4
Resolução exata do PAG

No capítulo anterior foram apresentadas duas formulações para o Problema de Alocação Generalizada, (PAG-C) e (PAG-Exp). Uma aplicação da primeira destas formulações para a resolução do PAG é relatada em de Farias [31]. Os resultados obtidos apresentaram, para a relaxação linear, um *gap* de integralidade proibitivo para várias classes de instâncias. Para reduzir esta dificuldade, [31] faz um estudo poliedral da formulação clássica e propõe desigualdades válidas fortes, que neste caso são facetas (ver Nemhauser e Wolsey[7] e Wolsey [41] para um texto completo sobre combinatória poliedral). Os resultados obtidos não são suficientes para resolver nenhuma instância das classes mais difíceis da literatura, embora acelerem a resolução de instâncias de classes menos complicadas.

Deste modo, a abordagem exata adotada neste trabalho, utiliza a formulação (PAG-EXP) e segue os passos apresentados em Savelsbergh [20]. Isto é, resolver (PAG-EXP) através de um algoritmo de *branch-and-bound*. Para tal, torna-se necessário a utilização de um algoritmo de geração de colunas para obter o limite inferior a ser utilizado em cada nó da árvore de *branch-and-bound*.

Gilmore e Gomory [2] introduziram o uso da técnica de geração de colunas para resolver um problema linear que servia como relaxação de um problema inteiro, no caso deles, o problema de *cutting stock*. Naturalmente, quando um programa inteiro relaxado é resolvido por geração de colunas, a solução não é necessariamente inteira. Para obter uma solução inteira ótima, ou até mesmo viável, do programa inteiro torna-se necessário que a técnica padrão de *branch-and-bound* possa ser combinada com a geração de colunas. Esta necessidade levou a uma intensa pesquisa nos últimos 15 anos [29], [22], [32], [14], [23], [18], [16], [30].

Os algoritmos resultantes foram batizados de *Branch-and-Price*, fazendo referência ao procedimento de determinação da variável a entrar na base durante o algoritmo Simplex, conhecido como *pricing*. Este capítulo descreve na seção 4.1.2 os elementos de um algoritmo de *branch-and-bound*. Na seção 4.2 o algoritmo de geração de colunas para a resolução da relaxação linear de (PAG-EXP) é apresentado. Devido a dificuldades presentes neste processo, este capí-

tulo também contém um estudo sobre métodos de estabilização para algoritmos de geração de colunas. Na seção 4.4 apresenta-se, em detalhe, o algoritmo de *branch-and-price* proposto. Finalmente, a última seção discursa sobre a experiência computacional realizada.

4.1 Geração de colunas em programação inteira

Problemas de programação inteira (ou inteira mista ou MIP como são frequentemente referidos na literatura) são resolvidos de forma genérica por um procedimento de *branch-and-bound* baseados em programação linear. Isto é, uma relaxação do problema obtida retirando-se as restrições de integralidade sobre o valor das variáveis definidas como inteiras no modelo é utilizada para prover uma expectativa do valor da melhor solução que se pode obter para o problema.

4.1.1 Relaxação Linear com um número exponencial de variáveis

A técnica de geração de colunas é aplicada para resolver programas lineares com um número muito grande de variáveis. Ela é baseada no fato do algoritmo simplex não necessitar conhecer todas as colunas do problema no momento de escolher uma variável para entrar na base. Basta que se encontre uma coluna correspondente a uma variável com custo reduzido negativo (ou positivo no caso de maximização), se esta coluna existir, para que o algoritmo prossiga para a próxima iteração. Quando se usa a abordagem por geração de colunas as variáveis de custo reduzido mais negativo são encontradas resolvendo-se um outro problema de otimização. Dessa forma, as colunas ficam implicitamente representadas pelo conjunto de soluções desse subproblema. Assim, a cada iteração é resolvido um problema contendo um subconjunto das variáveis do problema, denominado mestre. Para verificar se a solução do problema mestre corresponde à solução ótima para o conjunto completo das variáveis, resolve-se um subproblema, referenciado neste trabalho como subproblema de Geração de Colunas.

4.1.2 Enumeração, *Branch-and-bound* e *Branch-and-Price*

Um algoritmo de *branch-and-bound* nada mais é do que um método de enumeração implícita de todas as possíveis soluções para a instância de entrada.

Ele requer duas rotinas básicas, uma para calcular limites superiores e outra para calcular limites inferiores para o valor da solução ótima inteira. Inicialmente, aplicam-se as rotinas sobre a instância de entrada. Se o limite inferior for igual ao superior, o algoritmo termina. Se não for esse o caso, faz-se uma ramificação ou *branching*. Este procedimento consiste em particionar o problema em dois ou mais subproblemas que, juntos, são equivalentes ao problema original. No caso em que se usa um programa linear relaxado como método para calcular os limites inferiores, a ramificação é normalmente feita com base nos valores dessas variáveis nessa relaxação. Em um problema definido em variáveis 0-1, como o PAG, a ramificação seria escolher uma variável com valor fracionário e fixar seu valor a 0 em um dos subproblemas e em 1 no outro. A partir daí, cada subproblema é resolvido recursivamente. A melhor solução encontrada dentre todos os subproblemas é a solução do problema inicial. O *branch-and-bound* constrói uma árvore de resolução durante sua execução. Cada nó da árvore dá origem a dois ou mais nós filhos, a menos que os limites inferior e superior coincidam, neste caso o subproblema estará resolvido, ou quando as variáveis fixadas fizerem o problema ficar inviável. Nesta situação diz-se que o ramo que tem esse nó como raiz foi podado.

