UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DEPARTAMENTO DE MATEMÁTICA APLICADA

Método de Geração de Colunas aplicado ao problema de Satisfatibilidade Probabilística

MAP2060 - Trabalho de Formatura

Autor: Paulo Sérgio de Souza Andrade
Orientador: Prof^{or}. Dr^{or}. Fábio Gagliard Cozman
Departamento da Mecatrônica e Sistemas
Mecânicos da Escola Politécnica

São Paulo
Janeiro de 2007

Paulo Sérgio de Souza Andrade

Método de Geração de Colunas aplicado ao problema de Satisfatibilidade Probabilística

Monografia apresentada ao Instituto de Matemática e Estatística da Universidade de São Paulo para obtenção do título de Bacharel em Matemática Aplicada e Computacional

Habilitação em Mecatrônica e Sistemas Mecânicos

Orientador: Prof^{or}. Dr^{or}. Fábio Gagliard Cozman.

São Paulo
Janeiro de 2007

Agradecimentos

Gostaria de manifestar o meu agradecimento a todas as pessoas e instituições que contribuíram direta ou indiretamente para que a realização deste trabalho fosse possível.

Ao meu orientador, Prof^{or}. Dr^{or}. Fabio Gagliard Cozman, fico grato pela oportunidade que me deu, a orientação científica, ao compartilhamento de conhecimento e experiência, a confiança e a dedicação ao longo deste tempo e todas as sugestões feitas durante o desenvolvimento do trabalho.

Tenho ainda a agradecer-lhe, a disponibilidade das instalações para trabalhar junto ao Grupo de Telecomando e Tomada de Decisão, onde pude conhecer pessoas que sempre ajudaram a ganhar ânimo para continuar a trabalhar, mostraram-se companheiras e amigas, e que fizeram parte de discussões que foram de grande apoio. Em especial, ao Prof^{or}. Dr^{or}. José C. F. da Rocha, da Universidade Estadual de Ponta Grossa, agradeço a ajuda na geração de testes e apoio na elaboração do código, ao Prof^{or}. Dr^{or}. Cassio P. de Campos, da USP Leste, pelo apoio na utilização do programa ILOG CPLEX, e ao aluno de Doutorado Daniel Kikuti, a quem eu sempre recorri para esclarecer as dúvidas que tinha de programação.

Aos meus colegas de curso, Paulo C. Felix e Tiago H. de Carvalho, fico grato pelas dicas que recebi durante as conversas que tivemos.

Aos meus pais e irmãos, pelo encorajamento, apoio e compreensão.

Resumo

No problema de satisfatibilidade probabilística temos um conjunto de sentenças e para cada sentença temos especificada a probabilidde de que ela seja verdadeira. Assim o problema resume-se na verificação da consistência das probabilidades atribuídas. No modelo de programação linear proposto por Nilsson (1986), as probabilidades são consistentes se há uma distribuição de probabilidade sobre as atribuicões verdade, tal que a probabilidade de cada sentença é a medida do quanto ela é satisfeita no conjunto de atribuições.

Uma dificuldade associada ao modelo é devida ao número exponencial de variáveis resultante, que pode ser contornada com um algorítmo de geração de colunas, proposto por Jaumard, Hansen, Aragão (1991), e no qual focamos o presente trabalho com a implementação de tal algorítmo utilizando o programa ILOG CPLEX (ILOG, 2006) para resolução dos problemas, um de programação contínua e outro de programação não-linear avaliado em variáveis binárias, associados ao método.

São apresentados resultados de testes computacionais para um conjunto particular de instâncias, com uma visão exploratória de seu comportamento.

Sumário

1 Introdução

O problema de *satisfatibilidade probabilística* ($PSAT$[1]), não é um problema novo, pois sua origem data do século XIX dos trabalhos realizados por George Boole e foi redescoberto por Nilsson (1986), no contexto de Inteligência Artificial (IA), sobre o nome de *lógica probabilística* (Andersen; Pretolani, 2001).

A maioria das aproximações para tomada de decisão em um ambiente complexo e realístico envolvem a noção que um fato , ou sentença lógica, é verdadeiro com uma certa probabilidade (Georgakopoulos; Kavvadias; Papadimitriou, 1988), pois pode ser que não conhecemos com certeza a verdade de uma particular sentença, mas estamos hábeis a atribuir uma probabilidade empírica de validade. Uma possível aplicação para $PSAT$ em IA é avaliar se um grande conjunto de precisas ou imprecisas probabilidades definem ou não uma base de conhecimento consistente ou coerente quando atribuídas ao correspondente conjunto de sentenças.

Muitas propostas foram feitas para lidar com incerteza em IA, algumas delas são revistas ou referenciadas em Hansen e Jaumard (1996).

A que nos interessa expor aqui consta de uma rigorosa estrutura para lidar com probabilidade de sentenças lógicas da lógica proposicional proposta por Nilsson (1986), em que segue-se a suposição geral da teoria da probabilidade de que existe uma distribuição de probabilidade consistente sobre todos os estados possíveis, no domínio de interesse, que representa a presente base de conhecimento (Jaumard; Hansen; Aragão, 1991).

No artigo de Nilsson (1986) propõe-se que $PSAT$ seja formulado em termos de um modelo de programação linear. Dificuldades chegam nessa formulação, visto o número exponencial de variáveis associadas ao problema. Entretanto, há uma forma de contornar esse problema, ou seja, através de algorítmos de programação linear grandes instâncias podem ser resolvidas usando métodos de geração de colunas e programação não-linear avaliada em variáveis $0-1$ (Jaumard et al., 1991). Alguns desses métodos já foram testados e apresentaram bons resultados para poucas centenas de variáveis, como em Jaumard et al. (1991) que resolvem casos com 140 variáveis e 300 sentenças, em Hansen e Perron (2004) que resolvem casos com 200 variáveis e 800 sentenças e em Jovanović, Mladenović e Ognjanović (2006) casos chegando a 1000 sentenças, com a observação que todos esses autores empregaram heurísticas na resolução do problema de programação não-linear associado ao método de geração de colunas e obtiveram respostas para problemas como no caso que trabalharemos aqui ou mais complexos em tempo razoável.

[1] do inglês Probabilistic Satisfiability

É bem conhecido que $PSAT$ está na classe de complexidade dos problemas NP-completos, como pode ser visto em Georgakopoulos et al. (1988), que é uma sub-classe da classe NP onde os problemas são inerentemente difíceis e não possuem nenhum algorítmo de tempo polinomial, embora não haja prova disto (Russell; Norvig, 2003).

Dado que a aproximação baseada sobre lógica e probabilidade é uma das mais aceitas por depender de técnicas avançadas de programação linear e inteira mista (Hansen; Jaumard, 1996), o presente trabalho visa a implementação de um algorítmo em C para verificação da consistência de um conjunto de probabilidades atribuídas ao respectivo conjunto de sentenças, formulado na forma de decisão como em Hansen e Jaumard (1996), e que usa ILOG CPLEX 10.0 como programa para resolução do problema linear e do não-linear, numa versão linearizada, relacionados ao método numérico de geração de colunas.

A utilização de ILOG CPLEX, muitas vezes relatado na literatura de IA, é uma alternativa eficiente na resolução de $PSAT$, por isso nos motivamos a implementar tal método com uma visão exploratória do seu comportamento e para empregá-lo em possíveis futuras extensões onde seja viável.

2 $PSAT$ e programação linear

$PSAT$ na forma de decisão (Hansen, Jaumard, 1996), pode ser definido como segue:

Os operadores da lógica booleana $\vee$ *(soma)*, $\wedge$ *(produto)* e $\neg$ *(negação)*, estabelecem as relações lógicas sobre o conjunto dado de m *sentenças lógicas*, as quais particularizaremos para o caso em que todas são *cláusulas*, $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$, definidas sobre n *variáveis proposicionais*, $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$. Juntamente é dado um vetor $\pi \in \mathbb{R}^m$ que define as *probabilidades* de que as cláusulas são verdadeiras. Vale lembrar que uma cláusula é uma *disjunção* de *literais*, onde um literal é uma variável x_j ou uma variável negada $\neg x_j$. Por exemplo $x_1 \vee x_3 \vee \neg x_{10}$ é uma cláusula de comprimento três.

No problema de $PSAT$ as relações lógicas entre as variáveis estão ocultas e incerteza está associada com as cláusulas (Pretolani, 2005). Logo, a questão é verificar se o vetor de probabilidades é *consistente* com o conjunto de sentenças.

Uma *atribuição verdade* , ou, equivalentemente, um *mundo possível*, é uma atribuição de valores $\{verdadeiro, falso\}$ às variáveis em $\mathcal{X}$, ou equivalentemente, um vetor $w \in \{0,1\}^n$. Devemos notar que o número de mundos possíveis é $N = 2^n$ e assim denotamos $\mathcal{W} = \{w^{(1)}, w^{(2)}, \ldots, w^{(N)}\}$, o conjunto dos mundos possíveis e a matriz $W = [w^{(1)}|w^{(2)}|\ldots|w^{(N)}]$ de dimensão $n \times N$.

Uma cláusula C_i é *satisfeita* por um mundo possível se e somente se ela apresenta valor verdadeiro para pelo menos um literal em C_i, assim cada mundo possível associa ao conjunto de cláusulas um vetor $a \in \{0,1\}^m$, sendo a_i o valor verdade de C_i, ou seja, $a_i = 1$ se C_i é satisfeito e $a_i = 0$ caso contrário. Agora denotamos por $\mathcal{A} = \{a^{(1)}, a^{(2)}, \ldots, a^{(N)}\}$ o conjunto dos N vetores verdade correspondentes aos N mundos possíveis e a matriz $A = [a^{(1)}|a^{(2)}|\ldots|a^{(N)}]$ de dimensão $m \times N$. Como observado por Hansen e Jaumard (1996) o número máximo de colunas diferentes é dado por $2^{argmin\{m,n\}}$.

Uma atribuição de probabilidade para os mundos possíveis é um vetor $p \in \mathbb{R}^N$ tal que $\sum_{i=1}^{N} p_i = 1$. Assim dado uma atribuição de probabilidade conforme Pretolani (2005):

- p dá a combinação convexa dos vértices do hipercubo unitário no $\mathbb{R}^n$, isto é, ele define um vetor $v = Wp$ no hipercubo, cujo componente v_j é o valor da soma das probabilidades atribuídas para os mundos possíveis onde à x_j foi atribuído valor verdadeiro, e dá a probabilidade da variável x_j sobre a atribuição p.

- p dá a combinação convexa dos vetores em $\mathcal{A}$, ou seja ele define um vetor $y = Ap$ no hipercubo unitário no $\mathbb{R}^m$, cujo componente y_i é o valor da soma das probabilidades atribuídas para os mundos possíveis onde C_i é satisfeita, e dá a probabilidade da cláusula C_i sobre a atribuição p.

Assim, π é consistente se existe uma atribuição de probabilidade p satisfazendo $\mathcal{P} := \{p \in \mathbb{R}^N : Ap = \pi, \sum_{i=1}^{N} p_i = 1, p \geq 0\}$, e não caso contrário. Logo com a adição de uma fictícia função objetivo $0p$, que atribui custo zero para cada variável e, visando a simplificação, a restrição $\sum_{i=1}^{N} p_i = 1$ será considerada aqui como uma sentença lógica adicional correspondendo a uma tautologia que contará com a inclusão de uma linha A_0 na matriz A e de uma probabilidade $\pi_0 = 1$.

Agora, o problema $PSAT$ pode ser formulado como um problema de programação linear e a sua forma de decisão pode ser escrita como:

$$P_{PSAT} = \begin{cases} \min & \mathbf{0}p \\ \text{sujeito a} & \\ & Ap = \pi \\ & p \geq 0 \end{cases} \tag{1}$$

Então π é consistente se P_{PSAT} tem uma solução viável, isto é, o valor ótimo de P_{PSAT} é zero implicando que $\mathcal{P} \neq \emptyset$.

Nota: $PSAT$ é uma generalização do problema de *satisfatibilidade proposicional* (SAT), pois podemos obter SAT de $PSAT$ atribuindo $(\pi_1, \pi_2, \ldots, \pi_m) =$

mundos possíveis (x_3, x_2, x_1)	x_3	x_2	$\neg x_3 \vee \neg x_2 \vee x_1$
(0,0,0)	0	0	1
(0,0,1)	0	0	1
(0,1,0)	0	1	1
(0,1,1)	0	1	1
(1,0,0)	1	0	1
(1,0,1)	1	0	1
(1,1,0)	1	1	0
(1,1,1)	1	1	1

Tabela 1: Tabela da verdade para as sentenças nos mundos possíveis

$(1, 1, \ldots, 1)$ às m cláusulas, e nesse nosso interesse é verificar se existe um mundo possível para as n variáveis lógicas em $\mathcal{X}$ tal que todas as m cláusulas em $\mathcal{C}$ são simultaneamente satisfeitas (Andersen; Pretolani, 2001).

2.1 Exemplos

A seguir damos um exemplo que se aproxima do apresentado por Andersen e Hooker (1996).

Exemplo 1 .

Suponha que temos uma pequena base de conhecimento consistindo nas três sentenças abaixo:

$$\begin{aligned} C_1 &= x_3 \\ C_2 &= x_2 \\ C_3 &= \neg x_3 \vee \neg x_2 \vee x_1 \end{aligned} \tag{2}$$

onde x_1, x_2 e x_3 são as variáveis lógicas. Por exemplo x_3 e x_2 podem representar dois sintomas distintos e x_1 pode representar uma doença específica. A terceira sentença de (2) pode ser interpretada da seguinte forma: se uma pessoa tem os dois sintomas distintos isto implica que ela tem a particular doença específica também, sendo que a última sentença é equivalente a fórmula $(x_3 \wedge x_2) \rightarrow x_1$.

