>k} I_j$;

4) $d_q = (I_q, \ell)$.

Isto é, toda cláusula na seqüência usa somente proposições previamente

demonstradas, toda cláusula na seqüência demonstra proposições não demonstradas com anterioridade que devem ser usadas posteriormente e a cláusula final demonstra ℓ.

Na continuação estudaremos uma dessas relações.

3.5 – EQUIVALÊNCIA DE BUSCA E VIABILIDADE PL

O problema da existência de uma demonstração de ℓ em A é equivalente a existência de um hiperárvore com raiz ℓ e folhas ϕ, que por sua vez é a solução de um problema de viabilidade.

A existência de um fluxo unitário entre ϕ e ℓ pode ser demonstrada mediante a generação de um fluxo que induza uma divergência dada. Isto é, a solução de um problema de viabilidade nas equações de divergência.

Dado um problema de busca de $\ell \in N$ em A, o poliedro associado vem dado por:

$$\{x: Ax = b,\ x \geq 0\}$$

com $-b$ o vetor de propriedade de ℓ em N.

Se existe um ponto viável no poliedro associado, podemos construir uma hiperárvore tomando os arcos com fluxo estritamente positivo.

A existência de um hiperárvore implica na existência de um hipercaminho, basta escolher partindo–se do nó ℓ e, um único arco incidente em cada nó, tomando cuidado em evitar as referenciadas ciclicas, isto pode ser conseguido usando técnicas

de busca em amplitude.

A definição de hipercaminho coincide com a de demonstração, com a exceção de algums ciclos ilhados.

A equivalência entre o problema de busca e a viabilidade no poliedro associado se enuncia da seguinte maneira:

Teorema 3.1

Seja A a matriz associada ao sistema de Horn, $-b$ e vetor de propriedade sobre N de $\{\ell\}$, existe uma demonstração de ℓ em A se e somente se:

$$\exists x : Ax = b,\ x \geq 0$$

Por construção:

i) supondo que existe uma demonstração, mostraremos que existe $x \geq 0$ tal que $Ax = b$.

Construiremos um hipercaminho a partir da demonstração, começando com um fluxo zero, $x = 0$.

Colocamos um fluxo unitário ao hiperarco associado com a última cláusula na demonstração.

Observando que para qualquer cláusula $a = (I, i)$ na demonstração os elementos $j \in I$ aparecem uma única vez na demonstração em cláusulas da forma

(J, j).

Colocamos como fluxo a cada cláusula $a_j = (I, i)$ na demonstração o número de vezes que i aparece como elemento nas outras cláusulas na demonstração, construímos um fluxo unitário de ϕ a ℓ.

Basta observar que para todo nodo $i \in N$ o fluxo que entra é igual ao fluxo que sai, igual ao número de vezes que i aparece como elemento nos conjuntos antecessores das cláusulas na demonstração.

ii) supondo que existe $x \geq 0$ tal que $Ax = b$ mostraremos que existe uma demonstração.

A construção em outro sentido é direta, pois a existência de uma divergência unitária em nó ℓ, só pode vir de uma divergência no "nó vazio" associado as cláusulas de domínio, pois a divergência é zero em todos os outros nós.

Vários modelos poliedricos podem ser desenvolvidos ao redor destas idéias: um modelo em grafos foi desenvolvido por JEROSLOW, WANG [26] mediante a introdução de restrições auxiliares. O modelo desenvolvido por ARAOZ, TORRES [27] está baseado em hipergrafos usando o esquema de primeira fase Simplex; como veremos na continuação.

No presente trabalho se utiliza um modelo mixto: fluxo em hipergrafo e insumo–produto; modelo baseado na técnica de Big M para problemas de primeira fase, ponto interior [28].

3.6 – MODELOS COMBINATÓRIOS EM SISTEMAS DE HORN

3.6.1 – Modelos Versus Combinatória

Uma base de conhecimentos é basicamente uma estrutura de dados combinatória; algoritmos de busca exaustiva tem sido sido utilizados tradicionalmente na solução de problemas de busca e alterações em bases de conhecimento. Tais técnicas especificamente são as de caminhos de aumento Top–Down e Buttom–Up, que consistem basicamente em gerar ordenadamente algumas combinações. Estas, de natureza direta apresentam as vantagens de uma grande eficiência e economia de código; por outro lado, certas limitações inerentes a natureza exclusivamente combinatória do processo, podem ser assinaladas.

a) os estados de conhecimento e as demonstrações obtidas mediante o uso destes algoritmos são dependentes da ordem em que os dados são apresentados;

b) no caso de Bottom–Up o estado de conhecimento gerado é independente do elemento a ser demonstrado e no caso de Top–Down caminhos mais largos podem desprezar caminhos curtos na busca de uma demonstração; isto é conseqüência da unidirecionalidade na escolha das cláusulas no algoritmo;

c) não é possível agrupar estados de conhecimentos semelhantes;

d) estados de conhecimento semelhantes não estão colocados de uma maneira que permita mover–se livremente de um a outro;

e) trocas na base de conhecimento implicam em recomeçar desde o início;

f) o fato de existirem várias demonstrações não facilita o processo;

g) não se tem uma medida de buscas quase exitosas;

i) o usuário se perde com facilidade nos sistemas interativos.

Em resumo, a estrutura discreta do acercamento a busca limita a capacidade do usuário na navegação e manipulação de base de conhecimento.

Como uma base de conhecimentos é uma estrutura combinatória, uma idéia natural seria aplicar modelos combinatórios e utilizar a força dos números reais, conseqüência da relaxação ao contínuo.

Obtendo–se desta forma um conjunto de vantagens:

a) a capacidade de mover–se livremente de um estado de conhecimento a outro através do contínuo;

b) o modelo representa em forma espacial, eventualemnte natural, a base de conhecimento como um todo, permitindo que o usuário não se desoriente independentemente do tamanho do sistema;

c) buscas simultâneas, atualizações, listas dos estados de conhecimento próximos, representações gráficas, pseudo–verdades, buscas probabilística, são algumas das instruções que podem ser agregadas a um sistema inteligente implementado, usando técnicas de otimização combinatória;

d) um melhor aproveitamento do computador que é mais eficiente na manipulação de operações aritméticas que na manipulação de condições lógicas, sobretudo quando este dispõe de processadores aritméticos e vertoriais.

Grande parte das vantagens do uso de um modelo combinatório se perdem quando se usa o algoritmo Simplex, devido em parte as peculiaridades da estrutura combinatória do algoritmo Simplex e a construção do modelo de primeira fase no Simplex.

3.6.2 – Modelo Primeira Fase Simplex

Problema de viabilidade:

achar x

$$\text{sujeito a:} \begin{cases} Ax = b \\ x \geq 0 \end{cases}$$

Pode ser transformado o seguinte problema de programação linear do qual se conhece um ponto viável x_0:

$$x_0 = \begin{bmatrix} 0_n \\ -b \end{bmatrix}$$

$$\text{minimizar}_{x \in \mathbb{R}^{n+m}} (0_n \mid e_m)^t \bar{x}$$

$$\text{sujeito a:} \begin{cases} \bar{A} = (A, -I_m)\, \bar{x} = b \\ \bar{x} \geq 0 \end{cases}$$

Na realidade somente é necessário $m - |0(A)|$ variáveis artificiais, pois as

cláusulas de domínio podem ser usadas na base inicial.

O problema PL pode ser resolvido usando o algoritmo Simplex. Esta construção se conhece com o nome de Primeira Fase Simplex.

Em termos do problema de busca no sistema de Horn este modelo poderia ser descrito como agregar cláusula de verdade (ϕ, i) para todas as variáveis considerando as demais cláusulas como não participantes na demonstração. O algoritmo Simplex em cada iteração buscará eliminar as cláusulas de verdade agregadas incorporando cláusulas que combinam as verdades do sistema em forma apropriada.

