**Uma Aplicação do Método
Branch-and-Cut a um
Problema de Roteamento de Veículos**

Marko Loparić

DISSERTAÇÃO APRESENTADA
AO
INSTITUTO DE MATEMÁTICA E ESTATÍSTICA
DA
UNIVERSIDADE DE SÃO PAULO
PARA
OBTENÇÃO DO GRAU DE MESTRE
EM
MATEMÁTICA APLICADA

Área de Concentração: Ciência da Computação
Orientador: Prof. Dr. Carlos Eduardo Ferreira

Durante a elaboração deste trabalho o autor recebeu apoio financeiro da CAPES

- São Paulo, dezembro de 1995 -

Uma Aplicação do Método
Branch-and-Cut a um
Problema de Roteamento de Veículos

Este exemplar corresponde à redação
final da dissertação devidamente corrigida
e defendida por Marko Loparić
e aprovada pela comissão julgadora.

São Paulo, 09 de fevereiro de 1996.

Banca examinadora:

- Prof. Dr. Carlos Eduardo Ferreira (orientador) – IME-USP
- Prof. Dra. Yoshiko Wakabayashi – IME-USP
- Prof. Dr. Cid Carvalho de Souza – IMECC-UNICAMP

*À minha querida
Adriane*

Agradecimentos

À minha família, em especial aos meus pais e à Adriane, pelo imenso apoio dado. Nem sei como agradecer.

Ao Carlinhos, pela excelente orientação, pela amizade, dedicação e entusiasmo. Adorei trabalharmos juntos.

À Yoshiko, pela atenção especial e ajuda oferecida em diversos momentos.

Ao Arnaldo, pelo trabalho com matróides.

Ao Prof. Siang, pelo incentivo sempre demonstrado.

Ao Prof. Carlos Humes, por duas ou três observações sensíveis, que — mais do que ele deva ter percebido — me trouxeram um apoio decisivo.

Ao Prof. Brinati, pela atenção que me deu em apresentar o problema da Petrobrás e obter autorização para que eu tivesse acesso aos dados.

Aos meus colegas, que me estimularam e me deram apoio, em particular ao Álvaro, ao Fabio Kon, ao Flávio e ao Orlando.

À CAPES, pela bolsa concedida.

Resumo

O Problema de Roteamento de Veículos é o problema de determinar a melhor rota para que veículos levem mercadoria a clientes geograficamente dispersos, respeitando restrições a respeito da capacidade de transporte dos veículos e de tempo máximo de viagem. Nesse trabalho mostramos uma implementação de um algoritmo “branch-and-cut” para esse problema. Nosso algoritmo foi testado com dados de um problema real de transporte de funcionários da Petrobrás que trabalham nas plataformas de petróleo da Bacia de Campos. Os resultados de nossa abordagem mostram-se superiores aos obtidos anteriormente por métodos heurísticos.

Abstract

The Vehicle Routing Problem can be described as the problem of finding the best route for some vehicles to deliver goods to clients, satisfying some capacity constraint of the vehicle and a time bound for the trip. In this work we present an implementation of a *branch-and-cut* algorithm for this problem. We tested this algorithm with real world data obtained from an application to the transport of the employees from Brazilian Petroleum Company who work at the petroleum platforms in Campos Bay. Our results are superior to the ones previously obtained by heuristic methods.

Conteúdo

Introdução	i
0 Definições e resultados básicos	1
0.1 Poliedros e facetas	1
0.2 Grafos e vetores de incidência	4
1 Problemas clássicos de transporte	5
1.1 O Problema do Caixeiro Viajante (PCV)	5
1.2 O Problema de Roteamento de Veículos (PRV)	6
1.2.1 Variantes	8
1.2.2 Minimizando o número de viagens	10
1.3 Uma aplicação prática do PRV	11
1.3.1 Resultados da abordagem anterior	12
1.3.2 Variando a função objetivo	15
1.4 Conclusão	16
2 O PCV e o Método Branch-and-Cut	18
2.1 O PCV é “difícil”	18
2.2 A utilização do algoritmo simplex	19
2.3 Restrições de eliminação de subcircuitos	21
2.4 Métodos de planos-de-corte	28
2.5 Geração automática de restrições violadas	28
2.6 Qualidade dos cortes	30
2.7 Facetas do PCV	31
2.8 Identificação de restr. de elim. de subcircuitos	41
2.9 O método branch-and-cut	48
2.10 Aspectos críticos do método branch-and-cut	50

2.11	A “Olimpíada” do PCV	53
2.12	Conclusão	54
3	O Problema de Roteamento de Veículos	56
3.1	Formulação do PRV	56
3.2	Estudo do poliedro do PRV	59
3.3	Utilização de método de plano de corte	60
3.4	Heurísticas de separação	61
3.5	Outras Formulações	63
3.5.1	Formulação por viagens	63
3.5.2	Formulação por partição de conjuntos	65
3.5.3	Formulação por Partição em Caminhos	66
3.6	Conclusão	67
4	Resultados obtidos	68
4.1	Otimizando distância	68
4.2	Estrutura de branch-and-bound	69
4.3	Primeira separação	70
4.4	Buscando violações em subconjuntos	70
4.5	Verificando violação de restrição de tempo	72
4.6	Comparação de diversos estágios do algoritmo	74
4.7	Considerações práticas	77
4.8	Otimalidade do número de viagens	78
4.9	Resultados heurísticos	79
	Conclusão e comentários finais	89
	Bibliografia	91

Introdução

Um caixeiro viajante deseja visitar uma lista de cidades e retornar à cidade inicial, passando exatamente uma vez em cada uma das cidades. Ele pode visitar essas cidades em diversas ordens diferentes, percorrendo distâncias menores ou maiores. Qual é a sequência de visita que ele deve escolher de modo a distância ser a mínima possível?

Esse é o famoso *Problema do Caixeiro Viajante*, o mais clássico e o mais estudado problema da área conhecida como *Otimização Combinatória*. Os problemas dessa área apresentam a característica comum de que um elemento ótimo é buscado em uma classe *finita* de possibilidades.

Um problema semelhante ao Problema do Caixeiro Viajante, e de grande utilidade prática, é o *Problema de Roteamento de Veículos*. Nesse caso temos, em vez do caixeiro viajante, um veículo partindo de um depósito e entregando mercadoria a diversos clientes. Devido a limitações, como a capacidade do veículo ou o tempo máximo de viagem, os clientes serão visitados em várias viagens diferentes.

No início da década de 70, demonstrou-se que ambos os problemas citados acima, assim como um grande número de outros problemas de otimização combinatória, são *NP*-difíceis. Dizer que um problema é *NP*-difícil implica, do ponto de vista prático, que não se conhece nenhum algoritmo eficiente para resolvê-lo e que poucas esperanças se têm de que um tal algoritmo exista.

Muito antes disso, no entanto, pesquisadores desenvolveram para o Problema do Caixeiro Viajante algoritmos que, apesar de não poderem ser considerados eficientes, foram capazes de resolver instâncias consideradas grandes. Esses algoritmos baseiam-se na utilização do método simplex para a resolução de problemas de programação linear e na técnica de geração de planos-de-corte. Do desenvolvimento dessas técnicas surgiu o método *branch-and-cut* que hoje representa o mais poderoso instrumento de resolução de problemas difíceis de otimização combinatória.

Tivemos a oportunidade de dispor dos dados de uma interessante aplicação do Problema de Roteamento de Veículos ao planejamento do transporte de funcionários da Petrobrás que trabalham em plataformas de petróleo. Decidimos então desenvolver e implementar um algoritmo *branch-and-cut*, tendo como meta a resolução de problemas envolvendo esses dados.

O nosso trabalho traz motivações diversas, até certo ponto independentes. Por um lado tínhamos a intenção de estudar o método *branch-and-cut*, assim como obter uma implementação eficiente do mesmo. Por outro, o nosso desejo de resolver um problema real com que nos confrontamos era muito grande. Procuramos nessa dissertação abordar o método e a aplicação prática, podendo o leitor dedicar sua atenção ao que considerar mais importante.

No Capítulo 1, expomos os dois problemas de transporte com que estamos lidando, exibimos a aplicação prática da Petrobrás e mostramos uma abordagem adotada por outros autores para essa aplicação.

Em seguida, no Capítulo 2, apresentamos métodos de planos-de-corte, com destaque para o método *branch-and-cut*, ilustrando a utilização dessas técnicas ao Problema do Caixeiro Viajante. Utilizamos uma perspectiva histórica, acreditando ser a forma mais ilustrativa de expor as características desses métodos. O Problema do Caixeiro Viajante é aquele cuja aplicação de métodos de planos-de-corte é não somente a mais importante do ponto de vista histórico, como também aquela em que hoje esses métodos atingem o seu maior refinamento. Além disso, tal aplicação introduz o nosso tratamento do Problema de Roteamento de Veículos, uma vez que os dois problemas são próximos. São também tratados alguns aspectos teóricos relacionados com esses métodos, como a caracterização de facetas e o estudo do problema de separação.

No Capítulo 3, descrevemos a abordagem do Problema de Roteamento de Veículos via *branch-and-cut* utilizada na literatura e por nós empregada. Descrevemos também abordagens de destaque dadas a esse problema.

O desenvolvimento do nosso algoritmo *branch-and-cut* para o Problema de Roteamento de Veículos e sua aplicação aos dados da Petrobrás encontram-se no Capítulo 4. Embora soluções comprovadamente ótimas não tenham sido obtidas para todas as instâncias testadas, a utilização feita do algoritmo como método heurístico nos proporcionou soluções para o problema melhores do que as obtidas por outros autores. A melhoria consiste em que o roteamento correspondente às soluções que encontramos apresenta distância total inferior às distâncias obtidas pelas soluções anteriores. O relato da experiência

de desenvolvimento do algoritmo *branch-and-cut* feito nesse capítulo contém sugestões de como implementar e como testar o algoritmo. Acreditamos que essas sugestões podem ser úteis para a aplicação do *branch-and-cut* a outros problemas de otimização.

Acrescentamos, por último, uma seção de conclusão e comentários finais, em que tratamos dos resultados da pesquisa desenvolvida e apontamos direções em que ela poderá progredir.

Capítulo 0

Definições e resultados básicos

Apresentamos neste capítulo algumas definições e resultados básicos que serão usados nos capítulos seguintes. O leitor poderá consultar esse capítulo quando considerar necessário.

0.1 Poliedros e facetas

O leitor interessado em um tratamento mais detalhado de teoria dos poliedros deve procurar algum livro de programação inteira, como Schrijver (1986) ou Nemhauser e Wolsey (1988). Apresentaremos aqui rapidamente apenas alguns conceitos básicos.

Problema 0.1 *Problema de Programação Linear.*

Dados: inteiros n e m e vetores $c, a_1, \dots, a_m \in \mathbb{R}^n$ e reais $b_1, \dots, b_m$.

Pede-se: Minimizar cx sujeito a

$a_i x \leq b_i$, para $i = 1, \dots, m$.

Observação 1: O problema é equivalente se também forem utilizadas restrições envolvendo " $\geq$ " ou " $=$ ".

Damos o nome de *poliedro* ao conjunto de pontos de um espaço $\mathbb{R}^n$ que satisfazem um número finito de restrições lineares. Assim, um problema de programação linear pode ser descrito como um problema de minimizar uma função linear sobre um poliedro.

Quando for conveniente, escreveremos o conjunto de restrições $a_i x \leq b_i$, $i = 1, \dots, m$ sob a forma $Ax \leq b$, onde A é uma matriz cujas linhas são os vetores a_i e b é um vetor cujas coordenadas são os números b_i .

Dizemos que $x \in \mathbb{R}^n$ é uma *combinação convexa* de pontos $x_1, \dots, x_k \in \mathbb{R}^n$ se existem números reais não-negativos $\alpha_1, \dots, \alpha_k$, com $\alpha_1 + \dots + \alpha_k = 1$, tais que $x = \alpha_1 x_1 + \dots + \alpha_k x_k$.

Definimos o *casco convexo* de um conjunto de pontos S de $\mathbb{R}^n$, denotado por $\text{conv}(S)$ da seguinte forma:

$$\text{conv}(S) = \{x \in \mathbb{R}^n \mid x \text{ é uma combinação convexa de pontos de } S\}.$$

Temos o seguinte resultado:

Teorema 0.2 *O casco convexo de um número finito de pontos é um poliedro.*

Dizemos que uma inequação $ax \leq a_0$ é *válida* com relação a um poliedro P , se todo ponto de P satisfaz a inequação. Sempre que estiver claro a que poliedro estamos nos referindo diremos apenas que a inequação $ax \leq a_0$ é *válida*. Ainda nesse caso, chamaremos os pontos contidos no poliedro de pontos *viáveis*, caso contrário, de pontos *inviáveis*.

Seja P um poliedro e $ax \leq a_0$ uma inequação válida com relação a P . Dizemos que $F = \{x \in P \mid ax = a_0\}$ é a *face* definida pela inequação $ax \leq a_0$ em P . Se $F \neq \emptyset$, dizemos que a inequação $ax \leq a_0$ é uma *inequação suporte* de P . Além disso, se F é uma face própria maximal (i.e. se estiver contida propriamente em P e não estiver contida propriamente em nenhuma outra face de P), dizemos que F é uma *faceta* de P e que a inequação $ax \leq a_0$ *define faceta* em P .

Dependendo do contexto, poderemos chamar também inequações do tipo $ax \leq a_0$ de *restrições* ou *cortes*.

Além dos conceitos de independência linear e de posto de matriz que são bem conhecidos, necessitamos do conceito de independência afim. Dizemos que um conjunto $\{x_1, \dots, x_k\}$ é afim-independente se as equações $\lambda_1 x_1 + \dots + \lambda_k x_k = 0$ e $\lambda_1 + \dots + \lambda_k = 0$ implicam que $\lambda_i = 0$, $i = 1, \dots, k$. Dado um $S \subseteq \mathbb{R}^n$, a *dimensão* de S , denotada por $\dim(S)$, é a cardinalidade do maior conjunto afim-independente contido em S menos um.

