3
Algoritmo Branch-Cut-and-Price Robusto

3.1
Introdução

Problemas de Otimização Combinatória estão presentes tanto no âmbito industrial, como na área comercial e científica. As técnicas utilizadas com o intuito de encontrar soluções para esses problemas podem ser subdivididas em duas grandes classes: algoritmos exatos e algoritmos heurísticos. Os algoritmos exatos visam a encontrar soluções ótimas, provando matematicamente a otimalidade. Entretanto, em se tratando de problemas de complexidade exponencial, para instâncias de médio e de grande porte, com essa abordagem nem sempre será possível obter soluções de qualidade em tempo computacional razoável. Este capítulo descreve o algoritmo *Branch-Cut-and-Price* proposto neste trabalho. Primeiramente, é apresentado o subproblema *pricing* utilizado para resolver a relaxação linear através de geração de colunas. Isso implica a permissão de se gerar rotas não elementares, evitando a resolução de um problema fortemente NP-Hard. O subproblema resultante pode ser resolvido em tempo pseudo-polinomial. Afirma-se que *Branch-Cut-and-Price* é robusto do ponto de vista de eficiência quando a complexidade se mantém para todos os subproblemas de geração de colunas resolvidos durante toda a execução do BCP. Com o intuito de preservar essa propriedade, os cortes apresentados nas seções subsequentes são definidos sobre as variáveis da segunda formulação. As restrições de *branching* também mantém essa propriedade conforme descrição mais detalhada mostrada no decorrer deste capítulo.

3.2
Branch-and-Price

Na literatura, o principal algoritmo exato já proposto para o TOP foi o algoritmo *Branch-and-Price* de Boussier et al. (BFG07). Esse trabalho utiliza rotas elementares e *branching* sobre as variáveis de visitação dos vértices.

A seguir, é apresentada a formulação de Boussier et al. (BFG07) que é

usada no algoritmo exato.

$$\text{Maximizar} \quad \sum_{j \in R} p_j.\lambda_j \tag{3-1}$$

$$s.t. \quad \sum_{j \in R} a_{v\lambda}.\lambda_j \leq 1 \quad \forall v \in V \setminus \{v_0, v_{n+1}\} \tag{3-2}$$

$$\sum_{j \in R} \lambda_j \leq m \tag{3-3}$$

$$\lambda_j \in \{0,1\} \quad \forall j \in R \tag{3-4}$$

$$\tag{3-5}$$

onde:

– R é o conjunto de todas as rotas elementares de v_0 a v_{n+1};

– $a_{v\text{j}}$ indica se o vértice v é visitado pela rota j.

A Relaxação Linear dessa formulação é computada por Geração de Colunas em cada nó da árvore de *branch-and-price*.

Essa formulação é semelhante à terceira formulação, apresentada no capítulo 2, cuja relaxação é utilizada como problema mestre restrito no algoritmo BCP proposto neste trabalho. A diferença é que o conjunto R da formulação de Boussier et al. (BFG07) representa um conjunto de rotas elementares e o conjunto Q, da formulação deste trabalho contém rotas não elementares.

O subproblema de geração de colunas consiste em resolver um *Elementary Shortest Path Problem with Resource Constraints (ESPPRC)*, no qual o recurso restrito é o tempo de duração das rotas que é limitado pelo valor de L.

3.3 Subproblema de Geração de Colunas

O subproblema de geração de colunas ou *pricing subproblem* consiste em encontrar rotas, sejam elas elementares ou não elementares, com máximo custo reduzido (problema de maximização) e com duração limitada ao tempo máximo L.

No primeiro caso de rotas elementares, uma rota $R = (v_0, \ldots, v_{n+1})$ não pode visitar o mesmo vértice mais do que uma vez.

Entretanto, no segundo caso, de rotas não elementares, considera-se que o mesmo vértice pode ser visitado repetidas vezes pelo mesmo veículo. O problema de gerar a rota não elementar com o maior custo reduzido pode ser resolvido por programação dinâmica usando-se a recursão dada abaixo (3-6) :

$$r_c^l(j) = max\{r_c^l(j), r_c(i)^{l+l_{(i,j)}} + p(j) - \pi_{(i,j)}\} \tag{3-6}$$

O custo reduzido máximo em cada vértice j e a duração de rota l é dado por $r_c^l(j)$. O valor $\pi(i,j)$ é o custo dual do arco (i,j) e a soma de todas as contribuições duais do problema mestre restrito corrente. No nó raiz do *branch-cut-and-price* $\pi(i,j)$ corresponde à variável dual associada à *j-ésima* restrição (2-49). Como cortes são adicionados e *branching* é feito, outros custos duais irão contribuir para o valor de $\pi(i,j)$.

A rota, seja ela elementar ou não elementar, com maior custo reduzido é encontrada, observando-se os valores de $\pi(i,j)$, em $O(n^2L)$. É possível eliminar 2-ciclos nas rotas ((i,j) e (j,i) em sequência) sem alterar essa complexidade.