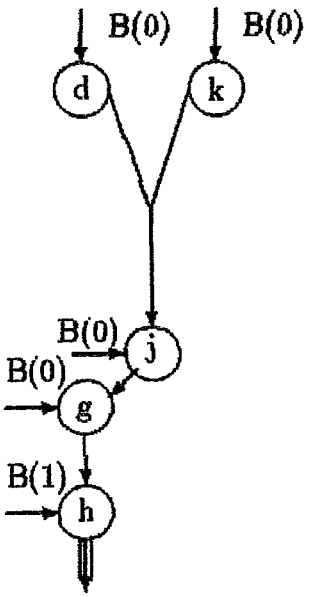
3.6.3 – Simplex e Bottom–Up

Para superar as limitações, o enfoque mediante poliedros combinatórios parecia ser a solução adequada.

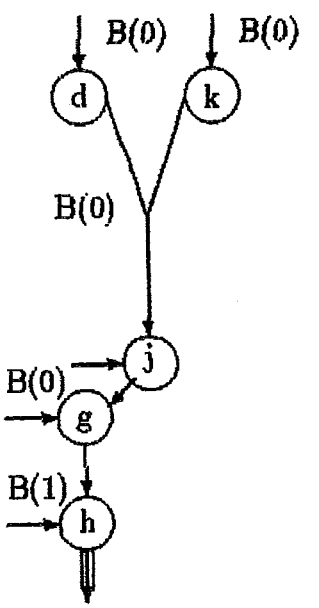
Recentemente Araoz estudou o comportamento da primeira fase Simplex quando aplicado ao modelo descrito: suas conclusões foram, simultaneamente, decepcionantes e surpreendentes: o algoritmo Simplex aplicado ao modelo é equivalente à técnica Bottom–Up.

Observemos este processo em um exemplo simples em que pretendemos demonstraria a proposição h tem como domínio neste caso, as proposições d, k.

Assinalaremos os hiperarcos correspondentes as variáveis básicas e seu fluxo, desta forma B(0) denota que o hiperarco é básico, que seu fluxo é zero; o fluxo nos hiperarcos não básico é obviamente zero.

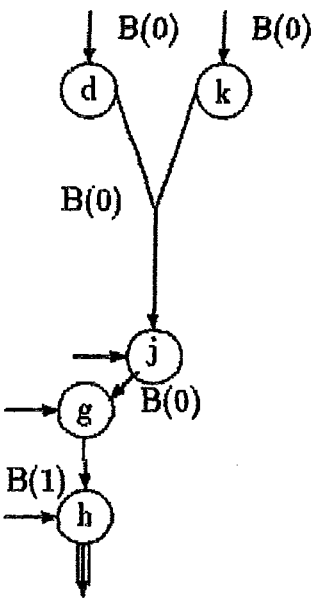


Posição inicial.



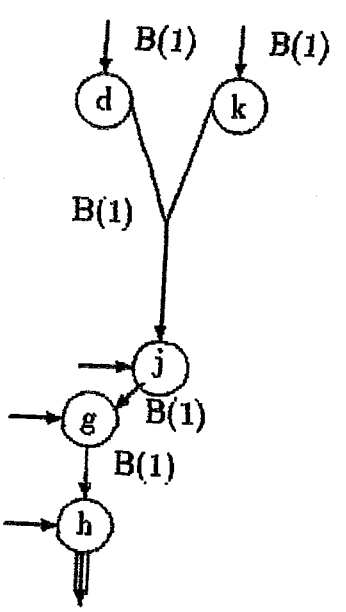
Primeira iteração.

Entra na base a cláusula $d \wedge k \rightarrow j$



Segunda iteração

Entra na base a cláusula j → g



Ponto viável

Se completa a demonstração pelo método Bottom–Up

Pode–se comprovar igualmente que o uso de Simplex–Dual reproduz o comportamento do algoritmo Top–Down.

As seguintes considerações nos levam a questionar o uso do modelo de primeira fase Simplex na solução do problema de busca em bases de conhecimento:

a) reproduz o comportamento de algoritmos combinatórios conhecidos de fácil implementação;

b) como os vértices do poliedro são inteiros, não gera soluções racionais que indicariam combinações de várias demonstrações;

c) não aceita variedade na escolha de um ponto inicial;

d) introduz muitas variáveis artificiais aumentando a dimensão do problema.

Todas estas características podem ser resumidas: Simplex aplicado ao modelo de primeira fase reproduz o comportamento do algoritmo Bottom–Up.

3.6.4 – Modelo Mixto

Este modelo está baseado na técnica de primeira fase ponto interior, sendo suscetível de ser abordado com mecanismos semelhantes aos de Big M [28].

Dado um sistema de cláusulas de Horn com m variáveis lógicas e n cláusulas, sua matriz associada A, o problema de busca (i.e. existência de uma demonstração de ℓ em A), é equivalente a existência de um fluxo unitário de ϕ a ℓ no hipergrafo

associado e viabilidade PL como foi provado.

Problema de viabilidade PL.

Dados:

$A \in \{0, 1, -1\}^{mxn}$ a matriz associada ao sistema

$$b \overset{def}{=} b_\ell = -1, b_j = 0, \quad \forall j \neq \ell, 0 < i,j < m$$

achar x

s.a. $Ax = b$

$x >= 0$

Este problema é transformado em um problema de otimalidade mediante o uso de uma variável artificial.

Determina-se um ponto inicial $x^0 \in \mathbb{R}^n$ qualquer. Apresentaremos agora a transformação do problema de viabilidade PL em um problema de otimalidade PL mediante ao uso de uma variável artificial.

Agregamos à matriz associada A, uma coluna com as larguras geradas pelo ponto inicial x^0:

$$\bar{A} = (A \quad b - Ax^0)$$

O ponto $\bar{x}^0 = (x^0, 1)$ é viável no poliedro $\bar{A}\bar{x} = b$. Isto nos leva a plantar um problema de programação linear com um custo unitário na variável artificial

$c = (0^n, \quad 1)$:

Problema de programação linear:

$$\begin{aligned} &\min c^t x \\ &\text{s.a.} \quad \bar{A}\bar{x} = b \\ &\qquad x >= 0 \end{aligned}$$

Este problema pode ser resolvido usando algoritmos de programação linear por direções viáveis.

Fica assim configurado o modelo de primeira fase que pode ser resolvido usando qualquer algoritmo de programação linear susceptível de arrancar desde o ponto interior.

Este modelo poliédrico, conjuntamente com o algoritmo BEGO, constituem as ferramentas fundamentais desenvolvidas no presente trabalho para a solução dos problemas de dedução em bases de conhecimentos. É chamado de modelo mixto, pois a diferença entre este e o modelo de primeira fase Simplex é que neste caso a matriz $\bar{A}$ não constitue a matriz de incidência de nenhum hipergrafo. O modelo possue duas partes: a parte hipergrafo, constituída pelas primeiras n colunas e a parte de insumo–produto, constituída pela coluna $n + 1$; no processo de solução do problema de busca, as partes interagem, decrescendo a componente insumo–produto e variando o fluxo nos hiperarcos associados às cláusulas do sistema de Horn.

A escolha do ponto inicial x^0 é fundamental devido a que a configuração do modelo depende dele.

O número de iterações que o algoritmo realiza até conseguir a solução e o estado de conhecimento atingido dependem de uma escolha adequada do ponto inicial; várias alternativas foram estudadas: fluxo zero em todos os arcos, fluxo unitário nos arcos de domínio, fluxo randômico. Estas alternativas forem descartadas porque ou bem regressavam o problema a suas características puramente combinatórias ou eram inconseqüentes.

Como queremos simular o processo humano de descarte sucessivo incialmente todas as cláusulas devem participar no estado de conhecimento e conseqüentemente seu fluxo inicial deve ser estritamente positivo, a alternativa de tomar o vetor $x^0 = e^n = (1, ..., 1) \in \mathbb{R}^n$ parece a mais adequada.

A variação e eventual eliminação do fluxo associado às cláusulas no sistema, manifesta o comportamento heurístico definido pela interação entre o algoritmo BEGO e o modelo mixto.

