5

O Algoritmo de *Branch-and-Cut-and-Price*

Nesse capítulo, apresenta-se em detalhes o algoritmo de *branch-and-cut-and-price* que foi implementado para o CVRP, a partir da formulação descrita na seção anterior. Descreve-se cada uma das partes do algoritmo de BCP: geração de colunas, geração de cortes, representação de variáveis e restrições, regra de *branching*, seleção de nós e limites superiores e, por fim, escolha dinâmica da geração de colunas.

Porém, antes de apresentar cada uma das partes do algoritmo em separado, é mostrado o funcionamento, de forma resumida, do algoritmo de BCP como um todo. O algoritmo de geração de colunas e cortes implementado para o CVRP funciona conforme o pseudocódigo a seguir.

1. Resolver o LP restrito (LP com apenas um subconjunto do número total de colunas).
2. Resolver subproblema de geração de colunas obtendo variáveis de custo reduzido negativo.
3. Adicionar as colunas obtidas em 2 ao LP restrito.
4. Se alguma coluna foi adicionada ao LP restrito, resolver o novo LP restrito.
5. Aplicar algoritmo de separação à solução obtendo as desigualdades violadas.
6. Adicionar as desigualdades encontradas em 5 ao LP.
7. Se alguma coluna foi adicionada em 3 ou algum corte foi adicionado em 6, retorne para 1. Caso contrário, pare.

O pseudocódigo anterior mostra o procedimento que é aplicado a cada nó da árvore de *branch-and-bound*. Maiores detalhes sobre cada etapa do algoritmo são apresentados nas próximas seções desse capítulo.

5.1 Geração de Colunas

O custo reduzido de uma coluna (variável λ, em (DWM)) é a soma dos custos reduzidos das arestas na q-rota correspondente. Sejam μ, υ, π, e ω as variáveis duais associadas com as restrições (4-34), (4-35), (4-36), e (4-37), respectivamente. O custo reduzido $\bar{c}_e$ de uma aresta e é dado por:

$$\bar{c}_e = \begin{cases} \ell_e - \mu_i - \mu_j - \sum\limits_{S|\delta(S)\ni e} \pi_S - \omega_e, & e = \{i,j\} \in E\backslash\delta(\{0\}) \\ \ell_e - \upsilon - \mu_j - \sum\limits_{S|\delta(S)\ni e} \pi_S, & e = \{0,j\} \in \delta(\{0\}) \end{cases}$$

Um corte adicional genérico $\sum_{j=1}^{p} (\sum_{e\in E} a_e q_j^e) \cdot \lambda_j \geq b$ em (DWM), com variável dual associada α, contribui com o valor de $-a_e \cdot \alpha$ para o cálculo de $\bar{c}_e$.

O sub-problema de geração de colunas consiste em encontrar q-rotas (colunas) de custo reduzido mínimo. Apesar desse problema ser NP-difícil em geral, ele pode ser resolvido em tempo pseudo-polinomial se todas as demandas forem inteiras [21].

5.1.1 Algoritmo básico

A estrutura de dados básica para o algoritmo é uma matriz M de dimensão $C\times n$. Cada entrada $M(d,v)$ representa o caminho mais barato que chega ao vértice v consumindo uma demanda de exatamente d. A entrada contém um *rótulo* consistindo do vértice (v), o custo do caminho, e um apontador para o rótulo que representa o caminho mais barato até o vértice anterior. A matriz é preenchida através de programação dinâmica, linha por linha, começando com $d = 1$. Para cada linha d, o algoritmo percorre cada entrada v e, para cada vizinho w de v, avalia o tamanho do caminho representado por $M(d,v)$ até w. Se $\bar{c}(M(d,v))+\bar{c}(v,w) < \bar{c}(M(d+d(w),w))$, $M(d+d(w),w)$ é atualizado.

Para armazenar os resultados é utilizado um arranjo de tamanho n, onde cada elemento é um rótulo, chamado *best*. O elemento $best(v)$ guarda o caminho mais negativo encontrado pelo algoritmo até o momento. Mais precisamente, logo antes do algoritmo começar a processar a d-ésima linha de M, $best(v)$ representa o caminho com custo mais negativo cuja demanda acumulada é *estritamente menor* que d (pode ter qualquer valor entre 1 e $d-1$). No fim, cada entrada $best(v)$ (para todos os vértices v) representa o caminho mais negativo com demanda acumulada de no máximo C que chega em v.

Estendendo o caminho resultante até o depósito (cuja demanda é zero), a q-rota correspondente é obtida. Todas as q-rotas negativas encontradas (serão no máximo n) são adicionadas ao programa linear. Existem nC entradas na matriz, e cada uma é processada em tempo $O(n)$, logo a complexidade do algoritmo é $O(n^2C)$.