Aplicar um procedimento padrão de *branch-and-bound* ao problema mestre com suas colunas já geradas não garante uma solução inteira ótima ou nem mesmo viável. Feita a ramificação, pode acontecer o caso em que uma alocação viável tenha um custo reduzido favorável a entrar na base, mas esta alocação não está presente no problema mestre. Desta forma, para encontrar uma solução ótima deve-se gerar colunas após cada ramificação.

O procedimento de *Branch-and-Price* consiste em fazer a ramificação e gerar novas colunas a cada nó. As técnicas para realizar este procedimento de forma eficiente começaram a ser desenvolvidas ao longo da década de 80. Isto justifica as dificuldades encontradas por Gilmore e Gomory na experiência com o problema de corte unidimensional. De forma objetiva, a dificuldade advém do fato de que cada ramificação pode aumentar a complexidade de resolução do problema de geração de colunas. Avanços neste ponto começaram a surgir ao longo dos anos 90. Este trabalho utiliza estas idéias, o que pode ser visto ao longo deste capítulo.

4.2
Relaxação Linear para o PAG

Conforme descrito acima, um algoritmo de geração de colunas mantém, no problema mestre, uma formulação reduzida com apenas um subconjunto de todas

as colunas do problema completo. Sempre que for necessário testar a otimilidade da solução obtida para o problema mestre, com relação ao problema completo, é resolvido um subproblema, chamado de subproblema de geração de colunas. Este método é então aplicado à formulação (PAG-EXP), reapresentada a seguir.

$$\min \sum_{i=1}^{m} \sum_{k=1}^{k_i} \left(\sum_{j=1}^{n} c_{ij} v_{ij}^k \right) y_i^k$$
$$s.a.$$
$$\sum_{i=1}^{m} \sum_{k=1}^{k_i} v_{ij}^k y_i^k = 1 \quad j \in \{1, \ldots, n\},$$
$$\sum_{k=1}^{k_i} y_i^k \leq 1 \quad i \in \{1, \ldots, m\},$$
$$y_i^k \in \{0,1\} \quad i \in \{1, \ldots, m\}, k \in K_i.$$

Aqui, $k_1, k_2, \ldots, k_m$ representam quantidade que crescem exponencialmente com o número de tarefas. Assim, para cada um dos m agentes será necessário resolver um subproblema de geração de colunas.

Para começar o procedimento de geração de colunas, um problema mestre reduzido inicial deve ser fornecido. Este problema mestre inicial deve ser um programa linear viável para garantir que as informações duais apropriadas sejam passadas para o subproblema.

Neste trabalho foi escolhido inicializar o problema mestre com uma coluna, artificial, para cada tarefa. Cada uma dessas colunas possue um 1 na linha relativa a restrição de associação da tarefa e zeros nas linhas relativas aos agentes. O custo atribuído à estas variáveis artificiais é o do par tarefa/agente mais caro adicionado de 1.

Após inserir as colunas descritas no parágrafo anterior, também são inseridas colunas relativas a uma solução inteira retornada por um resolvedor de programação linear inteira. Esta solução inteira é a primeira solução inteira encontrada pelo resolvedor usando a formulação (PAG-C).

4.2.1 Subproblema da geração de colunas

O subproblema da geração de colunas para a formulação por número exponencial de variáveis do PAG corresponde a um problema de mochila inteira para cada agente i, conforme formulado a seguir.

$$z\left(KP_i\right) = \max \sum_{1 \leq j \leq n} \left(u_j - c_{ij}\right) x_{ij}$$
$$s.a.$$
$$\sum_{1 \leq j \leq n} a_{ij} x_j^i \leq b_i$$

$$x_j^i \in \{0,1\} \qquad j \in \{1, \ldots, n\}$$

onde u_j é o valor da variável dual relativa a restrição de obrigatoriedade de alocação da tarefa j (a primeira na formulação acima) associada à solução ótima obtida para o problema mestre. Para se obter o menor custo reduzido entre todas as colunas em (PAG-Exp), ainda é necessário calcular o custo reduzido das alocações obtidas para cada agente. Para isto, ainda é necessário subtrair o valor v_i da variável dual associada à restrição de capacidade do agente i associada à solução ótima corrente do problema mestre, ou seja, determinar o valor da expressão abaixo.

$\min_{1 \leq i \leq m} \{z\left(KP_i\right) - v_i\}$

Conseqüentemente, se o valor objetivo de todas as soluções ótimas obtidas para os subproblemas associados a cada um dos agentes é maior ou igual a zero, então não há colunas correspondentes a variáveis com custo reduzido negativo e a solução ótima corrente para o problema mestre também é a solução ótima para o problema linear descrito por (PAG-Exp).

Assim, resolver o problema de geração de colunas envolve a resolução de vários problemas da mochila, o que poderia ser computacionalmente proibitivo. Felizmente, para o esquema de geração de colunas funcionar, não é necessário selecionar sempre a coluna com o custo reduzido mais negativo, qualquer coluna com custo reduzido negativo pode ser selecionada.

Deste modo, duas alternativas foram testadas neste trabalho. A primeira consiste em executar um algoritmo guloso para o problema da mochila, que primeiro coloca na mochila os elementos com a melhor relação custo por peso. Se este algoritmo não conseguir encontrar uma coluna com custo reduzido negativo é chamado então o algoritmo exato descrito em [28]. A segunda alternativa é chamar sempre o algoritmo exato descrito em [28]. Surpreendentemente, a segunda alternativa apresentou melhores resultados. Isto se deve ao fato do algoritmo exato ser muito eficiente.

O procedimento correspondente de geração de colunas proposto para o PAG é ilustrado no algoritmo 1, onde:

- $conjCol$ é o conjunto de colunas existentes no modelo de programação linear;
- $aumenta$ é uma variável booleana que identifica se um aumento do número de colunas aconteceu;
- $valorSolucao$ armazena o valor da solução do programa linear, retornada pela função resolve;
- $numMaxCols$ é uma constante que guarda um limite máximo do número de colunas no modelo.