Uma face de dimensão zero é chamada *vértice*. O seguinte resultado será útil:

Teorema 0.3 *Se um problema de programação linear tem uma solução ótima e o poliedro correspondente apresenta vértices, então tem uma solução ótima que é vértice do respectivo poliedro.*

O seguinte teorema nos será útil para mostrarmos que uma determinada inequação é faceta.

Teorema 0.4 *Seja $P \subseteq \mathbb{R}^n$ um poliedro tal que $P = \{x \in \mathbb{R}^n \mid Ax \leq a \text{ e } Bx = b\}$, onde A é uma matriz (m, n) , B é uma matriz (k, n) , $a \in \mathbb{R}^m$ e $b \in \mathbb{R}^k$. Seja F uma face não vazia de P . Então as seguintes afirmações são equivalentes:*

1. F é uma faceta de P .
2. F é uma face própria maximal de P .
3. $\dim(F) = \dim(P) - 1$
4. Existe uma inequação $cx \leq c_0$ válida com relação a P com as seguintes propriedades:
 - (a) $F \subseteq \{x \in P \mid cx = c_0\}$.
 - (b) Existe $\bar{x} \in P$ com $c\bar{x} < c_0$.
 - (c) Se existe uma outra inequação $dx \leq d_0$ válida com relação a P satisfazendo $F \subseteq \{x \in P \mid dx = d_0\}$, então existe um escalar $\mu \geq 0$ e um vetor $\lambda \in \mathbb{R}^k$ tais que

$$d = \mu c + \lambda B,$$

$$d_0 = \mu c_0 + \lambda b.$$

Se estamos investigando um poliedro, é de fundamental importância estudarmos as suas facetas. Uma das justificativas para isso é dada pelo teorema abaixo:

Teorema 0.5 *Seja P um poliedro descrito por $\{x \in \mathbb{R}^n \mid Ax \leq a \text{ e } Bx = b\}$, onde $Ax \leq a$ não contém implicitamente uma equação e B tem posto completo. Então essa descrição do poliedro é não-redundante (i.e. a remoção de uma inequação ou equação resulta em um poliedro diferente de P), se e somente se toda inequação de $Ax \leq a$ define faceta em P .*

0.2 Grafos e vetores de incidência

Dado um grafo $G = (V, E)$, dizemos que dois vértices u e v são *adjacentes* se (u, v) é uma aresta do grafo. Dizemos que $(u_1, e_1, u_2, e_2, \dots, e_{n-1}, u_n)$ é um *passeio* se temos que, para $i = 1, \dots, n$, u_i pertence a V e que, para $i = 1, \dots, n-1$, e_i pertence a E com $e_i = (u_i, u_{i+1})$. Uma *trilha* é um passeio onde nenhuma aresta aparece mais de uma vez, enquanto um *caminho* é uma trilha que não possui vértices repetidos. Uma trilha $(u_1, e_1, u_2, e_2, \dots, e_{n-1}, u_n)$ é um *circuito*, se $(u_1, e_1, u_2, e_2, \dots, e_{n-2}, u_{n-1})$ é um caminho e $u_n = u_1$. Um circuito é *hamiltoniano*, se contém todos os vértices do grafo.

Dizemos que um grafo $G = (V, E)$ é *completo* se todo par de vértices é aresta.

Dado um grafo $G = (V, E)$ completo e $x \in \mathbb{R}_+^E$, o *grafo induzido pelo suporte de x* , denotado por G_x , é o subgrafo $G = (V, F)$ onde $F = \{e \in E \mid x_e > 0\}$.

Seja G um grafo e u um vértice de G . Então a *vizinhança* de u é definida por $N(u) = \{v \in V : (u, v) \text{ é aresta de } G\}$.

Podemos decompor um grafo em subgrafos chamados *componentes conexas*, de modo que, para todo par de vértices u e v , existe um caminho ligando u e v , se e somente se u e v estão na mesma componente conexa e que cada vértice isolado seja uma componente conexa. Um grafo é *conexo* se pode ser decomposto em apenas uma componente conexa.

Dados $G = (V, E)$ e $X, Y \subseteq V$ disjuntos, denotamos por $[X : Y]$ o conjunto $\{(u, v) \in E \mid u \in X \text{ e } v \in Y\}$ e por $\delta(X)$ o conjunto $[X : V \setminus X]$. Dizemos que $F \subseteq E$ é um *corte* se $F = \delta(U)$ para algum $U \subseteq V$. Quando G é conexo, dizemos que um subgrafo $G' = (V', E')$ de G é uma *árvore geradora* de G , se $V' = V$, G' é conexo e não apresenta circuitos.

Definida uma função x , do conjunto de arestas de um grafo em $\mathbb{R}$, que, dependendo do contexto, é chamada de *capacidade* ou *peso*, podemos falar em corte de capacidade mínima ou em árvore de peso mínimo.

Dados dois conjuntos E e F onde $F \subseteq E$, dizemos que $\chi \in \{0, 1\}^E$ é o *vetor de incidência* de F com relação a E se para todo $e \in E$ temos que $\chi_e = 1$ se e só se $e \in F$.

Dados dois conjuntos E e F , $F \subseteq E$, e um vetor $x \in \mathbb{R}^E$, denotamos por x_F o vetor em $\mathbb{R}^F$ onde $(x_F)_i = x_i$, para $i \in F$.

Capítulo 1

Problemas clássicos de transporte

Neste capítulo apresentaremos dois problemas clássicos de transporte: o Problema do Caixeiro Viajante e o Problema de Roteamento de Veículos.

Exibiremos também uma instância real do Problema de Roteamento de Veículos, relacionada com o transporte de funcionários da Petrobrás que trabalham em plataformas de petróleo.

1.1 O Problema do Caixeiro Viajante (PCV)

Vamos supor que um caixeiro viajante tenha uma lista de cidades a visitar. Dependendo da sequência em que as cidades forem percorridas, a distância total poderá variar. O problema consiste justamente em escolher uma sequência em que as cidades deverão ser visitadas de forma que a distância total seja a menor possível. Não permitimos que uma cidade seja visitada mais de uma vez e incluímos o retorno à cidade de partida no cálculo da distância total.

De uma forma mais genérica formulamos o Problema do Caixeiro Viajante, denotado por PCV, do seguinte modo:

Problema 1.1 *Problema do Caixeiro Viajante (PCV)*

Dados: n : número de pontos a serem visitados

d_{ij} : distância entre cada par i, j de pontos

Pede-se: uma sequência em que os pontos devam ser visitados exatamente uma vez, com retorno ao ponto de partida, de modo que a distância total a ser percorrida seja a menor possível.

O Problema do Caixeiro Viajante é o mais clássico problema de otimização combinatória. Técnicas dos mais variados tipos foram desenvolvidas para a sua resolução. Além disso, diversas questões teóricas relacionadas com o problema foram estudadas. O livro “The Traveling Salesman Problem” editado por Lawler *et al.* (1985) traz um “survey” bastante completo do estado dessas pesquisas até aquele ano.

No capítulo 2, apresentamos alguns resultados relacionados com o PCV, principalmente sobre o ponto de vista poliédrico.

O PCV pode ser classificado sob os seguintes aspectos:

- Euclidiano ou não-euclidiano. O PCV é chamado de euclidiano quando as distâncias obedecem à desigualdade triangular, isto é, quando

$$d_{uv} \leq d_{uw} + d_{wv},$$

para quaisquer $u, v, w \in V$. No caso do PCV ser euclidiano, algumas abordagens exploram essa propriedade para atingir soluções de melhor qualidade, ou para obter soluções ótimas mais rapidamente (veja, por exemplo, o livro editado por Lawler *et al.* (1985)).

- Simétrico ou assimétrico. O PCV assimétrico é aquele em que a distância para ir de um ponto a outro pode variar dependendo do sentido do percurso. Nós estaremos interessados apenas no PCV simétrico, uma vez que em nossas aplicações o custo de uma viagem independe do sentido. O tratamento de problemas assimétricos tende a ser bastante diferente daquele que é dado aos problemas simétricos (veja, novamente, o livro editado por Lawler *et al.* (1985)).

1.2 O Problema de Roteamento de Veículos (PRV)

Desta vez o exemplo, que é típico e que dá origem ao nome, é o de um distribuidor de uma determinada mercadoria que precisa atender a pedidos de

diversos clientes. Existe, como no caso do caixeiro viajante, uma lista de pontos (clientes ou consumidores) a serem visitados, além de um ponto diferenciado, chamado depósito ou base, de onde parte o veículo com a mercadoria e para onde deve voltar no final. Ocorre, no entanto, que o veículo usado para transportar essa mercadoria tem capacidade limitada e, dessa forma, dependendo da demanda de mercadoria em cada ponto, vários veículos podem ser necessários para realizar a entrega. O problema é determinar que percurso cada veículo deve fazer de forma que a soma das demandas dos clientes a serem atendidos pelo veículo não ultrapasse a sua capacidade, e, além disso, a distância total percorrida pelos veículos seja a menor possível. Normalmente se supõe que todos os veículos tenham a mesma capacidade, que cada veículo só possa visitar um cliente no máximo uma vez, e que não possa haver mais de um veículo visitando o mesmo cliente.

Chamamos o percurso feito por cada veículo desde a saída do depósito até o seu retorno de *viagem*.

Pode ocorrer que exista um limite de tempo para cada viagem. Devemos então levar em conta o tempo gasto no percurso entre cada par de pontos. A distribuição de jornais é um exemplo típico em que esse tipo de restrição se aplica. Em alguns casos, além do tempo de percurso, devemos incluir também o tempo de descarga da mercadoria. Na distribuição de produtos a supermercados, por exemplo, o tempo de descarga costuma ser relevante.

O problema é equivalente no caso de existir apenas um veículo. Nesse caso, as diversas viagens correspondem a viagens sucessivas que esse veículo deverá fazer, sempre retornando ao depósito no final de cada uma delas para que seja recarregado de mercadoria.

O PRV, na versão que nos interessa, pode ser descrito do seguinte modo:

Problema 1.2 *Problema do Roteamento de Veículos (PRV)*

Dados:

n : número de pontos, incluindo os clientes e o depósito

d_{ij} : distância entre cada par i, j de pontos (para $i < j$)

C : capacidade de cada veículo

c_i : demanda associada aos pontos, com exceção do depósito ($c_i < C$, para todo i)

T : limite de tempo para cada viagem

v : velocidade de cada veículo

ρ : taxa de descarga (quantidade de mercadoria descarregada por unidade de tempo)

Pede-se: O percurso de viagem de cada veículo partindo do depósito e retornando a ele de forma que:

- *Todo ponto seja visitado exatamente uma vez.*
- *A soma das demandas dos clientes de cada viagem não ultrapasse a capacidade do veículo.*
- *O tempo de cada viagem – incluindo o tempo de deslocamento e o tempo de descarga – não ultrapasse o limite T .*
- *A distância total percorrida seja minimizada, respeitando-se as restrições acima.*

A Figura 1.1 ilustra uma solução de um PRV sem restrições de tempo. As demandas dos clientes estão indicadas dentro dos círculos e a capacidade total de cada veículo é 7. Note que duas das viagens da solução se cruzam. Trocando dois clientes entre as duas viagens não haveria esse cruzamento, mas a restrição de capacidade ficaria violada.

O PRV pode ser também classificado em euclidiano ou não-euclidiano e simétrico ou assimétrico. Algumas abordagens dirigem-se a um desses tipos de PRV, explorando propriedades específicas para atingir soluções mais rapidamente.

1.2.1 Variantes

Estão presentes na literatura diversas variantes do PRV descrito. As variações mais frequentes são as seguintes:

- Restrições de tempo não são consideradas, apenas restrições de capacidade.
- O tempo de descarga é desprezado (nesse caso, o limite de tempo por viagem pode ser encarado como um limite de distância).
- O tempo de descarga é fixo e não proporcional à demanda de cada local.

Figura 1.1: Exemplo de solução de um PRV.

- Janelas de tempo são consideradas, isto é, a visita a cada cliente deve ser feita dentro de limites pré-estabelecidos de tempo.
- O número de viagens é minimizado ao invés da distância total (veja comentário a esse respeito na próxima seção).

A bibliografia tratando da resolução do PRV é muito extensa. O “survey” mais clássico e mais completo é certamente o de Bodin, Golden, Assad e Ball (1983), que traz os mais interessantes métodos – exatos ou não – conhecidos na época e estabelece uma classificação das diversas variantes do problema. Laporte e Nobert (1987) apresentam um “survey” mais novo, tratando especificamente de métodos exatos. O último capítulo do livro sobre o PCV editado por Lawler *et al.* (1985) trata do PRV e constitui também um interessante “survey”.

1.2.2 Minimizando o número de viagens

Abordaremos agora a variante do PRV para a qual o número de viagens é minimizado, ao invés da distância, de particular importância para o problema que iremos tratar.

Pode-se imaginar que uma solução do PRV que minimiza a distância total percorrida tenderá a usar um número mínimo de viagens também. Isso pode ocorrer na maioria dos casos práticos, mas não ocorrerá sempre, como nos mostra o exemplo da Figura 1.2. Nesse exemplo, os clientes $v_1, \dots, v_4$ estão dispostos em uma circunferência de centro no depósito v_0 e de raio R , a capacidade de cada veículo é 2 e a demanda de cada cliente é 1. Assim, na solução A a distância total percorrida é $4R + d_{v_1v_2} + d_{v_3v_4}$ e na solução B a distância total é $6R + d_{v_2v_3}$. Se os pontos estiverem dispostos de forma que $d_{v_1v_2} + d_{v_3v_4} > 2R + d_{v_2v_3}$, a solução B, que minimizará a distância, não minimizará o número de viagens, que será minimizado pela solução A.

Figura 1.2: Uma solução minimizando a distância pode não minimizar o número de viagens.

1.3 Uma aplicação prática do PRV

O problema de roteamento de veículos que iremos estudar origina-se da necessidade de transporte dos funcionários da Petrobrás que trabalham nas plataformas de petróleo da Bacia de Campos. Em cada plataforma existem dois turnos de trabalho de doze horas, em que duas equipes se alternam. Essas equipes também alternam 14 dias na plataforma e 14 dias no continente de forma que a atividade contínua é proporcionada, em cada plataforma, por quatro equipes de mesmo tamanho. A cada semana é feita a troca de uma equipe. O dia dessa troca é livre e pode cair em qualquer dia de segunda a domingo.