5.1.2

Eliminação de Ciclos

Uma forma natural de fortalecer a formulação é encontrar q-rotas sem ciclos, ou seja, q-rotas em que cada vértice apareça no máximo uma vez. No fim, todas as rotas presentes na solução ótima do CVRP têm essa propriedade. Infelizmente, não existe nenhuma maneira de resolver esse problema em tempo pseudopolinomial. Ao invés disso, o que é feito para fortalecer a formulação é a eliminação apenas de s-ciclos (ciclos com no máximo s arestas) para valores pequenos de s.

O algoritmo funciona como descrito anteriormente, usando programação dinâmica para preencher a matriz $C \times n$ com caminhos parciais. Os rótulos possuem o mesmo significado de antes, porém, agora, cada entrada da matriz não contém mais um único rótulo, mas um *repositório* de rótulos. O repositório $M(d, v)$ não representa apenas o caminho sem s-ciclos mais curto com demanda de d até o vértice v, mas também caminhos alternativos que garantem que todas as possíveis extensões de s vértices começando em v com demanda acumulada de d são consideradas. Além disso, *best* é agora um arranjo de repositórios ao invés de um arranjo de rótulos.

O algoritmo segue o mesmo princípio de antes. Processa-se uma linha de cada vez (indo das com menor demanda acumulada para as com maior demanda acumulada). Em cada linha, processa-se cada vértice v,

estendendo-se cada caminho representado no repositório de v (cada extensão cria um novo rótulo). No algoritmo original, o caminho foi estendido a todos os possíveis vértices exceto v (o último vértice no caminho); agora *extensões para qualquer um dos últimos s vértices são proibidas*. Como um rótulo contém apenas o último vértice, deve-se seguir os vértices em *pred* (predecessores de v no caminho) para recuperar os outros $s-1$.

Isso explica o motivo de ser armazenado mais de um rótulo por repositório. Assuma que o caminho de menor custo com demanda acumulada d que chega em v termina em $u \rightarrow v$. Estender esse caminho até u iria criar um 2-ciclo, então a extensão não é feita. No entanto, pode ser o caso em que a q-rota de menor custo sem s-ciclos possui um subcaminho com demanda acumulada d no nó v que vai para u. A menos que seja armazenado pelo menos mais um rótulo no mesmo repositório (com diferente predecessor), nunca será possível encontrar esse caminho. Decidir que caminho manter é trivial para $s = 2$: basta manter o melhor caminho que não vem de u. Para $s \geq 3$, a seleção é bem mais complicada; neste trabalho usa-se uma abordagem semelhante à de Irnich e Villeneuve [41].

Dominância

Em geral, os rótulos armazenados em um repositório devem ser tais que qualquer possível s-extensão seja considerada. Em outras palavras, para cada possível s-extensão, deve-se garantir que o rótulo mais barato que permite essa extensão seja selecionado. Essa noção pode ser formalizada com o conceito de *dominância*. Diz-se que um certo rótulo ℓ é s-dominado por um conjunto L de rótulos se existe um caminho p de tamanho s que pode estender ℓ mas não pode estender nenhum rótulo em L. Uma seqüência de rótulos é *s-minimal* se nenhum elemento na seqüência é dominado pelo conjunto de todos predecessores (isso implica que nenhum elemento é supérfluo). O lema 1 e a conjectura 1 apresentados na próxima seção foram propostos em [41].

Lema 1 *Existe uma seqüência s-minimal com* $s!$ *rótulos.*

Prova. Tome o conjunto de todos os caminhos de tamanho $s+1$ que passam pelos vértices $1, 2, \ldots, s$ (em qualquer ordem) e terminam em v (assumindo $v > s$). Existem $s!$ caminhos como esse. Cada um desses caminhos possui uma extensão que não pode ser aplicada a qualquer um dos outros $s! - 1$

caminhos. Considere algum caminho $P = p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_s \rightarrow v$, onde $(p_1, p_2, \ldots, p_s)$ é uma permutação de $(1, 2, \ldots, s)$. Existe uma extensão válida de P (sem s-ciclos) que não é válida para nenhum dos outros $s! - 1$ possíveis caminhos: o próprio $p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_s$. Como p_1 é o primeiro vértice da extensão, o caminho original não pode ter p_1 entre seus últimos s elementos; como p_1 aparece em algum lugar, esse lugar deve ser na primeira posição. Então restringe-se a caminhos que comecem com p_1. Como p_2 é o segundo elemento da extensão, ele não pode estender caminhos que possuam p_2 entre seus últimos $s - 1$ elementos; portanto, restringe-se a caminhos que possuam p_2 como segundo elemento (p_2 também poderia ser o primeiro elemento, porém essa posição já é ocupada por p_1). O mesmo argumento mostra que o terceiro elemento deve ser p_3, o quarto p_4, e assim por diante. Logo o único caminho que pode estender $p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_s$ é $p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_s \rightarrow v$, como dito antes. □