– resolve é uma função que resolve o modelo de programação linear definido pelas colunas presentes em $conjCol$;

– numColunas é uma função que retorna o numero de colunas contidas em $conjCol$;

– retiraColunas é a função que retira as $range$ colunas de custo reduzido mais positivos;

– resolveSubProblema é a função que resolve o subproblema, esta função tem como retorno as colunas com custo reduzido negativos encontradas.

```
conjCol = mestreReduzidoInicial();
aumenta = true;
while aumenta do
    valorSolucao = resolve(conjCol);
    if numColunas(conjCol) > numMaxCols then
        retiraColunas(conjCol,range);
        resolve(conjCol);
    aumenta = false;
    colunas = 0;
    for i = 1; i < m; i++ do
        colunas+ = resolveSubProblema(i,&gerouColuna);
        if gerouColuna then
            aumenta = true;
    if aumenta then
        conjCol+ = colunas;
return valorSolucao;
```

Algoritmo 1: Algoritmo de Geração de Colunas

Conforme mencionado acima, o algoritmo de geração de colunas termina quando a resolução do problema da mochila para todos os agentes retornar valores objetivos maiores ou iguais a zero. Se o número de colunas no modelo ultrapassar $numMax$ é chamada a função *retiraColunas* que retira as $range$ colunas de custo reduzido mais positivos. Este procedimento é adotado para evitar que a resolução do problema mestre fique excessivamente lenta. Também foi experimentada outra estratégia para impedir que o número de colunas no modelo crescesse indiscriminadamente. Esta segunda estratégia consiste em retirar a cada intervalo fixo de iterações as colunas de custo reduzido positivos. A primeira estratégia demonstrou melhores resultados do que a segunda.

4.3 Estabilização da Geração de Colunas

A resolução do PAG por geração de colunas apresenta um sério problema de convergência, muitas iterações podem ser necessárias para se obter a resolução do problema mestre em cada nó. Surgiu então a necessidade de acelerar a convergência do procedimento de geração de colunas. Com este objetivo implementou-se um esquema de estabilização da geração de colunas proposto por Merle, Villeneuve, Desrosiers e Hansen [24].

Um método de geração de colunas precisa ser estabilizado quando o valor das variáveis duais associadas oscila muito. Assim, o subproblema de geração de colunas passa a obter, a cada iteração, colunas com características muito diferentes tornando lenta a convergência. A proposta para reduzir esta oscilação pode ser compreendida observando o programa dual do programa linear que está sendo resolvido por geração de colunas.

Seja um programa linear P viável e limitado e o seu dual D:

$$
\begin{array}{lll}
 & \min & \sum_{j=1}^{n} c_j x_j \\
(P) & s.a & \sum_{j=1}^{n} \alpha_{ij} x_j = b_i \quad i = 1, \ldots, m \\
 & x \geq 0 \;\; j = 1, \ldots, n \\
\hline
 & \max & \sum_{i=1}^{m} b_i \pi_i \\
(D) & s.a. & \sum_{i=1}^{m} \alpha_{ij} \pi_i \leq c_j \;\; j = 1, \ldots, n
\end{array}
$$

Uma maneira de se tentar acelerar a convergência da geração de colunas é através da redução da degenerecência. Isso é feito perturbando-se P através da adição de variáveis de excesso y_1 e de variáveis de folga y_2 limitadas respectivamente por pequenos valores ϵ_1 e ϵ_2. Este processo evita que as variáveis duais oscilem.

$$
\begin{array}{lll}
 & \min & \sum_{j=1}^{n} c_j x_j \\
(P_p) & s.a. & \sum_{j=1}^{n} \alpha_{ij} x_j - y_{1i} + y_{2i} = b_i \quad i = 1, \ldots, m \\
 & & x_j \geq 0 \quad j = 1, \ldots, m \\
 & & 0 \leq y_{1i} \leq \epsilon_1, 0 \leq y_{2i} \leq \epsilon_2 \;\; i = 1, \ldots, m
\end{array}
$$

Nota-se que o valor da solução do problema perturbado (P_p) é necessariamente menor ou igual que a solução do problema original (P) e, portanto, é um limite inferior válido para o problema inteiro. Fazendo-se os valores de ϵ_1 e ϵ_2 serem suficientemente pequenos, essa perda pode ser tornada pouco significativa.

O problema perturbado (P_p) evita que as variáveis duais oscilem penalizando-as quando elas saem do valor 0. O problema (P_p) tem o seguinte

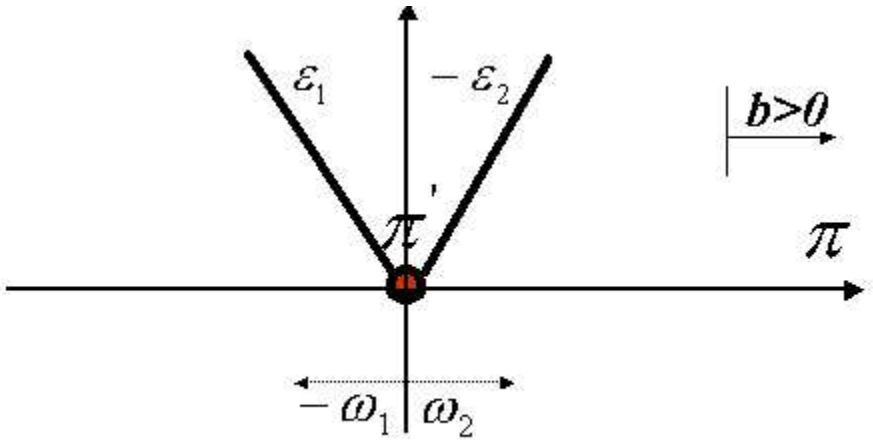

Figura 4.1: Espaço dual do problema (D_p)