O pseudo-código do algoritmo completo é apresentado abaixo:

Algorithm 1: Subproblema de Geração de Colunas

Input: n vertices
Input: L maximum duration of the routes
Output: generated routes Q^p

```
1  State[n][L] stateMatrix;
2  initializeMatrix(stateMatrix);
3  List<State> finalStates;
4  t = 0;
5  while t < L do
6      foreach state ∈ S_o pened do
7          if state.node = n + 1 then
8              finalState.add(state);
9              if stateMatrix[i][l].arc > stateMatrix[n + 1][L].arc then
10                 stateMatrix[n + 1][L].father ⟵ state;
11                 stateMatrix[n + 1][L].rc ⟵ state.rc;
12             continue;
13         foreach v ∈ V do
14             if state.point = v then
15                 continue;
16             t_a^1 ⟵ ConsumedTime(node.point, v);
17             t_a^2 ⟵ ConsumedTime(v, n + 1);
18             if t_a^1 + t_a^1 > L then
19                 continue;
20             if state.father != NULL AND state.father.point = v then
21                 continue;
22             value ⟵ state.profit - SumOfDuaisArc(node.point, v, l);
23             if state.rc + profit > stateMatrix[v][t + t_a^1].rc then
24                 stateMatrix[v][t + t_a^1].rc ⟵ state.rc + profit;
25                 stateMatrix[v][t + t_a^1].father ⟵ state;
26                 stateMatrix[v][t + t_a^1].opened ⟵ true;
27                 S_opened ⟵ S_opened + stateMatrix[v][t + t_a^1];
28         state.opened = false;
29         S_opened ⟵ S_opened − state ;
30     t++;
31 State bestState = stateMatrix[n + 1][L] ;
32 if bestState.rc < ZERO then
33     return;
34 Q^p ⟵ RecuperaRotas(finalStates);
```

As distâncias são aproximadas usando duas casas decimais (arredondando para baixo). Portanto, é possível gerar rotas inviáveis no subproblema de geração de colunas. Com o intuito de evitar esse problema, insere-se no conjunto de rotas somente as rotas viáveis. Sempre que o subproblema de geração de colunas gera somente rotas inviáveis, modifica-se o grafo da programação dinâmica de modo a eliminar o melhor caminho (inviável, nesse caso) e executa-se o algoritmo novamente com o grafo modificado. Esse processo se repete até que o algoritmo gere um caminho viável ou até não ser mais possível encontrar rotas com custo reduzido positivo.

O fato de utilizarmos uma precisão de duas casas decimais implica um maior custo computacional do algoritmo de geração de colunas, bem como do algoritmo de separação de cortes, visto que ambos utilizam o grafo estendido que possui n^2L nós. Sendo assim, surgiu a ideia de, no futuro, arredondar todas as distâncias para baixo e utilizar uma precisão menor - de uma casa decimal, por exemplo. Com isso, aumenta o risco de se ter soluções inviáveis em relação à restrição do tempo máximo das rotas. Por essa razão, poderiam ser adicionados os denominados *no-good cuts* sempre que for encontrada uma solução inteira, porém inviável por ultrapassar o tempo máximo permitido de duração das rotas. Esse corte seria simplesmente uma restrição que impede a utilização consecutiva dos arcos da(s) rota(s) que tenha(m) ultrapassado o valor de L, quando se verifica a viabilidade das rotas através do uso das distâncias euclidianas com todas as casas decimais.

3.3.1 Algoritmo de Remoção de Caminho

Com o objetivo de remover o caminho com o maior custo reduzido, foi utilizado o algoritmo proposto por Santos (S07) para encontrar o *k-ésimo* menor caminho em um grafo. A idéia principal considera a seguinte propriedade: o segundo menor caminho em uma rede N é o menor caminho em uma rede N', obtida a partir de N removendo o menor caminho e aplicando novamente um algoritmo de caminho mínimo após a remoção do referido caminho.

A remoção de um caminho $p = [s, v_1, ..., v_l, t]$, com origem s e destino t, em uma rede N pode ser realizada construindo-se uma nova rede N' de N da seguinte forma:

- faz-se uma cópia do caminho p criando-se cópias dos seus nós internos;
- conecta-se essas cópias de vértices aos mesmos vértices aos quais os vértices originais estão conectados, incluindo os arcos em p, mas sem conectar os novos vértices aos vértices originais correspondentes;

– conecta-se a origem somente a v_1 não conectando à sua respectiva cópia;

– conecta-se a cópia de v_l para o destino, sem conectar o original v_l.

Seja $T_s(T'_s)$ a menor árvore com raiz em s na rede $N(N')$, que é uma árvore formada pelos caminhos mínimos de s para todos os nós de $N(N')$. Se T_s já estiver computada, então T'_s pode ser obtida eficientemente.

3.4 Fortalecendo a Formulação

Como este trabalho utiliza rotas não elementares, que permitem a visita do mesmo vértice repetidas vezes, o limite superior obtido pela relaxação linear é maior que ou igual ao limite superior obtido quando se utiliza rotas elementares ao resolver a relaxação linear utilizando a técnica de geração de colunas.

Por essa razão, é necessário adicionar cortes à formulação de modo fortalecê-la eliminando os subciclos. Neste trabalho, foram utilizadas duas famílias de cortes: uma adaptação dos *Triangle Clique Cuts*, propostos por Pessoa et al. (PPU09) e *min-cut*, propostos por Poggi et al. (PVU10).

3.4.1 Triangle Clique Cuts

Seja $S \subset V^-$ um conjunto de vértices contendo exatamente três elementos. Considere agora todos os arcos em $A_1 \bigcup A_2$ que têm como ponto extremo um vértice em S, e suas cópias em l. Afirma-se que dois arcos são compatíveis quando existe uma rota que contém ambos.