Conjectura 1 *Nenhuma seqüência s-minimal tem mais do que s! rótulos.*

Considere uma seqüência $\sigma = \ell_1, \ell_2, \ell_3, \ldots, \ell_t$ (para algum t). Devemos construir uma subseqüência σ^+ de σ tal que $\ell_i \in \sigma^+$ se e somente se l_i não é dominado pelo conjunto $\{l_1, \ldots, l_{i-1}\}$. Inicialmente, σ^+ contém apenas ℓ_1. Então percorre-se σ verificando se l_i (o rótulo corrente) é dominado pelo conjunto atualmente em σ^+. Se é dominado, apenas descarta-se l_i, caso contrário, adiciona-se l_i a σ^+.

Isso significa que o problema se reduz a verificar dominância: deve-se saber determinar se um rótulo é dominado por seus predecessores ou não. Isso pode ser feito através de um algoritmo relativamente simples, mas que requer uma representação especial do conjunto de rótulos. A seguir, primeiro descreve-se a representação utilizada e, depois, o algoritmo em si.

Representação

Considere um repositório representando uma seqüência de caminhos que terminam em um certo vértice. Cada rótulo no repositório pode ser estendido de diversas formas. A união de todas as extensões permitidas por todos os rótulos é o conjunto de extensões válidas do repositório como um todo. Deseja-se encontrar uma forma compacta de representar esse conjunto. Por conveniência, representa-se o *complemento* desse conjunto, ou

seja, todas extensões que não podem estender nenhum rótulo do repositório. Irnich e Villeneuve [41] chamam esse conjunto de extensões proibidas de uma seqüência de rótulos $\Im$ de *hole set* de $\Im$.

Representa-se o *hole set* de um repositório por uma *árvore de cláusulas*. Cada nó nessa árvore representa uma *cláusula*. Folhas são *cláusulas terminais*, que podem ser tanto *cláusulas verdadeiras* como *cláusulas falsas*. Nós terminais correspondem a *cláusulas padrão*. Uma cláusula padrão $\mathcal{C}$ possui pelo menos dois filhos:

- Um ou mais filhos *rotulados*, cada um associado a um rótulo de um vértice $v(1 \leq v \leq n)$. Chama-se v de *guarda* do filho correspondente, e denota-se por $\mathcal{C}^v$. Constantemente os filhos rotulados serão referidos como *cláusulas guardadas*. Elas são mantidas em uma lista ordenada pelo guarda.
- Exatamente um filho sem rótulo. Esse filho é a *cláusula padrão* $\mathcal{C}^+$; Ele pode ser interpretado como uma cláusula guardada por todos os rótulos de vértices que não aparecem explicitamente como guardas de filhos rotulados.

Usa-se um exemplo para explicar como cláusulas são usadas para representar *hole sets*. Considere o caminho $7 \rightarrow 9 \rightarrow 6 \rightarrow v$, e assuma que tenta-se eliminar 4-ciclos. Como precisa-se fazer distinção apenas entre rótulos associados a v, não precisamos representar v explicitamente. Como para outros vértices, impõe-se as seguintes restrições: nenhuma extensão pode começar com 7, 9 ou 6; nenhuma extensão pode ter 9 ou 6 como seu segundo vértice; e nenhuma extensão pode ter 6 como seu terceiro vértice. Qualquer outra extensão é permitida. Isso pode ser representado como na figura 5.1.

A árvore de cláusulas pode ser usada para determinar se uma determinada extensão é válida ou não. Usa-se cada um dos elementos da extensão para encontrar o filho apropriado em cada nível até atingir-se uma folha. Se a folha é falsa, a extensão não é permitida; se ela é verdadeira, a extensão é válida.