3.6.5 – Comportamento Combinatório do Algoritmo BEGO Aplicado ao Modelo Mixto

Da mesma forma que o Simplex reproduz o comportamento do algoritmo Top–Down, os algoritmos de direções viáveis aplicados ao modelo mixto também reproduzem um certo comportamento de natureza combinatória, só que este caso é desconhecido na literatura.

O modelo de primeira fase Simplex começa com um fluxo associado aos arcos do sistema igual a zero, o que indica um estado de conhecimento vazio, por contraste em nosso caso de primeira fase ponto interior, todos os arcos possuem um fluxo estritamente positivo e o estado de conhecimento é todo o sistema clausular A sobre

N.

Como os algoritmos de direções viáveis apresentados começam com um conjunto ativo vazio $\beta = \phi$ e vão agregando paulativamente restrições ao conjunto β, o significado deste processo em termos de conjunto de cláusulas consiste na eliminação do estado de conhecimento das cláusulas associadas a restrições agregadas à β. Isto é, o comportamento do algoritmo BEGO quando aplicado ao modelo mixto vai progressivamente eliminando cláusulas, até conseguir resolver o problema de viabilidade. Além disso, quando todas as cláusulas no sistema pertencem a alguma demonstração, o algoritmo pode resolver o problema de busca em uma única iteração.

Este comportamento é completamente diferente dos algoritmos combinatórios baseados em caminhos de aumento e por certo diferente de qualquer algoritmo combinatório puro, aplicável na solução do problema de busca em sistemas clausulares.

Este comportamento é congruente com a definição de uma heurística aplicável na resolução do esquema algoritmico DEMO.

Dada a demonstração de complexidade do algoritmo BEGO podemos concluir que a heurística é suficientemente boa, como para assegurar a finitude do processo de eliminação e reincorporação de cláusulas no estado de conhecimento. Na prática se obtém resultados muito melhores, como veremos. A construção de um exemplo em que a heurística elimine erradamente uma cláusula útil possue certa dificuldade.

3.7 – CONCLUSÕES

O objetivo central deste trabalho é usar algoritmos de conjuntos ativos por

direções viáveis na solução de problemas de dedução em sistemas de Horn.

Na continuação detalharemos o alcance e as ramificações deste objetivo:

1) justificar a aplicação de algoritmos de direções viáveis na solução de problemas combinatórios mediante um estudo sistemático da estrutura combinatória dos algoritmos de PL;

2) desenvolver e implementar algoritmos para a solução de problemas de dedução em bases de conhecimento;

3) aprimorar o modelo de fluxo em hipergrafos mediante a criação de um modelo de primeira fase, mais adequado ao problema que a primeira fase Simplex. É importante assinalar que as características deste modelo são dependentes da escolha de um ponto inicial;

4) obter globalidade na solução. Quando existem várias demonstrações, a solução é uma combinação delas;

5) deduzir o comportamento combinatório dos algoritmos apresentados quando aplicados ao modelo, tal como foi realizado por Araoz para o Simplex;

6) apresentar um princípio algoritmico mais flexível e essencialmente diferente ao de caminhos de aumento;

7) assinalar opções de construções de sistemas provadores mais amigáveis, mediante o uso de poliedros combinatórios;

8) provar a convergência dos algoritmos apresentados.

O sentido essencialmente diferente do processo de dedução proposto, assim como a natureza diferente da solução, invalidam um estudo estatístico comparativo com algoritmos combinatórios puros. É de observar que a complexidade por iteração é a mesma que no Bottom–Up e que o número de iterações na prática pode ser menor ou maior dependendo do problema. É fácil construir exemplos nos quais o presente algoritmo resolve em uma iteração e que são resolvidos pelos algoritmos combinatórios em um número linear de iterações; sendo também fácil construir exemplos nos que Bottom–Up se comporta melhor.

No capítulo 4 estudaremos o sistema completo resolvendo problemas de dedução em sistemas de informação, comprovando que efetivamente funcionam dentro dos esquemas de Facho e Demo.

CAPÍTULO 4

SOLUÇÃO A PROBLEMAS

Será estudado o comportamento dos algoritmos de direções viáveis apresentados na solução de um conjunto de problemas em sistemas de informação:

1) problema de busca: é um problema SI–NO definido como a existência de uma demonstração de $\ell \in N$ em A;

2) demonstração generalizada: encontrar todas as cláusulas em A que pertence a alguma demonstração irrestrita de ℓ em A (com ciclos);

3) demonstração: encontrar uma demonstração de ℓ em A;

4) demonstração total: encontrar todas as cláusulas que pertencem a alguma demonstração restrita de ℓ em A (sem ciclos);

5) demonstração mínima: encontrar uma demonstração com o menor número de cláusulas;

6) problemas perturbados: dada a solução de um problema em A encontrar a solução de um problema em A', um sistema quase igual a A.

Estudaremos a solução destes problemas usando algoritmos de direções viáveis, as características dos processos serão estudadas usando as três visões apresentadas dos sistemas de informação:

1) como uma lista de cláusulas;

2) como um hipergrafo unitário;

3) como um problema PL na matriz associada ou matriz de incidência.

4.1 – O PROBLEMA DE BUSCA

A verificação de um predicado produz dois tipos de resposta "certo" ou "certo ou falso" usando o método de eliminação sucessiva de cláusulas.

4.1.1 – Solução do Problema de Busca

Dado um sistema clausular de Horn A sobre o conjunto N, seu hipergrafo associado A = (N, E) e a matriz de incidência A de m filas e n colunas, o problema de busca ou dedução de ℓ em A pode ser solucionado da seguinte maneira, usando as ferramentas desenvolvidas nos capítulos precedentes:

1) construir o modelo mixto:

$$x^0 = e^n = (1, ..., 1) \in \mathbb{R}^n$$

Ao partimos de um estado de conhecimento igual a todo o sistema, assumimos que todas as cláusulas participam de maneira unitária na demonstração.

Agreguemos a matriz de incidência A, a componente insumo–produto constituída por uma coluna com as folgas geradas pelo ponto inicial x^o

$$\bar{A} = (A\ b - Ax^0)$$

O ponto $\bar{x}^0 = \begin{bmatrix} x^0 \\ 1 \end{bmatrix}$ é viável no poliedro $\bar{A}x = b$, $x \geq 0$. Isto nos leva a colocar um problema de programação linear com um custo unitário na componente insumo–produto ou divergência artificial e zero nas colunas correspondentes a componente hipergrafo: $c = \begin{bmatrix} 0^n \\ 1 \end{bmatrix}$.

Problema de Programação Linear.

$$\begin{aligned} &\min c^t x \\ &\text{s.a.} \quad \bar{A}x = b \\ &\qquad\quad x >= 0 \end{aligned}$$

2) usar o algoritmo BEGO para resolver o problema PL com um ponto inicial interior conhecido $\bar{x}^0 = e^{n+1} = (1, ..., 1)$;

3) o algoritmo se detem depois de um número finito de iterações em um ponto ótimo $\hat{x}$ para o problema PL: se $\hat{x}_{n+1} = 0$ existe um fluxo unitário no hipergrafo associado dado por $\hat{x}$, portanto como foi provado existe uma demonstração de ℓ em A e o sistema emite a mensagem "certo". Caso contrário não existe demonstração e a mensagem é "certo ou falso".

No caso da mensagem ser "certo" as cláusulas correspondentes aos arcos cujo fluxo é estritamente maior que zero constituem um estado de conhecimento relevante, que pode ser usado para construir uma resposta útil por um determinado sistema de auxílio, isto é associada as cláusulas no estado de conhecimento existe informação relevante para o sistema. Se a mensagem é "certo ou falso" o estado de conhecimento

é em geral irrelevante e o sistema produzirá uma resposta genérica de não casamento.

4.1.2 – Interpretação Combinatória

Como assinalamos no primeiro capítulo, os algoritmos de programação linear possuem uma estrutura combinatória representada pelas faces que visitam no processo de obter um ótimo.

Os conjuntos ativos β gerados pelo algoritmo, definem faces do poliedro devido corresponderem a conjuntos não vazios.

Podemos observar que um ponto x pertence ao interior relativo de uma face se o conjunto de índices com componentes nulas é mínimo para todos os pontos na face.