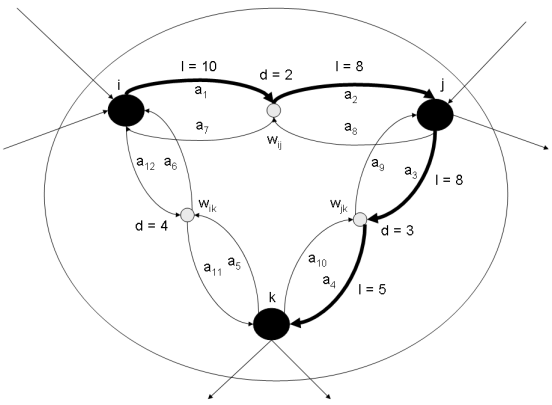

Figura 3.1: Arcos Compatíveis

A Figura 3.1 ilustra a idéia de compatibilidade de arcos. São exibidos vértices pretos e vértices cinzas. Os vértices na cor preta são os vértices

que estão em S. Os vértices na cor cinza são denominados de vértices intermediários. O índice l representa o tempo de partida para cada arco. O valor da demanda d dos vértices em cinza correspondem ao tempo de viagem dos arcos originais associado a esse vértice intermediário. Os arcos em negrito descrevem uma possível parte de uma rota e, portanto, são *compatíveis*. Vale ressaltar que a_5 e a_6 não são compatíveis com os arcos em negrito porque se houvesse um fluxo retornando para o vértice i, o tempo de chegada em i não seria igual a 10.

Os *triangle clique cuts* simplesmente declaram que a soma das variáveis associadas aos arcos (e suas cópias) em um conjunto onde cada par não é compatível pode ser no máximo 1. Isso pode ser visto como uma clique em um grafo de incompatibilidade onde há uma aresta conectando cada par de arcos incompatíveis. Outra visão dessa mesma estrutura é considerar conjuntos estáveis em um grafo de compatibilidade que, no caso, é muito menos denso.

Seja $G' = (V', E')$ o grafo de compatibilidades onde cada vértice de V' representa um arco indexado no tempo $a^l = (i,j)^l$ para $a \in A_1 \bigcup A_2$ e $l = 0, \ldots, L$. Nesse caso, uma aresta $e = (a_1^{l_1}, a_2^{l_2})$ pertence a E' se, e somente se $a_1^{l_1}$ e $a_2^{l_2}$ são *compatíveis*. Seja $S = \{i, j, k\}$. Há quatro casos:

Caso 1: Se $e = ((i, w_{ij})_1^l, (i, w_{ik})_2^l)$, então $e \notin E'$

Caso 2: Se $e = ((i, w_{ij})_1^l, (k, w_{kj})_2^l)$, então $e \notin E'$

Caso 3: Se $e = ((i, w_{ij})_1^l, (w_{ij}, k)_2^l)$, e $l_1 \neq l_2 - l(w_{ij})$, então $e \notin E'$

Case 4: Se $e = ((i, w_{ij})_1^l, (w_{ij}, k)_2^l)$, e $l_1 = l_2 - l(w_{ij})$, então $e \in E'$

Para qualquer conjunto independente $I \subset V'$, a seguinte desigualdade é válida: $\sum_{a^l \in I} x_a^l \leq 1$.

A rotina de separação para os *triangle clique cuts* encontra o conjunto independente $I \subset V'$ em G' que maximiza $\sum_{a^l \in I} \overline{x}_a^l$, onde $\overline{x}_a^l$ denota a solução ótima linear atual. Apesar do problema de encontrar o *maximum-weighted independent set* ser fortemente NP-Difícil, é possível explorar a estrutura específica de G' e encontrar um *maximum weighted independent set* em tempo polinomial.

Um conjunto I é um *maximum-weight independent set* para um conjunto de cadeias se, e somente se, é a união dos *maximum-weight independent set* de cada cadeia separadamente.

Encontra-se em tempo linear o *maximum-weight independent set* para cada cadeia H, usando-se um algoritmo baseado em programação dinâmica. Seja $a_i^{l_i}$ o *i-ésimo* vértice na cadeia H, numerada de 1 a $|H|$ de um extremo a outro da cadeia.

Define-se $I^*(i,1)$ como sendo o *maximum independent set* para a subcadeia contendo os primeiros i vértices de H que usam o *i-ésimo* vértice. Fi-

nalmente, seja $c(I) = \sum_{a^l \in I} \overline{x}^l_a$. Tem-se a seguinte recorrência:

$$c(I^*(i,1)) = \overline{x}^{l_i}_{a_i} + c(I^*(i-1,0))$$
$$c(I^*(i,0)) = max(c(I^*(i-1,0)), c(I^*(i-1,1)))$$

Vale ressaltar que a adição desses cortes é uma maneira de eliminar ciclos de tamanho fixo da solução do problema mestre restrito.

3.4.2 As novas desigualdades min-cut

A segunda formulação foi fortalecida através da adição de desigualdades denominadas *Min Cut Inequalities*. De acordo com o que foi pesquisado nesse trabalho, essas desigualdades não haviam sido propostas anteriormente. Os cortes são descritos sobre as variáveis x^l_a e y_v, variáveis originais da formulação de geração de colunas. Entretanto, as restrições resultantes são escritas em termos de variáveis λ_j, através da equação (2-47), e adicionadas à formulação de geração de colunas.

As desigualdades *Min Cut* são baseadas na intuição de que as soluções fracionárias para a segunda formulação irão entrar e sair em um vértice várias vezes, enquanto que soluções inteiras irão ter no máximo um valor de $x^l_{(i,j)}$ para todos os vértices i e para todos os instantes l maior do que zero, e consequentemente igual a um. De uma certa maneira, esse corte pode ser interpretado como uma restrição de eliminação de sub-ciclos, embora considere todas as rotas com valor não negativo da solução corrente.