problema dual (D_p) associado:

$$
\begin{aligned}
& \max \quad \sum_{i=1}^{m} b_i\pi_i - \sum_{i=1}^{m} \omega_{1i}\epsilon_1 - \sum_{i=1}^{m} \omega_{2i}\epsilon_2 \\
(D_p) \quad & s.a. \quad \sum_{i=1}^{m} \alpha_{ij}\pi_i \leq c_j \quad j = 1, \ldots, n \\
& \omega_{2i} \geq \pi_i \quad i = 1, \ldots, m \\
& \omega_{1i} \geq -\pi_i \quad i = 1, \ldots, m
\end{aligned}
$$

que tem o seu espaço dual representado pela figura 4.1.

Uma outra maneira de se tentar acelerar a convergência da geração de colunas, evitando que as variáveis duais oscilem excessivamente ao longo das iterações, é feita através de restrições que limitam o valor das variáveis duais a um intervalo de valores $[d_1, d_2]$.

$$
\begin{aligned}
& \max \quad \sum_{i=1}^{m} b_i\pi_i \\
(D_r) \quad & s.a. \quad \sum_{i=1}^{m} \alpha_{ij}\pi_i \leq c_j \quad j = 1, \ldots n \\
& d_{1i} \leq \pi_i \leq d_{2i} \quad i = 1, \ldots m
\end{aligned}
$$

Esses limites no dual correspondem ao seguinte problema primal:

$$
\begin{aligned}
& \min \quad \sum_{j=1}^{n} c_j x_j - \sum_{i=1}^{m} d_{1i} y_{1i} + \sum_{i=1}^{m} d_{2i} y_{2i} \\
(P_d) \quad & s.a. \quad \sum_{j=1}^{n} \alpha_{ij} x_j - y_{1i} + y_{2i} = b_i \quad i = 1, \ldots, m \\
& x_j \geq 0 \quad j = 1, \ldots, n \\
& y_{1i} \geq 0, y_{2i} \geq 0 \quad i = 1, \ldots, m
\end{aligned}
$$

Novamente, o valor da solução do novo problema é necessariamente menor ou igual a solução do problema original. Para evitar essa perda, deve-se achar uma solução dual que esteja estritamente contida dentro das faixas d_1 e d_2. Com o objetivo de encontrar tais valores devem ser feitos ajustes nos valores de π a cada iteração, como é ilustrado na figura 4.2.

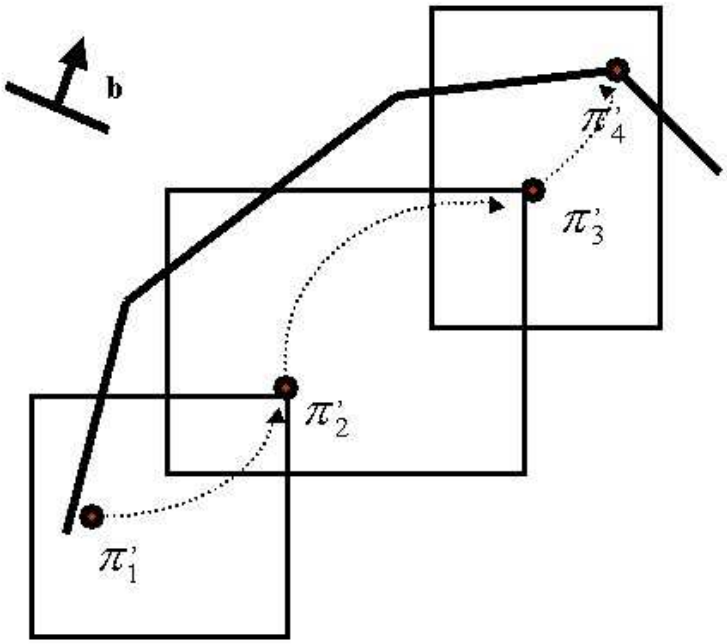

Figura 4.2: Algoritmo do Passo da Caixa

A junção das formulações P_p e P_d resulta na formulação do método de estabilização da geração de colunas P_e:

$$
\begin{array}{lll|l}
 & \min & \sum_{j=1}^{n} c_j x_j - \sum_{i=1}^{m} d_{1i} y_{1i} + \sum_{i=1}^{m} d_{2i} y_{2i} & \\
(P_e) & s.a & \sum_{j=1}^{n} \alpha_{ij} x_j - y_{1i} + y_{2i} = b_i \quad i = 1, \ldots, m & \pi \\
 & & 0 \leq y_{1i} \leq \epsilon_1 \quad i = 1, \ldots, m & -\omega_{1i} \leq 0 \\
 & & 0 \leq y_{2i} \leq \epsilon_2 \quad i = 1, \ldots, m & -\omega_{2i} \leq 0 \\
 & & x_j \geq 0 \;\; j = 1, \ldots, n &
\end{array}
$$

Deve ser notado que as variáveis de folga e excesso têm custo como em P_d e são limitadas por epsilon como em P_p. A exemplo dos dois métodos que o originaram o método de estabilização da geração de colunas também fornece uma solução de valor menor do que a do problema original. Para evitar esta perda os valores duais da solução devem estar dentro das faixas d_1 e d_2 ou ϵ_1 e ϵ_2 estarem iguais a 0.