Os pontos gerados pelo algoritmo BEGO são tais que em geral (salvo exceções causais) pertencem ao interior relativo da face ativa.

Desta forma, quando o algoritmo BEGO é aplicado a um modelo combinatório (isto é certo também para outros algoritmos de programação linear) podemos associar o conjunto de índices que se anulam em uma determinada iteração como uma particular seleção de elementos do problema combinatório. Além disso, como os elementos que se anulam não variam bruscamente, esta seleção obedece a um processo de seleção progressiva de elementos do problema combinatório.

Esta análise foi aplicada ao algoritmo SIMPLEX associado ao modelo de fluxo em hipergrafos.

Desta forma podemos associar o comportamento do SIMPLEX com o

comportamento do algoritmo combinatório BOTTOM–UP.

Quando o algoritmo SIMPLEX associado ao modelo de primeira fase inicia o fluxo associado a todas as cláusulas no sistema é zero e dado que ninguém pertence ao conjunto de variáveis básicas, consideramos que nos encontramos no estado de conhecimento correspondente ao conjunto vazio, o mesmo estado de conhecimento que o algoritmo BOTTOM–UP possue quando inicializa. Se alternativamente incluíssemos, no conjunto básico, as cláusulas de domínio para diminuir o número de variáveis artificiais o estado de conhecimento seria o correspondente ao conjunto de cláusulas de domínio do sistema estado que também poderia ser atingido pelo algoritmo Bottom–Up, caso as cláusulas de domínio fossem colocadas no começo da lista.

Ao contrário disto, o modelo mixto arranca associando fluxos estritamente positivos a todos os arcos e portanto o estado de conhecimento inicial corresponde a $E = \{a_1, ..., a_n\}$ todo sistema clausular.

Na primeira iteração o algoritmo gera uma direção com a componente insumo–produto negativa d_{n+1} componente a associada a divergência artificial, como resultado do teste de razão, uma restrição $i \in \{1, ..., n+1\}$ entra no conjunto ativo. Se essa restrição selecionada é $n + 1$ obtemos um ponto ótimo e a resposta ao problema de busca é certo com um estado de conhecimento final igual a E. Se a restrição i obtida pelo teste de razão correspondente a um hiperarco a_i, o fluxo associado $x_i = 0$ toma o valor zero eliminando–se a_i do estado de conhecimento, obtendo–se $E - \{a_i\}$.

No caso em que várias restrições se anulem simultaneamente, se eliminam todas elas do estado de conhecimento.

Certamente uma variável anulada pode regressar ao estado de conhecimento em iterações posteriores, afortunadamente isto não ocorre com freqüência na prática.

Em cada iteração uma restrição é obtida como resultado do teste de razão e subseqüentemente eliminada do estado de conhecimento. Este comportamento configura um dos resultados fundamentais do presente trabalho: desenvolver um algoritmo que resolva o problema de busca mediante a eliminação sucessiva de cláusulas.

4.1.3 – Resultados Associados à Inteligência Artificial

1) Eliminação de cláusulas inúteis

A eliminação progressiva de cláusulas é uma característica da forma humana de resolver problemas, compartida pelo procedimento aqui proposto.

A avaliação deste resultado no contexto da inteligência artificial consiste em possibilitar o processamento da informação de uma maneira que reproduza uma característica do pensamento humano.

Mesmo acentando que o ser humano utiliza um processo de eliminação sucessiva na resolução de problemas, não seria válido afirmar que o mecanismo aqui apresentado constitue uma aproximação da maneira humana de processar informações, pois compartilhar uma característica não implica uma similaridade global.

Por outro lado, o fato de permitir processar informações mediante a eliminação progressiva de cláusulas representa uma possibilidade de progresso no objetivo

fundamental da inteligência artificial: simular o pensamento humano.

2) Mínimas conclusões.

Outro resultado fundamental está associado ao princípio da compreensão das conclusões mínimas. Seguindo-se este princípio, na compreensão de histórias e diálogos o ser humano quando confrontado com uma nova informação altera seu estado de conhecimento o menos possível preservando toda a informação que não é diretamente anulada pelos novos conhecimentos.

Ilustraremos este conceito mediante um exemplo:

A diz a B que vai viajar a semana que vem.

Na mente de B se elabora um reconhecimento do plano de A de viajar na semana que vem, este plano é a união de muitos diferentes planos de viagem onde o único denominador comum é A viajando a alguma parte, utilizando algum meio de transporte, na companhia de pessoas.

B pergunta a A: A para onde vai viajar ?

– Vou a São Paulo.

Ao receber esta nova informação B reconhece um plano mais específico porém introduzindo as mínimas conclusões no plano precedente. Isto é, descarta a possibilidade do meio de transporte a ser usado, seja de barco, mas preserva as possibilidades: trem, ônibus, carro e avião.

O processo de construir uma demonstração mediante um algoritmo baseado em caminhos de aumento dificilmente reproduz o comportamento de mínimas conclusões, pois a unicidade só nos permite construir uma possibilidade como por exemplo a viagem a São Paulo de trem em companhia da esposa e não uma combinação de planos maximal nas possibilidades permitidas pelo volume de informação recebido.

Pelo contrário, o sistema apresentado, reproduz de maneira natural este princípio, pois a projeção como já sabemos gera a direção de máxima ou declive e isto em termos de fluxo consiste em distribuir os fluxos em forma proporcional entre os diferentes caminhos, preservando em geral os diferentes planos no estado de conhecimento.

Este resultado assim como o anterior e em geral todos os resultados relacionados com a inteligência artificial não podem ser questionados em termos absolutos. Por uma parte o algoritmo pode eventualmente eliminar demonstrações possíveis não respeitando o princípio de mínimas conclusões, mais por outra parte este princípio não é aplicado em forma inflexível pelos seres humanos: observemos que no exemplo anterior B não considerou a possibilidade de A viajar à São Paulo em bicicleta.

4.1.4 – Exemplo

Resolveremos o seguinte sistema de Horn:

A =

1) $\phi \rightarrow a$

2) $\phi \rightarrow b$

3) $\phi \rightarrow c$

4) $b \rightarrow e$

5) $b \wedge c \rightarrow f$

6) $d \rightarrow e$

7) $e \rightarrow g$

8) $f \rightarrow h$

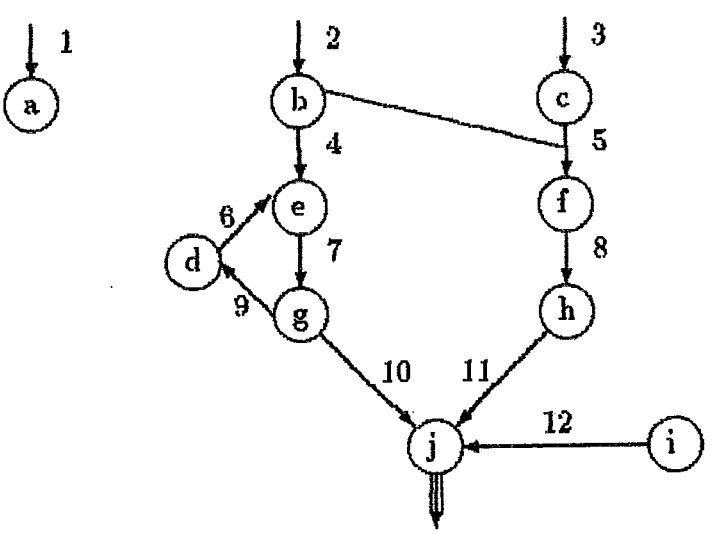
9) $g \rightarrow d$

10) $g \rightarrow j$

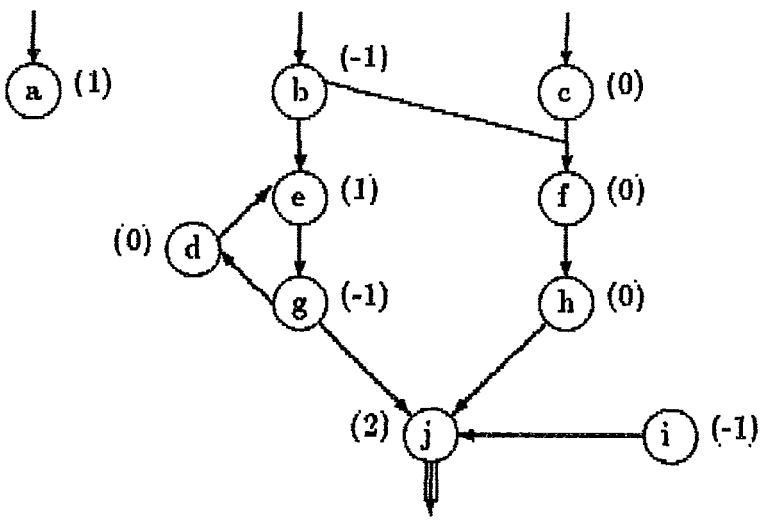
11) $h \rightarrow j$

12) $i \rightarrow j$

Associado ao hipergrafo



Divergência artificial inicial com $x^0 = (1, ..., 1)$



Gerando uma direção projetiva viável em $x^0 = (1,1,1,1,1,1,1,1,1,1,1,1,1)$ com o custo $c = (0,0,0,0,0,0,0,0,0,0,0,0,1)$ obtemos:

$h = -cp = (-0.22, 0, -0.06, -0.17, -0.06, -0.02, 0.04, -0.07, -0.02, -0.16,$

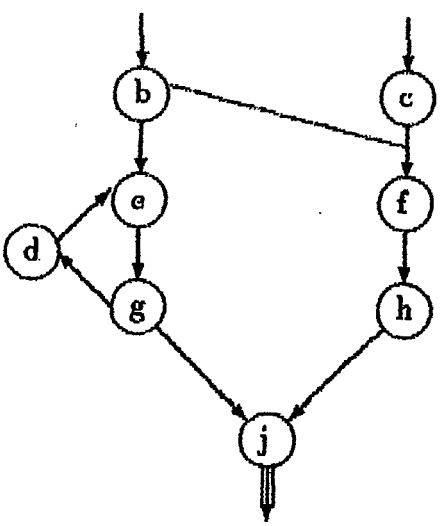
$0.07, -0.22, -0.2)$

Depois de uma iteração com $\lambda = 4.4$ obtemos o valor x^1

$x^1 = (0, 1, 0.7, 0.3, 0.7, 0.9, 1.2, 0.7, 0.9, 0.3, 0.7, 0, 0)$

Obtemos três restrições com fluxo anulado $x_1 = 0$, $x_{12} = 0$, $x_{13} = 0$.

Como a componente x_{13} associada a componente de insumo–produto se anula, a busca foi vitoriosa e o estado de conhecimento depois da busca é:



4.2 – DEMONSTRAÇÃO GENERALIZADA

Buscamos uma combinação de demonstrações para representar um estado de conhecimento associado a consideração de diversos planos ou possibilidades de encarar uma situação real.

4.2.1 – Solução ao Problema de Demonstração Generalizada

Achar um conjunto de cláusulas que seja a união das cláusulas em várias demonstrações irrestritas de ℓ em A.

A solução deste problema é um subproduto da solução ao problema de busca:

1) solucionar o problema de busca usando o modelo mixto;

2) toma–se como solução o conjunto de todos os hiperarcos com fluxo estritamente positivo.

Obtendo–se uma combinação dos vértices do poliedro viável com o que obtemos uma combinação linear de "demonstrações", pois os vértices do poliedro correspondem a demonstrações de ℓ em A.

4.2.2 – Problema da Máxima Demonstração Generalizada

Este é um problema equivalente a conseguir um ponto interior no conjunto viável. Existem vários mecanismos para conseguir um ponto interior quando se conhece um ponto na fronteira, aqui nós utilizaremos uma técnica relacionada com os algoritmos de direções viáveis, para manter uma consistência de estilo.

1) depois de solucionar o problema de busca e obter uma primeira demonstração generalizada; procederemos a incorporar cláusulas na demonstração generalizada até que não seja possível incorporar mais cláusulas;

2) utilizar o algoritmo Antipurif como explicado no capítulo um.

Este processo é oposto ao de purificação e pára, quando se obtém um ponto interior.

4.2.3 – Resultados Associados à Inteligência Artificial

Como vimos na seção anterior, o método apresentado para solução do problema de busca, tende, devido a característica do máximo declive, a preservar todas as demonstrações no estado de conhecimento, no entanto, a solução do problema de busca não nos garante que estamos considerando absolutamente todas as possibilidades.

O procedimento previamente descrito nos permite em primeiro lugar checar se a solução obtida pelo método do problema de busca corresponde com as mínimas conclusões e em segundo lugar tal procedimento nos permite incorporar progressivamente cláusulas.

4.2.4 – Exemplo

Observamos que no exemplo anterior o algoritmo gera a demonstração generalizada máxima.

4.3 – O PROBLEMA DE DEMONSTRAÇÃO E DEMONSTRAÇÃO MÍNIMA

Depois da solução do problema de busca resultar em uma resposta "certo", desejamos em ocasiões obter uma demonstração da proposição ℓ.

4.3.1 – Solução do Problema de Demonstração

1) depois de uma resposta "certo" do problema de busca, se elimina a coluna $n+1$ da matriz $\bar{A}$ regressando–se a matriz A, para a qual, se conhece um ponto viável;

2) dado que uma demonstração restrita corresponde a uma hiperárvore, o fluxo associado aos hiperarcos fica determinado pela correspondência a um vértice do poliedro, associado a matriz de incidência, então aplicaremos o algoritmo da PURIF para conseguir um vértice do poliedro.

Problema de Purificação:

Dado x^0 viável r $c \in \mathbb{R}^n$ achar x um vértice tal que:

$c^t x \leq c^t x^0$

s.a. $Ax = b$

$x >= 0$

O conjunto viável pode ser ilimitado, devemos colocar o custo estritamente positivo, por exemplo $c + c^n = (1, ..., 1)$.

4.3.2 – O Problema da Demonstração Mínima

Existem várias medidas para calcular a complexidade de uma demonstração algumas delas são mais facéis de obter em processos combinatórios tais como a altura da árvore de demonstração. Estas medidas não são implementadas por funções objetivos lineares. Outras medidas como o número de cláusulas na demonstração, o número de nós usados, são caracterizadas por funções lineares, considerando–se as várias aparições do mesmo arco como aumentos da complexidade da demonstração.

4.3.3 – Solução do Problema da Demonstração Mínima

i) Minimizar o número de cláusulas:

1) construir o vetor de custo $c = e^n = (1, ..., 1)$;

2) aplicar o algoritmo BEGO, tomando–se como ponto inicial o ponto resultante do problema de busca.

ii) Minimizar o número de elementos.

1) construir o custo tomando $c = (c_i =$ Número de nós em $a_i)$;

2) aplicar o algoritmo BEGO.

iii) Mínimos particulares

1) construir o custo associado a um conjunto de valores relacionados a um certo resultado empírico $c = (c_1, c_i, ..., c_n)$;

2) aplicar o algoritmo BEGO.

4.3.4 – Resultados

O resultado mais importante em relação à solução do problema de demonstração é a possibilidade de mover–se de uma demonstração à outra usando diferentes valores para o vetor custo.

Os resultados obtidos esclarecem o conceito de estados de conhecimento relacionados, pois observamos que o modelo combinatório junto com os algoritmos de direções viáveis nos permitem mover entre as faces correspondente aos estados de conhecimento relacionados. Isto é muito difícil de implementar por métodos combinatórios.

É importante assinalar que possuem uma flexibilidade adicional associada a liberdade de direcionar nossa busca a estados de conhecimento, particulares mediante a fixação do vetor custo.

Esta flexibilidade está associada a mais plausível explicação entre várias, em geral a explicação menos complexa é a que se considera mais apropriada, mais outros conjuntos de valores associados a arcos e nós geram uma medida da demonstração mais apropriada.