A Tabela (1) apresenta a avaliação para as sentenças em (2), em que atribuem-se os valores verdade com relação aos respectivos mundos possíveis.

Então, deixando $p = (p_1, p_2, \ldots, p_8)$ denotar a probabilidade dos oito mundos possíveis e $\pi = (\pi_1, \pi_2, \pi_3)$ denotar a probabilidade das sentenças em (2), na ordem em que aparecem.

Logo, o sistema resultante de $Ap = \pi, p \geq 0$ *é:*

$$\begin{array}{cccccccccccccccc} p_1 & + & p_2 & + & p_3 & + & p_4 & + & p_5 & + & p_6 & + & p_7 & + & p_8 & = & 1 \\ & & & & & & & & p_5 & + & p_6 & + & p_7 & + & p_8 & = & \pi_1 \\ & & & & p_3 & + & p_4 & + & & & & & p_7 & + & p_8 & = & \pi_2 \\ p_1 & + & p_2 & + & p_3 & + & p_4 & + & p_5 & + & p_6 & + & & & p_8 & = & \pi_3 \\ & & & & & & & & & & & & & & p_i & \geq & 0, \forall i \end{array} \tag{3}$$

A segunda equação em (3) diz que a probabilidade da sentença x_3 *deve ser igual a* π_1*, pois* x_3 *é verdadeiro nos últimos quatro mundos possíveis. Similarmente a segunda e terceira equações em (3) dizem que as probabilidades de* x_2 *e* $\neg x_3 \vee \neg x_2 \vee x_1$ *devem ser iguais a* π_2 *e* π_3*, respectivamente. A primeira e a última equações em (3) expressam o fato que p deve constituir uma distribuição de probabilidade. Neste exemplo para que tenhamos uma atribuição de probabilidades válida para as sentenças no banco de dados, o vetor* π *deve satisfazer, segundo Hansen, Jaumard e Aragão (1995):*

$$\begin{array}{l} \pi_1 + \pi_3 \geq 1 \\ \pi_2 + \pi_3 \geq 1 \\ \pi_i \leq 1, i = 1, 2e3 \end{array} \tag{4}$$

No exemplo a seguir, extraído de Nilsson (1986), procura-se explorar $PSAT$ segundo sua interpretação geométrica:

Exemplo 2 .

Considere as três sentenças abaixo:

$$\begin{array}{l} C_1 = x_1 \\ C_2 = \neg x_1 \vee x_2 \\ C_3 = x_2 \end{array} \tag{5}$$

e os vetores resultantes da avaliação das sentenças nos mundos possíveis:

$a^{(1)} = (010)^T, a^{(2)} = (100)^T, a^{(3)} = (011)^T, a^{(4)}(111)^T$.

Os valores de probabilidade destas sentenças são restringidos por $Ap = \pi$ *e* $p \geq 0$*, e a estas restrições pode ser dado uma simples interpretação geométrica. A equação matricial traça um espaço de probabilidade sobre os mundos possíveis dentro de um espaço de valores de probabilidade sobre sentenças. A traçagem é linear e, por esta razão, traça valores extremos de p dentro de valores extremos de* π*. Há quatro valores extremos de p a saber:*

$p^{(1)} = (1000)^T, p^{(2)} = (0100)^T, p^{(3)} = (0010)^T, p^{(4)}(0001)^T$

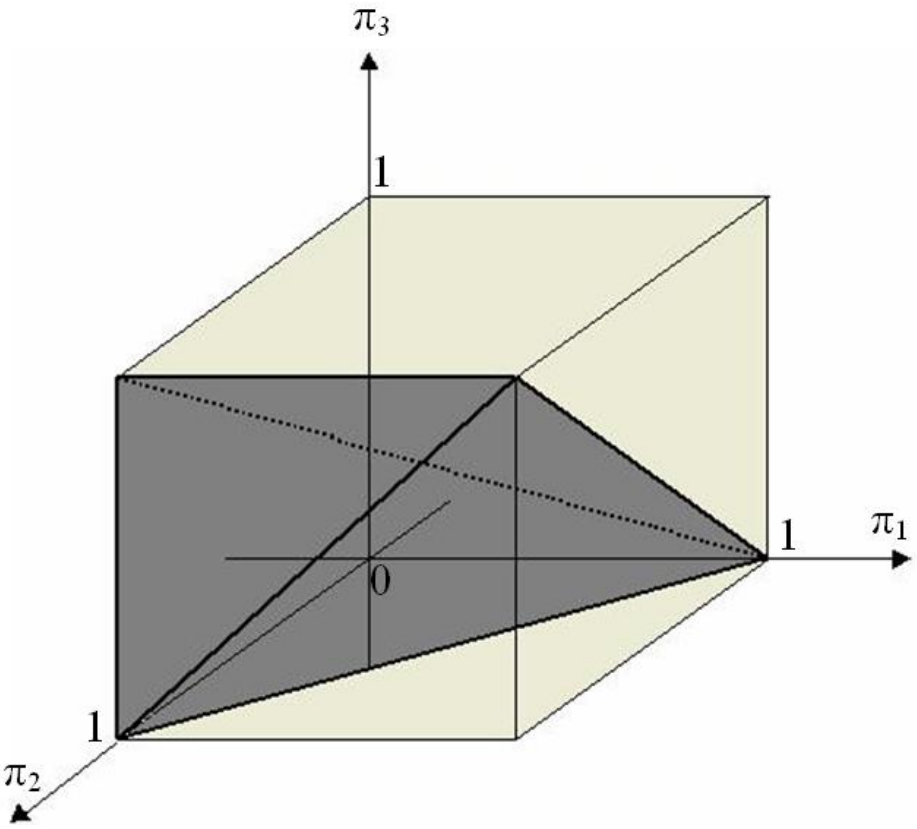

Figura 1: Região convexa de valores de probabilidade consistentes

Os vetores extremos $\pi^{(i)}$ correspondentes a estes vetores extremos $p^{(i)}$ são simplesmente os vetores $a^{(i)}$ que compõem a matriz A. Já para valores arbitrários de p, π deve pertencer ao fecho convexo dos valores extremos de π.

Na Figura 1 os valores extremos de π são os vértices da região em forma de pirâmide que apresenta-se destacada. Valores consistentes de π para as sentenças devem pertencer ao fecho convexo destes vértices.

3 Solução numérica por geração de colunas

3.1 Motivação

Visto que o problema P_{PSAT} (1) tem um número de colunas que cresce exponencialmente dado por $2^{argmin\{m,n\}}$, resolvê-lo olhando o problema explicitamente, como já apresentado na literatura de IA, é tarefa impraticável para grandes problemas, como observado por Nilsson (1986), e o mesmo recomenda olharmos para heurísticas.

O método que usaremos faz parte de uma avançada ferramenta de programação linear chamado de *geração de colunas* (Bertsimas; Tsitsiklis, 1997). Aqui, dois

programas estão associados com o programa linear original chamado de *Programa Mestre (PM)*: o *Programa Mestre Restrito (PMR)*, que é idêntico ao programa original mas com um pequeno número de colunas explícitas, e o *subproblema (SP)*, cuja função é determinar a coluna de entrada ou determinar o critério de parada, como no *simplex revisado*. Um específico problema de otimização deve ser resolvido para este propósito, uma vez que a coluna de entrada é determinada, obtém-se sua expressão no PMR atual e realizam-se iterações do simplex para resolução do PMR.

3.2 Revisão da geração de colunas em programação linear

Considere o problema na forma padrão, o qual chamamos de problema mestre (PM),

$$\begin{aligned} \min \quad & z = cx \\ \text{sujeito a} \quad & \\ & Ax = b \\ & x \geq 0 \end{aligned} \tag{6}$$

com $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, admitindo $n \geq m$ e supondo que A tem posto m.

Na sua solução atual no algorítmo simplex, dada pela solução do problema mestre restrito (PMR), que mantém explícitas somente as colunas das variáveis que compõem a base, temos fixados m elementos de $\{1, 2, \ldots, n\}$, compondo a lista $B = (j_1, j_2, \ldots, j_m)$ das *variáveis básicas*, e seu complementar, compondo a lista $N = (l_1, l_2, \ldots, l_{n-m})$ das *variáveis não básicas*, ambos, em alguns casos, podem ser referidos como conjuntos. Assim, reordenando as colunas de A ela pode ser dividida em duas matrizes $A = [A_B|A_N]$, onde A_B é uma matriz quadrada $(m \times m)$ não singular chamada de *matriz básica* e A_N a *matriz não básica*.

Como $\{A_i\}_{i \in B}$ é base, então qualquer que seja $x \in \mathbb{R}^n$, vale a equivalência:

$$b = Ax = A_B x_B + A_N x_N \Leftrightarrow x_B = A_B^{-1}(b - A_N x_N) = A_B^{-1} b - A_B^{-1} A_N x_N \tag{7}$$

Lembrando que B é *base viável* se e somente se B é base e $A_B^{-1} \geq 0$. Se B é base viável e x a *solução básica viável* ou *vértice associado*, então

$$\begin{aligned} x_B &:= A_B^{-1} b \geq 0 \\ x_N &:= 0 \end{aligned} \tag{8}$$

Fazendo uso de (7) na função objetivo, temos:

$$\begin{aligned} cx = c_B x_B + c_N x_N = c_B(A_B^{-1} b - A_B^{-1} A_N x_N) + c_N x_N = \\ = c_B A_B^{-1} b + (c_N - c_B A_B^{-1} A_N) x_N \end{aligned} \tag{9}$$

Isto nos leva a reescrever (6), usando (9), como:

$$\begin{array}{ll} \min & z = c_B A_B^{-1} b + (c_N - c_B A_B^{-1} A_N) x_N \\ \text{sujeito a} & \\ & x_B + A_B^{-1} A_N x_N = A_B^{-1} b \\ & x_B, x_N \geq 0 \end{array} \tag{10}$$

onde x_B, x_N são os vetores de variáveis básicas e não básicas, correspondendo às colunas das matrizes A_B e A_N, respectivamente, e c_B, c_N os correspondentes vetores de coeficientes na função objetivo.

Em cada iteração do algorítmo simplex procuramos por uma variável não básica com o menor *custo reduzido* para entrar na base, e lembrando que o custo reduzido das variáveis básica é zero, temos

$$c^* = c_k - c_B A_B^{-1} A^k = \min_{j \in N} c_j - c_B A_B^{-1} A^j = c_j - u^T A^j \tag{11}$$

onde $u = c_B A_B^{-1}$ é o atual vetor de *variáveis duais*.

O método simplex primal é um procedimento iterativo numérico que inicia de alguma solução básica viável no primal e tenta torná-la viável no dual.

Uma solução básica é chamada *solução básica viável dual* se todos os custos reduzidos $c^* \geq 0$, isto é, se não há variável não básica que mudando na direção viável pode melhorar a função objetivo z.

Solução básica que é viável no primal e no dual ao mesmo tempo é *solução ótima* do problema.

Uma busca explicita para resolução de (11) pode ser computacionalmente impossível quando o número de colunas da matriz A é grande. Na prática a resolução de (11) é realizada por um algorítmo específico, que não perde tempo olhando cada uma das colunas não básicas, ou seja, nele os coeficientes na coluna A^j são as variáveis. Se o mínimo deste problema de otimização é $c^* \geq 0$, então todos custos reduzidos não são negativos e nós temos uma solução ótima para o problema original. Se por outro lado o mínimo é negativo, a variável x_k, correspondente ao índice k resultante do SP, tem custo reduzido negativo e a coluna A^k pode entrar na base.

A coluna de entrada é obtida calculando-se $A_B^{-1} A^k$, e as demais operações prosseguem, verificação de otimalidade, verificação de ilimitação, escolha da variável de partida, atualização da solução e da inversa da base.

Com a usual precaução contra ciclagem do método simplex, geração de colunas é finito e exato com a vantagem de resolver um problema de otimização em (11) ao invéz de resolver um problema de enumeração, cuja vantagem torna-se mais apa-

rente quando lembramos que vetores coluna de A frequentemente codificam objetos combinatoriais.

4 O subproblema para P_{PSAT}

Para P_{PSAT} o SP (11) para ser resolvido em cada iteração, isto é, geração de coluna com custo reduzido negativo é:

$$\min_{j \in N} c_j - u^T A^j = \min_{x \in \{0,1\}} -u_0 - \sum_{i=1}^{m} u_i C_i \tag{12}$$

Então (12) é transformada em uma expressão aritmética envolvendo as variáveis lógicas que aparecem em C_i, com valores verdadeiro ou falso associados a 1 e 0, usando as relações:

$$\begin{aligned} x_i \vee x_j &\equiv x_i + x_j - x_i \times x_j \\ x_i \wedge x_j &\equiv x_i \times x_j \\ \neg x_i &\equiv 1 - x_i \end{aligned} \tag{13}$$

Logo (12) resulta num *programa não-linear ou multilinear sem restrições*, avaliado em variáveis 0-1 para colunas implícitas e para o exame do custo reduzido para as colunas explícitas.