O problema dual associado a P_e é como segue:

$$
\begin{array}{lll}
 & \max & \sum_{i=1}^{m} b_i \pi_i - \sum_{i=1}^{m} \omega_{1i} \epsilon_1 - \sum_{i=1}^{m} \omega_{2i} \epsilon_2 \\
(D_e) & s.a. & \sum_{i=1}^{m} \alpha_{ij} \pi_i \leq c_j \quad j = 1, \ldots, n \\
 & & d_{1i} - \omega_{1i} \leq \pi_i \leq d_{2i} + \omega_{2i} \;\; i = 1, \ldots, m \\
 & & \omega_{1i} \geq 0, \omega_{2i} \geq 0 \quad i = 1, \ldots, m
\end{array}
$$

Na figura 4.3 é mostrado o espaço dual do método de estabilização da

geração de colunas.

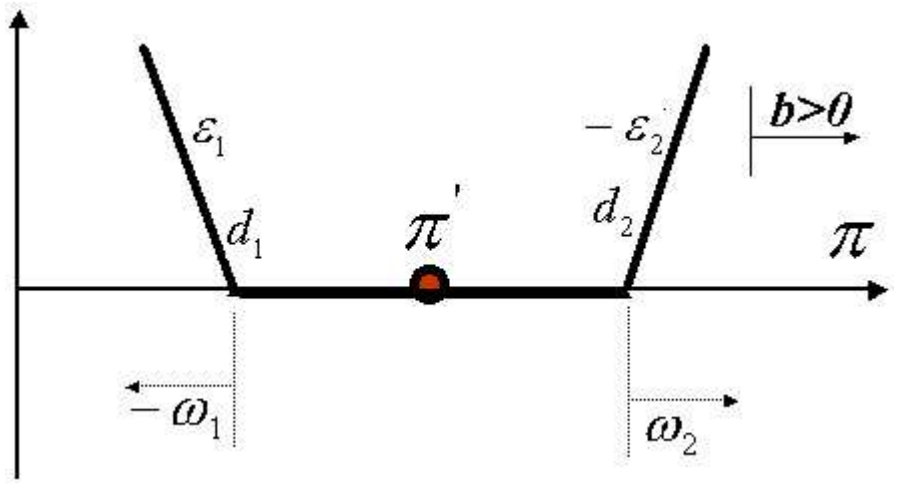

Figura 4.3: Espaço dual para D_e

Neste trabalho decidiu-se fazer $d_1 = d_2 = \pi'$, e executar 4 iteracoes com os valores 0.1, 0.01, 0.001 e 0 para ϵ. Na implementação realizada o valor de π' inicial é igual ao valor das variáveis duais da formulação clássica do PAG relaxado.

O espaço dual para as rodadas é mostrado nas figuras 4.4, 4.5, 4.6 e 4.7. A primeira rodada é feita com $\epsilon = 0.1$. Para este valor de ϵ qualquer mudança nos valores das variáveis duais é muito penalizada. A segunda rodada é feita com $\epsilon = 0.01$, neste caso a penalização pela mudança no valor da variável dual diminui. Mas o processo converge rapidamente porque foram geradas boas colunas na rodada anterior que são mantidas para esta rodada. Para $\epsilon = 0.001$ a penalização por mudança no valor da variável dual cai ainda mais. Finalmente, na quarta e ultima rodada o valor de ϵ é igual a 0, ou seja, não há penalização pela troca de valor da variavel dual. A convergência é rápida devido a qualidade das colunas que foram geradas nas três rodadas anteriores. O procedimento de estabilização é ilustrado no algoritmo 2, onde:

- π são os valores das variáveis duais correspondentes a penalidade zero.
- ϵ é a penalização pelas variáveis duais ao sair de π.
- geraColunas é a função que realiza o procedimento de geração de colunas descrito em 4.2.
- $cutoff$ é um valor limite passado pelo procedimento de *Branch-and-Bound*. Se o valor retornado estiver acima do $cutoff$ o procedimento de geração de colunas deve parar.

```
π = duais da Relaxação;
ϵ = 0.1;
valRetorno = geraColunas();
if valRetorno ≥ cuttoff − 1 + eps then
    return valRetorno
for i = 0; i < 3; i++ do
    π = novos duais calculados pela geraColunas;
    if i==2 then
        ϵ = 0;
    else
        ϵ = 0.1^(i+2);
    valRetorno = geraColunas();
    if valRetorno ≥ cuttoff − 1 + eps then
        return valRetorno
return valRetorno
```