Quando esta busca não necessariamente deve terminar na obtenção de um ótimo, mas bem expressar uma tendência mais ou menos desejável é possível usar o algoritmo de purificação. Pelo contrário, quando quizermos encontra a demonstração que maximize um determinado custo, devemos usar o algoritmo de programação linear e depois se necessário o de purificação.

4.3.5 – Exemplo

Consideremos agora um processo de purificação no qual procuramos uma demonstração que contenha a cláusula $a_3 = (\{b, c\}, f)$.

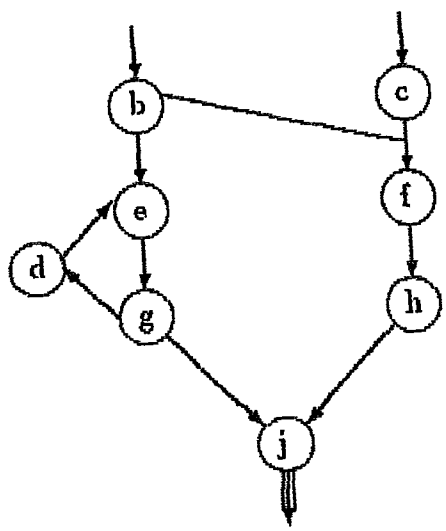
Para isso construímos um vetor custo apropriado:

$c = (0.1, 0.1, 0.1, 0.1, -1, 0.1, 0.1, 0.1, 0.1)$

Um vetor apropriado deve ser positivo com um valor "pequeno" nas variáveis não selecionadas e um valor negativo "grande" na variável selecionada. Estes valores dependem da dimensão do problema neste caso como o problema é de dimensão 10 tomamos $x_5 = -1$ e $x_i = 0.1$, $\forall i \neq 5$, em geral uma proporção n é suficiente para a 1ª também é apropriado.

Conhecemos um ponto viável para o problema solucionado na seção do problema de busca

$x^0 = (1, 0.7, 0.3, 0.7, 0.9, 1.2, 0.7, 0.9, 0.3, 0.7)$



Na primeira iteração com uma direção projetiva:

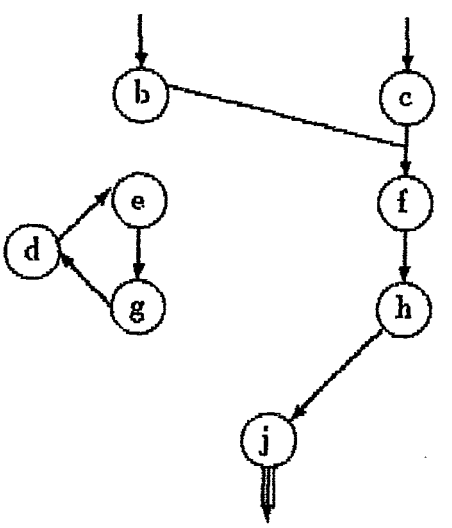
$$d^0 = (0.0,\ 0.13,\ -0.13,\ 0.13,\ -0.05,\ -0.19,\ 0.13,\ -0.05,\ -0.13,\ 0.13)$$

obtemos o seguinte ponto viável:

$$x^1 = (1,\ 1,\ 0,\ 1,\ 0.77,\ 0.77,\ 1,\ 0.77,\ 0{,}1)$$

com o passo $\lambda = 2.22$.

O estado de conhecimento intermediário:



Na segunda iteração incorporamos a restrição número 3 ao conjunto ativo, também se poderia incorporar a restrição número 9 pois nos encontramos em uma face degenerada e calculamos a direção projetiva na face caracterizada pelo conjunto ativo $\beta = \{3\}$

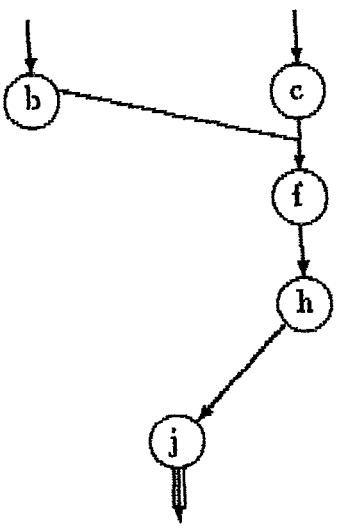
$$d^1 = (0,\ 0,\ 0,\ 0,\ -0.1,\ -0.1,\ 0,\ -0.1,\ 0,\ 0)$$

obtendo a demonstração:

$$x^2 = (1,\ 1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1)$$

com o passo $\lambda = 7.77$.

O estado do conhecimento resultante:



Se em vez de selecionarmos a cláusula selecionássemos uma cláusula pertencente a outra demonstração, obteríamos o seguinte resultado:

$$c = (0.1,\ 0.1,\ -1,\ 0.1,\ 0.1,\ 0.1,\ 0.1,\ 0.1,\ 0.1,\ 0.1)$$

$$d^0 = (0,\ -0.19,\ -0.19,\ -0.19,\ -0.16,\ 0.03,\ -0.19,\ -0.16,\ 0.19,\ -0.19),$$

$$x^1 = (1,\ 0,\ 1,\ 0,\ 0.307,\ 1.307,\ 0,\ 0.307,\ 1{,}0)$$

com $\lambda = 3.59$.

Na segunda iteração com β qualquer das variáveis anuladas obtemos:

$$d^1 = (0,\ 0,\ 0,\ 0,\ -0.1,\ -0.1,\ 0,\ -0.1,\ 0,\ 0),$$

$$x^2 = (1,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0,\ 0,\ 1,\ 0)$$

com $\lambda = 3.076$.

Se escolhessemos como variáveis uma das pertencentes ao ciclo d, e, g obteríamos um problema ilimitado.

4.4 – O PROBLEMA DE DEMONSTRAÇÃO TOTAL

Este é um problema basicamente de eliminação de ciclos, pois como vimos da solução do problema de busca, obtemos em geral uma demonstração generalizada ou podemos obté–la.

4.4.1 – Solução ao Problema de Demonstração Total

Para eliminar os ciclos usamos uma técnica particular, que consiste em agregar um arco de ℓ ao conjunto vazio. Em termos da matriz de incidência consiste em agregar uma coluna a_{n+1} igual ao vetor de propriedade de ℓ em A

$$a_{n+1} \in \mathbb{R}^m = (a_{(n+1,i)} = 0,\ \forall i \neq \ell,\ a_{(n+1,\ell)} = 1)$$

1) toma–se $\bar{A}$ igual a

$$\bar{A} = (A\ a_{n+1})$$

2) sendo $\hat{x} \in \mathbb{R}^n$ o ponto viável obtido da solução do problema de busca $A\hat{x} = b,\ \hat{x} \geq 0$ construir $x^0 \in \mathbb{R}^{n+1}$ tal que para $\bar{A}x^0 = b,\ x^0 \geq 0$ da seguinte maneira:

$$x^0 = \begin{bmatrix} \hat{x} \\ 0 \end{bmatrix}$$

3) construir um vetor $c \in \mathbb{R}^{n+1}$ com as seguintes características: seja M um valor real "grande"

$$c_i = 1\ \forall i \neq n+1$$

$$c_{n+1} = -M$$

4) calcula–se $cp = Pc$ a projeção do custo sobre o nulo de $\bar{A}$, os índices cujos valores correspondentes são negativos estão associados as cláusulas que

pertencem a ciclos.

Observaremos que este método só pode ser aplicado depois de se obter uma demonstração generalizada. Sua validez está garantida pelo fato de ser aplicado a um hiperbol com raiz ℓ. Quando aplicado a um hipergrafo qualquer, esta técnica não produz a eliminação de ciclos.

Este procedimento funciona em caso de uma demonstração generalizada devido a que unicamente dois tipos de hiperarcos sobrevivem ao processo de eliminação sucessiva: os que contribuem a levar o fluxo desde as cláusulas de domínio a ℓ e arcos que formam parte de ciclos, alguns destes não participam no processo de transladar o fluxo das cláusulas de domínio a ℓ e podem portanto ser detectados mediante a *geração de um vácuo ou absorção no nó* ℓ.