Podemos utilizar as leis de DeMorgan:

$$\begin{aligned} \neg(x_i \wedge x_j) &\equiv \neg x_i \vee \neg x_j \\ \neg(x_i \vee x_j) &\equiv \neg x_i \wedge \neg x_j \\ \neg(\neg x_i) &\equiv x_i \end{aligned} \tag{14}$$

e no caso onde todas as sentenças são cláusulas, definindo C_i^- e C_i^+ como o conjunto dos índices das variáveis negadas e não negadas, respectivamente, que aparecem na cláusula C_i, então de (13) e (14) obtemos uma nova expressão para avaliação da cláusula C_i:

$$C_i \equiv 1 - \prod_{j \in C_i^-} x_j \prod_{j \in C_i^+} \neg x_j \tag{15}$$

e de (12) e (15) temos:

$$c^* = \min_{x \in \{0,1\}} -u_0 - \sum_{i=1}^{m} u_i C_i = \min_{x \in \{0,1\}} -\sum_{i=0}^{m} u_i + \sum_{i=1}^{m} u_i \prod_{j \in C_i^-} x_j \prod_{j \in C_i^+} \neg x_j \tag{16}$$

Então, numa dada iteração, ao resolver o SP (16), a coluna de entrada a^k é obtida realizando-se a avaliação das cláusulas para o mundo possível resultante

de (16) e calculando-se $A_B^{-1}a^k$ e as demais operações prosseguem como no simplex revisado.

NOTA: O problema (12) pode ser visto como uma versão ponderada da *Satisfatibilidade Máxima* ($MAXSAT$): dado um conjunto de m cláusulas ponderadas sobre n variáveis lógicas determinar um mundo possível tal que a soma dos *pesos* das cláusulas satisfeitas é maior ou igual a um dado valor, onde o peso da cláusula C_i é u_i, valendo observar que as variáveis duais não são restringidas quanto ao sinal, pois as restrições no primal são de igualdade.

5 Resolvendo P_{PSAT}

Para resolvermos P_{PSAT} precisamos realizar somente a *fase 0* do algorítmo simplex com geração de colunas: minimização da soma de *variáveis artificiais* adicionadas as restrições. As correspondentes colunas das variáveis que estão na base são mantidas explicitamente, enquanto as demais podem ser descartadas. De forma a detectarmos se o conjunto é inviável, ou seja, $\mathcal{P} = \emptyset$.

5.1 Variante do simplex revisado envolvendo colunas retidas

No método que expusemos acima as colunas que saem da base são descartadas da memória do PMR e não desfrutam de um estado especial.

Na variante deste método, o algorítmo retém na memória todas ou algumas das colunas que foram geradas no passado e procede em termos do PMR que envolve somente colunas retidas (Bertsimas; Tsitsiklis, 1997).

Então, descrevemos o algorítmo como uma seqüência de iterações envolvendo o SP e o PMR. No início de uma dada iteração do SP, temos uma solução básica viável para o PMR e uma matriz básica associada, buscamos uma variável com custo reduzido negativo resolvendo (16), se nenhum é encontrado, o algorítmo termina, caso contrário existe algum $c^k \leq 0$, tal que formamos um conjunto de colunas A^i, $i \in \mathcal{I}$, o qual contém todos os índices das colunas básicas, a coluna de entrada A^k e possivelmente alguma outra coluna também.

Problema mestre restrito:

$$\begin{array}{ll} \min & z = \sum_{i\in\mathcal{I}} c_i x_i \\ \text{sujeito a} & \\ & \sum_{i\in\mathcal{I}} A^i x_i = b \\ & x \geq 0 \end{array} \tag{17}$$

Lembramos que a última solução básica viável do PMR pode ser usada como ponto de início, com o conjunto $\mathcal{I}$ tendo um elemento a mais, para sua próxima solução, quando, então, realizam-se tantas iterações do simplex quantas forem necessárias, até que o PMR seja resolvido otimamente. Neste ponto estamos prontos para realizar a próxima iteração do SP.

Tal método é um caso especial do método simplex revisado, em conjunto com alguma regra específica para escolha da variável de entrada que dá *prioridade* para as variáveis x_i, $i \in \mathcal{I}$. Somente quando o custo reduzido destas variáveis são todos não negativos, que ocorre numa solução ótima do PMR, que o algorítmo examina o custo reduzido das variáveis cujos índices estão no conjunto complementar de $\mathcal{I}$. A motivação é a de que podemos desejar dar prioridade para variáveis para as quais as correspondentes colunas já foram geradas e armazenadas na memória, ou para variáveis que são mais prováveis para ter custo reduzido negativo, embora não estamos certos disto. Há várias variantes deste método dependendo de como o conjunto $\mathcal{I}$ é escolhido em cada iteração.

A nossa proposta será deixarmos $\mathcal{I}$ ser o conjunto de índices de todas as variáveis que passaram por básicas em alguma iteração passada, ou seja, nenhuma variável sai do PMR e cada variável de entrada tem seu índice adicionado ao conjunto $\mathcal{I}$.

Bertsimas e Tsitsiklis (1997) observam que se o número de iterações for grande esta opção poderá ser problemática e quanto ao término desta variante, este é garantido na medida que é um caso especial do método simplex revisado.

5.2 Como obter a primeira base viável

A fase 0 consiste de usarmos *variáveis artificiais* para conseguirmos $Ap = \pi$. Como $\pi \geq 0$, basta tomarmos $y \in \mathbb{R}^m$, tal que $y := \pi$ e considerarmos o sistema $[A|I](p, y)^T = \pi$.

Assim, dado um problema na forma (1) montamos um problema auxiliar fase 0 que resolverá a questão de existência de ponto viável e o problema da base inicial.

O problema fase 0, sendo $B \in \mathbb{R}^{m \times m}$ a matriz identidade de ordem m é,

$$\begin{array}{ll} \min & 1^T y \\ \text{sujeito a} & \\ & Ap + By = \pi \\ & p, y \geq 0 \end{array} \tag{18}$$

que é equivalente ao problema

$$\begin{array}{ll} \min & (0^T, 1^T)z \\ \text{sujeito a} & \\ & A^*z = \pi \\ & z \geq 0 \end{array} \tag{19}$$

Inicialização é fácil para o problema auxiliar: deixando $z_N = 0$ e $z_B = \pi$ teremos uma solução básica viável e a correspondente base inicial sendo as últimas colunas de A^* e como B é matriz identidade $A^*_B = B$ e $A_B^{-1} = B$.

Se p é uma solução para P_{PSAT}, esta escolha de p juntamente com $y = 0$ fornece o valor ótimo de zero para o problema auxiliar. Mas se o valor ótimo no problema auxiliar não for zero, concluímos que P_{PSAT} é inviável. Se por outro lado, obtemos um valor ótimo zero como solução de (19), então ele deve satisfazer $y = 0$ e p é uma solução viável para P_{PSAT}.

Pode ocorrer no método simplex do problema (19) terminar com uma matriz básica consistindo exclusivamente de colunas da matriz A ou de alguma das variáveis artificiais permanecerem na matriz básica final, visto que o valor final das variáveis artificiais é zero, isto implica que nós temos ou uma solução básica degenerada para o problema auxilar, ou uma matriz cujas linhas são linearmente dependentes, ou seja, há restrições redundantes no problema, ou ambos podem estar ocorrendo.

5.3 Resolução do Subproblema para P_{PSAT}

Hansen e Jaumard (1996) observam que resolver exatamente o subproblema em cada iteração pode consumir muito tempo, sendo que para garantir convergência não é necessário resolvê-lo exatamente em cada iteração, ou seja, desde que um custo reduzido negativo, para minimização, seja encontrado, uma iteração do simplex revisado pode ser realizada. Se uma solução viável é obtida desta forma, a versão de decisão de $PSAT$ está resolvida.

Quando nenhum custo reduzido é dado pela heurística, devemos realizar uma iteração com o algorítmo exato para comprovar que não há solução viável para a versão de decisão de $PSAT$.

5.3.1 Resolução exata do subproblema

O procedimento que nos interessa expor aqui envolve uma técnica padrão e completamente natural para reduzir otimização não-linear em variáveis binárias

para programação linear inteira chamada de *linearização*, que, por exemplo, aparece aplicada em Boros e Hammer (2002) e em Hansen e Jaumard (1996).

A idéia básica, na transformação abaixo, é substituir um termo não-linear por uma nova variável, e usar desigualdades lineares para forçar as novas variáveis a tomar em toda solução viável o valor do correspondente termo.

Assim dado um termo na forma

$$c\prod_{j\in J} x_j, \tag{20}$$

onde $c \in \mathbb{R}$ e $x_j \in \{0,1\}$ para $j \in J$, então (20) é equivalente a:

$$\begin{array}{l} cy \\ \text{sujeito a} \\ \qquad \begin{array}{l} y \leq x_j, j \in J \\ y \geq \sum_{j\in J} x_j - |J| + 1 \\ y \geq 0 \end{array} \end{array} \tag{21}$$

com |J| denotando a cardinalidade do conjunto J.

Por exemplo, para forçar a igualdade $y = x_1x_2x_3$ para uma variável binária, $x_1, x_2, x_3 \in \{0,1\}$, podemos escrever:

$$\begin{array}{l} y \leq x_1 \\ y \leq x_2 \\ y \leq x_3 \\ y \geq x_1 + x_2 + x_3 - 2 \\ y \geq 0 \end{array} \tag{22}$$

É facil verificar que para toda atribuição de valores binários para x_1, x_2, x_3, a única atribuição viável para a variável y é o valor de $x_1x_2x_3$, pois a quarta restrição em (22) força y a ser igual a 1 quando todos x_i, $i = 1, 2, 3$ são 1 e as três primeiras restrições mais a última forçam y a ser igual a 0 caso um dos x_i seja iqual a 0.

Em nossa aplicação, em cada iteração resolveremos um problema de otimização, SP, que após o processo de linearização resultará, dado que todas as sentenças são cláusulas, num problema com $(n + m)$ variáveis e sujeito a $(m + \sum_{i=1}^{m} |C_i|)$ restrições e com a adicional imposição de que todas as variáveis são binárias.

Nota: Num problema de minimização as três primeiras restrições são redundantes se y tem um coeficiente positivo na função objetivo, e as duas últimas restrições são redundantes quando y tem um coeficiente negativo na função objetivo.

6 Desenvolvimento do algorítmo

A implementação do código para verificação da consistência das probabilidades atribuídas ao conjunto de sentenças está focada na utilização da linguagem C, principalmente pelo conhecimento prévio herdado das disciplinas que fazem parte do currículo do curso, e na utilização do programa comercial ILOG CPLEX versão 10.0, para resolução dos problemas de otimização sobre variáveis contínuas, PMR, e o de otimização sobre variáveis binárias, SP, que compõem o código do método de resoluçao numérica por geração de colunas.

6.1 Estrutura do código

Algorítmo 1 .

Inicialização

- *Definição do conjunto de variáveis relacionadas ao arquivo de entrada;*
- *Definição do ambiente ILOG CPLEX, dos problemas, PMR e SP, e das variáveis relacionadas;*
- *Leitura do arquivo de entrada omitindo comentários e atribuindo dados ao conjunto de variáveis relacionadas;*
- *Atribuição de dados aos problemas, por meio da geração de arquivos no formato* `LP` *e posterior leitura, sendo realizado o processo de linearização (21) em (16) para criação do SP e o PMR inicia contendo apenas variáveis artificiais.*

Aplicação da variante do método de geração de colunas envolvendo colunas retidas

Passo 1: *Resolve-se o PMR, obtendo-se uma solução que é ótima em termos das variáveis que nele estão incluídas.*

Se *Valor ótimo é $0 \Rightarrow Parar$. E concluírmos consistência das probabilidades atribuídas π.*

Senão *Na ausência de falha, segue-se para o passo seguinte.*

Passo 2: *Procura-se por variáveis, que não fazem parte do atual PMR, que apresentam custo reduzido negativo resolvendo-se o SP (16), linearizado, com os coeficientes da função objetivo, que acompanham as cáusulas, dados pelo atual vetor de valores das variáveis duais do PMR resolvido. Aqui o SP é resolvido sem a definição padrão dos parâmetros que influenciam na sua resolução.*

Se *Valor ótimo é $c^* \geq 0 \Rightarrow$, segue-se para o passo seguinte.*

Senão *Na ausência de falha, segue-se para o Passo 4.*

Passo 3: *Como no Passo 2, exceto pela definição padrão dos parâmetros para resolver o SP.*

Se *Valor ótimo é $c^* \geq 0 \Rightarrow Parar$. E concluírmos inconsistência das probabilidades atribuídas π.*

Senão *Na ausência de falha, realtera-se os parâmetros e segue-se para o passo seguinte.*

Passo 4: *Gera-se a coluna de entrada correspondente a atribuição resultante às variáveis proposicionais do SP, para isso deixa-se de lado as que estão envolvidas com o processo de linearização, insere-se a mesma no PMR e retorna-se ao Passo 1.*

Nota: *Falha constitui parada com informação adicional.*

6.2 Apresentação fragmentada

A seguir realizamos uma visão geral do código, suas funções e da utilização do progama ILOG CPLEX, sendo que o código poderá ser consultado no Apêndice A.

6.2.1 Bibliotecas

Na inclusão de bibliotecas destaca-se a responsável pela biblioteca acessível de funções em C do ILOG CPLEX, `<ilcplex/cplex.h>` , tornando possível criar, otimizar, modificar e interpretar resultados dos problemas de programação matemática.

O ILOG CPLEX apresenta uma arquitetura formada por um núcleo composto pela biblioteca acessível e seu banco de dados, sendo que este último inclui o ambiente de computação, seus canais de comunicação e seus objetos problema. Logo associaremos este núcleo com nossa aplicação ao chamar as rotinas desta biblioteca.

As rotinas utilizadas podem ser enquadradas em algumas categorias:

- de definição / modificação: definem um objeto problema e mudam-o depois de tê-lo criado dentro do banco de dados;
- de otimização: otimizam e geram resultados de um objeto problema;
- de questões sobre o problema: acessam informações sobre o objeto problema criado;

- de leitura de arquivos: movem dados de um sistema de arquivo para dentro da aplicação ou da aplicação para um sistema de arquivo.