Algoritmo 2: Algoritmo da Estabilização da Geração de Colunas

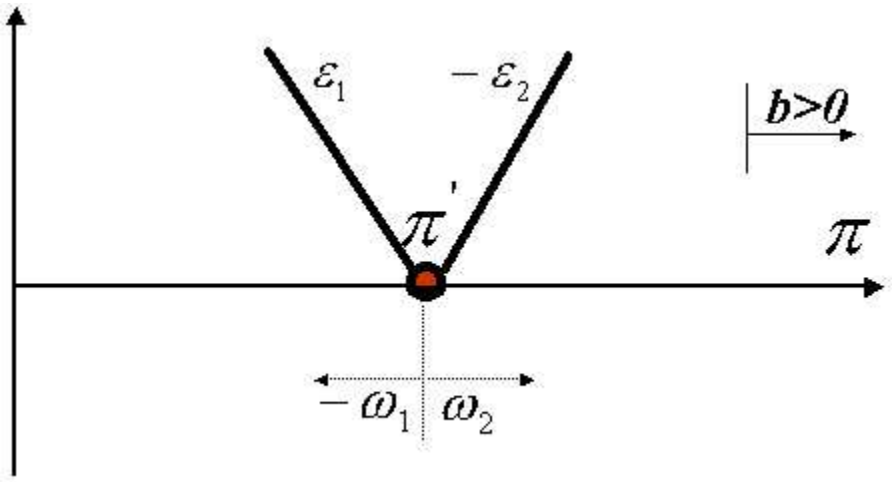

Figura 4.4: Espaço dual para rodada com $\epsilon = 0.1$

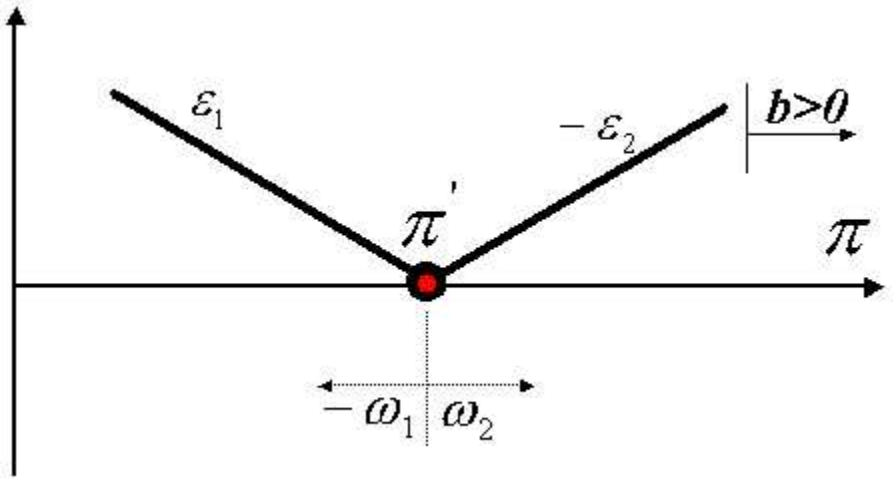

Figura 4.5: Espaço dual para rodada com $\epsilon = 0.01$

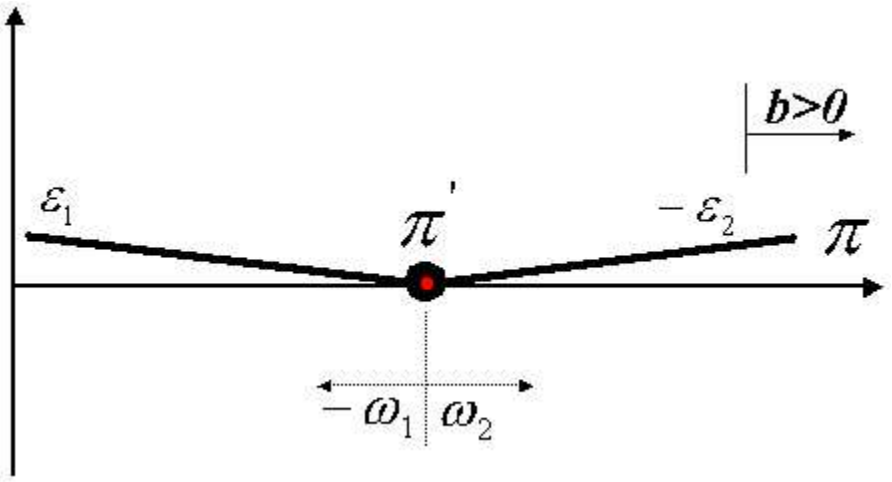

Figura 4.6: Espaço dual para rodada com $\epsilon = 0.001$

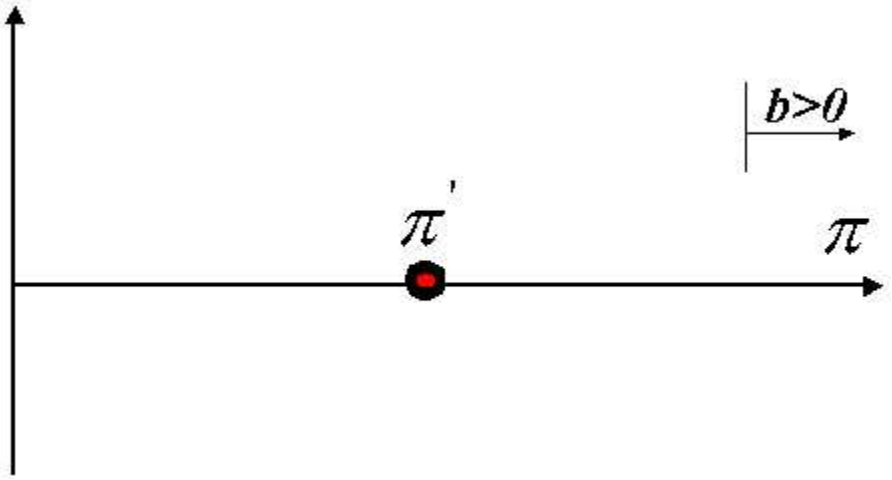

Figura 4.7: Espaço dual para rodada com $\epsilon = 0$

4.4 *Branch-and-Price* para o PAG

Após o procedimento de geração de colunas ter terminado, a solução para o problema mestre será a solução para o problema inteiro original somente no caso das variáveis y_i^k serem inteiras. No caso das variáveis y_i^k não serem inteiras, o que ocorre na maioria das vezes, se faz necessário a execução de um método de *branch-and-bound* para se obter a solução inteira do problema mestre.

Com o objetivo de evitar a geração de colunas que já foram retiradas pela regra da ramificação, deve ser escolhida uma regra que seja compatível com o subproblema. Ser compatível significa ser capaz de modificar o subproblema de maneira que as colunas que são inviáveis pela regra de ramificação não sejam geradas novamente e o subproblema de geração de colunas continue tratável.

O desafio na formulação desta regra é encontrar uma estratégia que exclua a solução corrente, particione de forma valida o espaço de soluções do problema e que gere um subproblema que seja tratável.