O problema da Petrobrás é decidir em quais plataformas fazer a troca de equipes em cada dia da semana, e, o que é mais importante, decidir quantas embarcações são necessárias para fazer as trocas das equipes. No caso da aplicação em estudo, o número de embarcações e o tipo utilizado são de fundamental importância. Deseja-se escolher um modelo de embarcação, entre os existentes no mercado, de modo a minimizar o gasto com a aquisição de frota. Observe-se que cada modelo de embarcação possui características específicas tais como limite de passageiros, velocidade e preço. Assim, se optarmos por um determinado tipo de embarcação, para atender a todas as trocas de equipes precisaremos fazer mais ou menos viagens, podendo necessitar de usar mais ou menos embarcações.

As plataformas que podem ser atendidas em uma viagem devem atender a dois tipos de restrições. Por um lado, não se pode transportar em uma embarcação mais funcionários do que a sua capacidade permite. Por outro lado, cada viagem não pode levar mais do que um certo número de horas, o que torna a velocidade em que a embarcação consegue navegar um dado relevante para o problema.

Ainda por causa da restrição do tempo de viagem, outro dado tem que ser levado em conta: a taxa de transferência de passageiros, ou seja, o número de passageiros por hora que podem ir da embarcação para a plataforma e vice-versa. Isso vai determinar o tempo que a embarcação vai perder em cada plataforma. Dois sistemas de transferência podem ser usados apresentando taxas bem diferentes: o sistema de cesto, mais simples e barato, em que 60 passageiros são trocados por hora e o sistema de rampa, mais eficiente, mas tecnologicamente mais complicado, em que a taxa de transferência é de 360 passageiros por hora.

A modelagem utilizada para esse problema inclui algumas simplificações. A primeira consiste em considerar a frota homogênea. Isso simplifica o problema já que o tempo de percurso entre as cidades não depende da embarcação. Além disso, a capacidade de uma embarcação também é constante. Como está definido que cada embarcação deverá fazer apenas uma viagem por dia e como podemos usar todos os dias da semana, o número mínimo de embarcações corresponde então ao menor inteiro maior ou igual ao número mínimo de viagens necessárias dividido por sete. Outra simplificação que fazemos diz respeito à transferência de passageiros. Supomos que todas as plataformas estão adotando o mesmo sistema de transferência.

Para oferecer à Petrobrás condições de efetuar escolha de um entre os diversos tipos de embarcação e de um dos dois tipos de transferência de passageiros torna-se necessário resolver um PRV para cada combinação possível de embarcação e tipo de transferência. A decisão final se daria em função das soluções desses diversos PRV, combinadas com fatores que aqui não nos interessam, como preço e qualidade de cada tipo de embarcação, custo, conforto e segurança de cada tipo de transporte.

Essa modelagem foi adotada por Brinati *et al.* (1988), que desenvolveram esse projeto junto à Petrobrás.

1.3.1 Resultados da abordagem anterior

O problema da Petrobrás foi abordado por Brinati *et al.* (1988), em um estudo encomendado pela Petrobrás. Esse estudo, envolvia outros aspectos do problema relacionados com Engenharia Naval que não dizem respeito a otimização. Os métodos propostos são heurísticos, isto é, não oferecem garantia de que as soluções encontradas sejam ótimas, nem de qual distância máxima do ótimo elas se encontrem.

Uma outra observação a ser feita a respeito do estudo de Brinati *et al.* é que as heurísticas utilizadas visaram a minimização da distância total, enquanto, conforme afirmam os autores do trabalho, o objetivo era minimizar o número de viagens. Como vimos na seção anterior, uma solução heurística, ainda que corresponda de fato à solução ótima com relação à minimização da distância, pode não representar uma utilização mínima de viagens.

Para chegar a uma solução, os autores utilizaram uma combinação de diversos métodos heurísticos e de um método de melhoria.

A primeira heurística utilizada foi a *técnica de varredura*, que adota a es-

estratégia de agrupar primeiro e rotear depois. Uma semi-reta com origem no depósito é girada em um determinado sentido, agrupando, em uma primeira viagem, as plataformas contidas no ângulo formado, até que as restrições do problema impeçam a inclusão de uma nova plataforma. Essa plataforma é incluída em uma segunda viagem e a varredura prossegue, até que todas as plataformas estejam agrupadas em viagens. Em seguida, é necessário decidir a rota de cada um dos agrupamentos, que corresponde a vários PCV's. A heurística utilizada é a estratégia do vizinho mais próximo aprimorada pelo algoritmo 2-opt. Esse algoritmo testa, para cada par de arestas $\{(u, v), (w, t)\}$, se a troca dessas arestas pelo par $\{(u, w), (v, t)\}$ é vantajosa e forma um novo circuito hamiltoniano, executando a troca em caso positivo.

A segunda heurística escolhida, devida a Clarke e Wright (1964), é chamada de *método de economia*. A cada aresta (u, v) ligando plataformas é associada a economia $e_{uv} := d_{v_0u} + d_{v_0v} - d_{uv}$. Partimos de uma solução em que cada viagem corresponde ao percurso de ida e volta a cada uma das cidades. Em seguida, as arestas ligando plataformas são inseridas em ordem decrescente de suas economias, sempre que possível, de forma que em cada inserção duas viagens são juntadas em uma só. Note que essa inserção será possível sempre que a aresta estiver ligando duas plataformas pertencentes a extremos de duas viagens distintas e que a nova viagem a ser formada não viole as restrições do problema. O processo é repetido até que a inserção de todas as arestas tenha sido tentada.

A terceira heurística utilizada é baseada no algoritmo de *partição múltipla da roda gigante*, devido a Golden *et al.* (1980). Adota a estratégia de primeiro rotear e depois agrupar. O PCV incluindo todas as plataformas e excluindo o depósito é resolvido (do mesmo modo que os PCVs da primeira heurística). Em seguida, essa rota é particionada de forma a minimizar a distância total. Essa partição é escolhida, dentre todas as partições que podem ser obtidas agrupando conjuntos maximais de cidades ao longo da rota obtida.

Os autores mencionam ainda a utilização de uma quarta heurística. Especificam apenas que ela se baseia em um algoritmo de Fisher e Jaikumar (1981) e que só foi utilizada em casos isolados, devido ao fato de que ela não incorpora restrições de tempo.

Após executar independentemente cada uma dessas heurísticas, a solução apresentando menor distância total é tomada e um procedimento de melhoria é aplicado. Para tanto, as k rotas da solução escolhida são reunidas numa única rota através de $k - 1$ replicações do ponto correspondente ao depósito.

Em seguida, o algoritmo 2-opt é aplicado, observando-se as restrições do problema do caso de troca de arestas pertencentes (na solução original) a rotas diferentes. No final, os diversos pontos correspondentes ao depósito são reunidos em um só, dando origem a uma nova solução, possivelmente melhor. Ao realizar essa reunião, verifica-se a possibilidade de reunir duas viagens contíguas, baixando-se assim o número total de viagens.

É fácil imaginarmos exemplos, até bastante simples, em que essas heurísticas não obtêm solução ótima. A Figura 1.3 traz um deles. Nesse exemplo, as plataformas são numeradas de 1, 2, 3 e 4 e esses números correspondem às suas demandas. A capacidade de cada embarcação é 5. As plataformas estão dispostas na ordem (1,2,4,3) ao redor de uma circunferência de raio 2, com o depósito no centro, de forma que a distância entre cada par de clientes consecutivos é 1. Temos assim que a única solução com duas viagens é $\{(1,4),(2,3)\}$. Notemos que essa solução é ótima, já que sua distância total é menor do que 12 e que qualquer solução com 3 ou mais viagens apresenta distância total maior ou igual a 13. Qualquer uma das heurísticas gera soluções com pelo menos 3 viagens. A varredura trabalha agrupando os clientes consecutivos e só pode agrupar as plataformas 1 e 2 ou as plataformas 1 e 3. O método da economia começa tentando agrupar clientes próximos, de forma que também agrupa os clientes 1 e 2, e não progride além disso. O método da roda gigante obtém (1,2,4,3) como solução do PCV, da qual a solução ótima não é uma partição. Assim, a mesma solução não-ótima é devolvida. O procedimento de melhoria tampouco poderá exibir a solução ótima, dado que as trocas necessárias para a sua obtenção não correspondem a uma diminuição de distância na rota única montada a partir da solução não-ótima que as heurísticas geram.

Os autores Brinati *et al.* estudaram a influência de diversos parâmetros na determinação do roteamento. Informações mais detalhadas a respeito do trabalho encontram-se em um relatório que nos foi gentilmente fornecido pelo Prof. Dr. Marco Antonio Brinati. Cada conjunto de parâmetros dá origem a um PRV diferente. Apresentamos na Tabela 1.1 as instâncias que mais nos interessam, que são aquelas para as quais dispomos do valor da distância total da solução heurística obtida pelos autores. No caso das outras instâncias apenas o número de viagens foi apresentado. Em todas elas, o número de plataformas é sempre 49, a capacidade da embarcação é de 250 passageiros.

A Tabela 1.2 traz a localização das plataformas e a sua demanda. A origem do sistema é a base e a unidade usada é o metro.

Figura 1.3: Instância do PRV que não é solucionada pelas heurísticas apresentadas.

Decidimos trabalhar com três instâncias: uma para a qual a restrição de tempo é importante (instância A), outra para a qual essa restrição é fraca (instância F), e uma instância intermediária (instância C).

1.3.2 Variando a função objetivo

No exemplo prático em que estamos interessados, o mais importante é a minimização do número de viagens, pois isso corresponde à minimização da frota, cuja aquisição parece representar um valor muito superior ao gasto com combustível. No entanto, se o gasto com combustível não for insignificante e puder ser comparado com o gasto da aquisição de veículos, pode ser utilizada uma função objetivo que combine os dois gastos. Dessa forma o algoritmo poderia eventualmente optar por realizar mais viagens do que o mínimo possível se isso significar uma redução acentuada na distância total percorrida.

Uma outra possibilidade de levar em conta o gasto de combustível é resolver duas vezes o PRV, na primeira vez minimizando-se o número de viagens, e na segunda vez minimizando-se a distância total com o número de viagens fixado no mínimo obtido na primeira otimização.

Se acreditarmos que a solução minimizando a distância total minimiza, além disso, o número de viagens, podemos também otimizar inicialmente a

instância	limite de tempo (horas)	velocidade (nós)	taxa de transf. (por hora)	distância total (km)	n. de viagens	n. de embarcações
A	10	25	60	4940,0	17	3
C	12	23	60	4018,2	14	2
D	10	23	360	3696,1	13	2
E	11	22	360	3696,1	13	2
F	12	20	360	3696,1	13	2
G	9	25	60	6722,6	23	4

Tabela 1.1: Resultados obtidos por Brinati *et al.*

distância e, em seguida, mostrar que o problema, com um número de viagens menor que o obtido, é inviável.

1.4 Conclusão

Neste capítulo foram apresentados tópicos que servem de base para os capítulos seguintes. Definimos os dois problemas de transporte com que estaremos lidando: o Problema do Caixeiro Viajante e o Problema de Roteamento de Veículos. Expusemos uma aplicação real de transporte de funcionários das plataformas de petróleo na bacia de Campos cujos dados serviram de base para nossos testes computacionais. Mostramos também a abordagem original dada a esse problema por Brinati *et al.*

número	x	y	demanda	número	x	y	demanda
1	-33720	96469	30	26	-5019	139734	40
2	-35704	102125	40	27	-3590	147120	100
3	-37510	100790	30	28	-3692	138424	145
4	-39736	98601	50	29	-3692	138424	40
5	-46233	103076	60	30	-3119	132670	165
6	-43843	104110	140	31	-3119	132670	40
7	-43843	104110	100	32	1318	125784	25
8	-43843	104110	40	33	3390	139109	120
9	-43843	104110	40	34	7976	153264	55
10	-39524	112157	55	35	2005	168645	45
11	-39514	112132	50	36	14451	173070	50
12	-38690	110844	50	37	17523	166838	70
13	-34770	108430	45	38	25785	184957	35
14	-33837	111111	175	39	34075	190644	35
15	-33836	111111	40	40	31848	183202	55
16	-36451	114689	35	41	27250	153167	35
17	-33955	115222	50	42	27250	153167	45
18	-30959	120053	40	43	25592	151719	45
19	-29054	123778	45	44	25592	151868	45
20	-29770	126170	50	45	23992	150669	45
21	-18761	127980	55	46	16675	147986	100
22	-8108	129288	40	47	19026	139160	45
23	-6749	133914	165	48	17735	136322	45
24	-6749	133914	40	49	63494	123096	30
25	-5019	139734	145				

Tabela 1.2: Coordenadas e demandas das 49 plataformas.

Capítulo 2

O Problema do Caixeiro Viajante e o Método Branch-and-Cut

Vamos falar da história do PCV, que se confunde com a história do desenvolvimento dos métodos de plano de corte e do branch-and-cut. Em paralelo com a história estaremos introduzindo os conceitos e resultados relacionados com o problema. Este capítulo baseia-se principalmente no livro “The Traveling Salesman Problem” e nos artigos de Dantzig, de Fulkerson e Johnson (1954), de Padberg e Rinaldi (1990b, 1991) e de Applegate *et al.* (1994).

2.1 O PCV é “difícil”

As origens do PCV são incertas. A primeira referência conhecida foi feita em Viena na década de 1920 pelo matemático e economista Karl Menger. Esse pesquisador já observou um aspecto da sua dificuldade: o algoritmo trivial de ir ao vértice mais próximo não resolve em geral o problema, como mostra a Figura 2.1. Nos anos 30, o problema reapareceu entre os matemáticos de Princeton e, no fim dos anos 40, dele muito se falou na Rand Corporation, empresa americana que, na época, chegou a oferecer um prêmio para um resultado importante que tratasse da resolução do PCV. As expectativas de um método eficiente para o PCV ficaram frustradas e o prêmio nunca foi dado. Na verdade, um único resultado significativo a respeito do PCV —

aparecido bem mais tarde, em um artigo de 1972 escrito por Karp, Lawler e Tarjan — apontou em sentido contrário, mostrando que o PCV é NP-difícil (veja o livro de Garey e Johnson (1979)).