Tome, por exemplo, uma possível extensão $(5, 7, 4)$. O primeiro elemento é representado no nó raiz; 5 não ocorre explicitamente, então vai-se para o filho padrão. O novo nó representa a segunda posição no caminho (7, no caso). Como 7 é representado como um filho rotulado, segue-se o link

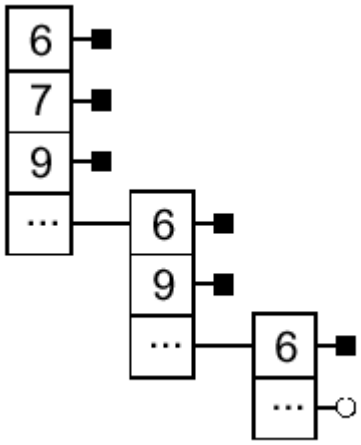

Figura 5.1: Árvore de cláusulas representando o caminho terminando em $7 \rightarrow 9 \rightarrow 6 \rightarrow v$. Nós internos, representados por retângulos numerados, são as cláusulas padrão. Os números são os guardas, e o filho padrão é indicado por três pontos(...). Cláusulas falsas são representadas por quadrados preenchidos, e cláusulas verdadeiras por círculos vazios.

correspondente. Isso leva a uma cláusula falsa, o que indica que o caminho não é válido.

Agora considere a extensão $(4, 5, 9)$. O primeiro elemento é 4. Como ele não está explicitamente representado na raiz, segue-se a cláusula padrão. Deve-se verificar agora se o segundo elemento (5) é representado nesse nó; ele não é representado, logo deve-se seguir a cláusula padrão de novo. Verifica-se o elemento 9, que também não é representado. Seguindo o link padrão, chega-se a uma cláusula verdadeira, o que significa que $(4, 5, 9)$ é uma extensão válida do conjunto corrente de rótulos.

União

Representa-se a união de dois *hole sets* como a união de suas correspondentes árvores de cláusulas. A união de duas árvores de cláusulas é definida de forma natural. Primeiro, a união de uma cláusula verdade com qualquer outra cláusula é uma cláusula verdadeira. Já a união de uma cláusula falsa com qualquer outra cláusula tem como resultado a outra cláusula. O caso não-trivial é a união de duas cláusulas padrão. Considere duas cláusulas S e T, e seja U a união das duas. Existirá uma cláusula U^v (uma cláusula guardada por v em U) se e somente se existir uma cláusula guardada por v em S, T, ou ambos. Mais precisamente, tem-se o seguinte:

– Se v é um guarda de ambos S e T, então $U^v = S^v \cup T^v$.

– Se v é um guarda somente de S, então $U^v = S^v \cup T^+$.

– Se v é um guarda somente de T, então $U^v = S^+ \cup T^v$.

Pode parecer que a árvore de cláusulas resultante terá pelo menos o mesmo número de nós das árvores de cláusulas originais. Isso não é necessariamente verdade, já que U é submetido a uma etapa final de processamento:

1. (eliminação de guarda) Se uma cláusula guardada é idêntica a uma cláusula padrão, elimina-se a cláusula guardada.

2. (eliminação de nível) Se não existe cláusula guardada e a cláusula padrão é verdadeira, substitui-se a cláusula inteira U por uma cláusula verdade.

Esses testes devem ser aplicados não somente a cláusula U, mas também devem ser aplicados recursivamente a cada uma de suas sub-cláusulas. Portanto, eles asseguram que as cláusulas estarão sempre na *forma normal*, com o número de nós estritamente necessário.

5.1.3

Heurísticas

São usadas três heurísticas para acelerar o algoritmo. Elas encontram apenas q-rotas sem s-ciclos, logo qualquer rota negativa encontrada pode ser adicionada à formulação. Chama-se de rota negativa uma rota cuja soma das arestas é negativa (custo reduzido). No entanto, se nenhuma rota negativa é encontrada, o algoritmo exato deve ser executado para verificar se nenhuma existe ou, então, encontrar uma.

Redução de granularidade

Quando a redução de granularidade é usada, são consideradas apenas $g > 1$ unidades de demanda de cada vez. O algoritmo exato é executado com uma demanda modificada $d'_v(g) = \lceil d_v/g \rceil$ para todo vértice v, e uma capacidade modificada $C' = \lfloor C/g \rfloor$ para os veículos. O tempo de execução será proporcional a C' ao invés de C, porém as soluções encontradas permanecerão válidas na configuração original.

Esparsificação

Intuitivamente, arestas pequenas no grafo original são mais prováveis de aparecerem na solução. Levando em conta isso, é pré-computado um conjunto de cinco florestas geradoras disjuntas usando uma extensão do algoritmo de Kruskal similar a heurística gulosa de arestas descrita em [76]. O algoritmo ordena todas arestas (exceto as incidentes no depósito) em ordem crescente de custo; cada aresta é inserida na primeira floresta em que ela não cria ciclo. O algoritmo pára quando todas as cinco florestas se tornam árvores, ou quando não existem mais arestas para inserir. Considerando somente arestas dessas árvores e arestas incidentes ao depósito, a programação dinâmica é limitada a um conjunto restrito de vizinhos quando cada vértice é processado.