Como mencionado na seção 3.1.2, qualquer solução viável da formulação por número exponencial de variáveis tem uma solução viável correspondente para a formulação clássica, seção 3.1.1. Foi dito também que se uma solução para a formulação por número exponencial de variáveis é fracionária, então a solução correspondente para a formulação clássica também é fracionária.

A idéia é fazer as ramificações usando a formulação clássica enquanto, na verdade, trabalha-se com a formulação com número exponencial de variáveis. Estratégias de ramificação para programas lineares 0-1 são baseadas na fixação de variáveis, ou na fixação de uma variável por vez ou na fixação de um conjunto de variáveis. Assim sendo, para se trabalhar dessa maneira é necessário mostrar que fixando-se uma variável ou um conjunto de variáveis na formulação clássica existe uma fixação equivalente na formulação por número exponencial de variáveis, e que o esquema de ramificação resultante é compatível com o subproblema.

Na formulação clássica, fixar a variável x_{ij} a 0 proíbe a tarefa j de ser alocada ao agente i. Fixar a variável x_{ij} a 1 exige que a tarefa j seja alocada ao agente i. Na formulação por número exponencial de variáveis isso pode ser atingido como segue:

- Para proibir que uma tarefa j seja alocada ao agente i, todas as colunas associadas com o agente i que tem um 1 na linha correspondente à tarefa j são fixadas a zero, isto é, se $v_{ij}^k = 1$, então $y_i^k = 0$ para todo $k \in K_i$.
- Para assegurar que uma tarefa j seja alocada ao agente i, todas as colunas associadas com o agente i que não tem um 1 na linha correspondente

à tarefa j são fixadas a 0, isto é, se $v_{ij}^k = 0$, então $y_i^k = 0$ para todo $k \in K_i$, e todas as colunas não associadas ao agente i que tem um 1 na linha correspondente à tarefa j são fixadas a 0, isto é, se $v_{lj}^k = 1$, então $y_l^k = 0$ para $1 \leq l \neq i \leq m$ e $k \in K_l$.

O esquema de ramificação resultante é compatível com o subproblema. Este envolve a resolução de um problema da mochila para cada agente. Seja J_i^0 o conjunto de todos os índices j tais que x_{ij} foi fixado à 0 e J_i^1 o conjunto análogo para fixação em 1. Seja também $J_{\bar{i}}^1$ o conjunto de todas as tarefas que foram fixadas aos agentes diferentes de i, ou seja j tais que $x_{i'j}$ foi fixado em 1 para $i \neq i'$. O subproblema de geração de colunas que leva em conta estas fixações fica então:

$$\max \sum_{j \in \{1,\ldots,n\} \setminus (J_i^0 \bigcup J_i^1 \bigcup J_{\bar{i}}^1)} (u_j - c_{ij})\, \alpha_{ij} + \sum_{j \in J_i^1} (u_j - c_{ij})$$

$$s.a.$$

$$\sum_{j \in \{1,\ldots,n\} \setminus (J_i^0 \bigcup J_i^1 \bigcup J_{\bar{i}}^1)} a_{ij} \alpha_j^i \leq b_i - \sum_{j \in J_i^1} a_{ij}$$

$$\alpha_j^i \in \{0,1\} \qquad j \in \{1,\ldots,n\}$$

Neste trabalho foi usado um esquema para a escolha da variável a ser fixada que leva em consideração o quanto a variável fracionária está proxima de 0.5 e o seu custo. Seja $dist$ o quanto a variável está distante de 0.5 e c o custo da variável. A função de escolha utilizada retorna a variável com o melhor valor para $(1 - dist) * 100 + c$.

Determinada a variável que será fixada, o algoritmo a fixa primeiro a 0 e posteriormente a 1.

O Algoritmo de Ramificação e geração de novas colunas, ou seja, de *branch-and-price*, resultante é mostrado no algoritmo 3, onde:

- geraColunasEstabilizada é o procedimento de geração de colunas estabilizadas descrito em 4.3.
- $valMelhorSolucao$ é o valor da melhor solução inteira encontrada (seu valor inicial é o da solução utilizada para a inicializar o problema mestre na relaxação linear do nó raiz).
- determinaProximoFixado é uma função que determina a próxima variável x_{ij} a ser fixada. Se esta função fizer $i = 0$ e $j = 0$ significa que não existe variável a ser fixada e que a solução corrente é inteira.
- usandoColunas é uma função que retorna $true$ se alguma das colunas artificiais inseridas pelo problema mestre inicial está sendo usada na solução inteira corrente. Caso esteja usando a solução é inviável.

- retirarColunasViolamZero é uma funcão que retira do modelo todas as colunas que violam a fixação de x_{ij} em 0.
- fixarZero é uma função que fixa x_{ij} em 0.
- chamarAlgoritmoRecursivamente() representa a chamada do procedimento de *Branch-and-Bound* recursivamente.
- recolocarColunasViolamZero é uma função que recoloca no modelo as colunas que foram retiradas pela função retirarColunasViolamZero.
- desFixarZero é uma função que retira a fixação de x_{ij} em 0.
- retirarColunasViolamUm é uma funcão que retira do modelo todas as colunas que violam a fixação de x_{ij} em 1.
- fixarUm é uma função que fixa x_{ij} em 1.
- recolocarColunasViolamUm é uma função que recoloca no modelo as colunas que foram retiradas pela função retirarColunasViolamUm.
- desFixarUm é uma função que retira a fixação de x_{ij} em 1.


```
valSolucao = geraColunasEstabilizada(valMelhorSolucao);
if valSolucao ≥ valMelhorSolucao − 1 + ϵ then
    return;
determinaProximoFixado(i,j);
if i == 0 && j == 0 then
    //Solução Inteira Encontrada
    if !usandoColunas() then
        if valSolucao < valMelhorSolucao then
            valMelhorSolucao = valSolucao;
    return;
//fixando em 0
retirarColunasViolamZero(i,j);
fixarZero(i,j);
chamarAlgoritmoRecursivamente();
recolocarColunasViolamZero(i,j);
desFixarZero(i,j);
//fixando em 1
retirarColunasViolamUm(i,j);
fixarUm(i,j);
chamarAlgoritmoRecursivamente();
recolocarColunasViolamUm(i,j);
desFixarUm(i,j);
```