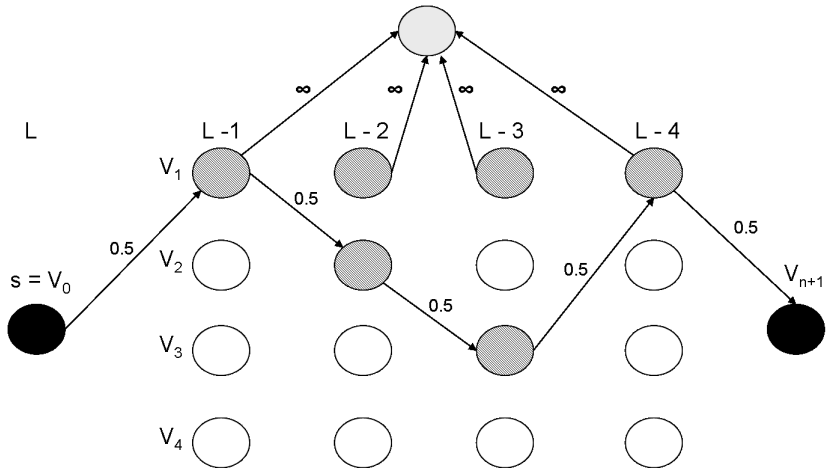

Figura 3.2: Solução fracionária violando uma desigualdade min-cut

A Figura 3.2 apresenta uma situação na qual o corte *Min-Cut* pode ser aplicado. Primeiramente, observa-se que exatamente uma unidade de

fluxo entra no vértice V_1, satisfazendo as restrições (2-40), mas violando a integralidade. Em seguida, verifica-se que o corte mínimo, a partir do depósito inicial V_0 para todas as cópias de V_1 (ou para o vértice convergente no topo da figura), é 0.5. Esse corte é dado pelo conjunto de todos os vértices em cinza e o valor mínimo para ele é um (ou o valor corrente da variável y_1).

Seja S o conjunto dos vértices associados a um corte mínimo, considerando o vértice v. A desigualdade correspondente é dada por:

$$\sum_{a,l|a\in\delta^-(S)} x_a^l \geq y_v \tag{3-7}$$

A identificação de desigualdades violadas é dada pela resolução de um problema corte mínimo s-t. Considere o grafo com conjunto de vértices $\{0, n+1\} \bigcup \{(i,l)|i \in V^-, \quad l = 0,\ldots,L\}$ e o conjunto de arcos $\{(i^l, j^{l-l_{i,j}})|x^l_{(i,j)} > 0\}$, no qual a capacidade dos arcos é dada pelos valores das variáveis $x^l_{(i,j)}$ na solução fracionária atual. Para esse grafo, adicione um vértice destino e arcos de todas as cópias de um vértice v, v^l para $l = 0,\ldots,L$, e atribua uma capacidade infinita. Para obter o corte mínimo s-t, resolve-se o problema do fluxo máximo sobre esse grafo com vértice 0 como origem e o vértice artificial requerido como destino.

A desigualdade resultante (3-7) é definida somente sobre variáveis da segunda formulação. Portanto, as variáveis duais associadas permitirão a atribuição dos seus valores para os custos duais dos arcos e o subproblema de geração de colunas permanece inalterado.

O Algoritmo 3 detalha a separação do *Min-Cut*.

A Figura 3.3 ilustra uma solução fracionária que é a entrada do algoritmo de separação de cortes *min-cut*. Esta figura retrata o grafo estendido no qual cada vértice $v \in V$ do grafo original é replicado L vezes, ou seja, cada célula desta matriz representa um vértice em um determinado instante de tempo. Nesse exemplo, há 32 vértices, sendo o vértice 0 e o vértice 31 marcados em preto por serem os pontos inicial e final respectivamente. Nessa instância, o tempo máximo de duração das rotas é 20. Como trabalha-se com tempo de viagem igual a distância euclidiana entre os pontos e como os valores das distâncias são números de ponto flutuante, tanto as distâncias como o tempo máximo L foram multiplicados por 100 de modo a assegurar uma precisão de duas casas decimais. Por isso, no exemplo, as colunas, representando os instantes de tempo, começam de 2000 até 0. Para fins didáticos e de visualização, a figura omitiu as colunas cujos instantes não aconteceram visitas a nenhum vértice.

A última coluna mostra os valores das variáveis de visitação dos vértices y_v da solução fracionária. As cores das rotas não elementares indicam o valor

das variáveis λ_j que representam as rotas. Nesse exemplo, todas as variáveis de rota possuem valor igual a 0.5.

Agora, observe a Figura 3.4. Nessa figura, a linha referente ao vértice 25 foi destacada pelo fato de que em bora a variável y_v tenha valor igual 1, há um subciclo de modo que o vértice é visitado duas vezes, entrando em cada vez um fluxo no valor de 0.5. O objetivo é, portanto, encontrar um corte que elimine esse subciclo indesejado.

O algoritmo de separação cria um grafo semelhante a esse da Figura 3.4 conforme já explicado anteriormente, com capacidades iguais aos valores das variáveis da solução fracionária (no caso, 0.5 para todos os arcos da solução, exceto os dois arcos que conectam os pontos da linha 25 ao vértice artificial t que têm capacidade infinita). Executa-se o fluxo máximo de 0 para t e o resultado é apresentado na Figura 3.5.