Uma possibilidade interessante que não havia sido estudada é a de realizar o processo de busca e a eliminação de ciclos em forma simultânea aumentando em duas colunas a matriz de incidência, uma coluna de insumo–produto a ser diminuida progressivamente, dado que o custo associado a essa componente é positivo e uma componente de absorção com custo negativo que aumenta progressivamente. Isto nos permite controlar o processo de solução mediante a manipulação de dois parâmetros.

4.4.2 – Resultados

Do ponto de vista da inteligência artificial a presença de ciclos não é contraproduzente, pois eles fazem parte da visão humana da compreensão. Mas do ponto de vista estritamente lógico, os ciclos constituem uma presença indesejada associada à tautologia.

Do ponto de vista humano, um conjunto de atividades não diretamente relacionadas com um determinado plano não são descartadas quando se dá um reconhecimento. No exemplo da viagem a São Paulo a pessoa não descarta a possibilidade de parar para visitar um povoado, um museu, almoçar em um restaurante e outras atividades que não estão diretamente relacionadas com o objetivo principal. Tais atividades permanecem no estado de conhecimento do ouvinte, quando reconhece o plano do falante, isto é, viajar a São Paulo. Porém, na vida real, estas atividades não estão diretamente relacionadas com o plano que interrompem momentaneamente a continuidade do plano ou diálogo, elas formam parte integrante dos mecanismos de compreensão.

Um dos trabalhos de inteligência artificial estudado se refere à comida oriental e uma das dificuldades associadas à compreensão do diálogo entre o professor e o aluno (de cozinha) era a constante saida do tema para estabelecer comentários de natureza cultural e pessoal para logo retomar a preparação da receita no ponto em que foi abandonada. Isto quer dizer que ciclos inconseqüentemente são introduzidos em forma natural na comunicação.

O objetivo de permitir ou eliminar à vontade estas disperções poderia ser eventualmente resolvido por meio desta técnica de vazio ou absorção.

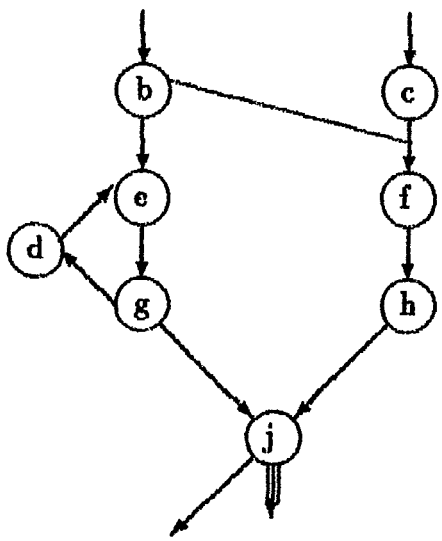
4.4.3 – Exemplo

Agregando o arco $a_{n+1} = (\ell, \phi)$ que por certo não corresponde com uma cláusula de Horn obtemos o seguinte hipergrafo:

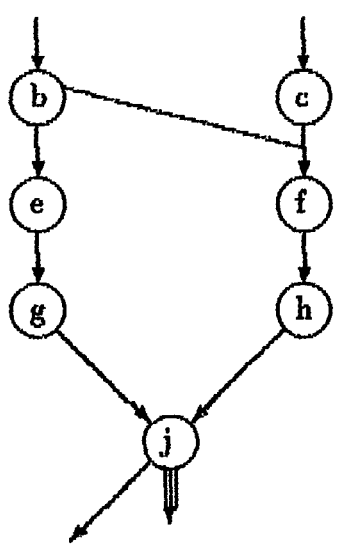
Com um custo c = (0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, −100)

Obtemos: cp = Pc = (22, 11, 16, 11, −5.5, 11, 11, −5.5, 16, 11, 27).

Que como esperávamos tem componentes negativas associadas aos arcos correspondente ao ciclo.



Estado de conhecimento de demonstração total



4.5 – PERTURBAÇÃO

Três tipos de perturbações serão analizados:

i) depois de uma busca bem sucedida, se elimina uma das cláusulas e se pergunta se existe uma demonstração que não use essa cláusula.

Este problema pode ser resolvido procurando–se uma demonstração que não use a cláusula eliminada.

Seja a_k a cláusula eliminada do sistema clausular de Horn, seja $\bar{A}$ o sistema perturbado.

1) como se conhece um ponto viável resultante do problema de busca, se fixa um custo que gere direções negativas na variável que se quizer eliminar, para isto o custo mais adequado é $c_i = 0, \forall i \neq k$ e $c_k = 1$;

2) use o algoritmo BEGO para conseguir o ótimo;

3) se a componente x_k é zero no ótimo, então sabemos que a variável pode ser eliminada existindo uma demonstração de ℓ em $\bar{A}$. Se no ótimo a componente x_k não se anula, então não existe uma demonstração de ℓ em $\bar{A}$.

ii) despois de uma busca não sucedida, se agrega a cláusula a_k resultando no sistema $\bar{A}$.

1) como a busca foi não sucedida a componente associada à parte insumo–produto do sistema ainda não foi eliminada se procede simplesmente a continuar o processo de busca com a nova cláusula.

iii) depois de uma busca bem sucedida se agrega uma cláusula A_k e se pergunta se existem demonstrações de ℓ em $\bar{A}$ que usem essa cláusula.

1) este problema pode ser resolvido em uma iteração, partindo–se do ponto viável, resultante da busca anterior e se calcula uma direção projetiva associada ao custo c definido da seguinte maneira: $c_i = 0$, $\forall i \neq k$ e $c_k = -1$. Se a direção $d = -cp$ tem a componente $d_k > 0$ a resposta é sim, caso contrário não existe demonstração onde ℓ em $\hat{A}$ que use a cláusula a_k recém agregada.

4.5.1 – Resultados

O algoritmo se comporta melhor na eliminação de cláusulas que as técnicas combinatórias conhecidas, pois não é necessário recomeçar o processo desde o começo.

Ao agregar–se novas cláusulas, o algoritmo usa os valores de fluxo obtidos para proceder a obtenção de uma solução ao problema de busca.

A continuação do processo em uma situação associada à solução do problema antes da perturbação constitue uma vantagem do ponto de vista da inteligência artificial, pois isto nos permite associar a situação presente com a situação anterior, fenômeno que poderia ser usado vantajosamente na solução de diálogos. O trabalho de LITMAN [22] apresentado no capítulo dois, orientado a solução de diálogos, procura guardar numa pilha a situação anterior, para utilizá–la na solução posterior; os valores associados às cláusulas resultantes da busca anterior poderiam ser usados em substituição a esta pilha.

Outros tipos de perturbação podem ser resolvidos de maneira semelhante.

4.6 – EFICIÊNCIA DA HEURÍSTICA

Uma pergunta fundamental em relação ao método apresentado é com que freqüência o algoritmo elimina erradamente cláusulas úteis que devem ser incorporadas novamente.

Do ponto de vista teórico uma cláusula pode ser erradamente eliminada e incorporada um número exponencial de vezes, dado que a convergência do algoritmo só assegura que se tem em um número finito de passos numa solução ótima.

Na prática devido as características específicas do modelo, o algoritmo dificilmente elimina cláusulas úteis.

Depois de construirmos um exemplo em 12 dimensões, no qual uma cláusula útil reduz seu fluxo na primeira iteração, para se conseguir um exemplo em que uma cláusula útil é eliminada e reincorporada imediatamente, foi necessário uma instância com mais de 100 dimensões.

O algoritmo foi implementado nas linguagens de desenvolvimento GAUSS e MAT–LAB que não permitem matrizes de mais de 200 dimensões. Nessa categoria o comportamento da heurística é quase infalível. No entanto será necessário esperar por uma implementação que aceite matrizes maiores para definitivamente estabelecer a eficiência da heurística na deteção de cláusulas inúteis.

Por outro lado, o exemplo construído é muito desbalanceado, pois foi desenvolvido repetindo–se todas as cláusulas, menos aquela útil que reduz seu fluxo, isto faz pensar que um pré–escalamento poderia melhorar ainda mais a eficiência do algoritmo.