6.2.2 Considerações sobre funções

A maioria das funções, da nossa aplicação, devolvem um tipo inteiro e apresentam um único ponto de saída, indicando que algum tipo de falha ocorreu, 1 (um), sendo a mesma informada na tela, ou que ela completou com sucesso todos os passos cabíveis, 0 (zero). O restante delas não devolvem.

Algumas funções que solicitam endereço da variável ficaram caracterizadas pela utilização do sufixo `_p` no nome do parâmetro, e não há variáveis globais, tendo em vista a possível utilização do código em outras aplicações.

Quanto as rotinas do ILOG CPLEX, a maioria delas devolvem um valor inteiro, 0 (zero) indicando sucesso da chamada ou um valor diferente de zero indicando uma falha, sendo cada valor único. Entretanto, algumas rotinas são exceções a esta regra e seus valores devolvidos serão explicitados. É válido lembrar que as informações que acompanham a chamada das rotinas foram omitidas da tela pela definição de um dado parâmetro e que certas rotinas aceitam argumentos opcionais permitindo a passagem do ponteiro `NULL` no lugar do argumento opcional.

6.2.3 Constantes

Neste grupo as constantes criadas na aplicaçao são poucas e não apresentam uma funcionalidade relevante, salvo `EPS` que atribui uma tolerância ao valor ótimo resultante da resolução dos problemas, quando formos tomar decisões de parada.

Já o comportamento do ILOG CPLEX é controlado por uma variedade de parâmetros que são acessíveis e definíveis. Para a biblioteca acessível estas contantes são capitalizadas e iniciadas com `CPX_PARAM_`, e ILOG CPLEX recomenda a utilização destas constantes simbólicas ao invés de seus valores, com a finalidade de garantir portabilidade para versões futuras de ILOG CPLEX.

6.2.4 Licença

ILOG CPLEX roda sobre o controle do seu gerenciador de licença, assim, antes de rodar nossa aplicação que chama ILOG CPLEX devemos estabelecer uma licença válida.

Tivemos um pequeno problema ao rodar os testes e para contorná-lo criamos uma constante `PAUSA`, que é responsável por estabelecer quantos intervalos de "espera" quantos sejam necessários até que a licença seja liberada.

6.2.5 Utilização da biblioteca acessível do ILOG CPLEX

Sintetizando, devemos inicializar o ambiente ILOG CPLEX instanciar os objetos problema e completá-los com os dados. Então podemos chamar os respectivos otimizadores do ILOG CPLEX para otimizar o objeto problema criado, logo após podemos modificá-los e reotimizá-los.

ILOG CPLEX foi desenvolvido para suportar essa seqüência de operações, modificação e reotimização de problemas de programação, eficientemente por reusar a solução viável atual de um problema como ponto de início para a próxima otimização, quando aplicável. Após finalizarmos o uso de ILOG CPLEX, liberamos seus objetos problema e ambiente.

6.2.6 Parte inicial

Aqui são inicializadas todas as variáveis que, direta ou indiretamente, compõem o código, passam-se as informações de início e nome do arquivo de entrada passado à aplicação, cujos detalhes encontram-se descritos em Formato de Arquivo no Apêndice E. Logo após, chama-se a função `ledados` que chama a função `limpa`, para remover possíveis comentários presentes no arquivo de entrada, e posteriormente atribui valores as variáveis relacionadas com tal arquivo. E para finalizar esta parte, um conjunto das variáveis inicializadas são alocadas.

6.2.7 Criação do ambiente do ILOG CPLEX

ILOG CPLEX precisa de certas estruturas de informação internas para operar. Devemos inicializar estas estruturas antes de nossa aplicação chamar alguma rotina da biblioteca do ILOG CPLEX.

A primeira chamada de uma rotina, `CPXopenCPLEX`, é para inicializar um ambiente, sendo que a mesma realiza a verificação da licença e devolve um ponteiro C para o ambiente que é criado. Este ponteiro é passado para todas as demais rotinas da biblioteca de ILOG CPLEX.

6.2.8 Instanciando os objetos problema

Uma vez que inicializamos o ambiente nosso próximo passo é inicializar os objetos problema, chamando `CPXcreateprob`. Tal rotina devolve um ponteiro C para os objetos problemas, `lp` para o PMR e `mip` para o SP, que serão passados para outras rotinas da biblioteca do ILOG CPLEX.

6.2.9 Definindo os dados nos objetos problema

Ao instanciar os objetos problema, eles eram originalmente vazios, ou seja, não haviam restrições, variáveis, função objetivo, etc.

ILOG CPLEX oferece várias formas alternativas de incluir tais dados nos objetos problema e a qual adotamos resume-se em criar um arquivo no formato `LP` e chamar a função `CPXreadcopyprob` para ler tal arquivo e copiar os dados para os objetos problema.

6.2.9.1 Arquivo no formato LP

É um específico arquivo formatado do ILOG CPLEX para entrada de problemas em uma forma algébrica e orientada por linhas. Em outras palavras o formato `LP` permite a entrada de problemas de uma forma simples em termos de suas restrições, que é equivalente a entrada de problemas no otimizador Interativo que acompanha ILOG CPLEX e apresentou-se útil no processo de familiarização com ILOG CPLEX, na identificação de métodos e rotinas que usamos em nossa aplicação e como ferramenta de auxílio na busca de erros.

No caso de nossa aplicação a mesma, através das funções `preenchemip` e `preenchelp`, gera o arquivo problema de uma forma simples e fácil no formato algébrico, sendo que as regras para sua criaçao estão descritas na documentação do ILOG CPLEX, juntamente com as regras de outros vários formatos suportados.

Mas é válido atentar que tal facilidade e redução do código do algorítmo ao ler arquivos no formato `LP` podem aumentar o tempo de execução e o espaço de disco requeridos quando comparados com outros métodos para atribuição de dados ao problema.

Uma outra observação importante a ser feita é que na leitura desses arquivos ILOG CPLEX representa um problema internamente em formato ordenado por colunas. Esta diferença entre a forma que ILOG CPLEX aceita um problema no formato `LP` e a forma que ele armazena internamente, pode ter um impacto sobre o uso da memória e sobre a ordem na qual as variáveis são registradas, pois na medida que ele percorre uma certa linha e encontra uma nova variável, então uma nova coluna é adicionada. Assim, como a primeira linha a ser lida é a função objetivo, basta impor uma ordem conhecida das variáveis nessa linha forçando o aparecimento de todas.

6.2.9.2 Comentários sobre os problemas criados

Na etrada de problemas de programação linear interira mista há um campo, que difere do problema sobre variáveis contínuas, onde indicamos os tipos das variáveis presentes. Assim quando ILOG CPLEX lê o arquivo, automaticamente ele

reconhece o tipo de problema que será tratado. O processo de linearização embutido em `preenchemip`, caracteriza-se pela aplicação dos resultados descritos em (21) para a função (16).

Já a criação do PMR em `preenchelp`, caracteriza-se pelo problema contendo inicialmente apenas variáveis artificiais e a ausência do campo $y \geq 0$ causa sua definição automática, quando ILOG CPLEX lê o arquivo, para apresentar apenas limite inferior em 0.

6.2.10 Otimizando os problemas

ILOG CPLEX afirma que a maioria dos problemas são bem resolvidos pelos otimizadores na forma padrão em que são fornecidos, mas pode ocorrer de querermos exercer uma influência mais direta sobre o processo de solução, sendo que ILOG CPLEX disponibiliza um grande número de estruturas que podem ser de interesse.

Nossa aplicação procurou não alterar tais definições padrão, salvo quando as mesmas possívelmente inibiam um bom desempenho.

6.2.10.1 Otimização sobre variáveis contínuas

Como dissemos acima, a seleção automática padrão para escolha do método, permitiu que ILOG CPLEX determinasse a escolha do algorítmo para otimizar nosso problema, sendo que na maioria dos casos o otimizador escolhido por `CPXlpopt` é o simplex dual.

Uma exceção para esta regra é quando uma base avançada está presente, quando então o otimizador escolhido é o simplex primal, lembrando que isto sempre ocorerá após modificarmos o PMR anteriormente resolvido e tentarmos resolvê-lo novamente.

O otimizador simplex dual do ILOG CPLEX, quando utilizado, faz uso da relação direta que a solução ótima do problema formulado em termos do dual tem com a solução ótima do problema primal, com o lado positivo de ainda devolver a solução em termos do problema primal.

A rotina `CPXlpopt` é a responsável pela otimização do PMR e é usada no interior da função `otimizalp`, e informações sobre a solução do problema são dados por `CPXsolution`.

6.2.10.2 Otimização sobre variáveis binárias

Sua otimização envolve:

- achar uma sucessão de soluções viáveis inteira, ou seja soluções que satisfazem as restrições lineares e as condições de integralidade, enquanto
- também trabalha na direção de uma prova que nenhuma melhor solução viável existe e ainda não foi descoberta.

A rotina `CPXmipopt` é a responsável pela otimização do SP e é usada no interior da função `otimizamip`, tendo alguns dos seus parâmetros controlados pelos arquivos que se encontram no Apêndice D, que são definidos através da chamada de `CPXreadcopyparam`, e informações sobre a solução são obtidas por rotinas inicializadas por `CPXget`.

6.2.10.3 Pré-processamento

Na definição padrão ILOG CPLEX pré-processa problemas simplificando as restrições, reduzindo o tamanho do problema e eliminando redundâncias, com o objetivo de aumentar a velocidade durante o processo de obtenção da solução, mesmo incluindo o tempo gasto neste processo e com a vantagem de devolver a solução do problema pré-processado em termos da formulação do problema original.

Durante este processo constrói-se um novo problema e armazena-se informação suficiente para converter a solução do problema para o problema original.

Sobre o problema PMR, dado que a cada iteração estaremos partindo de uma base avançada, o pré-processamento não afeta o problema modificado pela inclusão da nova coluna, ou automaticamente é inviabilizado pelos fatos de base avançada e natureza da modificação.

Já sobre o SP o pré-processamento foi executado em cada iteração com sua definição padrão e foi de grande utilidade quanto ao tempo de resposta.

6.2.11 Mudando um objeto problema

Cabe a nós realizarmos modificações nos objetos problema, no `mip` por alterar os coeficientes da função objetivo com o vetor de variáveis duais resultante da atual iteração do `lp`, que é realizada no interior da função `otimizamip` antes de otimizá-lo, e então, após análise da resposta do `mip`, possivelmente realizarmos a inclusão de uma nova coluna no `lp` através da função `atualizalp`.

Quando fazemos tais mudanças devemos manter ILOG CPLEX informado sobre as modificações para que então ILOG CPLEX possa eficientemente reotimizar o objeto problema alterado. Para isso basta usarmos as rotinas de modificação de problema da biblioteca de ILOG CPLEX. Rotinas com esse propósito e que foram utilizadas em nossa aplicação apresentam nomes como `CPXchgobj`, para mudar os

coeficientes da função objetivo de um objeto existente no modelo, ou `CPXaddcols` para adicionar nova coluna ao modelo.

Por exemplo se uma dada alteração é realizada, então ILOG CPLEX iniciará a próxima otimização da base ótima anterior, se for cabível. Se aquela base é ainda ótima com respeito a alteração, então ILOG CPLEX devolverá aquela informação sem precisar refatorar a base.

6.2.11.1 Tamanho do problema e alocação de memória

Não há necessidade de alocarmos espaço extra antecipando uma futura modificação do problema. Limites sobre o tamanho são determinados pelos recursos do sistema e a subjacente implementação da função de sistema `malloc` e não por limites artificiais no ILOG CPLEX.

Quando modificamos um objeto problema através da chamada das rotinas de modificação da biblioteca do ILOG CPLEX, ele automaticamente manuseia as alocações de memória para acomodar o aumento de tamanho do problema.

6.2.12 Critério de parada

O laço presente no `main` iniciando o método de geração de colunas caracteriza-se por três tipos de parada:

- O resultado da otimização do PMR por `otimizalp` apresentou valor ótimo 0, implicando problema viável, sendo tal informação enviada para tela e no arquivo de saída como RSL S;
- O resultado da otimização do SP por `otimizamip` apresentou valor ótimo $c^* \geq 0$, com a definição padrão de seus parâmetros, implicando que não há variável fora da base com custo reduzido negativo, sendo a informação de problema inviável enviada para tela e no arquivo de saída como RSL N;
- Houve falha em algum ponto que será informado na tela e enviado para o arquivo de saída como RSL F.

Em todos os casos fornece-se informações adicionais que encontram-se descritas no Apêndice B, e o arquivo de saída é finalizado.

6.2.13 Critério para geração de colunas

Novamente, no laço presente no `main` temos o seguinte critério para geração e inclusão de uma nova coluna no PMR:

- O resultado da otimização do SP por `otimizamip` apresentou valor ótimo $c^* \leq 0$, com a definição padrão ou não de seus parâmetros, implicando que há variável fora da base com custo reduzido negativo, cuja coluna é definida como função da atribuição resultante às variáveis lógicas, que pode entrar na base após a chamada de `coluna`, para geração de tal coluna, e de `atualizalp`, para inclusão da mesma no PMR.

Lembrando que após tal inclusão vamos para o Passo 1 do algorítmo apresentado em 6.1.

6.2.14 Liberação da memória alocada

Com o uso das rotinas `CPXfreeprob` destruímos os objetos problema quando nossa aplicação não precisa mais dos mesmos, e depois que todas as chamadas para a biblioteca do ILOG CPLEX foram completadas liberamos o ambiente por chamar a rotina `CPXcloseCPLEX`, sendo que a mesma estabelece ao ILOG CPLEX que:

- todas as chamadas para a biblioteca do ILOG CPLEX foram completadas;
- ILOG CPLEX deve liberar a memória alocada para a criação de seu ambiente;
- a aplicação abandonou a licença do ILOG CPLEX para esta execução, tornando-a avaliável para o próximo usuário.