Vale a pena ressaltar que o tamanho original das arestas é levado em conta como custo de cada aresta e não o custo reduzido (variável dual associada à aresta). Apesar dos custos reduzidos poderem ser uma medida melhor do quanto uma aresta será útil durante uma rodada de geração de colunas, usar o tamanho original das arestas tende a ser melhor quando considera-se um conjunto maior de iterações. Como o objetivo é encontrar o conjunto "correto"de colunas que pertencem a solução, usar os tamanhos originais das arestas tende a reduzir o número de iterações de geração de colunas.

Poda de repositórios

A última heurística de aceleração consiste em guardar menos de $s!$ rótulos por repositório. O problema potencial disso é que algumas extensões válidas serão bloqueadas. Para minimizar esse problema, é levado em conta não somente o custo, mas também a diversidade quando é decidido que

rótulos guardar. Dada uma seqüência de rótulos ordenados pelo custo, é tomada uma subseqüência que possui pelo menos dois vértices diferentes em cada posição. O primeiro rótulo é sempre escolhido, e os outros são mantidos somente se eles possuem um vértice diferente em qualquer uma das posições que possuem somente um vértice até o momento. Essa regra nunca toma mais que s rótulos.

Parâmetros

O algoritmo começa usando todas as três técnicas de aceleração descritas. Quando falha em encontrar q-rotas negativas, uma das acelerações é desativada (primeiro redução de granularidade, depois esparsificação e por fim poda de repositórios). Se o algoritmo falha em encontrar uma rota de custo reduzido negativo, pode-se afirmar de forma segura que nenhuma q-rota negativa existe. Por outro lado, se somente um subconjunto de heurísticas está em uso e é encontrada uma q-rota negativa, outras heurísticas são ativadas novamente.

Além disso, a granularidade do algoritmo é auto-ajustada. Após uma execução do procedimento de geração de colunas com uma certa granularidade g, é computada sua *taxa de sucesso* r. Essa taxa é uma comparação entre o número de caminhos negativos encontrados e n (o número total de caminhos que podem ser potencialmente encontrados). A nova granularidade é definida por:

$$g^{'} = g(0.1 + r)$$

Também é garantido que a granularidade é um inteiro que nunca é maior que $n/4$ e nunca menor que 1. Se a fórmula acima não retorna um valor na faixa de valores permitida, $g^{'}$ é fixado ao valor extremo mais próximo. Se $g = 1$, isso significa que a redução de granularidade não é usada.

5.2

Geração de Cortes

Inicialmente, o LP para o CVRP inclui somente as restrições de grau. Cortes adicionais violados são adicionados durante a execução do algoritmo. Além dos cortes de capacidade (4-13) e dos cortes de limite(4-

14), também são utilizados (utilizaremos os nomes em inglês, seguindo o padrão utilizado em [46]): *framed capacity*, *strengthened comb*, *multistar*, *partial multistar*, *generalized multistar* e *hypotour*, todos utilizados em [46]. Essas desigualdades não são triviais e os algoritmos de separação para as mesmas são geralmente bastante complicados. Como não é objetivo dessa dissertação implementar algoritmos de separação para essas desigualdades utiliza-se o pacote CVRPSEP[52] para separar os cortes violados para cada uma das famílias de desigualdades citadas. O pacote CVRPSEP implementa os algoritmos de separação utilizados no algoritmo de *branch-and-cut* de [46]. Como os cortes são definidos em termos das variáveis x_e^* da formulação no formato Mestre Explícito e trabalha-se com a formulação (DWM), converte-se as variáveis λ_j^* para as correspondentes variáveis x_e^* (e vice-versa) quando necessário. A seguir, apresenta-se uma visão das famílias de desigualdades utilizadas no algoritmo de BCP para o CVRP. Para maiores detalhes sobre as mesmas é sugerida a leitura de [46].

5.2.1

Cortes de Capacidade (*Capacity Cuts*)

As desigualdades de capacidade (4-3) são as mais conhecidas dentre as desigualdades para o CVRP. A idéia dessa desigualdade é para um dado subconjunto S de clientes, garantir que eles sejam servidos pelo número de veículos necessários. Calcular o lado direito da desigualdade exatamente requer a resolução de um problema de empacotamento em bins (BPP). No entanto, a desigualdade permanece válida se $r(S)$ no lado direito da desigualdade for substituído pelo limite inferior para o BPP $\lceil d(S)/C \rceil$. Mesmo com essa substituição, a separação das desigualdades de capacidade é fortemente NP-difícil [1]. Em [46], são utilizadas heurísticas para realizar a separação. No total são utilizadas quatro heurísticas para separar as desigualdades. Para uma descrição de como as heurísticas funcionam, veja [46].