Os resultados das experiências empíricas realizadas em matrizes de 200 dimensões produz uma eficiência de cerca de 100%.

4.7 – CONCLUSÕES

Os resultados de maior interesse estão associados a diferentes especialidades da computação utilizadas na solução do problema de busca em sistemas de Horn:

1) na área da programação linear se desenvolve um algoritmo para ser usado na solução de modelos combinatórios poliédricos, donde as múltiplas degenerações tanto primais como duais são tratadas de uma forma uniforme e transparente. Um conjunto de lemas para os quais nenhuma hipótese de não degeneração foi considerada asseguram a correção dos diferentes passos do algoritmo BEGO e justificam uma política de mínimas variações no conjunto ativo. A convergência do algoritmo é formalmente provada;

2) na área de modelos, se utiliza a técnica BIG M para algoritmos de ponto interior na construção de modelos combinatórios poliédricos que apresentam vantagens em relação com os modelos de primeira fase Simplex até agora utilizados. O uso de algoritmos de direções viáveis se propõe como alternativa válida ao Simplex na solução de modelos combinatórios. Se observa que a relação modelo–algoritmo determina um comportamento combinatório específico;

3) na área da combinatória se propõe um novo processo como alternativa dos algoritmos Bottom–Up e Top–Down e um novo princípio como alternativa ao de caminhos de aumento;

4) na área de grafos um conjunto de definições em hipergrafos como hipergrafo unitário, hiperárvore, hipercaminhos e hiperciclo nos permitem construir mecanismos para detectar estas estruturas;

5) na área da inteligência artificial comportamentos característicos da forma humana de processar informações, como a eliminação de cláusulas, o princípio das mínimas conclusões e a associação com estados de conhecimento prévios são desenvolvidas de forma natural pelo método proposto.

4.8 – DESENVOLVIMENTOS FUTUROS

Do ponto de vista teórico, parece interessante realizar uma investigação que permita estabelecer o comportamento do modelo quando se utilizam algoritmos de direções de busca, em vez de algoritmos projetivos.

Deve–se analisar também a influência de escalamento na eficiência do algoritmo. Técnicas de "pricing" dinâmico podem também ser usadas. Estes artifícios podem resultar na construção de um sistema infalível na deteção de cláusulas inúteis.

Um estudo de natureza teórica com vistas a determinar as causas que determinam a eliminação e cláusulas úteis. Deve ser feito um estudo empírico para se determinar estatisticamente a eficiência da heurística na eliminação de cláusulas inúteis.

Uma implementação do algoritmo usando matrizes esparsas e linguagens eficientes como C ou FORTRAN em substituição das implementações em linguagens de desenvolvimento como GAUSS e MAT–LAB desenvolvidas no presente trabalho

que só permitem o uso de matrizes densas e de dimensões maiores.

Uma implementação do método na solução de problemas reais na área de linguagens naturais, reconhecimento de planos ou sistemas especialistas.

Desenvolver uma linguagem de cláusulas de Horn tipo LISP ou PROLOG em que os mecanismos de recursão automática sejam substituídos pelo método de otimização em modelos de fluxo em hipergrafos aqui apresentado para desenvolver aplicações na área de inteligência artificial.

Estudar o comportamento do algoritmo em um modelo com duas variáveis artificiais, uma associada à parte insumo–produto e outra associada à técnica de vácuo apresentada no capítulo 4 para a deteção de ciclos.

Uma implementação em paralelo do algoritmo poderia resultar em um método competitivo com os métodos combinatórios na solução do problema de busca em sistemas clausulares.

Parece em termos gerais, razoável pensar que os resultados aqui obtidos abrem um conjunto de possibilidades interessantes a ser desenvolvidas para melhorar nossos conhecimentos dos mecanismos de processamento de informação em geral e em particular da dedução em sistemas clausulares.

BIBLIOGRAFIA

[1] P. E. GILL and W. MURRAY, 1977 – The Computation of Lagrange Multiplier Estimates for Constrained Minimization Mathematical Programming 17 (1989), 32–60.

[2] C. GONZAGA, 1992 – Interior Point Methods – A ser publicado.

[3] KHACHIAN, 1979 – A Polinomial Algorithm in Linear Programming. Diklady Akademii Nauk 244, 191–194.

[4] N. KARMARKAR, 1984 – A New Polynomial–Time Algorithms for Linear Programming. Combinatoria 4, 373–395.

[5] R. FRISCH, 1957 – The Multiplex Method for Linear Programming Sankhya: The Indian Journal of Statistics 18, 329–367.

[6] J. B. ROGEN, 1960 – Gradient Projection Method for Non–Linear Programming: Part I, Linear Constraints Journal of SIAM 8, 181–217.

[7] K. O. KORTANEK and Z. JUSHAN, 1988 – New Purification Algorithms for Linear Programming, Department of Management Science, University of Iowa.

[8] E. KLOTZ, 1988 – Dynamic Pricing Criteria in Linear Programming. Systems Optimization Laboratory Technical Report 88–15, Stanford University.

[9] G. W. BROWN and T. C. KOOPMANS 1951 – Computational Suggestions

for Miximizing a Linear Function. Chapter XXV in Activity Analysis of Production and Allocation. Cowles Comission Monograph 13, J. Willey and Sons.

[10] J. L. NAZARETH, 1987 – Pricing Criteria in Linear Programming. University of California, Berkeley.

[11] W. RODDER and M. BLAUTH, 1981 – Project: An Alternative Linear Programming Algorithm. Congress of Mathematical Programming.

[12] T. ROCKAFELLAR, 1970 – Convex Analysis. Princeton Mathematics Series, PrincetonUniversity.

[13] M. MINSKY, 1975 – A Framework for Representing Knowledge, in the Bychology of Computer Vision, McGraw–Hill.

[14] A. NEWELL and H. SIMON, 1963 – A Program that Simulates Human Throught, McGraw–Hill.

[15] J. McCARTHY and P. HAYES, 1969 – Some Philosophical Problems from the Standpoint of Artificial Intelligence in Machine Intelligence U. B. Meltzer and D. Michie (Ed.), Edinburgh University Press.

[16] C. SCHMIDT, N. SEIDHARAM and J. GOODSON, 1978 – The Plan Recognition Problem. Artificial Intelligencell, 45–83.

[17] B. BRUCE, 1981 – Plans and Social Section, in Theoretical Issues in Reading Comprehension. Laurence Earbaum, Hillsdale.

[18] R. SHANK, 1975 – Conceptual Information Processing. American Eisevier.

[19] J. AUSTIN, 1962 – How to do Things with Words. Oxford University Press.

[20] J. SEARLE, 1969 – Speach Acts, an Essay in the Philosophy of Language. Cambridge University Press.

[21] P. R. COHEN and C. PERRAULT, 1979 – Elements of Plan Based Theory of Speach Acts, Cognitive Science 33, 177–212.

[22] J. F. ALLEN and C. R. PERRAULT, 1980 – Analysing Intention in Utterances, Artificial Intelligence 15, 3, 143–178.

[23] D. LITMAN and J. ALLEN, 1984 – A Plan Recognition Model for Clarification Subdialogues, JR 141, Department of Computer Science, University of Rochester.

[24] M. POLLACK – A Model of Plan Inferences. Proceedings of the ACL–86, New York.

[25] E. CHARNIAK and D. McDERMONT, 1985 – Introduction to Artificial Intelligence. Addison Wesley, Reading, MA.

[26] R. JEROSLOW e J. WANG, 1989 – Dynamic Programming, Integar Polyhedra and Horn Clause Knowledge Bases. ORSA Journal on Computing 1: 1, 7–19.

[27] A. TORRES, J. ARAOZ, 1988 – Combinatorial Model for Searching in Knowledge Bases. Matematicas Acta Cientifica Venezolana 39: 387–394.

[28] I. ADLER, N. KARMARKAR, M. RESENDE, G. VEIGA, 1986 – An Implementation of Karmarkar's Algoritm for Linear Programming.