Finalmente liberamos o restante das variáveis alocadas chamando a função `desaloca`.

6.3 Compilação e vínculos

Tais informações podem ser encontradas em Leia-me que pode ser conferido no Apêndice B.

7 Análise do comportamento

Para analisarmos o comportamento do código geramos testes que foram executados sobre um Intel Xeon 2.8Ghz com 4MB de cache L2 e 4GB de RAM, onde cada arquivo teste gerado seguia o seguinte padrão:

- As cláusulas foram geradas da seguinte forma:

  Passo 1: Definia-se um conjunto $\mathcal{X}$ com n variáveis;

Passo 2: Para cada k-cláusula, sendo k o número de variáveis que compõem a cláusula, selecionava-se sem reposição k variáveis entre as n de $\mathcal{X}$. Aleatoriamente;

Passo 3: Para criar uma nova cláusula, reposicionava-se as variáveis previamente selecionadas e repetia-se o Passo 2.

- As probabilidades foram geradas de duas formas:

  1. Na primeira para cada cláusula foi gerado um número aleatório entre 0.1 e 0.9.
  2. Na segunda, supondo que havia uma distribuição uniforme sobre os mundos possíveis, se realizassemos uma contagem dos mundos em que C_i era satisfeita e multiplicassemos pela probabilidade individual do mundo, neste caso $p_i = 1/2^n \ \forall i$, teríamos a seguinte expressão que fornece o valor para π_i e que depende apenas da cardinalidade de C_i:

$$\pi_i = \sum_{j=1}^{|C_i|} \frac{2^{n-j}}{2^n} = \sum_{j=1}^{|C_i|} \frac{1}{2^j} \tag{23}$$

  assim, era garantido que o problema seria consistente, visto que garantíamos a existência de um vetor p que atenderia as restrições do probleblema.

O comportamento dos resultados, inicialmente obtidos, mostrou-nos que ILOG CPLEX encontrava uma boa solução inteira rapidamente, mas precisava verificar muitos outros nós da arvore de busca para provar otimalidade, e isto levou-nos a três seguintes definições de parâmetros:

1. Nos testes, cujos resultados encontram-se na Tabela 2, alteramos os valores dos parâmetros de “gap“ e “gap absoluto“ para 1, mas não alteramos o parâmetro que cuida da aplicação de heurística, do ILOG CPLEX, em certos nós para resolução do SP. Dessa forma não tínhamos prova sobre otimalidade, e sim, apenas um ponto viável era obtido tão logo fosse encontrado.

2. Já nos testes, cujos resultados encontram-se na Tabela 3, realizamos a mesma alteração para “gap“ e “gap absoluto“, salvo que desta vez, ILOG CPLEX resolveu o SP aplicando seu método de heurística em todas as iterações.

3. Quanto aos testes, cujos resultados encontram-se na Tabela 4, foram os únicos que foram gerados com π aleatório e neste caso as definições dos parâmetros que utilizamos foram as mesmas do item 1.

Nos três casos acima os problemas eram os mesmos, no que diz respeito às cláusulas, e a coluna ARQ das tabelas resume o tamanho do problema, por exemplo 14–50–1 define um problema com 14 variáveis 50 cláusulas e relativo ao primeiro arquivo deste tipo. RSL indica se a avaliação do problema resultou em consistente S ou não N.

Da análise dos resultados temos que nas três tabelas o tempo total de processamento de todos os PMR (TMPC) não é significante, pois o tempo total do método de geração de colunas recaiu sobre o tempo total de processamento de todos os SP (TMPI), cujo valor apresentou-se sempre inferior para os problemas com π aleatório, sendo refletido pelo número total de iterações para resolvê-los (ITEI). Este último valor apresentou-se inferior para os problemas do item 2 acima, mas não se refletiu em TMPI, quando comparamos as tabelas referentes aos itens 1 e 2.

Nos problemas do item 3 observou-se que o número de iterações do método de geração de colunas (ITGC) apresentou-se, no máximo, ligeiramente superior a metade do número de cláusulas que haviam no problema, em contraste com os resultados obtidos para os itens 1 e 2 em que ele assumiu um valor ligeiramente superior ao número de cláusulas.

Quanto ao número total de iterações de todos os PMR (ITEC) o mesmo apresentou-se com uma média de 17,3 vezes o ITGC, para a Tabela 2, 18,2 para a Tabela 3 e 1,5 para a Tabela 4, mas, como já observado, tais iterações não consumiram tempo significativo.

Já o número de vezes que o SP precisou ser resolvido com a definição padrão dos parâmetros (ITEX), este apresentou-se alto com π aleatório, isto se lembrarmos que ITGC foi baixo para este caso, salvo onde ITEX esteve próximo de 1, pois isto reflete a iteração para prova de inconsistência. ITEX também nos leva a refletir sobre os valores que alteramos para resolução aproximada do SP, pois se o seu valor for alto significa que a aproximação não é boa o bastante e temos que recorrer a solução exata do SP.

8 Possíveis extensões

Várias extensões são revistas em Hansen e Jaumard (1996), uma delas, referida como problema na forma de otimização, pelos autores anteriores, e como problema de implicação por Nilsson (1996), e que não se apresenta distante do trabalho que realizamos seria a seguinte:

- Assumindo que a atribuição de probabilidades π é consistente para as cláusulas em $\mathcal{C}$, então nós devemos responder a seguinte questão: qual é a probabilidade

mínima (máxima) que podemos atribuir a uma sentença adicional dado que as cláusulas em $\mathcal{C}$ tem a probabilidade π?

Esta questão pode ser respondida por minimizar (maximizar) o problema P_{PSAT} (1), exceto que na função objetivo deverá constar os coeficientes que avaliam a cláusula adicional, semelhante ao que fazemos para obter uma linha da matriz A associada uma dada cláusula.

Outras extensões, que não são imediatas, envolveriam o tratamento de sentenças na forma normal conjuntiva (FNC), isto é, sentenças formadas por conjunções de cláusulas através do operador $\wedge$, pois sentenças genéricas podem ser reduzidas à FNC (Nilsson, 1998). Casos mais realísticos não apresentam uma atribuição de probabilidade π consistindo de valores pontuais, nesses casos temos intervalos $[\underline{\pi_i}, \overline{\pi_i}]$ para a validade da cláusula C_i. Esta ligeira extensão de $PSAT$, também pode ser formulada como um problema de programação linear, com mais linhas do que a formulação de P_{PSAT} (1), e é obtida do anterior através de simples modificações, podendo ser conferido em Jaumard, Hansen, Aragão, Chauny e Perron (2000). E para finalizarmos, uma extensão adicional é para o caso onde temos probabilidades condicionais. Por exemplo, suponha que $\pi_{i,j}$ é a probabilidade que a cláusula C_i é verdadeira dado que a cláusula C_j é verdadeira, e novamente este último problema é similar a P_{PSAT} (1) com o mesmo número de linhas e variáveis (Jaumard et al., 1991).

E no intuito de possivelmente obter uma melhora quanto ao tempo de execução, pode-se realizar a inclusão de outros parâmetros, nos arquivos de parâmetros, e acompanhar seus efeitos na presença ou ausência de outros e na faixa onde podem ser definidos.

9 Conclusão

Neste trabalho apresentou-se o problema de satisfatibilidade probabilística formulado como um problema de programação linear (Nilsson, 1986) e realizou-se a implementação do método numérico de geração de colunas, como proposto por Jaumard et al. (1991).

Para termos um ligeiro conhecimento do comportamento do programa, realizaram-se testes computacionais sobre um pequeno grupo de instâncias, em que num caso tínhamos conhecimento prévio da resposta e no outro não. Assim, seu desempenho apresentou-se bastante diferente para as instâncias estudadas, pois o problema mostrou-se bem mais fácil quando não tínhamos conhecimento da resposta por ter gerado as probabilidades das sentenças aleatoriamente.

Na resolução do problema de progamação inteira, associado ao método de geração de colunas, mostrou-se que ILOG CPLEX (2006) apresenta um conjunto de parâmetros que podem ser manipulados visando melhorias nos tempos de resolução das instâncias, mas uma exploração mais detalhada destas potencialidades do ILOG CPLEX não foram nesta fase testadas e podem estar relacionadas com o prosseguimento deste trabalho, assim como a implementação das extensões do problema de satisfatibilidade probabilística e a realização de testes computacionais mais exaustivos procurando-se abranger uma gama de instâncias mais heterogênea.

Referências

[1] ANDERSEN, K.A., HOOKER, J.N. 1996. Determining Lower and Upper Bounds on Probabilities of Atomic Propositions in Sets of Logical Formulas Represented by Digraphs. *Annals of Operations Research* 65 1–20

[2] ANDERSEN, K.A., PRETOLANI, D. 2001. Easy Cases of Probabilistic Satisfiability. *Annals of Mathematics and Artificial Intelligence* 33 (1): 69–91

[3] BERTSIMAS, D., TSITSIKLIS, J.N. 1997. Introduction to Linear Optimization. Athena Scientific, Belmont, Mass.

[4] BOROS E., HAMMER, P. 2002. Pseudo-Boolean Optimization. *Discrete Applied. Mathematics* 123: 155 – 225

[5] GEORGAKOPOULOS, G., KAVVADIAS, D., PAPADIMITRIOU, C.H. 1988. Probabilistic Satisfiability. *Journal of Complexity* 4 (1): 1–11

[6] HANSEN, P.,JAUMARD, B. 1996. Probabilistic Satisfiability. Technical Report G-96-31, *Les Cachiers du GERAD*, École Polytechnique de Montreal, 69 p.

[7] HANSEN, P., PERRON, S. 2004. Merging the Local and Global Approaches to Probabilistic Satisfiability. Technical Report G-04-31, *Les Cachiers du GERAD*, 13 p.

[8] ILOG 2006. CPLEX versão 10.0. www.ilog.com/products/cplex

[9] JAUMARD, B., HANSEN, P., ARAGÃO, M.P. 1991. Column Generation Methods for Probabilistic Logic. *ORSA Journal on Computing* 3: 135–148

[10] JAUMARD, B., HANSEN, P., ARAGÃO, M.P. 1995. Boole's conditions of possible experience and reasoning under uncertainty. *Discrete Applied Mathematics* 60 (1-3): 181–193

[11] JAUMARD, B., HANSEN, P., ARAGÃO, M.P., CHAUNY, F., PERRON, S. 2000. Probabilistic satisfiability with imprecise probabilities. *Int. J. Approx. Reasoning* 24 (2-3): 171–189

[12] JOVANOVIĆ, D., MLADENOVIĆ, N., OGNJANOVIĆ, Z. 2006. Variable Neighborhood Search for the Probabilistic Satisfiability Problem. Technical Report G-06-49, *Les Cahiers du GERAD*, 19 p.

[13] NILSSON, N.J. 1986. Probabilistic logic. *Artificial Intelligence* 28 (1): 71–88

[14] NILSSON, N.J. 1998. Artificial Intelligence: A New Synthesis. Morgan Kaufmann Publishers, San Francisco

[15] PRETOLANI, D. 2005. Probability Logic and Optimization SAT: The PSAT and CPA Models. *Annals of Mathematics and Artificial Intelligence* 43 (1-4): 211–221

[16] RUSSELL, S., NORVIG P. 2003. Artificial Intelligence: A Modern Approach. Prentice-Hall, Englewood Cliffs, NJ