5.2.2

Framed Capacity

Em Augerat(1995)[8], as desigualdades conhecidas como *framed capacity* foram apresentadas. Para algum conjunto $S \subseteq V_C$, seja $\Omega =$

$\{S_1, \cdots, S_p\}$ uma partição de S. Seja $k(S, \Omega)$ igual ao número mínimo de veículos necessários para servir S dado que a desigualdade de capacidade para cada S_i vale como igualdade. Um bom limite inferior para $k(S, \Omega)$ pode ser encontrado resolvendo um problema de empacotamento em bins(BPP). As desigualdades *framed capacity* (FCI) possuem a seguinte forma:

$$x(\delta(S)) + \sum_{i=1}^{p} x(\delta(S_i)) \geq 2k(S, \Omega) + 2\sum_{i=1}^{p} k(S_i).$$

Intuitivamente, essa desigualdade funciona da seguinte forma: se todas as desigualdades de capacidade para todos os conjuntos S_i são satisfeitas na igualdade, então o somatório do lado esquerdo é igual ao somatório do lado direito, e então $x(\delta(S))$ deve ser pelo menos $2k(S, \Omega)$, como requerido. Note que, se qualquer S_i é simples (só um elemento), então os i-ésimos termos nos somatórios podem ser omitidos devido às restrições de grau. Para o caso especial onde $S = V_C$, a desigualdade é chamada de *generalized capacity* em [59]. Um procedimento de busca em profundidade é utilizado para separar as FCIs. Esse procedimento é descrito em [46].

5.2.3

Strengthened Comb

As desigualdades *comb* [37, 38] são conhecidas para o TSP. Muitos autores tentaram adaptá-las para o CVRP [1, 9].

Uma *comb* é formada por: um conjunto de vértices $H \subset V_C$, chamado *handle*, e outros conjuntos de vértices $T_1, \cdots, T_t$ com $t \geq 2$ chamados *teeth*, tal que:

- $H \cap T_j \neq \emptyset$ e $T_j \setminus H \neq \emptyset$ para $j = 1, \cdots, t$;
- para cada par $\{i, j\} \subset \{1, \cdots, t\}$, ou $T_i \cap T_j \subset H$ ou $T_i \cap T_j \cap H = \emptyset$ (ou ambos).

Essa definição é mais geral que a equivalente para o TSP, porque permite intersecção de *teeth* e não requer que o número de *teeth* seja ímpar.

Para qualquer conjunto $S \subset V$, seja $\widetilde{k}(S)$ igual a $k(S)$ se $0 \notin S$ e $k(V \setminus S)$ caso contrário. Então pode-se definir a quantidade:

$$S(H, T_1, \cdots, T_j) := \sum_{j=1}^{t} (\widetilde{k}(T_j \cap H) + \widetilde{k}(T_j \setminus H) + \widetilde{k}(T_j)).$$

Se $S(H, T_1, \cdots, T_t$ é ímpar, então a seguinte desigualdade *strengthened comb* é válida para o CVRP:

$$x(\delta(H)) + \sum_{j=1}^{t} x(\delta(T_j)) \geq S(H, T_1, \cdots, T_t) + 1.$$

As desigualdades *strengthened comb* dominam as desigualdades *comb*, exceto para o caso especial mencionado em [9]. Em [46] é descrita uma heurística de separação utilizada para separar as desigualdades *strengthened comb*.

5.2.4

Multistar, Partial Multistar e Generalized Multistar

As desigualdades *Multistar* foram originalmente definidas por Araque, Hall e Magnanti [5] para o CVRP com demandas unitárias. Essas desigualdades têm a mesma forma:

$$\alpha x(E(N)) + \beta x(E(N : S)) \leq \gamma.$$

onde $N \subset V_C$ é chamado *núcleo*, $S \subseteq V_C \setminus N$ é o conjunto de *satélites*, e α, β e γ são constantes que dependem de $|N|$ e $|S|$. Os mesmos autores também introduziram as desigualdades *partial multistar*, que têm a forma:

$$\alpha x(E(N)) + \beta x(E(C : S)) \leq \gamma,$$

onde $C \subset N$ é o conjunto chamado de *vértices conectores*, e, de novo, α, β e γ são constantes que dependem de $|N|$, $|S|$ e $|C|$.