Como o valor do fluxo máximo (0.5) é inferior ao valor da variável y_v do vértice 25 ($y_v = 1$), nesse caso, foi encontrada uma violação. O valor do fluxo máximo foi 0.5 conforme mostrado na legenda. Através do algoritmo do fluxo máximo, obtém-se o corte mínimo, conforme o teorema *min-cut max-flow*. Na 3.5, a aresta mais espessa (na cor laranja) representa a aresta do *min-cut*, pois essa aresta conecta os dois conjuntos de vértices separados pelo corte mínimo (poderia ser mais de uma aresta, nesse exemplo, é essa aresta).

A restrição a ser adicionada deve forçar que o somatório das arestas que conectam os dois conjuntos de vértices separados pelo corte mínimo deve ser maior do que ou igual ao valor da variável y_v.

Como a formulação é em termos de variáveis de rotas não elementares (λ_j), faz-se necessário traduzir cada arco do corte no somatório das rotas que o percorrem.

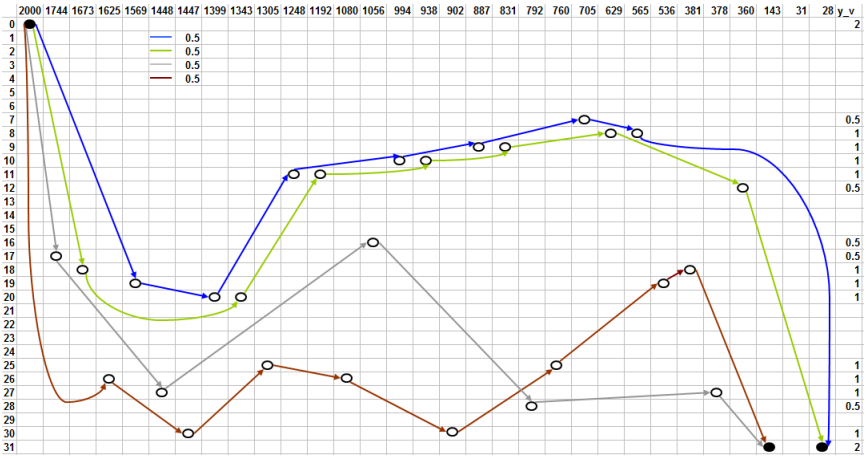

Figura 3.3: Solução da Relaxação Linear

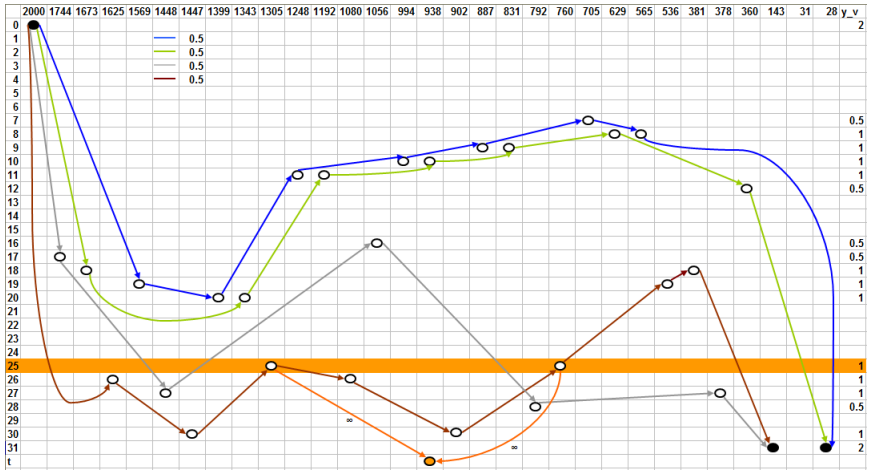

Figura 3.4: Entrada do Algoritmo de Fluxo Máximo

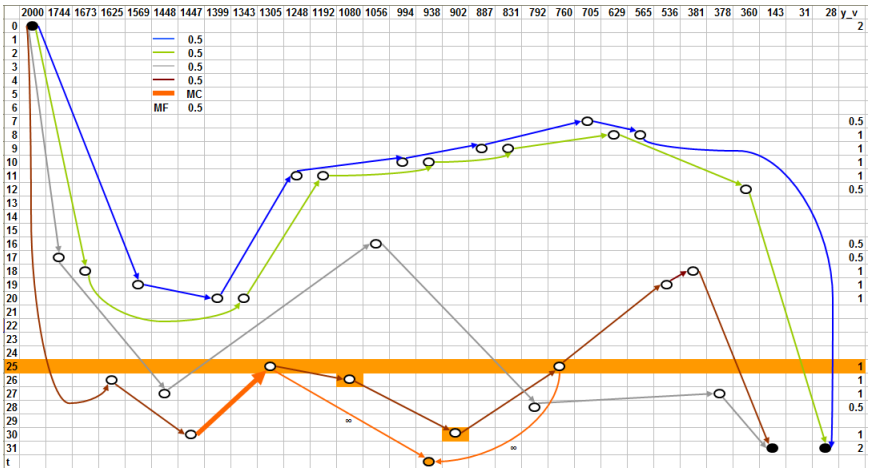

Figura 3.5: Saída do algoritmo de Separação do Min-Cut

3.5 Detalhes do Branch-and-Bound

O algoritmo *Branch-Cut-and-Price* inicia com uma fase de geração de colunas. Um vez que uma solução ótima da relaxação linear é encontrada, cortes são separados ou é realizado *branching*. Em ambos os casos, o subproblema de geração de colunas deve ser resolvido novamente até que uma outra solução ótima para o problema relaxado seja obtida.