Apêndices

A Código fonte em C

```
/* Arquivo: geracol.c
// Versao 1.0
//
// Autor: Paulo Sergio de Souza Andrade                n.usp: 4963109
// Janeiro de 2007
//
// geracol.c
// Aplicacao para verificacao da consistencia das probabilidades
// atribuidas a um conjunto de sentencas implementando o metodo de
// geracao de colunas e utilizando o programa de otimizacao ILOG CPLEX
// para resolucao dos problemas envolvidos com o metodo.
// Recomenda-se a leitura de formatos.ps, para esclarecimento do
// argumento que deve ser passado para o programa, e de leiame.txt,
// para instrucoes de compilacao e notas sobre saidas.
////////////////////////////////////////////////////////////////// */

/* Bibliotecas Utilizadas
////////////////////////////////////////////////////////////////// */
// a primeira inclui stdio.h juntamente com cplex.h
#include </usr/ilog/cplex100/include/ilcplex/cplex.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include <time.h>

/* Constantes
////////////////////////////////////////////////////////////////// */
# define TRUE 1       // para utilizacao em lacos
# define MAXLIN 10    // quebra de linha
# define EPS 1.0e-6   // erro admitido
# define PAUSA 1.0e+5 // para aguardar liberacao da licenca

/* Prototipos de Funcoes
////////////////////////////////////////////////////////////////// */
int otimizalp (CPXCENVptr env, CPXLPptr lp, double *objval_p,
               double *x, double *pi);
int otimizamip (CPXCENVptr env, CPXLPptr mip, int numcnf, int numvar,
```

```
                double *x, int *indices, double *pi, double *objval_p,
                int *mundo);
int atualizalp (CPXCENVptr env, CPXLPptr lp, int numcnf, int numvar,
                double coefobj, double *colentr, int *indlin,
                int *mundo, char **nomecol);
int preenchelp (CPXENVptr env, CPXLPptr lp, int numcnf, double *prob);
int preenchemip (CPXENVptr env, CPXLPptr mip, int numvar, int numcnf,
                 int **CLA, int *tamcla, int **CNF, int *tamcnf);
int ledados (char *nomearq, int *numcla_p, int *numvar_p,
             int *numcnf_p, int ***CLA_p, int **tamcla_p,
             int ***CNF_p, int **tamcnf_p,double **prob_p);
FILE *limpa (FILE *in, FILE *arq);
void coluna (int *mundo, double *colentr, int numcnf, int **CLA,
             int *tamcla, int **CNF, int *tamcnf);
void desaloca (char **ptr);

/* Programa Propriamente Dito
////////////////////////////////////////////////////////////////// */
int main (int numargs, char **v) {
  int status = 0;
  int i;

  /* Dados associados ao arquivo de entrada */
  int numcla;          // numero de clausulas,
  int numvar;          // de variaveis e
  int numcnf;          // de expressoes com probabilidade
  int **CLA = NULL;    // CLA[i][] contera a i-esima clausula
  int *tamcla = NULL;  // tamcla[i] sera o numero de variaveis
                       // que compoem a i-esima clausula
  int **CNF = NULL;    // CNF[i][] contera a i-esima CNF
  int *tamcnf = NULL;  // tamcnf[i] sera o numero de clausulas
                       // que compoem a i-esima CNF
  double *prob = NULL; // prob[i] sera a probabilidade da i-esima CNF

  /* Cria ponteiros para ambiente e problemas do cplex */
  CPXENVptr env = NULL; // ambiente
  CPXLPptr lp = NULL;   // problema de programacao continua
  CPXLPptr mip = NULL;  // problema de programacao inteira

  /* Vetores e variaveis relacionadas ao problema PMR (= lp) */
  double objval;      // valor otimo do lp
  double *x = NULL;   // vetor solucao de valores do primal
  double *pi = NULL;  // vetor de valores do dual
  int itlp = 0;       // numero de iteracoes para resolver lp
  // para coluna de entrada
  int *indlin = NULL;     // indices das linhas
  double coefobj = 0;     // coeficiente da objetiva
```

```
double *colentr = NULL; // coluna de entrada
char **nomecol = NULL;  // nome da coluna de entrada

/* Vetores e variaveis relacionadas ao subproblema (= mip) */
double objvalmip;    // valor otimo
double *xmip = NULL; // vetor solucao
int itmip = 0;       // numero de iteracoes para resolver mip
int *mundo = NULL;   // atribuicao dada as variaveis {0, 1}
int *indices = NULL; // indices para substituir objetiva

int itgc = 0; // numero de iteracoes do metodo de geracao de colunas

/* Variaveis para controle do tempo aproximado de execucao */
double fim;
double inicio;
double tempototallp = 0;
double tempototalmip = 0;
time_t t;
char *horario;

/* Para arquivo de escrita */
FILE *saida = NULL;
char *arquivo = "saida.txt";

/* Informacoes iniciais para arquivo de saida */
saida = fopen (arquivo, "a");
if (saida == NULL) {
  printf ("** ERRO ** Arquivo de escrita nao encontrado.\n");
  status = 1;
  goto FINAL;
}
t = time(NULL);
horario = asctime (localtime (&t));
fprintf (saida,"\nINICIO \nHINI %sARQ %s", horario, v[1]);

/* Verifica argumentos da linha de comando */
if (numargs != 2) {
  printf ("** ERRO ** Lista de argumentos errada.\n");
  goto FINAL;
}

/* Le arquivo de entrada */
status = ledados (v[1], &numcla, &numvar, &numcnf, &CLA,
                  &tamcla, &CNF, &tamcnf, &prob);
if (status) goto FINAL;
// dado que o problema apresenta uma restricao adicional
// 1*p = 1, que correspondera a ultima linha da matriz
```

```
// e pode ser vista como uma tautologia
numcnf++;

/* Alocacoes */
x = (double *) malloc (numcnf*sizeof (double));
pi = (double *) malloc (numcnf*sizeof (double));
indlin = (int *) malloc (numcnf*sizeof (int));
colentr = (double *) malloc (numcnf*sizeof (double));
nomecol = (char **) malloc (sizeof (char *));
xmip = (double *) malloc ((numvar+numcnf-1)*sizeof (double));
mundo = (int *) malloc (numvar*sizeof (int));
indices = (int *) malloc ((numcnf-1)*sizeof (int));
if (x == NULL || pi == NULL || indlin == NULL || colentr == NULL ||
    nomecol == NULL || xmip == NULL || mundo == NULL ||
    indices == NULL) {
  printf ("** ERRO ** Malloc devolveu NULL!\n");
  status = 1;
  goto FINAL;
}
nomecol[0] = (char *) malloc (sizeof (char));
if (nomecol[0] == NULL) {
  printf ("** ERRO ** Malloc devolveu NULL!\n");
  status = 1;
  goto FINAL;
}

/* Inicializa ambiente cplex */
while (TRUE) {
  env = CPXopenCPLEX (&status);
  if (status == 0) break;
  for (i = 0; i < PAUSA; i++);
}

/* Desliga saida do CPLEX para tela */
status = CPXsetintparam (env, CPX_PARAM_SCRIND, CPX_OFF);
if (status) {
  printf ("Falha ao desligar saida para tela, erro %d.\n", status);
  goto FINAL;
}

/* Cria os objetos problema */
lp = CPXcreateprob (env, &status, "PMR");
if (lp == NULL) {
  printf ("Falha ao criar LP.\n");
  goto FINAL;
}
```

```
mip = CPXcreateprob (env, &status, "Subprob");
if (mip == NULL) {
  printf ("Falhou para criar MIP.\n");
  goto FINAL;
}

/* Preenche dados iniciais dos problemas */
status = preenchelp (env, lp, numcnf, prob) ;
if (status) {
  printf ("Falhou para preencher LP.\n");
  goto FINAL;
}

status = preenchemip (env, mip, numvar, numcnf-1, CLA,
                      tamcla, CNF, tamcnf);
if (status) {
  printf ("Falhou para preencher MIP.\n");
  goto FINAL;
}

/* Parte relativa a fase0 do simplex */
// indices dos pesos para mip alterar objetiva
for (i = 0; i < numcnf-1; i++)
  indices[i] = numvar+i;
// indices das linhas para adicionar coluna no lp
for (i = 0; i < numcnf; i++)
  indlin[i] = i;

/* Altera parametro para resolver mip aproximadamente */
status = CPXreadcopyparam (env, "altparam.prm");
if (status) {
  printf ("Falha ao mudar parametros, erro %d.\n", status);
  goto FINAL;
}

/* Metodo de geracao de colunas para decisao da consistencia
// por determinar existencia de ponto viavel */
for (i = 1; TRUE; i++) {
  /* Otimiza LP e obtem solucao */
  inicio = (double) clock ();
  status = otimizalp (env, lp, &objval, x, pi);
  fim = (double) clock ();
  tempototallp += fim - inicio;
  if (status) {
    printf ("Falhou para otimizar LP.\n");
    goto FINAL;
  }
```

```
// obtem numero de iteracoes para resolver LP
itlp += CPXgetitcnt (env, lp);

/* Verifica se atingiu 0, implicando problema viavel */
if (objval < EPS) {
  printf ("Problema e viavel.\n");
  break;
}

/* Otimiza MIP e obtem solucao */
inicio = (double) clock ();
status = otimizamip (env, mip, numcnf, numvar, xmip, indices,
                     pi, &objvalmip, mundo);
fim = (double) clock ();
tempototalmip += fim - inicio;
if (status) {
  printf ("Falhou para otimizar MIP.\n");
  goto FINAL;
}
// obtem numero de iteracoes para resolver MIP
itmip += CPXgetmipitcnt (env, mip);

/* Verifica se deve resolver exatamente, para obter uma
// prova de que nao ha mais variavel com custo reduzido
// negativo dado pela aproximacao */
if (objvalmip > -EPS) {
  itgc++;
  /* Altera parametro para definicao padrao */
  status = CPXreadcopyparam (env, "norparam.prm");
  if (status) {
    printf ("Falha ao redefinir parametros, erro %d.\n", status);
    goto FINAL;
  }

  /* Otimiza MIP e obtem solucao */
  inicio = (double) clock ();
  status = otimizamip (env, mip, numcnf, numvar, xmip, indices,
                       pi, &objvalmip, mundo);
  fim = (double) clock ();
  tempototalmip += fim - inicio;
  if (status) {
    printf ("Falhou para otimizar MIP.\n");
    goto FINAL;
  }
  // obtem numero de iteracoes para resolver MIP
  itmip += CPXgetmipitcnt (env, mip);
```

```
    /* Verifica se nao ha mais variavel com custo reduzido
    // negativo, implicando problema inviavel */
    if (objvalmip > -EPS) {
      printf ("Problema e inviavel.\n");
      fprintf (saida,"\nRSL N");
      goto FINAL;
    }

    /* Realtera parametro para resolver mip aproximadamente */
    status = CPXreadcopyparam(env, "altparam.prm");
    if (status) {
      printf ("Falha ao mudar parametros, erro %d.\n",status);
      goto FINAL;
    }
  }

  /* Gera coluna de entrada */
  coluna (mundo, colentr, numcnf-1, CLA, tamcla, CNF, tamcnf);

  /* Insere coluna no LP */
  status = atualizalp (env, lp, numcnf, numvar, coefobj,
                       colentr, indlin, mundo, nomecol);
  if (status) {
    printf ("Falhou para atualizar simplex.\n");
    goto FINAL;
  }
}

/* Termino da fase0 com sucesso => problema viavel */
fprintf (saida,"\nRSL S");

FINAL:

////////////////////////////////////////////////////////
/* Curiosidade
status = status = CPXwriteprob (env, lp, "proPMR.lp", NULL);
if (status)
  printf ("Falhou para obter arquivo com problema LP.\n");
status = CPXmbasewrite (env, lp, "basePMR.bas");
if (status)
  printf ("Falhou para obter arquivo com base.\n");
status = CPXsolwrite (env, lp, "solPMR.sol");
if (status)
  printf ("Falhou para obter arquivo com solucao.\n");
////////////////////////////////////////////////////////*/

/* Interrupcao da fase0 por algum motivo, aconselhavel
```

```
// verificar informacao enviada para tela */
if (status)
  fprintf (saida,"\nRSL F");

/* Envia informacoes finais ao arquivo de saida */
fprintf (saida, "\nTMPC %e", tempototallp/CLOCKS_PER_SEC);
fprintf (saida, "\nTMPI %e", tempototalmip/CLOCKS_PER_SEC);
fprintf (saida, "\nITEC %d", itlp);
fprintf (saida, "\nITEI %d", itmip);
fprintf (saida, "\nITCG %d", i);
fprintf (saida, "\nITEX %d", itgc);
/* Desaloca problemas */
if (lp != NULL) {
  status = CPXfreeprob (env, &lp);
  if (status) {
    printf ("CPXfreeprob falhou, codigo de erro %d.\n", status);
  }
}

if (mip != NULL) {
  status = CPXfreeprob (env, &mip);
  if (status) {
    printf ("CPXfreeprob falhou, codigo de erro %d.\n", status);
  }
}

/* Desaloca ambiente */
if (env != NULL) {
  status = CPXcloseCPLEX (&env);
  if (status) {
    char  errmsg[1024];
    printf ("Falha ao fechar o ambiente CPLEX.\n");
    CPXgeterrorstring (env, status, errmsg);
    printf ("%s", errmsg);
  }
}

/* Desaloca vetores e matrizes de dados */
desaloca ((char **) &x);
desaloca ((char **) &pi);
desaloca ((char **) &indlin);
desaloca ((char **) &colentr);
desaloca ((char **) &nomecol[0]);
desaloca ((char **) &nomecol);
desaloca ((char **) &xmip);
desaloca ((char **) &mundo);
desaloca ((char **) &indices);
```

```
  if (CLA != NULL) {
    for (i = 0; i < numcla; i++)
      desaloca ((char **) &CLA[i]);
  }
  desaloca ((char **) &CLA);
  if (CNF != NULL) {
    for (i = 0; i < numcnf; i++)
      desaloca ((char **) &CNF[i]);
  }
  desaloca ((char **) &CNF);
  desaloca ((char **) &tamcla);
  desaloca ((char **) &tamcnf);
  desaloca ((char **) &prob);

  /* Informacoes finais para arquivo de saida */
  t = time(NULL);
  horario = asctime(localtime(&t));
  fprintf (saida,"\nHFIM %sFIM", horario);

  fclose (saida);

  return EXIT_SUCCESS;
}

/* Otimiza lp, fornecendo o valor otimo da objetiva em objval_P, o
// vetor com os valores das variaveis que compoem a base x, e o vetor
// de valores do dual pi, do problema lp no ambiente env.
// Devolve 1 se uma falha ocorre e 0 caso contrario.
////////////////////////////////////////////////////////////////// */
int otimizalp (CPXCENVptr env, CPXLPptr lp, double *objval_p,
               double *x, double *pi) {
  int status = 0;
  int solstat; // status da solucao

  /* Otimiza LP e obtem solucao */
  status = CPXlpopt (env, lp);
  if ( status ) {
    printf ("Falhou para otimizar LP.\n");
    goto FINAL;
  }
  /* OBS: Obter numero de colunas e alocar x caso
  // haja interesse em obte-lo */
  status = CPXsolution (env, lp, &solstat, objval_p,
                        NULL, pi, NULL, NULL);