Em Letchford et. al (2002)[45], é mostrado como generalizar essas desigualdades para o caso geral do CVRP, dando origem às chamadas desigualdades *multistar* e *partial multistar* homogêneas. Também são dadas heurísticas de separação para essas desigualdades. Essas heurísticas utilizam

vários métodos gulosos e métodos baseados em fluxo máximo para selecionar para selecionar o núcleo N e, depois, heurísticas gulosas para selecionar os satélites S (e os conectores C para o caso das *partial multistars*).

Um conjunto de desigualdades relacionado, mas não idêntico, foi proposto por Gouveia [34] para o CVRP com demandas gerais:

$$Qx(E(N)) + \sum_{j \in V_C \setminus N} q_j x(E(N : \{j\})) \leq Q|N| - q(N).$$

Essas desigualdades são conhecidas como *generalized large multistar*[3, 45]. O problema de separação para as mesmas pode ser resolvido em tempo polinomial.

5.2.5

Hypotour

Seja $F \subset E$ tal que qualquer solução para o CVRP usa pelo menos uma aresta de F. Então a desigualdade chamada *hypotour* é: $x(F) \geq 1$, que foi introduzida no contexto do CVRP em [8]. Diversas variações foram propostas, incluindo a *extended hypotour*, que também foi introduzida em [8]. Outras variações são apresentadas em [1].

Em [46], apenas uma dessas variações, que é chamada de *2-edges extended hypotour(2EH)*, é considerada. Para um dados $W \subset V_C$ e duas arestas distintas $e_1, e_2 \in \delta(W)$, a 2EH é:

$$x(\delta(W)) + 2x(F) \geq 2x_{e_1} + 2x_{e_2},$$

onde $F \subset E$. Seja $e_i = \{u_i, v_i\}$ para $i = 1, 2$ com $v_1, v_2 \notin W$. Então, v_1 e v_2 são os *terminais* de 2EH.

Uma desigualdade 2EH expressa que em qualquer solução viável para o CVRP com $x(\delta(W)) = 2$ e $x_{e_1} = x_{e_2} = 1$, pelo menos uma aresta de F deve ser usada. Isto é, qualquer rota viável visitando todos os clientes em $W \cup \{v_1, v_2\}$, começando e terminando nos terminais dados, deve usar pelo menos uma aresta de F. A identificação de F é essencialmente baseada no fato de que esse caminho Hamiltoniano por $W \cup \{v_1, v_2\}$ deve ser estendido por dois caminhos para o depósito, de forma que esses caminhos possuam

vértices diferentes exceto por sua intersecção no depósito, e de forma que a demanda total no ciclo criado não exceda a capacidade do veículo.

Em [46], é apresentado um procedimento de separação para 2EH com complexidade polinomial.

5.3 Representação de variáveis e restrições

Para implementar-se um algoritmo de *branch-and-cut-and-price* é necessário que tenha-se uma distinção clara entre variáveis e colunas, e também entre restrições e linhas. Cada variável λ é associada a uma q-rota, que é armazenada em um pool de variáveis. Toda vez que um novo corte é adicionado, é necessário acessar todas as q-rotas correspondentes às colunas no LP corrente para calcular os coeficientes corretos para a linha que será inserida. Similarmente, cada restrição expressa em termos de variáveis x é armazenada em um pool de restrições. Toda vez que uma nova variável é computada, é necessário acessar cada linha no LP corrente para calcular os coeficientes da coluna que será inserida.

Os *pools* de colunas e linhas são tais que permitem a computação do produto interno de uma q-rota e uma restrição expressa em termos de variáveis x de maneira eficiente. Como muitas operações desse tipo devem ser realizadas toda vez que uma nova coluna ou nova linha é adicionada, uma implementação ruim dessa parte pode tornar o algoritmo lento. Os *pools* implementados no algoritmo de BCP são indexados utilizando tabelas *hash* auxiliares. Dada uma aresta e, pode-se determinar de forma muita rápida quais colunas no LP são associadas às q-rotas contendo e e quais linhas são associadas às restrições com coeficientes não-zero em x_e.

5.4 Regra de *Branching*

O algoritmo de BCP começa adicionando colunas à formulação, até que não seja encontrada mais nenhuma coluna com custo reduzido negativo para ser adicionada. Então o algoritmo busca todos os cortes violados e adiciona-os à formulação. Essa iteração é repetida até que a geração de colunas e de cortes falhem em encontrar colunas e cortes, respectivamente.

Então o algoritmo verifica se a solução do nó corrente é inteira ou se a solução do nó é maior do que o melhor limite superior encontrado (para esses casos não se faz *branching*). Caso nenhuma das afirmações seja verdadeira, o algoritmo faz *branching*.