Primeiramente, realiza-se *branching* sobre os vértices, como no trabalho de Boussier et al. (BFG07), decidindo se eles serão visitados ou não. Posteriormente, se necessário for, realiza-se *branching* sobre os arcos. Esse é um esquema de *branching* robusto porque não afeta o subproblema de geração de colunas. *Bounding* é feito usando-se os valores das soluções viáveis encontradas em Boussier et al. (BFG07).

A formulação usada no problema mestre é uma relaxação linear da formulação apresentada em 2.3.4. Sempre que uma variável qualquer é fixada ($y_v = 1$), em um nó da árvore de *branch-and-bound*, deve necessariamente existir uma rota que visite o vértice v. Quando tal rota inexiste, o problema torna-se inviável. Assim, para realizar *branch* sobre as variáveis y_v, foi necessário adicionar às restrições varáveis de folga artificiais f^+, f^- e q, com alto custo.

Dessa forma, garante-se a viabilidade do problema mestre restrito. A formulação modificada pode ser escrita da seguinte forma:

$$\text{maximizar} \quad \sum_{v \in V^-} p_v \cdot y_v - \sum_{v \in V^-} M \cdot f_v^+ - \sum_{v \in V^-} M \cdot f_v^- - M \cdot q \tag{3-8}$$

$$\sum_{a \in \delta^-(v)} \sum_{j \in Q} \sum_{l=0}^{L} g_a^{lj} \cdot \lambda_j + f_v^+ - f_v^- = y_v \qquad \forall v \in V \tag{3-9}$$

$$\sum_{a \in \delta(0)^+} \sum_{j \in Q} g_a^{0j} \cdot \lambda_j + q = m \tag{3-10}$$

Nesse caso, uma inviabilidade é detectada quando, resolvendo-se o LP na otimalidade, existe alguma variável de folga com valor positivo.

A abordagem de realizar *branching* apenas sobre as variáveis y_v, pode ainda resultar em uma solução fracionária em termos de arcos, já que as rotas são não-elementares. Nesse caso, com o intuito de obter uma solução inteira também em termos de arcos, realiza-se *branching* sobre as variáveis x_a^l até que uma solução inteira seja alcançada.

Fixar somente a variável x_a também não garante que a solução será inteira em termos de x_a^l, já que pode existir uma rota que percorra o mesmo arco repetidas vezes, porém em instantes diferentes. Apesar disso, realizar *branching* sobre as variáveis x_a é interessante pois o número de variáveis é L vezes menor do que o número de variáveis x_a^l o que em geral acelera o término do algoritmo *BCP*.

O critério de seleção de variável escolhida para ser ramificada foi a variável cujo valor for mais próximo de 0.5. Em caso de empate, a escolha é arbitrária entre as variáveis empatadas. O mesmo critério é adotado tanto para variáveis y_v como para as variáveis x_a e x_a^l.

3.6 Resultados

Esta seção lista e tece comentários acerca dos resultados obtidos com a abordagem exata.

Os resultados relacionados à abordagem exata com os limites superiores obtidos e as soluções inteiras encontradas pelo algoritmo BCP são listados nas Tabelas 3.1, 3.2, 3.3, 3.4 e 3.5. As Tabelas de 3.1 a 3.4 têm as seguintes colunas:

- *Instance* é o nome do arquivo da instância;
- m é o número de veículos;
- L é a duração máxima das rotas;
- LB é o melhor limite inferior, isto é, o valor da melhor solução para a instância do TOP;
- CG contém o valor da relaxação linear resolvida por geração de colunas;
- $ROOT\ UB$ apresenta o limite superior obtido no nó raiz quando as desigualdades *Min Cut* são separadas;
- *Our UB* é o valor do melhor limite superior encontrado pelo algoritmo *Branch-Cut-and-Price*;
- *BFG UB* é o limite superior apresentado em Boussier et al. (BFG07);
- *Our IS* é o valor da melhor solução inteira pelo algoritmo apresentado nesta tese;
- *BFG IS* é a solução inteira encontrada por Boussier et al. (BFG07);
- CGT é o somatório do tempo gasto em cada execução do subproblema de geração e colunas;
- CT é o somatório do tempo gasto em cada execução dos algoritmos de separação de cortes;
- OT é o tempo gasto na resolução de problemas de programação linear - problema mestre restrito;
- NN indica o número de nós explorados no algoritmo *Branch-Cut-and-Price* sobre as variáveis de visitação dos vértices;
- NN_BB é o número de nós explorados no algoritmo *Branch-Cut-and-Price* sobre as variáveis que indicam a utilização dos arcos;

A Tabela 3.5 tem uma coluna extra indicando o número de vértices da instância.

A análise da abordagem exata será concentrada nas Tabelas 3.1, 3.2 e 3.5, já que as instâncias listadas nas Tabelas 3.3 e 3.4 parecem ser de mais fácil resolução.

Nessas tabelas, com instâncias de 64, de 66 e de 102 vertices, é importante destacar que com a adição dos cortes *min-cut* consegue-se melhorar os limites superiores obtidos na geração de colunas.

Além disso, a solução ótima inteira foi provada pela abordagem apresentada nesta tese em 16 de 22 instâncias. Vale destacar que o tempo de CPU gasto foi consistentemente inferior a 3 minutos. Esse tempo é compatível com

os tempos de execução do algoritmo *Branch-and-Price* proposto por Boussier et al. (BFG07).