  if ( status ) {
    printf ("Falhou para obter solucao.\n");
    goto FINAL;
```

```
  }
  // interpreta status da solucao
  if (solstat == CPX_STAT_OPTIMAL || solstat == CPX_STAT_NUM_BEST);
  else {
    printf ("Solucao otima nao esta avaliada.\n");
    status = 1;
    goto FINAL;
  }

  FINAL:
  return status;
}

/* Otimiza subproblema mip, porem antes altera coeficientes da
// objetiva com os vetores pi[0..nuncnf-1] e indices[0..numcnf-2],
// e fornece o valor otimo da objetiva em objval_p, a solucao otima
// titulada no vetor x[0..(numvar+numcnf)-1] e a que define o mundo
// possivel no vetor  mundo[0..numvar-1] presentes no ambiente env.
// Devolve 1 se uma falha ocorre e 0 caso contrario.
////////////////////////////////////////////////////////////////// */
int otimizamip (CPXCENVptr env, CPXLPptr mip, int numcnf, int numvar,
                double *x, int *indices, double *pi, double *objval_p,
                int *mundo) {
  int status = 0;
  int solstat;    // status da solucao
  int numcol;     // numero de colunas no problema
  double cte = 0;
  int i;

  /* Obtem constante */
  for (i = 0; i < numcnf; i++)
    cte += (-1)*pi[i];

  /* Altera coeficientes da objetiva */
  status = CPXchgobj (env, mip, numcnf-1, indices, pi);
  if (status) {
    printf ("Falhou para mudar MIP.\n");
    goto FINAL;
  }

  /* Otimiza o problema e obtem solucao */
  status = CPXmipopt (env, mip);
  if (status) {
    printf ("Falhou para otimizar MIP.\n");
    goto FINAL;
  }
  solstat = CPXgetstat (env, mip);
```

```
  // interpreta status da solucao
  if (solstat == CPXMIP_OPTIMAL || solstat == CPXMIP_OPTIMAL_TOL ||
      solstat == CPXMIP_SOL_LIM);
  else {
    printf ("Solucao otima nao esta avaliada.\n");
    status = 1;
    goto FINAL;
  }
  status = CPXgetobjval (env, mip, objval_p);
  if (status) {
    printf ("MIP nao forneceu valor objetivo.  Exiting...\n");
    goto FINAL;
  }
  // soma constante que havia ficado de fora
  *objval_p += cte;

  // obtem numero de colunas, interessante quando nao sabemos!
  numcol = CPXgetnumcols (env, mip);
  status = CPXgetx (env, mip, x, 0, numcol-1);
  if (status) {
    printf ("Falhou para obter solucao inteira.\n");
    goto FINAL;
  }

  /* Obtem mundo possivel resultante */
  for (i = 0; i < numvar; i++)
       mundo[i] = (int) x[i];

  FINAL:
  return status;
}

/* Insere coluna no problema lp, inclui coeficiente da objetiva
// coefobj para nova variavel, o vetor coluna de entrada
// colentr[0..numcnf-1] com o auxilio do vetor com o indice das linhas
// indlin[0..numcnf-1] e do vetor mundo[0..numvar-1] para compor o
// nome da nova variavel em nomecol do problema lp no ambiente env.
// Devolve 1 se uma falha ocorre e 0 caso contrario.
////////////////////////////////////////////////////////////////// */
int atualizalp (CPXCENVptr env, CPXLPptr lp, int numcnf, int numvar,
                double coefobj, double *colentr, int *indlin,
                int *mundo, char **nomecol) {
  int status = 0;
  int indice = 0; // para compor nome da variavel que entra no lp
  int indcol = 0; // auxiliar na inclusao da nova coluna
  int i;
```

```
  /* Obtem indice da nova variavel na base */
  for (i = 0; i < numvar; i++)
    indice += (int) mundo[i]*pow(2,i);
  indice++;

  /* Compoe nome da nova variavel no lp e insere coluna */
  sprintf (nomecol[0], "x%d", indice);
  status = CPXaddcols (env, lp, 1, numcnf, &coefobj, &indcol,
              indlin, colentr, NULL, NULL , nomecol);
  if ( status ) {
    printf ("Falhou para inserir coluna.\n");
    goto FINAL;
  }

  FINAL:
  return status;
}

/* Preenche os vetores e variaveis associados ao arquivo de entrada
// cujo nome e nomearq, definindo numero de clausulas numcla_p, numero
// de variaveis logicas numvar_p numero de sentencas numcnf_p, a
// matriz de clausulas CLA_p e o correspondente tamanho de cada
// clausula no vetor tamcla_p, a matriz de sentencas CNF_p e os
// vetores correspondentes ao numero de clausulas na sentenca tamcnf_p
// e a probabilidade associada prob_p.
// Devolve 1 se uma falha ocorre e 0 caso contrario.
// OBS: o arquivo deve conter os dados informados em formato.ps, salvo
//      que cada sentenca apresenta uma unica clausula associada.
////////////////////////////////////////////////////////////////// */
int ledados (char *nomearq, int *numcla_p, int *numvar_p,
             int *numcnf_p, int ***CLA_p, int **tamcla_p,
             int ***CNF_p, int **tamcnf_p, double **prob_p) {
  FILE *arq = NULL, *in = NULL; // para arquivo de entrada e
                                // o sem comentarios
  char barraene;                // para verificar final de linha
  char *arquivo = "limpo.txt";
  int i = 0, j, aux;
  int status = 0;

  arq = fopen (arquivo, "w+"); // escrita e leitura
  in = fopen (nomearq, "r");
  if (arq == NULL || in == NULL) {
    printf ("** ERRO ** Arquivo de entrada nao encontrado.\n");
    status = 1;
    goto FINAL;
  }
```

```
/* Elimina comentarios */
arq = limpa (in, arq);

/* Leitura dos dados */
fscanf (arq, "%d %d %d", numvar_p, numcla_p, numcnf_p);
// alocacao das variaveis relacionadas ao arquivo de entrada
*CLA_p = (int **) malloc (*numcla_p*sizeof (int *));
*CNF_p = (int **) malloc (*numcnf_p*sizeof (int *));
*tamcla_p = (int *) malloc (*numcla_p*sizeof (int));
*tamcnf_p = (int *) malloc (*numcnf_p*sizeof (int));
*prob_p = (double *) malloc (*numcnf_p*sizeof (double));
if (*CLA_p == NULL ||*CNF_p == NULL || *tamcla_p == NULL ||
    *tamcnf_p == NULL || *prob_p == NULL) {
  printf ("** ERRO ** Malloc devolveu NULL!\n");
  status = 1;
  goto FINAL;
}
for (i = 0; i < *numcla_p; i++) {
  (*CLA_p)[i] = (int *) malloc (*numvar_p*sizeof (int));
  if ((*CLA_p)[i] == NULL) {
    printf ("** ERRO ** Malloc devolveu NULL!\n");
    status = 1;
    goto FINAL;
  }
}
for (i = 0; i < *numcnf_p; i++) {
  (*CNF_p)[i] = (int *) malloc (*numcla_p*sizeof (int));
  if ((*CNF_p)[i] == NULL) {
    printf ("** ERRO ** Malloc devolveu NULL!\n");
    status = 1;
    goto FINAL;
  }
}
// leitura e atribuicao de dados
fscanf (arq, "%c", &barraene);
if (barraene != '\n') {
  printf ("** ERRO ** Leitura da linha 0 do arquivo de entrada.\n");
  status = 1;
  goto FINAL;
}
for (i = 0; i < *numcla_p; i++) {
  for (j = 0, fscanf (arq, "%d", &aux);
       aux != 0; j++, fscanf (arq, "%d", &aux)) {
    (*CLA_p)[i][j] = aux;
    if (abs (aux) > *numvar_p)
      printf ("** ERRO ** Numero de literal > numvar.\n");
  }
```

```
    (*tamcla_p)[i] = j;
    fscanf (arq, "%c", &barraene);
    if (barraene != '\n') {
      printf ("** ERRO ** Leitura da linha %d do arquivo"
              " de entrada.\n", i+1);
      status = 1;
      goto FINAL;
    }
  }
  for (i = 0; i < *numcnf_p; i++) {
    fscanf (arq, "%lf", (*prob_p)+i);
    for (j = 0, fscanf (arq, "%d", &aux);
         aux != 0; j++, fscanf (arq, "%d", &aux))
      (*CNF_p)[i][j] = aux;
    (*tamcnf_p)[i] = j;
    fscanf (arq, "%c", &barraene);
    if (barraene != '\n') {
      printf ("** ERRO ** Leitura da linha %d do arquivo"
              " de entrada.\n", i+1);
      status = 1;
      goto FINAL;
    }
  }

  FINAL:
  fclose(arq);
  fclose (in);
  remove (arquivo);
  return status;
}

/* Retira comentarios do arquivo in e devolve o endereco para primeira
// posicao do arquivo sem comentarios arq.
////////////////////////////////////////////////////////////////// */
FILE *limpa (FILE *in, FILE *arq) {
  char aux;

  /* Suponho que uma linha de comentarios nao comeca com numero
  // nem sinal de menos (-) */
  do {
    for (aux = getc(in); (aux < '0'|| aux > '9') && aux != '-'
     && aux != EOF; aux = getc(in))
      while (aux != '\n') aux = getc(in);
    if (aux != EOF) {
      for (aux; aux != '\n'; aux = getc(in))
        putc (aux, arq);
      putc (aux, arq);
```

```
    }
    else break;
  } while (TRUE);
  rewind (arq);
  return arq;
}

/* Preenche dados iniciais do problema lp (=PMR), construindo
// I(numcnf x numcnf)y[0..numcnf-1] = (prob[0..nuncnf-2] | 1) com
// y >= 0, no ambiente env.
// Devolve 1 se uma falha ocorre e 0 caso contrario.
////////////////////////////////////////////////////////////////// */
int preenchelp (CPXENVptr env, CPXLPptr lp, int numcnf, double *prob){
  int status = 0;
  FILE *arq = NULL;           // contera problema no formato lp
  char *arquivo = "fase0.lp";
  int i;

  arq = fopen (arquivo, "w");
  if (arq == NULL) {
    printf ("Arquivo %s nao encontrado.", arquivo);
    fclose (arq);
    status = 1;
    goto FINAL;
  }

  /* Controi modelo para fase0  */
  // Funcao objetivo
  fprintf (arq, "\nMinimize\n");
  for (i = 0; i < numcnf; i++)
    fprintf (arq, "+ y%d ", i);
  // Restricoes
  fprintf (arq, "\nSubject to\n");
  for (i = 0; i < numcnf-1; i++)
    fprintf (arq, "y%d = %f\n", i, prob[i]);
  // Equivale a somatoria das probabilidades dos mundos
  fprintf (arq, "y%d = 1", i);
  fprintf (arq, "\nEnd\n ");

  fclose (arq);

  /* Le e copia problema para lp  */
  status = CPXreadcopyprob (env, lp, arquivo, NULL);
  if (status)  goto FINAL;

  FINAL:
  remove (arquivo);
```

```
  return status;
}

/* Preenche dados iniciais do problema mip (= subproblema), utilizando
// as matrizes de clausulas CLA e de sentencas CNF[0..numcnf][], e
// seus vetores auxiliares de tamanho tamcla e tamcnf para aplicarem
// o processo de linearizacao e definir o problema mip com
// ((numcnf-1)+numvar) variaveis binarias no ambiente env.
// Devolve 1 se uma falha ocorre e 0 caso contrario.
////////////////////////////////////////////////////////////////// */
int preenchemip (CPXENVptr env, CPXLPptr mip, int numvar, int numcnf,
                 int **CLA, int *tamcla, int **CNF, int *tamcnf) {
  int status = 0;
  FILE *arq = NULL;         // contera problema no formato lp
  int numneg;               // numero de literais negados
  int indcla;               // indice da clausula
  char *arquivo = "mip.lp";
  int i, j, k;

  arq = fopen (arquivo, "w");
  if (arq == NULL) {
    printf ("Arquivo %s nao encontrado.", arquivo);
    fclose (arq);
    status = 1;
    goto FINAL;
  }

  /* Lineariza as clausulas negadas para gerar problema */
  // Nota: estamos negando cada clausula, logo literal nao
  // negado na clausula aparecera negado ao gerar a restricao.
  // Funcao objetivo
  fprintf (arq, "\nMinimize\n");
  for (i = 0; i < numvar; i++)
    fprintf (arq, "+ 0 x%d ", i);
  for (i = 0; i < numcnf; i++)
    fprintf (arq, "+ y%d ", i);
  // Restricoes
  fprintf (arq, "\nSubject to\n");
  for (i = 0; i < numcnf; i++) {
    numneg = 0;
    // A principio uma cnf e uma clausula => tamcnf[i] == 1
    for (j = 0; j < tamcnf[i]; j++) {
      indcla = CNF[i][j]-1;
      for (k = 0; k < tamcla[indcla]; k++) {
if (CLA[indcla][k] < 0)
  fprintf (arq, "- x%d + y%d <= 0\n", abs (CLA[indcla][k])-1, i);
       else {
```

```
  fprintf (arq, "x%d + y%d <= 1\n", CLA[indcla][k]-1, i);
  numneg++;
        }
      }
      for (k = 0; k < tamcla[indcla]; k++) {
if (CLA[indcla][k] < 0)
          fprintf (arq, "+ x%d ", abs (CLA[indcla][k])-1);
        else
          fprintf (arq, "- x%d ", CLA[indcla][k]-1);
      }
      fprintf (arq, "- y%d <= %d\n", i, (tamcla[indcla]-(numneg+1)));
    }
  }
  // Tipos
  fprintf (arq, "Binaries\n");
  for (i = 0; i < numvar; i++) {
    if (i%MAXLIN == 0)
      fprintf (arq, "\n", i);
    fprintf (arq, "x%d ", i);
  }
  for (i = 0; i < numcnf; i++) {
    if (i%MAXLIN == 0)
      fprintf (arq, "\n", i);
    fprintf (arq, "y%d ", i);
  }
  fprintf (arq, "\nEnd\n ");

  fclose (arq);

  /* Le e copia problema para mip  */
  status = CPXreadcopyprob (env, mip, arquivo, NULL);
  if (status) {
    printf ("\nErro ao tentar copiar MIP para cplex.\n ");
    goto FINAL;
  }

  FINAL:
  remove (arquivo);
  return status;
}

/* Gera vetor coluna de entrada colentr[0..numcnf-1] que e uma funcao
// do vetor mundo aplicado nas matrizes de sentencas CNF e de
// clausulas CLA com seus vetores auxiliares de tamanho tancnf
// e tamcla.
////////////////////////////////////////////////////////////////// */
void coluna (int *mundo, double *colentr, int numcnf, int **CLA,
```

```
            int *tamcla, int **CNF, int *tamcnf) {
  int i, j, k;
  int indcla;  // indice da clausula

  for (i = 0; i < numcnf; i++) {
    // percorre a i-esima cnf
    for (j = 0; j < tamcnf[i]; j++) {
      indcla = CNF[i][j]-1;
      // percorre a j-esima clausula que compoe a cnf
      for (k = 0; k < tamcla[indcla]; k++) {
       // verifica literal negado
        if (CLA[indcla][k] < 0) {
          // verifica se mundo[|literal|-1] == 0 e valida
          if (mundo[abs (CLA[indcla][k])-1] < 1) break;
}
        else {
          // verifica se mundo[literal-1] == 1 e valida
          if (mundo[CLA[indcla][k]-1] > 0) break;
}
      }
      // verifica se percorremos a clausula e nao houve validacao
      if (k == tamcla[indcla]) {
        colentr[i] = 0;
        break;
      }
    }
    // verifica se todas as clausulas foram validadas
    if (j == tamcnf[i])
      colentr[i] = 1;
  }
  // ultima linha corresponde a uma tautologia
  colentr[i] = 1;
}

/* Desaloca ptr e define para NULL
////////////////////////////////////////////////////////////// */
void desaloca (char **ptr){
  if (*ptr == NULL) {
    free (*ptr);
    *ptr = NULL;
  }
}
```