A regra de *branching* utilizada no algoritmo de BCP para o CVRP é uma adaptação da regra utilizada em [46]. Escolhe-se conjuntos candidatos para o *branching* S tais que $2 < x^*(\delta(S)) < 4$ e faz-se o *branching* impondo a disjunção $(x(\delta(S)) = 2) \vee (x(\delta(S)) \geq 4)$. Para escolher os conjuntos candidatos para o *branching*, utiliza-se o procedimento disponível no pacote CVRPSEP [52], que retorna os conjuntos que satisfazem $2 < x^*(\delta(S)) < 4$.

Para escolher qual dos conjuntos retornados será utilizado para realizar o *branching*, utiliza-se a técnica de *strong branching*. Essa técnica consiste em:

- Selecionar um conjunto de possibilidades de fazer o *branching*;
- Avaliar o limite inferior produzido por cada uma das possibilidades;
- Selecionar o *branching* a ser feito considerando algum critério (por exemplo, a regra que produz a melhor média entre o limite inferior dos nós filhos).

Como computar o limite inferior para cada nó filho pode ser muito dispendioso em termos de tempo, o limite inferior é estimado realizando um pequeno número de iterações de geração de colunas. Para escolher em qual conjunto fazer *branching*, apenas p conjuntos candidatos são avaliados ($5 \leq p \leq \max\{10 - profundidade, 5\}$). Isso limita o tempo gasto fazendo *strong branching*, principalmente quando a profundidade da árvore de *branching* é maior. A prioridade na escolha dos conjuntos é dada para valores menores de $|x^*(\delta(S)) - 2.7|/d(S)$. Usa-se 2.7 porque a restrição $x(\delta(S)) = 2$ tende a ter um impacto maior na relaxação do LP que $x(\delta(S)) \geq 4$, podendo levar a um desbalanceamento na árvore de *branch-and-bound*. Utilizando valores mais próximos de 2 do que de 4, melhora-se o impacto de impor a desigualdade $x(\delta(S)) \geq 4$.

5.5 Seleção de nós e limites superiores

Nenhuma heurística foi implementada com o objetivo de encontrar soluções viáveis (limites superiores) para o CVRP. O limite primal utilizado foi o valor da melhor solução conhecida para a instância.

A estratégia de seleção de nós no *branch-and-bound* utilizada foi a busca em profundidade. O critério utilizado para a escolha dessa estratégia foi a quantidade de memória requerida. A busca em profundidade requer menor quantidade de memória do que outras estratégias, como a de busca em largura, por exemplo.

5.6 Escolha Dinâmica para a Geração de Colunas

Em geral, a combinação de geração de cortes e colunas gera ganhos significativos no limite inferior obtido. No entanto, em algumas instâncias, o ganho no valor do limite inferior não é significativo o bastante para justificar o tempo adicional gasto. Para esses casos, o algoritmo de *branch-and-cut* puro possui um desempenho melhor do que o algoritmo de *branch-and-cut-and-price*. Nosso algoritmo de BCP para o CVRP se adapta de forma dinâmica para considerar esses casos.

O algoritmo começa resolvendo o nó raiz utilizando somente *branch-and-cut* puro. Depois o nó raiz é resolvido utilizando *branch-and-cut-and-price* com geração de colunas eliminando ciclos de tamanho 2 ($s = 2$). Depois, o nó é resolvido novamente para $s = 3$. Para o resto da árvore de *branch-and-bound*, o algoritmo utiliza a estratégia com o melhor balanço entre tempo de execução e qualidade do limite inferior.

Mais precisamente, se o tempo gasto usando $s = 2$ é menor do que 10 vezes o tempo gasto sem geração de colunas e a melhoria no limite inferior é melhor do que 0.3%, considera-se que é melhor utilizar a geração de colunas e deve-se escolher entre utilizar $s = 2$ ou $s = 3$, caso contrário utiliza-se o *branch-and-cut* puro. Se utilizar $s = 3$ leva menos tempo e dá um limite inferior melhor que $s = 2$, escolhe-se $s = 3$. Caso contrário, se a melhora no limite inferior usando $s = 3$ é pelo menos 0.1% e o tempo gasto é menor do que 10 vezes o tempo gasto utilizando $s = 2$, continua-se escolhendo $s = 3$.

O algoritmo de BCP pode ser facilmente convertido em um algoritmo de BC comum de maneira simples. Introduz-se variáveis λ especiais, que não correspondem a q-rotas, mas sim a arestas simples em E. O subproblema de geração de colunas não é resolvido. Com essas variáveis, a transformação de restrições expressas em termos de variáveis x em restrições expressas em termos de λ se torna uma transformação feita utilizando a matriz identidade.