Figura 2.1: O algoritmo que toma sempre o vértice mais próximo não resolve o PCV. Temos à esquerda a solução heurística dada por esse algoritmo, partindo de v , e à direita a solução ótima.

É interessante observar que para um “parente” próximo do PCV, o problema da árvore de custo mínimo, são conhecidos desde 1926 algoritmos muito eficientes. A diferença, aparentemente pequena, que existe entre procurar uma árvore geradora de peso mínimo e um circuito hamiltoniano de peso mínimo, transforma o primeiro problema em um protótipo de problema *fácil* de otimização combinatória, assim como o PCV é um protótipo de problema *difícil* dessa área. Note que dizer que o PCV é difícil, porque existe um grande número de circuitos hamiltonianos em um grafo completo, não é um bom argumento, já que existe um número muito maior de árvores geradoras.

2.2 A utilização do algoritmo simplex

Em 1954 os pesquisadores George Dantzig, Ray Fulkerson e Selmer Johnson publicaram a descrição de um método de resolução do PCV que utilizava o algoritmo simplex de resolução de problemas de programação linear, inventado pelo próprio Dantzig em 1947. Para mostrar o poder do método, foi resolvido um PCV de 49 cidades (envolvendo a capital dos Estados Unidos, Washington D.C, e as capitais dos estados, que na época eram 48), que pode ser considerado bastante grande para a época.

O simplex é na prática um algoritmo bastante eficiente para a resolução de problemas de programação linear e é ainda o método mais utilizado hoje em

dia. É curioso notar que o algoritmo não atende à definição de eficiência usada para computação, que é o de apresentar tempo de pior caso polinomial. Klee e Minty (1972) exibiram problemas de programação linear em que o tempo de execução do simplex é exponencial. A descoberta de um algoritmo polinomial para programação linear só foi feita em 1979 por Khatchian. Um resultado que, de certa forma, explica o bom desempenho do simplex na prática, devido a Borgwardt (1982), garante que o tempo médio de execução do simplex é polinomial.

A eficiência prática do algoritmo simplex fez com que os autores procurassem usá-lo para a resolução do PCV. Para que isso seja possível, torna-se necessário formular o PCV em termos de poliedros (veja Seção 0.1). Assim, consideramos o grafo completo $G = (V, E)$, onde V é o conjunto das n cidades, com $|V| = n$ e $|E| = n(n-1)/2$. Representamos então um circuito hamiltoniano do grafo por um vetor 0/1 indexado pelas arestas, em que uma componente vale 1 se e somente se a respectiva aresta está no circuito. Se chamarmos de $\mathcal{S}$ o conjunto dos vetores que representam os circuitos hamiltonianos de G , e chamarmos de d o vetor de distâncias também indexado pelas arestas, o PCV pode ser então assim formulado:

$$\begin{aligned} &\text{minimize } \sum_{e \in E} d_e x_e \\ &\text{sujeito a } x \in \mathcal{S}. \end{aligned}$$

Ocorre que o conjunto $\mathcal{S}$ não é um poliedro. Tomamos, assim, no lugar do conjunto $\mathcal{S}$, o seu casco convexo, $\text{conv}(\mathcal{S})$, já que é sabido que o casco convexo de um número finito de pontos é um poliedro (veja Teorema 0.2). Definimos, assim, o poliedro que chamaremos de poliedro do PCV para n cidades:

$$PCV(n) = \text{conv}(\mathcal{S}).$$

Não é difícil provar que o conjunto dos vértices de $PCV(n)$ é precisamente o conjunto $\mathcal{S}$. É possível haver pontos em $PCV(n)$ que maximizem $\sum_{e \in E} d_e x_e$, mas que não sejam vértices do poliedro. No entanto, pode-se mostrar (Teorema 0.3) que todo problema de programação linear apresentando solução ótima cujo poliedro tem vértice admite um vértice como solução ótima. O funcionamento do algoritmo simplex baseia-se nesse princípio, de modo que sempre que uma solução ótima devolvida, ela corresponde a um vértice do poliedro.

A principal dificuldade está em apresentar um conjunto de restrições que defina o poliedro de $PCV(n)$. Na verdade, como veremos adiante, uma tal descrição não é conhecida. Além disso, mesmo que fosse conhecida, não poderia ser usada como entrada para o algoritmo simplex, porque envolveria um número muito grande (exponencial em n) de restrições.

Os autores decidiram então aplicar o algoritmo simplex usando apenas parte das restrições necessárias para descrever o poliedro $PCV(n)$. As restrições eram

$$0 \leq x_e \leq 1, \forall e \in E \quad (2.1)$$

e

$$x(\delta(v)) = 2, \forall v \in V. \quad (2.2)$$

O primeiro tipo de restrição determina apenas que as coordenadas dos pontos em $PCV(n)$ têm que estar entre 0 e 1. O segundo tipo estabelece que, para que um vetor x corresponda a um circuito hamiltoniano, é necessário (mas não suficiente) que haja para cada cidade exatamente duas arestas incidentes a ela. Chamaremos o poliedro definido pelas restrições desses dois grupos de P_0^n .

Observemos que o poliedro P_0^n não é igual, mas contém o poliedro $PCV(n)$. Isso significa que uma solução devolvida pelo simplex fornece um limitante inferior para o valor ótimo do PCV. Em particular, se o vetor x em P_0^n devolvido pelo simplex também pertencer ao poliedro $PCV(n)$, então x é solução do PCV. Na Figura 2.2, mostramos esquematicamente os poliedros $PCV(n)$ e $P_0(n)$. Note que os poliedros não têm esse formato.

Uma descrição parcial do problema tal como essa, dando origem a um poliedro que contém o poliedro do problema, é chamada de *relaxação* do problema, justamente pelo fato de suas restrições serem mais fracas, isto é, por serem satisfeitas não apenas por pontos do problema, mas eventualmente também por pontos inviáveis.

2.3 Restrições de eliminação de subcircuitos

É fácil imaginar pontos em P_0^n que não estão em $PCV(n)$. A Figura 2.3 dá um exemplo de uma solução que corresponde a um ponto de P_0^8 que não é um circuito hamiltoniano.

Figura 2.2: Neste exemplo, o valor ótimo do PCV é 512, enquanto o vértice ótimo de P_0^n apresenta valor da função objetivo de 510,4.

Figura 2.3: Ponto de P_0^8 que não está em $PCV(8)$.

Observemos que se x representa um circuito hamiltoniano C , então para qualquer subconjunto próprio W de V , $W \neq \emptyset$, existem pelo menos duas arestas em C pertencentes ao corte $\delta(W)$, isto é, x tem que satisfazer

$$x(\delta(W)) \geq 2.$$

Note que, no exemplo, temos que $x(\delta(\{1, 2, 8\})) = 0$.

Essas restrições são chamadas *restrições de eliminação de subcircuitos*. Observemos que essas restrições podem também ser escritas da seguinte forma, supondo que $x \in P_0^n$:

$$x(E(W)) \leq |W| - 1, \quad (2.3)$$

para todo W , $\emptyset \subset W \subset V$.

Proposição 2.1 *Seja $x \in P_0^n$. Então $x(\delta(W)) \geq 2r$ se e somente se $x(E(W)) \leq |W| - r$.*

Prova: De fato, se $x \in P_0^n$, então

$$\sum_{v \in W} x(\delta(v)) = 2|W|.$$

Observemos que a somatória acima conta duas vezes as arestas internas a W e uma vez as arestas pertencentes ao corte de W , ou seja,

$$\sum_{v \in W} x(\delta(v)) = 2x(E(W)) + x(\delta(W)). \quad (2.4)$$

Logo,

$$x(\delta(W)) = 2(|W| - x(E(W))),$$

de onde segue o resultado. □

Definimos então um novo poliedro, P_1^n correspondendo aos pontos de P_0^n que satisfazem as restrições do tipo 2.3.

De posse dessas restrições, conseguimos estabelecer uma nova formulação do PCV:

Teorema 2.2 *Um vetor $x \in \mathbb{R}^n$ é uma solução do PCV com vetor de distância d sobre o grafo completo $G = (V, E)$ se e somente se x minimiza $\sum_{e \in E} d_e x_e$ entre os pontos inteiros de P_1^n , i.e. se e somente se x é solução de*

$$\begin{aligned} & \text{minimize } \sum_{e \in E} d_e x_e \\ & \text{sujeito a } \begin{aligned} & x(\delta(v)) = 2, && \text{para todo } v \in V, \\ & x(E(W)) \leq |W| - 1, && \text{para todo } W, \emptyset \subset W \subset V, \\ & x_e \in \{0, 1\}, && \text{para todo } e \in E. \end{aligned} \end{aligned}$$

Prova: É claro que se x representa um circuito hamiltoniano, então x satisfaz as restrições do teorema.

Vamos supor agora que x satisfaz as restrições acima e mostrar que x representa um circuito hamiltoniano. Para tanto, vamos escolher um vértice v_0 qualquer de V e construir uma trilha usando arestas do grafo G_x (veja definição na Seção 0.2). Como x obedece à primeira restrição, ao atingir um vértice ainda não visitado, existe sempre uma segunda aresta não utilizada, com a qual podemos aumentar a nossa trilha. Prosseguindo dessa forma, construímos uma trilha até atingir novamente uma vértice já visitado $\bar{v}$. Observamos que $\bar{v} = v_0$, caso contrário teríamos um vértice de grau três em G_x , o que contradiz a primeira restrição. Se, então, v_0 é atingido sem que todos os vértices de V tenham sido visitados, temos que, para o conjunto W dos vértices visitados, vale que $x(E(W)) = |W|$, o que viola uma restrição do segundo tipo. Assim, devemos ter visitado todos os vértices de V antes ao atingir v_0 , o que significa que x representa um circuito hamiltoniano. $\square$

Esse teorema garante que, usando as restrições de eliminação de subcircuitos juntamente com a restrição de integralidade $x_e \in \{0, 1\}$ e a restrições do tipo 2.2, conseguimos uma caracterização de $\mathcal{S}$. Em outras palavras, temos que os elementos de $\mathcal{S}$ correspondem exatamente aos pontos inteiros de P_1^n .

No entanto, existem dois problemas que impedem a utilização dessa formulação pelo algoritmo simplex. Um deles é que o simplex não aceita restrições de integralidade como a de $x_e \in \{0, 1\}$, isto é, a formulação acima não corresponde a um problema de programação linear. A outra é que existe um número exponencial de restrições do tipo (2.3) e isso inviabiliza na prática que se monte um programa linear com todas elas, como entrada para o simplex.

Figura 2.4: Ponto fracionário de P_0^8 que não está em P_1^8 .

A estratégia usada por Dantzig, Fulkerson e Johnson — uma versão pioneira de método de plano de corte — foi a seguinte:

1. Monte o programa linear com as restrições do tipo (2.1) e (2.2).
2. Resolva o programa linear (i.e. aplique o método simplex).
3. Se houver W , $\emptyset \subset W \subset V$, tal que a respectiva restrição de eliminação de subcircuitos está violada, adicione tal restrição ao programa linear e volte para 2.

Observemos que restrições de eliminação de subcircuitos podem estar violadas não apenas por soluções inteiras como na Figura 2.3, mas também por soluções fracionárias como na Figura 2.4, onde temos que $x(E\{1, 2, 8\}) = 2,5$, violando $x(E(\{1, 2, 8\})) \leq 2$.

Os autores relatam que obtiveram 24 restrições de eliminação de subcircuitos, adicionadas ao programa linear conforme o método descrito. Após isso, no entanto, o simplex devolveu uma solução fracionária (veja Figura 2.5), para a qual os autores não conseguiram encontrar nenhuma restrição de eliminação de subcircuitos violada. De fato, nenhuma restrição desse tipo está violada. O leitor poderá verificar que, para qualquer corte $\delta(W)$ que tomar no grafo da Figura, a restrição $x(\delta(W)) \geq 2$ estará satisfeita.

Os autores só conseguiram prosseguir graças à ajuda de I. Glickberg da Rand Corporation, que apontou restrições válidas para o poliedro do PCV que estavam violadas pela solução obtida. Com a introdução de duas dessas restrições, os autores obtiveram finalmente a solução do PCV de 49 cidades.

Figura 2.5: Solução fracionária que não viola nenhuma restrição de eliminação de subcircuitos.

Figura 2.6: Restrição ponte violada pela solução fracionária obtida por Dantzig *et al.*

A restrição

$$x(E(H)) + x(E(T_1)) + x(E(T_2)) + x(E(T_3)) \leq 19, \quad (2.5)$$

onde

$$H = \{10, 11, 12, 24, 25\},$$

$$T_1 = \{9, 10\},$$

$$T_2 = \{24, 25, 26, 27\} \text{ e}$$

$$T_3 = \{11, 12, \dots, 23\}$$

é uma restrição válida, que não está satisfeita por essa solução (veja Figura 2.6). Essa restrição é do tipo ponte e será apresentada adiante, na Seção 2.7.

2.4 Métodos de planos-de-corte

O procedimento criado pelos três autores foi uma das versões pioneiras daquilo que passou a ser chamado de método de planos-de-corte. Esse nome decorre do fato de que cada restrição violada que é introduzida é um hiperplano de $\mathbb{R}^n$ que reduz (“corta”) o poliedro do programa linear com o qual o simplex trabalha. Esse procedimento pode ser descrito de uma forma mais geral do seguinte modo:

Procedimento 2.3 Método de Planos-de-Corte

Consideremos um problema de otimização de uma função linear sobre um poliedro P cujos vértices correspondem às soluções do problema.

1. Monte um programa linear inicial com restrições válidas para P .
2. Resolva o programa linear obtendo uma solução $\bar{x}$.
3. Se $\bar{x}$ pertence a P , pare. Se não, introduza no programa linear uma ou mais restrições válidas para P violadas por $\bar{x}$ e volte para 2.

A Figura 2.7 ilustra duas iterações desse método. Note que cada corte separa o poliedro do problema do poliedro relaxado, e que os dois cortes introduzidos não são suficientes para a obtenção da solução ótima do problema.