Algoritmo 3: Algoritmo de Ramificação e Geração de Colunas *Branch-and-Bound*

4.5 Resultados Computacionais

As instâncias utilizadas como *benchmark* do PAG na literatura([21], [17]) estão divididas em cinco classes: A, B, C, D e E. As classes utilizadas atualmente para avaliação e comparação de métodos de resolução são C, D, e E. A experiência computacional realizada concentra-se nestas classes. Em particular, as instâncias das classes D e E são geralmente mais difíceis do que as de classe C, dado que c_{ij} e a_{ij} são inversamente correlacionados. Nesta experiência, o algoritmo proposto foi testado em 18 instâncias, 6 de cada classe (a OR-Library[1] disponibiliza todas as instâncias aqui mencionadas).

Os resultados mostrados na tabela 4.1 foram obtidos em um PC Pentium IV 2GHz com 512 MB de memória RAM. Foi utilizado como resolvedor de programação linear o CPLEX versão 7.1 [38]. Os parâmetros do procedimento de geração de colunas $numMaxCols$ e $range$ foram ajustados em 40000 e 10000 respectivamente. O algoritmo de *branch-and-bound* utilizou a estratégia de Busca em Profundidade (primeiro o nó de maior profundidade) para percorrer a árvore de busca.

Na tabela 4.1 a coluna tipo informa qual a classe da instância. As colunas m e n informam, respectivamente, o número de agentes e de tarefas da instância. A coluna LB mostra o Limite Inferior obtido par a instância através da relaxação da formulação (PAG-Exp). A coluna melhor-conhecida contém os valores da melhor solução conhecida até o momento para a instância. A coluna algoritmo-deste-trabalho mostra os valores obtidos pela implementação realizada neste trabalho. A coluna tempo-melhor-solução informa o tempo em segundos que o algoritmo implementado neste trabalho demorou para encontrar a solução ótima. A coluna tempo-para-provar-ótimo mostra o tempo em segundos que o algoritmo implementado consumiu para provar a otimalidade da solução encontrada. O número de nós visitados até a solução ótima ser encontrada é mostrado pela coluna nó-melhor-solução e o número total de nós visitados para provar a otimalidade é mostrado na coluna total-nós-visitados.

Os valores marcados com $**$ foram os melhorados através do algoritmo implementado neste trabalho provando otimalidade. $*$ indica que a solução obtida obteve o mesmo valor da ótima conhecida, provando também otimalidade. Os valores marcados com um † indicam que o algoritmo correspondente o encontrou pela primeira vez.

Nesta tabela, pode-se observar que as instâncias D10100 e D05200 estavam em aberto e foram resolvidas otimamente pela primeira vez neste trabalho.

[1] URL da OR-Library: http://mscmga.ms.ic.ac.uk/jeb/orlib/gapinfo.html

Além disso, todos os melhores valores conhecidos das instâncias já resolvidas foram atingidos.

tipo	m	n	LB	melhor conhecida	algoritmo deste trabalho	tempo melhor solução	tempo para provar ótimo	nó melhor solução	total nós visitados
C	5	100	1930	†1931	*1931	0.93	1.17	3	5
C	5	200	3455	†3456	*3456	420.38	422.98	46	47
C	10	100	1400	†1402	*1402	1.12	1.84	9	17
C	10	200	2804	†2806	*2806	224.78	266.17	22	35
C	20	100	1242	†1243	*1243	1.92	2.12	25	29
C	20	200	2391	†2391	*2391	53.81	55.38	22	23
D	5	100	6350	†6353	*6353	61.74	96.30	106	171
D	5	200	12741	12743	**†12742	312.68	583.68	11	57
D	10	100	6342	6349	**†6347	705.76	818.62	2416	2687
D	10	200	12426	12433					
D	20	100	6177	6196	**†6185	902.36	1043.92	2345	2807
D	20	200	12230	12244					
E	5	100	12673	†12681	*12681	27.68	30.09	57	63
E	5	200	24927	†24930	*24930	1080.56	1305.62	27	31
E	10	100	11568	†11577	*11577	12.53	22.85	47	87
E	10	200	23302	†23307	*23307	135.62	670.62	19	155
E	20	100	8431	†8436	*8436	6.60	6.81	36	37
E	20	200	22377	†22379	*22379	36.79	40.74	18	21

Tabela 4.1: Resultados do *Branch-and-Price* Estabilizado para o PAG.

As instâncias D10200, D20200 não foram resolvidas pelo algoritmo implementado porque o *gap* entre a melhor solução conhecida e o limite inferior ainda é muito grande. Os valores apresentados na coluna LB-novo da tabela 4.2 são os novos limites inferiores obtidos pelo algoritmo proposto, os antigos valores são mostrados na coluna LB-antigo. A coluna tempo mostra o tempo em segundos consumido pelo algoritmo implementado para melhorar tais limites.

tipo	m	n	LB antigo	LB novo	tempo
D	10	200	12426	12430	47197.26
D	20	200	12230	12234	38469.81

Tabela 4.2: Limites inferiores melhorados