A Tabela 3.1 apresenta os resultados para instâncias com 33 vértices. No caso com 4 (quatro) veículos os limites superiores encontrados foram idênticos aos apresentados por Boussier et al. (BFG07), com apenas uma instância com limite acima do valor da solução ótima. Nos casos com 2 e 3 veículos abordagem deste trabalho obteve melhores limites superiores em 7 de 9 instâncias. Novamente, a separação dos cortes melhorou significativamente os limites superiores.

Os resultados apresentados para as instâncias de 100 vértices são apresentados na Tabela 3.2 e são similares. Nas instâncias nas quais não há empate com o algoritmo *branch-and-price* de Boussier et al. (BFG07), os resultados obtidos neste trabalho são melhores em 5 de 6 instâncias. Isso foi especialmente verdadeiro para as instâncias mais difíceis p4.4.j e p4.4.k. Novamente, nesses casos a separação dos cortes foi crucial. Finalmente, a Tabela 3.5 apresenta os resultados para as instâncias mais difíceis, para as quais Boussier et al. (BFG07) não encontraram soluções inteiras.

3.7 Conclusão

O algoritmo *Branch-Cut-and-Price* apresentado neste capítulo é a principal contribuição desta tese pelos seguintes motivos:

- É o primeiro algoritmo *Branch-Cut-and-Price* da literatura voltado para o TOP. O trabalho mais próximo é o *Branch-and-Price* de Boussier et al. (BFG07) com o qual o algoritmo aqui proposto foi comparado. Os cortes *min cut* são também uma novidade, tanto por ser uma nova família de desigualdades válidas que podem ser aplicadas a outros problemas, como também por ser uma novidade inserir cortes em formulações do TOP.
- A abordagem obteve resultados significativos, haja vista que novos limites superiores foram encontrados. Algumas soluções inteiras de boa qualidade também foram encontradas, capazes inclusive de competir com os melhores limites inferiores encontrados por heurísticas como as apresentadas em Archetti et al. (AHS07).
- O BCP aqui apresentado tem papel fundamental no desempenho da abordagem heurística de buscas em grandes vizinhanças (circulares e elipsoidais), apresentada no capítulo 5.

Algorithm 2: Recupera Rotas

Input: $finalStates$ é a lista de estados finais.
Output: generated routes Q^p

```
1 foreach state ∈ finalStates do
2     Route route;
3     while state.father != NULL do
4         route.add(state.point);
5         state = state.father;
6         Q^p ⟵ Q^p+ route;
```

Algorithm 3: Algoritmo de Separação - MIn Cut

Input: X é o conjunto das variáveis x_a^l da solução fracionária, onde $a = (i, j)$, $i, j \in V$.
Input: Y é o conjunto das variáveis y_v da solução fracionária, onde $v \in V$.
Input: L é o tempo máximo de duração das rotas.
Output: List<Cut> são os cortes gerados

```
1  List<Vertice> V^s;
2  List<Arc> A^s;
3  id ⟵ 0;
4  foreach x_a^l ∈ X do
5      t ⟵ ConsumedTime(i, j);
6      if x_a^l > 0 then
7          Vertice v_i ⟵ CreateVertice(i,l);
8          id++;
9          V^s ⟵ V^s + v_i;
10         Vertice v_j ⟵ CreateVertice(j,l − t);
11         id++;
12         V^s ⟵ V^s + v_j;
13         capacity = x_a^l;
14         Arc a^s ⟵ CreateArc(v_i,v_j, capacity);
15         A^s ⟵ A^s + a^s;
16 Vertice v_t ⟵ CreateExtraVertice();
17 foreach y_v ∈ V do
18     if y_v = 0 then
19         continue;
20     lineId ⟵ v.id;
21     foreach 0 ≤ l ≤ L do
22         Vertice v_id^l ⟵ GetVertice(lineId,l);
23         if v_id^l ∈ V^s then
24             capacity ⟵ ∞ ;
25             CreateArc(v_id^l, v_t, capacity);
26     V^s ⟵ V^s + v_0;
27     Graph G^s ⟵ CreateGraph(V^s, A^s);
28     maxFlowValue ⟵ maxFlowAlgorithm(v_0, v_t, G^s);
29     if maxFlowValue ¡ y_v then
30         List<Arc> cutArcs ⟵ GetMinCutArcs(); Cut cut;
31         cut.addArcs(cutArcs);
32         cuts.add(cut);
33 return cuts;
```

Tabela 3.1: Resultados para instâncias com 33 vértices

Instance	m	L	LB	CG	ROOT UB	Our UB	BFG UB	Our IS	BFG IS	CGT	CT	OT	NN	NN_BB
p3.2.h	2	25	410	430.645	414.167	**410**	417.5	**410**	**410**	77.72s	1.20s	0.10s	3	1
p3.2.k	2	32.5	550	575.088	554	**550**	566.667	**550**	**550**	316.20s	3.45s	0.55s	5	1
p3.3.j	3	20	380	403.333	396.667	**380**	390	-	**380**	75.62s	1.05s	0.16s	9	1
p3.3.l	3	23.3	480	503.333	491.429	**485**	488	**480**	**480**	151.26s	2.99s	0.25s	8	5
p3.3.m	3	25	520	537.5	520.26	**520.26**	526.667	**520**	**520**	132.45s	2.68s	0.18s	4	3
p3.4.k	4	16.2	350	350	350	**350**	**350**	**350**	**350**	2.12s	0.00s	0.01s	1	1
p3.4.l	4	17.5	380	395	384.5	**380**	**380**	350	**380**	30.51s	1.02s	0.10s	7	1
p3.4.m	4	18.8	390	405	400	**390**	**390**	380	**390**	40.83s	1.26s	0.10s	7	1
p3.4.n	4	20	440	461.667	446.667	**446.667**	**446.667**	**440**	**440**	51.24s	1.20s	0.11s	6	48
p3.4.p	4	22.5	560	566.667	560	**560**	**560**	**560**	**560**	18.32s	0.37s	0.05s	1	1