B Leia-me

Para compilação basta ter o arquivo Makefile, apresentado no Apêndice C, no diretório do código, verificar se o caminho para biblioteca do ILOG CPLEX corresponde a:

/usr/ilog/cplex100/include/ilcplex/cplex.h

e digitar na linha de comando makefile, que gerará o executável *geracol*.

Para chamada do programa, é preciso ter os arquivos para modificação dos parâmetros no formato PRM, altparam.prm e norparam.prm, no mesmo diretório, que são apresentados no Apêndice D e basta digitar:

geracol arquivo.txt

onde arquivo.txt deve conter os dados descritos no arquivo formato.ps, apresentado no Apêndice E, salvo que a cada sentença esta associada uma única cláusula.

O programa apresenta dois tipos de saída:

- Informações sobre a tela que afirmarão que o problema é consistente ou que não, ou que alguma falha ocorreu;
- As informações apresentadas acima e adicionais num formato codificado como abaixo, que são anexadas ao arquivo saida.txt:

  INICIO
  HINI horário de início
  ARQ nome do arquivo
  RSL para consistente S, inconsistente N e falha F
  TMPC tempo total aproximado de processamento do PMR
  TMPI tempo total aproximado de processamento do SP
  ITEC número total de iterações para resolução do PMR
  ITEI número total de iterações para resolução do SP
  ITGC número de iterações do método de geração de colunas
  ITEX número total de vezes que o SP foi resolvido com a
  definição padrão dos parâmetros.
  HFIM horário de término
  FIM

Exemplo de saída:

Informação enviada à tela: `Problema e viavel.`

Informações anexadas ao arquivo saída.txt:

```
INICIO
```

```
HINI Wed Jan 17 21:07:25 2007
ARQ 15_130_2.txt
RSL S
TMPC 3.900000e-07
TMPI 3.368000e+01
ITEC 2491
ITEI 79121
ITCG 139
ITEX 0
HFIM Wed Jan 17 21:07:59 2007
FIM
```

C Para compilar

Arquivo Makefile:

```
SYSTEM     = x86-64_rhel4.0_3.4
LIBFORMAT  = static_pic
#------------------------------------------------------------------------
# Set CPLEXDIR to the directories where CPLEX is installed.
#------------------------------------------------------------------------
CPLEXDIR      = /usr/ilog/cplex100
# ------------------------------------------------------------------------
# Compiler selection
# ------------------------------------------------------------------------
CC  = gcc
# ------------------------------------------------------------------------
# Compiler options
# ------------------------------------------------------------------------
COPT  = -fPIC
# ------------------------------------------------------------------------
# Link options and libraries
# ------------------------------------------------------------------------
CPLEXLIBDIR   = $(CPLEXDIR)/lib/$(SYSTEM)/$(LIBFORMAT)

CLNFLAGS  = -L$(CPLEXLIBDIR) -lcplex -lm -lpthread

compile:
make compile_c

CPLEXINCDIR   = $(CPLEXDIR)/include
CFLAGS  = $(COPT)  -I$(CPLEXINCDIR)
#------------------------------------------------------------------------
#  make compile      : to compilar
#------------------------------------------------------------------------
C_EX = geracol
```

```
compile_c: $(C_EX)

geracol: geracol.o
$(CC) $(CFLAGS) geracol.o -o geracol $(CLNFLAGS)
```

D Arquivos de alteração de parâmetros

Arquivo altparam.prm

```
CPLEX Parameter File Version 10.0
# Altera criterio de parada
CPX_PARAM_EPGAP 1.0
CPX_PARAM_EPAGAP 1.0
# Aplica heurística em todos os nos
CPX_PARAM_HEURFREQ 1
```

Arquivo norparam.prm

```
CPLEX Parameter File Version 10.0
# Redefine valores padrão
CPX_PARAM_EPGAP 1.0e-4
CPX_PARAM_EPAGAP 1.0e-6
CPX_PARAM_HEURFREQ 0
```

E Formato do arquivo

Formato de Arquivo

Thiago A. M. Delatorre

8 de fevereiro de 2006

1 Exemplo e Explicações

```
# comentário
v c C
 1  3  4    0
-1  3  5    0
-3  4  2 -1 0

0.1 1 3 0
0.2 3 0

# v representa o número de variáveis no problema
# c representa o número de cláusulas nas próximas linhas
# C representa o número de expressões com probabilidade
```

Os comentários podem ser utilizados em qualquer parte do arquivo. Um parser para este formato deve ignorar qualquer símbolo entre "#" (inclusive) e o fim da linha.

A primeira linha de dados possui três parâmetros: "v", "c" e "C", que no exemplo poderiam ser 4, 3 e 2, respectivamente . O primeiro parâmetro é um inteiro positivo que representa o maior número de variável que pode aparecer nas cláusulas do problema. O segundo parâmetro é um inteiro positivo que representa o número de cláusulas da instância. O terceiro parâmetro é um inteiro não negativo que representa o número de expressões com probabilidade.

As "c" linhas seguintes representam as cláusulas de uma instância representada em CNF. Um sinal "-" significa que a variável aparece negada. Os âtomos[1] podem aparecer em qualquer ordem em uma mesma cláusula, mas cada cláusula não pode conter mais de um âtomo com a mesma variável. A variáveis são representadas por inteiros no intervalo $[1 : v]$. O 0 no final da linha serve para indicar o final de uma cláusula.

As "c" cláusulas da instância devem ser duas a duas lógicamente distintas.

Após as "c" primeiras linhas de dados segue uma linha em branco e, em seguida, "C" linhas de dados com as expressões com probabilidade. Nestas "C" linhas o primeiro número é um real no intervalo $[0 : 1]$ e os números seguintes são inteiros positivos no intervalo $[1 : c]$ que representam as cláusulas da instância. Ao final de cada linha o número 0 indica o final da expressão.

[1] Na instância, um atômo representa uma variável negada ou uma variável não negada.

F Tabelas de resultados

ARQ	RSL	TMPC(s)	TMPI(s)	ITEC	ITEI	ITGC	ITEX
14–50–1	S	0.06	0.74	1455	3395	56	0
14–50–2	S	0.06	0.67	906	3872	60	0
14–60–1	S	0.08	1.97	621	6377	73	3
14–60–2	S	0.02	1.49	668	4678	67	0
14–70–1	S	0.09	3.55	1045	10314	88	2
14–70–2	S	0.14	3.35	921	9147	81	0
14–80–1	S	0.14	5.85	2148	15890	94	0
14–80–2	S	0.11	5.17	886	14839	88	1
14–90–1	S	0.11	7.79	1142	22712	97	0
14–90–2	S	0.13	9.03	1357	27972	107	3
14–100–1	S	0.21	14.69	1609	45025	114	1
14–100–2	S	0.19	13.54	1567	41573	113	0
14–110–1	S	0.24	18.54	1748	64716	126	3
14–110–2	S	0.30	18.13	1717	58801	118	0
15–60–1	S	0.06	1.75	605	4960	64	0
15–60–2	S	0.17	1.14	2587	4203	70	0
15–70–1	S	0.09	2.99	885	9385	81	1
15–70–2	S	0.08	2.85	1408	7548	79	1
15–80–1	S	0.45	4.18	7049	10512	86	0
15–80–2	S	0.13	5.01	1073	12890	89	0
15–90–1	S	0.20	7.68	1252	21540	106	1
15–90–2	S	0.19	8.20	1584	26784	99	0
15–100–1	S	0.18	11.67	1453	41092	106	0
15–100–2	S	0.18	11.96	1545	38971	108	0
15–110–1	S	0.31	21.36	1821	66658	137	2
15–110–2	S	0.30	18.21	1865	59993	136	2
15–120–1	S	0.38	23.37	2385	76112	130	0
15–120–2	S	0.31	25.41	2166	96408	132	0
15–130–1	S	0.51	34.31	2782	109598	149	0
15–130–2	S	0.58	31.33	2628	113203	147	3

Tabela 2: Testes com parâmetro de heurística padrão

ARQ	RSL	TMPC(s)	TMPI(s)	ITEC	ITEI	ITGC	ITEX
14–50–1	S	0.04	0.71	1461	3219	56	0
14–50–2	S	0.10	0.69	906	3988	60	0
14–60–1	S	0.03	1.56	651	4373	68	0
14–60–2	S	0.02	1.54	610	4301	66	0
14–70–1	S	0.08	3.26	1091	7485	78	0
14–70–2	S	0.06	3.22	811	7056	77	0
14–80–1	S	0.19	5.65	2063	12882	92	0
14–80–2	S	0.10	5.78	977	13175	88	1
14–90–1	S	0.16	9.55	1196	24950	105	2
14–90–2	S	0.11	8.99	1252	20038	102	1
14–100–1	S	0.17	13.39	1392	30669	107	0
14–100–2	S	0.23	13.05	1605	31078	105	0
14–110–1	S	0.29	18.76	1766	43439	120	1
14–110–2	S	0.31	23.28	1716	59678	121	0
15–60–1	S	0.06	1.75	603	4774	65	0
15–60–2	S	0.13	1.18	3682	4071	71	0
15–70–1	S	0.12	3.33	896	6978	83	0
15–70–2	S	0.11	3.23	978	7277	80	1
15–80–1	S	0.55	4.49	8566	11507	91	0
15–80–2	S	0.12	5.04	1049	11332	87	0
15–90–1	S	0.18	8.83	1275	18177	102	0
15–90–2	S	0.11	8.62	1652	18940	97	0
15–100–1	S	0.18	12.13	1540	29214	105	0
15–100–2	S	0.16	14.12	1511	33404	112	0
15–110–1	S	0.25	19.27	1726	46246	123	2
15–110–2	S	0.40	19.37	1935	44397	133	2
15–120–1	S	0.28	26.79	2312	65041	126	0
15–120–2	S	0.33	31.75	2223	82394	137	0
15–130–1	S	0.45	34.80	2602	80879	138	0
15–130–2	S	0.47	33.76	2491	79121	139	0

Tabela 3: Testes aplicando heurística do ILOG CPLEX

ARQ	RSL	TMPC(s)	TMPI(s)	ITEC	ITEI	ITGC	ITEX
14–50–1	N	0.01	0.13	15	697	14	2
14–50–2	N	0.01	0.44	35	1677	26	11
14–60–1	N	0.01	0.60	41	2812	22	9
14–60–2	N	0.00	0.75	40	2910	28	12
14–70–1	N	0.02	1.49	85	6350	45	13
14–70–2	N	0.01	0.85	33	3644	25	11
14–80–1	N	0.00	0.92	33	3956	26	9
14–80–2	N	0.00	0.77	21	3335	19	7
14–90–1	N	0.01	1.92	55	7315	37	14
14–90–2	N	0.01	1.68	45	4749	34	16
14–100–1	N	0.02	2.98	79	14036	46	15
14–100–2	N	0.02	2.12	85	8227	43	10
14–110–1	N	0.02	3.76	69	16903	43	21
14–110–2	N	0.00	2.17	51	7423	29	15
15–60–1	N	0.00	0.24	15	779	14	4
15–60–2	N	0.01	0.61	33	2291	23	11
15–70–1	N	0.00	0.32	21	1232	18	7
15–70–2	N	0.00	0.83	31	2899	26	12
15–80–1	N	0.01	1.55	77	5160	39	15
15–80–2	N	0.01	1.41	56	6948	35	14
15–90–1	N	0.05	2.70	93	10608	58	23
15–90–2	N	0.02	1.70	66	5757	36	15
15–100–1	N	0.01	1.75	52	6146	30	12
15–100–2	N	0.01	2.85	73	12296	42	18
15–110–1	N	0.02	2.48	33	11293	30	12
15–110–2	N	0.02	2.21	42	8723	29	12
15–120–1	N	0.02	3.18	50	10420	34	16
15–120–2	N	0.02	3.95	61	16105	42	20
15–130–1	N	0.05	4.46	64	20764	38	15
15–130–2	N	0.04	7.56	114	33674	67	29

Tabela 4: Testes com π aleatório e com a definição padrão para heurística