2.5 Geração automática de restrições violadas

Para resolver as instâncias do PCV consideradas, os autores não fizeram uso do computador para procurar as restrições válidas violadas. Na resolução de instâncias maiores poderia ser difícil “enxergar” essas restrições. Assim, o próximo passo natural para o aprimoramento do método usado pelos autores é o de escrever um algoritmo que gere automaticamente essas restrições pelo computador. Esse algoritmo deve resolver o seguinte problema:

Problema 2.4 Problema da Separação: *Dado um poliedro P e um ponto $\bar{x} \notin P$, obtenha uma restrição $ax \leq b$ válida para P que não é satisfeita por $\bar{x}$.*

Em 1958, Gomory obteve uma solução desse problema para o caso em que $\bar{x}$ é uma solução do programa linear, como ocorre no método de plano de corte. O autor definiu, em Gomory (1958), uma classe de restrições, chamadas *cortes de Gomory*, que podem ser geradas por um algoritmo. Além disso, ele mostrou que o método de planos-de-corte utilizando essa classe de restrições obtinha, após um número finito de passos, a solução do problema. Seu resultado podia ser aplicado ao PCV, bem como a qualquer outro problema abordado da mesma forma via programação linear.

Esse algoritmo, que prometia tornar tratáveis problemas difíceis como o PCV, revelou-se, no entanto, ineficiente na prática. Isso se deu porque os cortes que precisavam ser gerados pelo algoritmo apresentavam números fracionários que necessitavam de muita memória para serem representados, de modo que a memória usada simplesmente “explodia”. Apesar da garantia teórica apresentada de que o algoritmo obteria a solução, nenhuma garantia era oferecida quanto ao limite de memória e de tempo necessários.

Um resultado importante de Grötschel, Lovász e Schrijver (1981) mostra que a complexidade de um problema de otimização é a mesma do problema da separação correspondente. Isso nos dá poucas esperanças de encontrar um algoritmo polinomial para separação de problemas NP-completos (a menos que $P=NP$).

2.6 Qualidade dos cortes

A dificuldade prática encontrada na utilização dos cortes de Gomory fez com que os pesquisadores passassem a se preocupar com a “qualidade” das restrições a serem geradas, isto é, com a sua capacidade de fazer com que o método de planos-de-corte atinja a solução em menor tempo possível.

Um indicador de qualidade da restrição seria o de saber se o respectivo plano de corte “toca” o poliedro do problema (i.e. se é uma restrição suporte) e, melhor ainda, se “toca” o poliedro em um número máximo de vértices (i.e. se a restrição define uma faceta do poliedro — veja Teorema 0.4). Na Figura 2.8 a solução do poliedro relaxado é separada por 3 cortes. O primeiro não toca o poliedro; o segundo toca, correspondendo a uma inequação suporte; e o terceiro define uma faceta.

Observou-se então que os cortes de Gomory em geral não definem facetas e nem mesmo “tocam” o poliedro, isto é, nem mesmo são inequações suporte.

Figura 2.8: Cortes de diferentes qualidades

No entanto, é natural esperar que os planos-de-corte definindo facetas sejam os de melhor qualidade. Uma indicação nesse sentido está no fato de que uma descrição de um poliedro é não-redundante se e somente se ela é composta por restrições definindo facetas (Teorema 0.5).

Um novo problema de separação passou então a ser estudado.

Problema 2.5 *Identificação de facetas: Dado um poliedro P e um ponto $\bar{x} \notin P$, obtenha uma restrição $ax \leq b$ definindo faceta em P que não é satisfeita por $\bar{x}$.*

Esse problema foi estudado por Karp e Papadimitriou (1982). Revelou-se, assim como o anterior (2.4), intratável na prática. Por esse motivo, os pesquisadores passaram a tentar resolver um problema mais restrito:

Problema 2.6 *Identificação de facetas de uma certa classe: Dado um poliedro P , uma classe $\mathcal{L}$ de facetas de P e um ponto $\bar{x} \notin P$, obtenha, se existir, uma restrição $ax \leq b$ da classe $\mathcal{L}$ em P que não é satisfeita por $\bar{x}$.*

Tomando esse problema como subrotina de problemas difíceis, então, quanto maior o número de classes de facetas conhecidas e quanto melhor o algoritmo resolvendo o problema, melhor seria funcionamento do algoritmo de plano-de-corte. Iniciou-se, então, um grande esforço de pesquisa direcionada ao estudo de facetas de poliedros relacionados com problemas difíceis e de algoritmos para a identificação dessas facetas.

2.7 Facetas do PCV

O poliedro do PCV é provavelmente o que mais foi estudado até hoje. Um número grande de classes de facetas foi encontrado. Mostrou-se também

que as restrições de eliminação de subcircuitos (2.3), utilizadas pelos autores Dantzig, Fulkerson e Johnson, constitui uma classe de facetas. A Tabela 2.1 apresenta outras classes de facetas descobertas.

ano	classe de facetas	autor
1954	restrições de elim. de subcircuitos	Dantzig <i>et al.</i>
1965	restrições de 2-emparelhamento	Edmonds
1973	restrições pente simples	Chvátal
1979	restrições pente	Grötschel e Padberg
1985	restrições de árvores de cliques	Grötschel e Pulleyblank
1988	restrições de estrela	Fleischmann

Tabela 2.1: Classes de facetas do PCV

Os autores das três primeiras classes de inequações não mostraram que elas definem facetas do PCV. Esta prova está em Grötschel e Padberg (1979).

Para cada uma dessas classes de facetas, diversos algoritmos de identificação foram desenvolvidos. Só é conhecido algoritmo polinomial exato para identificação de facetas da primeira classe (veja próxima seção). Para as demais classes são usados algoritmos heurísticos descritos em Padberg e Hong (1980), Crowder e Padberg (1980), Padberg e Rao (1982), Grötschel e Holland (1991), Padberg e Rinaldi (1990b) e Applegate *et al.* (1994) (as três últimas referências trazem os algoritmos mais aprimorados).

Apresentamos agora as restrições de 2-emparelhamento.

Teorema 2.7 *Para todo $W \subseteq V$ e $F \subseteq \delta(W)$, a restrição*

$$x(E(W)) + x(F) \leq |W| + \left\lfloor \frac{|F|}{2} \right\rfloor \quad (2.6)$$

é válida para o PCV.

Prova: Vamos supor que x representa um circuito hamiltoniano. De (2.4), temos

$$x(E(W)) = |W| - \frac{x(\delta(W))}{2}.$$

Assim, como

$$x(F) \leq \min\{|F|, x(\delta(W))\},$$

segue que

$$\begin{aligned} x(E(W)) + x(F) &\leq |W| - \frac{x(\delta(W))}{2} + \min\{|F|, x(\delta(W))\} \\ &= |W| + \min\{|F| - \frac{x(\delta(W))}{2}, \frac{x(\delta(W))}{2}\} \\ &\leq |W| + \left\lfloor \frac{|F|}{2} \right\rfloor. \end{aligned}$$

□

O Teorema 2.9, apresentado a seguir, generaliza o resultado desse teorema. Optamos por apresentar essa demonstração, porque ela expõe a idéia básica que está por trás dos dois teoremas, facilitando a compreensão da demonstração do próximo.

Como vimos, as restrições de 2-emparelhamento se referem a um conjunto W , chamado *cabo*, e a arestas de $\delta(W)$, que chamamos *dentes*. A generalização de Chvátal (1973) consistiu em considerar dentes de maior tamanho (não apenas de dois vértices), tendo ainda um vértice em comum com o cabo. Na restrição podemos ter, então, um clique com número variado de vértices, ao invés de uma aresta — que é um clique de dois vértices. Essa restrição passou a ser chamada de restrição de *pente simples*.

Já para as restrições pente gerais, admite-se que os dentes interceptem o cabo em mais de um vértice. Essa generalização é devida a Grötschel e Padberg (1979). Na Figura 2.6 apresentamos uma restrição desse tipo.

As restrições de árvores de clique trazem a novidade que vários cabos passam a existir em vez de somente um. Assim como os dentes, os cabos não se interceptam, mas agora um dente pode interceptar mais do que um cabo. A Figura 2.9 ilustra essas generalizações das restrições de 2-emparelhamento.

Definimos árvore de cliques do seguinte modo:

Definição 2.8 *Dado um grafo completo G de n vértices, uma árvore de cliques é um subgrafo conexo de G correspondendo a uma união de cliques que apresentam as seguintes propriedades:*

Figura 2.9: A: 2-emparelhamento B: comb simples C: comb D: árvore de cliques

1. Para cada clique, seu conjunto de vértices é um cabo ou um dente.
2. Os cabos não se interceptam.
3. Os dentes não se interceptam.
4. Cada dente tem no mínimo um vértice não pertencendo a nenhum cabo.
5. A remoção de um clique que intercepta dois outros cliques desconecta o subgrafo.

As restrições de árvores de clique são as seguintes:

Teorema 2.9 Dada uma árvore de cliques definida pelos cabos $H_1, \dots, H_r$ e pelos dentes $T_1, \dots, T_s$, a restrição

$$\sum_{i=1}^r x(E(H_i)) + \sum_{j=1}^s x(E(T_j)) \leq \sum_{i=1}^r |H_i| + \sum_{j=1}^s (|T_j| - t_j) - \left\lfloor \frac{s}{2} \right\rfloor, \quad (2.7)$$

onde t_j representa o número de cabos interceptados pelo conjunto T_j , é válida para o PCV.

Prova: Seja x um vetor representando um circuito hamiltoniano. Por simplicidade, dada uma árvore de clique C , estaremos denotando o lado esquerdo da respectiva restrição por $e(x, C)$ e o direito por $d(C)$.

Usaremos indução no número de cabos. Se a árvore de clique não tem cabos a restrição correspondente é uma restrição de eliminação de subcircuitos, cuja validade já foi mostrada.

Vamos supor então que o teorema vale para todas as árvores de clique com r cabos, e assumir que C é uma árvore de clique composta por $r + 1$ cabos. Seja H um cabo qualquer de C e sejam $T_1, \dots, T_k$ os dentes de C que interceptam H . É fácil ver que, retirando as arestas em $E(H) \setminus \bigcup_{i=1}^k E(T_i)$ de C , obtemos k árvores de clique. Para $i = 1, \dots, k$, seja C_i a árvore que contém T_i e seja s_i o número de dentes nessa árvore. Observando que cada conjunto T_i intercepta um cabo a mais em C do que em C_i , e que $\sum_{i=1}^k s_i = s$ temos:

$$\begin{aligned} \sum_{i=1}^k d(C_i) &= d(C) - |H| + k - \left\lfloor \frac{s}{2} \right\rfloor + \sum_{i=1}^k \left\lfloor \frac{s_i}{2} \right\rfloor \\ &\geq d(C) - |H| + k - \left\lfloor \frac{s}{2} \right\rfloor + \left\lfloor \frac{\sum_{i=1}^k s_i}{2} \right\rfloor - \left\lfloor \frac{k}{2} \right\rfloor \\ &= d(C) - |H| + k - \left\lfloor \frac{s}{2} \right\rfloor + \left\lfloor \frac{s}{2} \right\rfloor - \left\lfloor \frac{k}{2} \right\rfloor \\ &= d(C) - |H| + \left\lfloor \frac{k}{2} \right\rfloor \end{aligned}$$

Examinemos agora o caso em que, para $T \in \{T_1, \dots, T_k\}$, vale que $x(\delta(H) \cap E(T)) = 0$. Seja D a árvore de clique em $\{C_1, \dots, C_k\}$ contendo T e seja D' a árvore de clique onde o clique do conjunto T é substituído pelo clique do conjunto $T \setminus H$. Observemos que $e(x, D) = e(x, D') + x(E(T \cap H))$,

já que $x(E(T)) = x(E(T \setminus H)) + x(E(T \cap H))$, e que $d(D) = d(D') + |T \cap H|$. Aplicando então a hipótese de indução para D' e a restrição de eliminação de subcircuito a $T \cap H$, temos que a restrição $e(x, D) \leq d(D)$ não está satisfeita com igualdade, isto é, $e(x, D) \leq d(D) - 1$.

Seja t o número de dentes T em $\{T_1, \dots, T_k\}$ tais que $x(\delta(H) \cap E(T)) = 0$. Como $x(\delta(H)) \geq k - t$ temos, pela Proposição 2.1, que $x(E(H)) \leq |H| - \left\lceil \frac{k-t}{2} \right\rceil$ (na demonstração do Teorema 2.7 tínhamos o equivalente a $k - t = x(F)$). Portanto,

$$\begin{aligned} e(x, C) &= \sum_{i=1}^k e(x, C_i) + x(E(H)) \\ &\leq \sum_{i=1}^k e(x, C_i) + |H| - \left\lceil \frac{k-t}{2} \right\rceil \\ &\leq d(C) + \left\lceil \frac{k}{2} \right\rceil - t - \left\lceil \frac{k-t}{2} \right\rceil \\ &\leq d(C) \end{aligned}$$

□

Observemos que na restrição (2.5) apresentada na Figura 2.6 temos uma restrição pente, já que existe apenas um cabo, mas não é pente simples, pois os dentes o cabo em mais de um vértice. Temos assim $r = 1$, $t_1 = t_2 = t_3 = 1$. Com base no teorema que acabamos de mostrar, podemos assim facilmente verificar que a restrição é de fato válida.

Para que, além de serem válidas, as restrições de árvores de cliques definam facetas é necessário que o número de dentes interceptando cada cabo seja ímpar e maior do que 3. A prova desse resultado está em Grötschel e Pulleyblank (1986).

As restrições de estrela também generalizam as restrições pente e continuam apresentando um único cabo. Os vértices dos dentes, no entanto, são particionados em *círculos* e essa partição é levada em conta na formulação da desigualdade. O enunciado dessas restrições e o resultado que apresenta condições em que essas restrições definem facetas estão em Fleischmann (1988).

As demonstrações de que as restrições de árvores de cliques correspondem a facetas é bastante trabalhosa, mesmo para os seus casos particulares.

Daremos aqui o resultado equivalente para as restrições de eliminação de subcircuitos.

Existem duas estratégias para provar que um determinado tipo de restrição é faceta. Vamos supor que seja dada uma restrição $ax \leq \alpha$ válida para um poliedro P e queremos mostrar que $ax \leq \alpha$ é define faceta de P .