Tabela 3.2: Resultados para instâncias com 100 vértices

Instance	m	L	LB	CG	ROOT UB	Our UB	BFG UB	Our IS	BFG IS	CGT	CT	OT	NN	NN_BB
p4.3.e	3	30	468	470.1	470.1	469.2	**468.75**	-	**468**	441.94s	2.44s	0.23s	12	1
p4.4.e	4	22.5	183	183	183	**183**	**183**	**183**	**183**	0.70s	0.00s	0.00s	1	1
p4.4.f	4	25	324	324	324	**324**	**324**	**324**	**324**	4.16s	0.00s	0.03s	1	1
p4.4.j	4	35	732	749.41	743.562	**736.408**	741.472	-	**732**	2044.74s	35.16s	1.21s	8	51
p4.4.k	4	37.5	821	841.799	832.3	**825.411**	831.945	-	**821**	5357.18s	24.37s	3.52s	12	1

Tabela 3.3: Resultados para instâncias com 66 vértices

Instance	m	L	LB	CG	ROOT UB	Our UB	BFG UB	Our IS	BFG IS	CGT	CT	OT	NN	NN_BB
p5.2.b	2	5	20	20	20	**20**	**20**	**20**	**20**	0.17s	0.00s	0.00s	1	1
p5.2.c	2	7.5	50	50	50	**50**	**50**	**50**	**50**	0.23s	0.00s	0.00s	1	1
p5.2.d	2	10	80	80	80	**80**	**80**	**80**	**80**	0.42s	0.00s	0.00s	1	1
p5.2.e	2	12.5	180	180	180	**180**	**180**	**180**	**180**	1.31s	0.00s	0.01s	1	1
p5.2.g	2	17.5	320	320	320	**320**	**320**	**320**	**320**	10.55s	0.00s	0.01s	1	1
p5.3.m	3	21.7	650	650	650	**650**	**650**	**650**	**650**	28.11s	0.00s	0.02s	1	1
p5.3.n	3	23.3	755	755	755	**755**	**755**	**755**	**755**	51.63s	0.00s	0.03s	1	1
p5.3.p	3	26.7	990	990	990	**990**	**990**	**990**	**990**	80.34s	0.00s	0.03s	1	1
p5.4.t	4	25	1160	1160	1160	**1160**	**1160**	**1160**	**1160**	70.11s	0.00s	0.04s	1	1
p5.4.v	4	27.5	1320	1320	1320	**1320**	**1320**	**1320**	**1320**	112.82s	0.00s	0.07s	1	1

Tabela 3.4: Resultados para instâncias com 64 e 102 vértices

Instance	m	L	LB	CG	ROOT UB	Our UB	BFG UB	Our IS	BFG IS	CGT	CT	OT	NN	NN_BB
p6.2.f	2	20	588	588	588	**588**	**588**	**588**	**588**	11.41s	0.00s	0.02s	1	1
p6.2.g	2	22.5	660	660	660	**660**	**660**	**660**	**660**	17.92s	0.00s	0.01s	1	1
p6.4.j	4	15	366	366	366	**366**	**366**	**366**	**366**	0.91s	0.00s	0.01s	1	1
p6.4.k	4	16.2	528	528	528	**528**	**528**	**528**	**528**	1.74s	0.00s	0.01s	1	1
p6.4.n	4	20	1068	1068	1068	**1068**	**1068**	**1068**	**1068**	31.68s	0.00s	0.04s	1	1
p7.3.f	3	40	247	247	247	**247**	**247**	**247**	**247**	3.96s	0.00s	0.02s	1	1
p7.3.g	3	46.7	344	349	344	**344**	**344**	**344**	**344**	22.61s	0.33s	0.02s	1	1
p7.4.i	4	45	366	369	366	**366**	**366**	**366**	**366**	18.34s	0.23s	0.02s	1	1
p7.4.j	4	50	462	476.192	462	**462**	**462**	**462**	**462**	53.19s	1.41s	0.05s	1	1

Tabela 3.5: Resultados para instâncias mais difíceis

Instance	n	m	L	LB	CG	ROOT UB	Our UB	BFG UB	Our IS	BFG IS	CGT	CT	OT	NN	NN_BB
p1.2.p	32	2	37.5	240	251.594	250	**250**	**250**	**250**	-	79.14s	2.80s	0.08s	1	1
p3.2.l	33	2	35	570	606.606	605.629	**590**	605	**590**	-	4012.61s	11.92s	6.46s	16	1
p3.3.s	33	3	35	710	748.311	743.538	**720**	738.913	**720**	-	10696.96s	13.00s	11.52s	20	1
p3.3.t	33	3	36.7	720	777.104	760.169	**754.286**	763.688	**740**	-	2688.65s	15.82s	50.22s	30	2
p5.4.s	66	4	23.8	960	1069.17	1069.17	1060	**1055**	**1030**	-	2252.82s	17.17s	0.47s	41	12