A primeira estratégia é conhecida como *método direto*. Baseia-se no item 3 do teorema 0.4 e consiste em apresentar vetores afim-independentes pertencentes ao conjunto $\{x \in P \mid ax = \alpha\}$.

A segunda, chamada de *método indireto*, baseia-se no item 4 do mesmo teorema e divide-se em duas partes. Na primeira, assumimos que existe uma restrição válida $bx \leq \beta$ tal que $\{x \in P \mid ax = \alpha\} \subseteq \{x \in P \mid bx = \beta\}$. Usando um sistema de equações $Ax = d$ conhecido para P , determinamos λ , tal que o vetor $b' = b + \lambda A$ tenha certas características especiais, como apresentar coordenadas coincidindo com as de a . Na segunda parte, utilizando propriedades conhecidas a respeito de pontos x de P satisfazendo $ax = \alpha$, determinamos iterativamente os coeficientes de b' ainda desconhecidos. Exibindo dois vetores $\mu \geq 0$ e λ' tais que $b' = \mu a + \lambda' A$, concluímos a prova.

Para as demonstrações não-triviais utiliza-se em geral o método indireto. A aplicação do método direto só é viável quando existe uma estrutura relativamente simples dos vetores compondo a faceta.

Teorema 2.10 *As restrições de eliminação de subcircuitos (2.3) para $2 \leq |W| \leq n - 2$ definem facetas do poliedro do PCV.*

Prova: Esta demonstração é uma versão esmiuçada da demonstração do Teorema 11 em Grötschel e Padberg (1985).

Vamos demonstrar o teorema para $n \geq 6$ e $3 \leq |W| \leq n - 3$. Os casos em que $n = 3, 4$ ou em que $|W| = 2$ ou $|W| = n - 2$ podem ser resolvidos por inspeção. Por simplicidade, vamos supor que $W = \{1, \dots, k\}$. Representamos a restrição $x(E(W)) \leq |W| - 1$ por $ax \leq \alpha$ e denotamos por $Ax = \mathbf{2}$ o conjunto das restrições 2.2, onde $\mathbf{2}$ é o vetor de n entradas valendo 2.

Utilizaremos o método indireto, aplicando o item 4 do teorema 0.4. Para tanto, vamos supor que $bx \leq \beta$ é uma restrição válida que satisfaz $\{x \in PCV(n) \mid ax = \alpha\} \subseteq \{x \in PCV(n) \mid bx = \beta\}$ e mostrar que existem $\mu \geq 0, \lambda' \in \mathbb{R}^n$, tais que $b = \mu a + \lambda' A$.

Consideremos o conjunto de arestas $F := \{(1, i) \mid i = 2, \dots, n\} \cup \{(2, 3)\}$. É fácil observar que a matriz A_F , correspondente às colunas das arestas em

F , é não-singular. Tomando-se $\lambda' \in \mathbb{R}^n$ tal que $\lambda'_F = (a_F - b_F)A_F^{-1}$ e $\lambda'_e = 0$, para $e \in E \setminus F$, temos que $b' = b + \lambda'A$ satisfaz $b'_e = a_e$, para e em F . Lembrando que a é o vetor de incidência de $E(W)$, temos que

$$\begin{aligned} b'_{1i} &= 1 \quad (= a_{1i}), \quad i = 2, \dots, k, \\ b'_{1i} &= 0 \quad (= a_{1i}), \quad i = k+1, \dots, n, \\ b'_{23} &= 1 \quad (= a_{23}). \end{aligned}$$

Tomando $\beta' := \beta + \lambda'2$, temos que $bx = \beta$ se e somente se $b'x = \beta'$.

Iniciamos com o estudo dos coeficientes de b' correspondentes às arestas de $E(W)$ ainda não determinados. Seja $j \in \{4, \dots, k\}$, e tomemos um caminho C_1 passando pelos vértices $\{4, \dots, j-1, j+1, \dots, k\}$, um caminho C_2 passando por $\{k+1, \dots, n\}$, e C'_2 o caminho reverso de C_2 . Sejam τ_1 o circuito hamiltoniano $1-j-C_1-3-2-C_2-1$ e τ_2 o circuito hamiltoniano $1-3-C'_2-j-2-C_2-1$, onde C'_1 denota o caminho inverso de C_1 (veja Figura 2.10 — as linhas grossas indicam arestas cujos coeficientes já haviam sido determinados e estão sendo usados para a obtenção de um novo). Então, os respectivos vetores de incidência χ^{τ_1} e χ^{τ_2} satisfazem $ax \leq \alpha$ com igualdade já que possuem $k-1$ arestas de $E(W)$. Logo, satisfazem também $bx \leq \beta$ e $b'x \leq b'_0$ com igualdade. Isso implica que $0 = b'_0 - b_0 = b'\chi^{\tau_1} - b'\chi^{\tau_2} = b'_{1j} + b'_{23} - b'_{13} - b'_{2j} = 1 - b'_{2j}$. Assim, $b'_{2j} = 1$, para $j \in \{4, \dots, k\}$.

Figura 2.10: Determinando b'_e para $e \in E(W)$.

De posse desses valores, podemos repetir essa operação para i de 3 a $k-1$. O vértice i assume o papel que foi o do vértice 2 e o vértice 2 assume o papel do 3. Para cada i , tomamos j variando de $i+1$ a k . Chegamos assim a situação:

$$b'_{ij} = 1, \quad 1 \leq i < j \leq k.$$

Em seguida, passamos ao estudo dos coeficientes de b' correspondentes às arestas de $\delta(W)$ ainda não determinados. Seja então $j \in \{k+1, \dots, n\}$, e consideremos um caminho C_3 passando por $\{4, \dots, k\}$, um caminho C_4 passando por $\{k+1, \dots, j-1, j+1, \dots, n\}$ e C'_4 o caminho reverso de C_4 . Sejam τ_3 o circuito hamiltoniano $1-C_3-2-3-C_4-j-1$ e τ_2 o circuito hamiltoniano $1-C_3-2-j-C'_4-3-1$ (veja Figura 2.11). Como seus vetores de incidência também apresentam $k-1$ arestas em $E(W)$, então eles também satisfazem $ax = \alpha$ e $b'x = b'_0$. Obtemos $0 = b'\chi^{\tau_3} - b'\chi^{\tau_4} = b'_{23} + b'_{1j} - b'_{13} - b'_{2j} = -b'_{2j}$. Aplicando essa operação repetidamente com i no papel de 2 variando de 3 a k e i' qualquer em $\{3, \dots, k\}$ diferente de i no papel de 3 chegamos a

$$b'_{ij} = 0, \quad i \in \{1, \dots, k\}, \quad j \in \{k+1, \dots, n\}$$

Figura 2.11: Determinando b'_e para $e \in \delta(W)$.

Por último, vamos estudar os coeficientes de b' correspondentes às arestas de $E(V \setminus W)$. Podemos mostrar que tais coeficientes correspondem a um único valor γ . Vamos simplificar a notação escrevendo $k' := k+1$ e fazer $\gamma := b'_{k'n}$. Tomando j em $\{k', \dots, n\}$, um caminho C_5 passando por $\{2, \dots, k\}$, um caminho C_6 passando por $\{k', \dots, j-1, j+1, \dots, n-1\}$ e C'_6 o caminho reverso de C_6 e considerando os circuitos $1-C_5-k'-n-C_6-j-1$ e $1-C_5-k'-j-C'_6-n-1$ (veja Figura 2.12), obtemos $0 = b'_{1j} + b'_{k'n} - b'_{k'j} - b'_{1n} = 0 + \gamma - b'_{k'j} - 0$, donde $b'_{k'j} = \gamma$. Aplicando a mesma construção para i percorrendo os valores de $\{k+2, \dots, n-1\}$ no papel de k' ; fazendo k' assumir o papel de n ; e variando, para cada i , j de $i+1$ a n , obtemos:

$$b'_{ij} = \gamma, \quad k+1 \leq i < j \leq n.$$

Figura 2.12: Determinando b'_e para $e \in E(V \setminus W)$.

Temos então que $b'x = x(E(W)) + \gamma x(E(V \setminus W))$. Como $a\chi^{\tau_1} = \alpha$ então $b'\chi^{\tau_1} = b'_0$. Assim, $b'_0 = |W| - 1 + \gamma(|V| - |W| - 1)$. Vamos tomar $\mu = 1 + \gamma$, $\lambda''_i = -\gamma/2$, para $i = 1, \dots, k$ e $\lambda''_i = \gamma/2$, para $i = k + 1, \dots, n$, e mostrar que

$$b' = \mu a + \lambda'' A$$

vale. Para tanto, notemos que para cada aresta e , a coluna A_e apresenta apenas duas entradas não-nulas (valendo 1), correspondendo aos dois extremos da aresta. Daí temos que para $e \in E(W)$, $a_e = 1$ e $\lambda'' A_e = -\gamma/2 - \gamma/2 = -\gamma$; para $e \in \delta(W)$, $a_e = 0$, $\lambda'' A_e = \gamma/2 - \gamma/2 = 0$; e para $e \in E(V \setminus W)$, $a_e = 0$, $\lambda'' A_e = \gamma/2 + \gamma/2 = 0$.

Precisamos mostrar ainda que $\mu \geq 0$. Como supusemos $|W| > 3$ e $n \geq 6$ podemos tomar um circuito hamiltoniano τ tal que $\chi^{\tau}(\delta(W)) = 4$. Temos então que $\chi^{\tau}(E(W)) = |W| - 2$ e $\chi^{\tau}(E(V \setminus W)) = |V \setminus W| - 2$, de modo que $b'\chi^{\tau} = |W| - 2 + \gamma(|V \setminus W| - 2)$. Como $b'x \leq b'_0$ é uma restrição válida, temos também que $b'\chi^{\tau} \leq |W| - 1 + \gamma(|V \setminus W| - 1)$. Assim, $0 = b'\chi^{\tau} - b'\chi^{\tau} \leq 1 + \gamma = \mu$.

Finalmente,

$$b = b' + \lambda' A = \mu a + \lambda'' A + \lambda' A = \mu a + (\lambda' + \lambda'') A,$$

como queríamos. □

Observamos que foi necessário exibir um grande número de circuitos hamiltonianos, o que dá uma indicação de que o método indireto é, de certa

forma, semelhante ao método direto. De fato, só foi possível determinar os coeficientes de b porque conseguimos obter vetores de incidência de circuitos hamiltonianos afim-independentes. Ocorre, todavia, que exibir os coeficientes de b obtidos é mais simples do que mostrar a independência afim desses vetores.

Goemans (1995) aponta que as diversas classes de facetas do PCV têm qualidades distintas. No artigo, o autor mostra que o aumento do limitante inferior usando tipos diferentes de inequações difere entre as diversas facetas.

2.8 Identificação de restrições de eliminação de subcircuitos

Abordaremos nessa seção as pesquisas realizadas com relação à classe de facetas de restrições de eliminação de subcircuitos. Essas pesquisas oferecem uma visão dos esforços realizados para o aprimoramento dos algoritmos de planos-de-corte para o PCV.

O problema é então o seguinte:

Problema 2.11 (*Identificação de restrições de eliminação de subcircuitos*)
Dado $x \in P_0^n$, obtenha, se houver, um ou mais conjuntos W , $\emptyset \neq W \subset V$, tais que as respectivas restrições do tipo (2.3) estejam violadas.

Como vimos na Proposição 2.1, quando $x \in P_0^n$ a restrição (2.3) está violada para um conjunto $W \subseteq V$, se e somente se $x(\delta(W)) < 2$. Desta forma, podemos resolver o nosso problema com um algoritmo que obtenha, no grafo com capacidades (G, x) , um corte de capacidade mínima (veja Seção 0.2). Gomory e Hu (1961) desenvolveram um algoritmo para esse problema que apresenta complexidade de tempo polinomial, mas que requer o cálculo de $|V| - 1$ (s, t) -cortes de capacidade mínima em (G, x) . Esses cortes podem ser obtidos em tempo próximo a $O(n^3)$ usando um algoritmo de fluxo máximo. Assim, torna-se muito caro executar diversas vezes esse algoritmo de separação. Crowder e Padberg (1980) propuseram, então, alterações ao algoritmo para torná-lo mais rápido, elaborando um procedimento que se baseia na operação de contração de arestas.

Definição 2.12 Dado um grafo com capacidades (G, x) a operação de contração de uma aresta $e = (u, v)$ de G é definida pelos seguintes passos:

Passo 1: Substitua os vértices u e v por um novo vértice u' .

Passo 2: Troque cada par de arestas (u, w) , (v, w) por uma única aresta (u', w) e faça $x_{u'w} := x_{uw} + x_{vw}$.

Passo 3: As demais arestas (u, s) e (v, t) são substituídas, respectivamente, por (u', s) e (u', t) .

O grafo obtido é denotado por G/e e as novas capacidades por x/e , de modo que temos um novo grafo com capacidades $(G/e, x/e)$. Dizemos que a aresta (u', w) de $(G/e, x/e)$ induz as arestas (u, w) e (v, w) de (G, x) ; que arestas (u', s) e (u', t) de $(G/e, x/e)$ induzem respectivamente as arestas (u, s) e (v, t) de (G, x) ; e que as demais arestas (s, t) de $(G/e, x/e)$, com $s, t \neq u'$, induzem as arestas (s, t) de (G, x) . Dizemos também, por extensão, que um conjunto E_1 de arestas de $(G/e, x/e)$ induz um conjunto E_2 de arestas de (G, x) , se E_2 corresponde ao conjunto das arestas induzidas pelas arestas de E_1 .

Procedimento 2.13 Entrada: Grafo $(G, x) = (V, E, x)$.

faça $L_u := u$, para todo $u \in V$;

faça $(G', x') := (V', E', x') := (V, E, x)$;

enquanto (G, x) tem arestas de capacidade maior ou igual a 1 faça

seja $e := (u, v)$ uma aresta de capacidade maior ou igual a 1;

crie u' e faça $L_{u'} := L_u \cup L_v$;

faça $(G', x') := (G'/e, x'/e)$;

se $x(\delta(u')) < 2$

guarde $L_{u'}$; /* restrição de eliminação de subcircuito violada */

se $|V'| > 1$ então

aplique o algoritmo Gomory-Hu a (G', x') ;

A idéia por trás desse procedimento está no fato de que a contração de arestas com capacidade maior ou igual a 1 tende a reduzir bastante o tamanho do grafo, tornando a busca de um corte mínimo mais rápida. Além disso, essa operação preserva propriedades do grafo, permitindo que o corte mínimo do grafo contraído corresponda a um corte mínimo do grafo não contraído. Mostraremos a seguir essas idéias.

Definição 2.14 Dado um grafo com capacidades (G, x) , dizemos que uma aresta e de G pode ser contraída se existe um corte mínimo em $(G/e, x/e)$ que induz um corte mínimo em (G, x) .

Teorema 2.15 *O Procedimento 2.13 resolve o problema da identificação de restrições de eliminação de subcircuitos.*

Prova: Os fatos abaixo garantem que um corte de capacidade mínima é obtido sempre que tal corte de capacidade menor do que 2 existir. O teorema segue então da Proposição 2.1. Tomamos os vértices u , v e u' tais como na definição de contração.

Fato 2.16 *Dado um grafo com capacidades (G, x) , uma aresta e de G pode ser contraída se existe um corte de capacidade mínima que não contém e .*

Prova: Basta observar que se F é um conjunto de arestas de $(G/e, x/e)$ e F' é o conjunto de arestas de (G, x) induzido por F , então $x(F) = x(F')$ e se F é um corte de $(G/e, x/e)$, então F' é um corte de (G, x) .

Fato 2.17 *Dado um grafo com capacidades (G, x) que satisfaz*

$$x(\delta(v)) \leq 2, \text{ para todo } v \in V \quad (2.8)$$

e uma aresta de G tal que $x_e \geq 1$, temos que $(G/e, x/e)$ também satisfaz (2.8).

Prova: Considerando $x' = x/e$, então temos que $x'(\delta(u')) = x(\delta(u)) - x_e + x(\delta(v)) - x_e \leq 2 - 1 + 2 - 1 = 2$. Para os outros vértices v de $(G/e, x/e)$, a igualdade $x'(v) = x(v)$ vale como consequência direta da definição.

Fato 2.18 *Dado um grafo com capacidades (G, x) que satisfaz (2.8), se $e = (u, v)$ for uma aresta de G tal que $x_e \geq 1$, então e pode ser contraída.*

Prova: Consideremos $W \subseteq V$ tal que $\delta(W)$ seja um corte mínimo de (G, x) contendo e . Digamos que $u \in W$. Temos então que

$$x(\delta(W - u)) = x(\delta(W)) + x([u : W - u]) - x([u : V - W])$$

Como $e \in [u : V - W]$, $\delta(u) = [u : W - u] + [u : V - W]$, $x_e \geq 1$ e $x(\delta(u)) \leq 2$, então $x([u : W - u]) \leq 1$ e $x([u : V - W]) \geq 1$, donde se segue que $x(\delta(W - u)) \leq x(\delta(W))$.

Assim $\delta(W - u)$ também é um corte mínimo, e então, pelo Fato 2.16, a aresta e pode ser contraída.

Isso conclui a demonstração do Teorema 2.15.

□

Eliminando-se a chamada, feita no final, ao algoritmo de Gomory-Hu, esse procedimento torna-se muito rápido e consegue obter em muitos casos restrições violadas. Em contrapartida, deixa de ser exato. Segundo Padberg e Rinaldi (1990b), no entanto, experimentos computacionais indicam que a resolução exata do problema da identificação das restrições de eliminação de subcircuitos melhora a convergência do algoritmo que resolve o PCV. Para grandes instâncias, no entanto, o grafo (G', x') obtido após o término das contrações ainda é grande demais para poder ser tratado eficientemente pelo algoritmo Gomory-Hu.

Em vista disso, esses dois autores propuseram um novo procedimento que, segundo eles, ocasionou uma “dramática” redução do número de computações de (s, t) -cortes. Esse procedimento baseia-se nas proposições abaixo, que fornecem novas condições para que uma aresta possa ser contraída.

Proposição 2.19 *Seja (G, x) um grafo com capacidades não-negativas e $e = (u, v)$ uma aresta qualquer G . Se existe $Y \subseteq N(u) \cap N(v)$ tal que*

(a) $x(\delta(u)) \leq 2x([u : W + v])$ para todo $W \subseteq Y$ ou

(b) $x(\delta(v)) \leq 2x([v : W + u])$ para todo $W \subseteq Y$,

então ou um dos cortes $\delta(u)$ ou $\delta(v)$ é um corte de capacidade mínima ou a aresta e pode ser contraída.

Prova: Se e não pertence a nenhum corte de capacidade mínima, então e pode ser contraída. Vamos admitir, então, que existe um corte de capacidade mínima $\delta(U)$ contendo e , com $u \in U$.

Vamos supor que a condição (a) é satisfeita. Se $U = \{u\}$, $\delta(u)$ é um corte de capacidade mínima, e não há nada a ser mostrado. Caso contrário, temos que para $X := U \setminus \{u\}$, $\delta(X)$ define um corte. Seja $W := Y \cap (V \setminus U)$. De

$$[u : W + v] \subseteq [u : V \setminus U],$$

$$[u : X] \cup [u : W + v] \subseteq \delta(u),$$

$$[u : X] \cap [u : W + v] = \emptyset$$

(veja a Figura 2.13) e da não-negatividade das capacidades segue-se que

$$x([u : V \setminus U]) \geq x([u : W + v]) \text{ e}$$

$$x([u : X]) \leq x(\delta(u)) - x([u : W + v]).$$

Assim, temos

$$\begin{aligned} x(\delta(X)) &= x(\delta(U)) - x([u : V \setminus U]) + x([u : X]) \\ &\leq x(\delta(U)) - x([u : W + v]) + x(\delta(u)) - x([u : W + v]) \\ &\leq x(\delta(U)), \end{aligned}$$

o que significa que $\delta(X)$, também é um corte de capacidade mínima. Se é a condição (b) que vale, a prova é a mesma, com u e v trocando os papéis.

□

Figura 2.13: Representação dos conjuntos Y , W e X , de $e = (u, v)$ e dos cortes $[u : X]$ e $[u : W]$.

Para $x \in P_0^n$, as arestas de capacidade maior ou igual a 1 satisfazem as condições da Proposição 2.19. Portanto o Procedimento 2.13 consiste em um caso especial da aplicabilidade da proposição. A seguir, a definição e a proposição abaixo nos fornecem novas condições para a contração de arestas.

Proposição 2.20 *Dada uma aresta $(u, v) \in E$ e um corte $\delta(U)$ em G não contendo (u, v) , se*

$$x(u, v) + \sum_{w \in N(u) \cap N(v)} \min\{x(u, w), x(v, w)\} \geq x(\delta(U)),$$

então a aresta (u, v) pode ser contraída.

Prova: Se $\delta(X)$ é um corte contendo e , então para cada vértice w de $N(u) \cap N(v)$, ou (u, w) ou (v, w) está em $\delta(X)$. Então,

$$x(\delta(X)) \geq x(u, v) + \sum_{w \in N(u) \cap N(v)} \min\{x(u, w), x(v, w)\}.$$

Assim, como $x(\delta(U)) \leq x(\delta(X))$, tem que haver em G um corte de capacidade mínima não contendo e .

□

O procedimento de identificação de restrições de eliminação de subcircuitos desenvolvido pelos autores é o seguinte:

Procedimento 2.21 *Entrada:* Grafo $(G, x) = (V, E, x)$.

faça $L_u := u$, para todo $u \in V$;

faça $(G', x') := (V', E', x') := (V, E, x)$;

enquanto $|V'| > 1$ faça

se existe e satisfazendo as condições das Proposições 2.19 e 2.20 então
escolha uma tal aresta $e = (u, v)$;

senão

escolha uma aresta $e = (u, v)$ qualquer;

calcule o (u, v) -corte de cap. min. $[U : V' - U]$;

se $x(\delta(U)) < 2$

*guarde U ; /*restrição de eliminação de subcircuito violada */*

crie u' e faça $L_{u'} := L_u \cup L_v$;

faça $(G', x') := (G'/e, x'/e)$;

se $x(\delta(u')) < 2$

*guarde $L_{u'}$; /*restrição de eliminação de subcircuito violada */*

Teorema 2.22 *O Procedimento 2.21 resolve o problema da separação de restrições de eliminação de subcircuitos.*

Prova: Como no Teorema 2.15, basta mostrar que um corte de capacidade mínima com capacidade menor do que 2 é obtido sempre que tal corte existir. Para garantir que, no decorrer do algoritmo, as condições das Proposições 2.19 e 2.20 estão satisfeitas para o grafo (G', x') , basta observar que:

Fato 2.23 *Dado um grafo com capacidades não-negativas (G, x) e uma aresta qualquer e de G , $(G'/e, x'/e)$ também é um grafo com capacidades não-negativas.*

Para concluir a justificativa do procedimento de contração de arestas falta ainda observar que a contração da aresta escolhida ao acaso não impede que o corte mínimo seja obtido, que é o que garante o fato abaixo:

Fato 2.24 *Dado um grafo com capacidades (G, x) , consideremos dois vértices distintos u e v , e um (u, v) -corte de capacidade mínima $\delta(U)$. Então, se existe no grafo um corte $\delta(X)$ de capacidade menor do que a de $\delta(U)$, ou $\{u, v\} \subseteq X$ ou $\{u, v\} \subseteq V - X$.*

Prova: Supondo o contrário, temos que $\delta(X)$ é um (u, v) -corte, o que contradiz a hipótese de minimalidade da capacidade de $\delta(U)$. □

A verificação da existência de um conjunto Y de tamanho arbitrário satisfazendo as condições da Proposição 2.19 é um problema NP-completo, como está indicado em Padberg e Rinaldi (1990b). Restringindo a busca para conjuntos Y vazios ou com apenas 1 elemento, o Procedimento 2.21 ainda tem a mesma complexidade de pior caso que o algoritmo de Gomory-Hu. Padberg e Rinaldi testaram as condições da Proposição 2.19 para conjuntos Y de cardinalidade não maior do que 2, afirmando que experimentos deveriam ser realizados para verificar se essa é de fato a opção mais interessante.

2.9 O método branch-and-cut

A inexistência de um algoritmo eficiente para a resolução do problema geral de identificação de facetas (2.5) nos deixa sem garantias de que o algoritmo de plano-de-corte obtenha solução ótima. Pode acontecer, por exemplo, que o ponto devolvido pelo simplex não pertença ao poliedro $PCV(n)$ e não exista nenhuma faceta dentre aquelas para as quais dispomos de algoritmos de identificação que separam o ponto do poliedro; ou ainda, que uma tal faceta exista, mas que os procedimentos heurísticos disponíveis para a respectiva classe de facetas não a encontrem.

Para garantir a obtenção de uma solução, o algoritmo de planos-de-corte foi combinado com o algoritmo *branch-and-bound*, que também era utilizado por diversos autores para a resolução do PCV (desde Little, Murty, Sweeney, Karel (1963)), passando então a ser chamado de *branch-and-cut*.

A idéia é executar o algoritmo de planos-de-corte até um certo ponto. Escolhemos em seguida uma aresta e do grafo e passamos a buscar separadamente (i) um circuito hamiltoniano de custo mínimo que utilize e (i.e. fixamos $x_e = 1$); e (ii) um circuito hamiltoniano de custo mínimo que não utilize e (i.e. fixamos $x_e = 0$). Novamente aplicamos o algoritmo de planos-de-corte a esses dois problemas, novos *branchings* são eventualmente realizados, e assim por diante. Cria-se dessa forma uma árvore binária, que chamamos árvore de *branching*, onde cada nó representa um problema de programação linear diferente, apresentando um conjunto de variáveis fixadas em 1 ou 0.

O método branch-and-cut trabalha com um limitante inferior e um limitante superior do valor ótimo do problema.

Como já vimos, a solução de um problema linear correspondente a um nó não representa em geral uma solução viável para o PCV. No entanto, sabemos que, se existe uma solução ótima do problema compatível com as variáveis fixadas naquele nó, então o valor da função objetivo naquele nó é menor ou igual ao valor ótimo do problema.

Dessa forma, podemos notar que, apesar do valor ótimo do problema poder ser inferior ao valor da função objetivo de uma certa folha, ele não pode ser inferior ao valor da função objetivo de todas as folhas ao mesmo tempo. Assim, o mínimo dentre os valores das funções objetivo das folhas da árvore de *branching* é utilizado como limitante inferior.

Por outro lado, o valor da função objetivo de um nó cuja solução é um ponto viável do PCV oferece um limitante superior para o valor ótimo do

problema. O algoritmo guarda o menor desses valores, juntamente com a solução correspondente.

Durante a execução do algoritmo, se o limitante inferior atingir o valor do limitante superior, o algoritmo pára. A solução viável correspondente ao limitante superior será uma solução ótima do problema.

Além disso, quando dispõe de um limitante superior, o algoritmo é capaz de eliminar certos nós da árvore. Isso se dá, quando a função objetivo de um nó supera o valor do limitante superior, o que indica que o conjunto dos valores fixados para as arestas, que deu origem àquele nó, não pode estar numa solução ótima.

Dessa forma, temos que a obtenção de um limitante superior de boa qualidade (i.e. próximo ao valor ótimo do problema) no início da execução será um dos principais fatores na determinação do tempo que o algoritmo vai levar para a concluir a otimização. Para esse fim, em vez de aguardar o aparecimento do primeiro ponto viável como solução de algum nó da árvore para somente então inicializar o limitante superior, pode ser interessante executar um método heurístico para a resolução do problema, que nos fornecerá uma solução viável e servirá para essa inicialização. A heurística pode utilizar como entrada o valor da solução da relaxação linear atual, podendo, então, ser executado diversas vezes.

É interessante notar que a justificativa da utilização de *branching* não está somente na garantia da obtenção de uma solução. Do ponto de vista prático, pode ocorrer que, depois da inclusão de muitas restrições no programa linear, o algoritmo de planos-de-corte não progrida mais. Ou seja, o valor da função objetivo pode não aumentar depois de algumas iterações ou esse aumento pode ser muito pequeno. Nesse caso, pode ser preferível usar já o *branching* mesmo que novas restrições violadas possam ainda ser obtidas. É isso que se chama, na literatura, evitar o “tailing-off”.

O método pode ser esquematizado da forma abaixo. Estamos denotando por $P(\mathcal{L}, F_0, F_1)$ o programa linear formado pelas restrições básicas (2.1) e (2.2) e pelas restrições do conjunto $\mathcal{L}$, apresentando as variáveis pertencentes a $F_0 \subseteq E$ fixadas em 0 e as variáveis pertencentes a $F_1 \subseteq E$ fixadas em 1.

1. (Opcional) Obtenha uma solução heurística x^* .
2. Faça $\mathcal{T} = \{\langle \emptyset, \emptyset \rangle\}$, $\mathcal{L} = \emptyset$.

3. Se $\mathcal{T} = \emptyset$, pare. Caso contrário tome um par $\langle F_0, F_1 \rangle$ de $\mathcal{T}$ e faça $\mathcal{T} := \mathcal{T} - \langle F_0, F_1 \rangle$.
4. Resolva o programa linear $P(\mathcal{L}, F_0, F_1)$. Se o programa for inconsistente, vá para o passo 3. Caso contrário, seja $\bar{x}$ a solução ótima devolvida.
5. Se existir x^* e $c\bar{x} \geq cx^*$, vá para o passo 3.
6. Encontre uma ou mais restrições violadas por $\bar{x}$.
7. Se nenhuma for encontrada, vá para o passo 8. Caso contrário, adicione as restrições a $\mathcal{L}$ e vá para o passo 4.
8. Se $\bar{x}$ for inteiro, faça $x^* := \bar{x}$ e vá para o passo 3.
9. (Opcional) Encontre uma solução heurística $\hat{x}$ com base em $\bar{x}$. Se não existir x^* ou se $c\hat{x} \leq cx^*$, faça $x^* := \hat{x}$.
10. Tome uma aresta e tal que $0 < x_e < 1$. Faça $\mathcal{T} := \mathcal{T} + \langle F_0 + e, F_1 \rangle + \langle F_0, F_1 + e \rangle$ e vá para o passo 3.

O nome *branch-and-cut* apareceu pela primeira vez em Padberg e Rinaldi (1987). O método, no entanto, já havia sido utilizado em Grötschel, Jünger e Reinelt (1984) para a resolução do Problema de Ordenação Linear.

2.10 Aspectos críticos do método branch-and-cut

O método branch-and-cut tornou possível a resolução de grandes instâncias do PCV, como veremos na próxima seção. Os autores que obtiveram esses resultados apontaram diversos aspectos a serem observados que se revelaram críticos para o bom funcionamento do algoritmo.

Além de ser o problema que deu origem ao método branch-and-cut, o PCV é também, até hoje, o problema para o qual esse método foi mais aprimorado. Por isso, o estudo de tais problemas pode ser bastante útil, também para quem deseja aplicar o método branch-and-cut a outros problemas.

O principais aspectos críticos apontados são os seguintes:

1. *O método heurístico.* Como vimos, um solução heurística atingindo valor próximo do ótimo diminui o tempo de execução do algoritmo. O método de Lin e Kernighan (1973) e suas variações são as opções mais usadas, ainda que seja necessário um tempo de execução significativo para se obter boas soluções viáveis.
2. *Algoritmos de separação eficientes.* A eficiência aqui tem dois significados: os algoritmos devem ser rápidos e ao mesmo tempo devem ser capazes de encontrar, em boa parte das vezes, restrições violadas pelo ponto devolvido pelo simplex.
3. *Remoção e inserção de colunas.* Boa parte do tempo de execução do algoritmo branch-and-cut é tomada pelo simplex ou seu equivalente. Isso torna atraentes as técnicas existentes para se trabalhar somente com algumas das colunas da matriz. O custo reduzido das variáveis cujas colunas não estão sendo utilizadas indica se a função objetivo pode ser melhorada com a inserção daquela coluna. Vale notar que inserir novas colunas quando algumas novas restrições foram incluídas pode ser uma tarefa não trivial.
4. *Remoção e inserção de linhas.* Essa é uma outra técnica que visa a redução do tempo gasto pelo simplex. Após vários nós da árvore de branching serem otimizados, muitas das restrições obtidas por separação não estão mais ativas, ou seja, não estão mais tendo função de restringir soluções. Isso pode ser medido pela variável de folga (*slack variable*), que é a diferença entre o lado direito e o lado esquerdo da restrição. Removendo-se então as restrições inativas, o programa linear diminui de tamanho e tende a ser resolvido mais rapidamente. Um cuidado deve ser tomado, já que uma restrição que não está ativa em um nó da árvore pode estar ativa em outro nó. Por isso, deve-se guardar as restrições removidas em um reservatório, chamado de *pool* de restrições, para que, ao se mudar de nó, elas possam ser testadas e eventualmente reinseridas no programa linear. As restrições ativas num nó pai são fortes candidatas a continuarem ativas num nó filho. Por isso pode ser interessante guardar a informação de quais restrições estavam ativas ao final do processamento de cada nó.
5. *Escolha de um nó para processamento.* O crescimento da árvore *branching* pode depender da forma como são escolhidos os nós para o pro-

cessamento (passo 3 do algoritmo). A tática mais usada aqui é a de se tomar o nó cuja função objetivo atinge o valor mais baixo. Espera-se que assim a árvore de *branching* cresça o mínimo possível. Além disso, tenta-se aumentar o limitante inferior do problema.

6. *Escolha de uma aresta para branching.* A estratégia de escolha da aresta a ser usada para *branching* também pode influenciar de forma significativa o crescimento da árvore de *branching*. As estratégias mais utilizadas tendem a escolher a aresta cuja variável esteja o mais próximo possível de 0,5. Alguns critérios de desempate são propostos para o caso em que existam várias arestas cujos valores apresentem distâncias iguais ou aproximadamente iguais ao valor 0,5.
7. *Fixação de variáveis.* Em certos casos, é possível chegar à conclusão de que existe uma solução ótima que inclui ou que exclui uma determinada aresta e , de forma que possamos fixar seu valor em 1 ou 0. Dada a solução $\bar{x}$ obtida para algum nó, pode-se mostrar que se $\bar{x}_e$ vale 0 ou 1 e o custo reduzido correspondente r_e àquela variável satisfaz $|r_e| > c\bar{x} - c\bar{x}$, para algum ponto viável x^* , então a variável x_e pode ser fixada em seu valor. Fixando o valor de variáveis correspondentes a certas arestas, reduzimos o tamanho do problema de programação linear a ser resolvido e, portanto, podemos atingir mais rapidamente a solução ótima do problema.
8. *Estruturas de dados especiais para o armazenamento de restrições.* Para a solução de PCVs com mais de duas mil cidades, em que o número de arestas supera um milhão, torna-se desaconselhável o uso de vetores indexados pelas arestas. Por isso, estruturas de dados especiais devem ser utilizadas para o armazenamento da informação relativa às arestas, como é o caso dos coeficientes das restrições.

Applegate *et al.* (1994) apontam ainda a paralelização do método *branch-and-cut* como uma forma utilizada para aumentar sua velocidade.

O conteúdo dessa seção é abordado com maior detalhamento em um artigo de Jünger, Reinelt e Thienel (1995). O método *branch-and-cut* é apresentado de forma geral, aplicado a diversos problemas difíceis. É interessante notar que as soluções técnicas apresentadas são quase todas originárias de implementações do PCV.

2.11 A “Olimpíada” do PCV

O quadro abaixo mostra os maiores PCVs resolvidos em cada época. Corresponde a uma série de recordes quebrados. Como os problemas resolvidos eram em sua maioria artificiais, no sentido de que não representavam um problema surgido de uma necessidade real, Applegate, Bixby, Chvátal e Cook (1994) deram o nome de Olimpíada do PCV a essa sucessiva quebra de recordes.

ano	autores	tamanho
1954	Dantzig, Fulkerson e Johnson	49
1970	Held e Karp	60
1974	Helbig, Hansen e Krarup	80
1975	Camerini, Fratta e Maffioli	100
1977	Grötschel	120
1980	Crowder e Padberg	318
1988	Grötschel e Holland	666
1987	Padberg e Rinaldi	2392
1994	Applegate, Bixby, Chvátal e Cook	4461

Tabela 2.2: A Olimpíada do PCV

Os três primeiros resultados da década de 70 foram obtidos por meio de algoritmos de *branch-and-bound* utilizando-se relaxação lagrangiana. Essa abordagem está descrita em Balas e Toth (1985).

O resultado de Grötschel do ano de 1977 retomou a técnica utilizada em 1954 e ainda obteve restrições violadas “visualmente”, isto é, sem auxílio de um algoritmo de separação. Também não foi necessária a utilização de *branching* para esse problema. A diferença básica entre a abordagem de Grötschel e a primeira abordagem consistia em que novas classes de facetes eram conhecidas e havia uma maior compreensão da estrutura poliédrica do PCV.

O resultado de Crowder e Padberg de 1980 utilizou uma variante do método de *branch-and-cut* chamada “primal”, que não foi abordada nesse trabalho por ter caído em desuso. Esse método e a sua comparação com o método que descrevemos está em Grötschel e Padberg (1985).

Para os três últimos recordes, foi utilizado o algoritmo *branch-and-cut* tal como descrevemos. Apesar do artigo de Grötschel e Holland ter sido publicado posteriormente ao de Padberg e Rinaldi, a época em que os algoritmos foram desenvolvidos coincide. A partir da experiência adquirida na resolução do penúltimo recorde, Padberg e Rinaldi escreveram um detalhado artigo (1991), descrevendo aspectos técnicos do método *branch-and-cut*, no qual boa parte da seção anterior está baseada. Grötschel e Holland apresentam seu algoritmo em um artigo de 1991.

O algoritmo desenvolvido por Applegate *et al.* que deu origem ao atual recorde contém certamente inovações técnicas em relação ao anterior. Infelizmente o que os autores publicaram até o momento traz muito pouca informação. Em relatório técnico preliminar, porém, eles mencionam o uso de paralelismo (diversas estações UNIX usadas concomitantemente) e de uma estratégia *branching*, em que vários *branchings* diferentes são tentados, e, em seguida, aquele que proporciona maior aumento da função objetivo é escolhido. Os autores observam que essa técnica consome muito tempo — uma vez que um problema de programação linear tem que ser resolvido para cada *branching* experimentado —, mas defendem, mesmo assim, a sua utilização. O relatório apresenta ainda uma nova heurística de obtenção de restrições pente violadas, baseada em árvores $P-Q$.

Esses autores obtiveram a solução ótima de diversos problemas — além da do problema de 4461 cidades — que não tinham sido obtidas anteriormente, incluindo a solução ótima de um problema de 7397 cidades. Como a disposição das cidades nesse problema o tornava mais fácil do que se poderia esperar, considerando-se um número tão grande de cidades, os autores preferiram apontar para o problema de 4461 cidades como o novo recorde obtido.

2.12 Conclusão

Nessa seção, vimos como se desenvolveu o método *branch-and-cut* e, paralelamente, apresentamos a história da resolução de grandes instâncias do PCV.

O método *branch-and-cut* constitui hoje em dia a melhor alternativa para a resolução de diversos problemas difíceis de otimização combinatória e é o método que decidimos utilizar para a resolução do nosso problema real apresentado no Capítulo 1. Decidimos apresentar o método de uma forma

histórica, para proporcionar uma melhor compreensão da necessidade de suas características.

Apresentando juntamente a história do PCV, exibimos a trajetória de investigação desse problema que além de ter dado origem ao método *branch-and-cut* e de ter estimulado os maiores aprimoramentos técnicos desse método, está bastante próximo do PRV.

Capítulo 3

O Problema de Roteamento de Veículos

Neste capítulo vamos apresentar a formulação do PRV que utilizaremos em nossa abordagem e o modo como essa formulação foi utilizada por outros autores em algoritmos *branch-and-cut*. Outras formulações serão exibidas para possibilitar comparações.

3.1 Formulação do PRV

Apresentaremos uma formulação do PRV, devida a Laporte et al. (1985). Esta formulação utiliza programação inteira e se aproxima bastante da formulação do PCV apresentada no capítulo anterior. A notação utilizada foi definida no Capítulo 1. No entanto, tendo em vista a aplicação do PRV ao problema da Petrobrás, estaremos falando em “base”, “embarcações” e “plataformas”, ao invés de usarmos outros termos mais frequentes na literatura, como “depósito”, “veículos” e “clientes”. Além disso, como cada embarcação deverá realizar várias viagens diferentes, estaremos falando em número de viagens, ao invés de número de veículos.

Seja n o número de plataformas e consideremos o grafo $G = (V, E)$, completo, com $V = \{v_0, \dots, v_n\}$, $|E| = n(n+1)/2$, onde o vértice v_0 corresponde à base, e os vértices $v_1, \dots, v_n$ correspondem às plataformas. Lembramos que tratamos o problema no caso simétrico, e, portanto, a direção em que a aresta é percorrida não é importante.

Definimos a variável x , de forma semelhante ao PCV. No entanto, como podemos ter viagens em que apenas uma plataforma é visitada, algumas arestas podem estar sendo usadas duas vezes. Por isso, as coordenadas de x podem assumir, além de 0 e 1, também o valor 2. Note que esse caso só ocorre quando a coordenada corresponde a uma aresta incidente à base. Assim, definimos

$$x_{vw} = \begin{cases} 1, & \text{quando } (v, w) \text{ é usada uma única vez em alguma viagem;} \\ 2, & \text{quando } (v, w) \text{ é percorrida numa viagem de ida e volta;} \\ 0, & \text{quando nenhuma viagem executa o trajeto entre } v \text{ e } w. \end{cases}$$

Cada plataforma deve ser visitada exatamente uma vez. Assim devemos ter:

$$x(\delta(v)) = 2, \quad (3.1)$$

para toda plataforma v (i.e. $v \neq v_0$).

Para a base, no entanto, lembrando que t representa a variável correspondente ao número total de viagens, devemos ter:

$$x(\delta(v_0)) = 2t. \quad (3.2)$$

Assim como no caso do PCV, as restrições (3.1) e (3.2) não impedem a formação de circuitos contendo apenas plataformas, e, consequentemente, permitem que x não represente um roteamento válido. Além disso, alguns circuitos são inválidos, pois a soma das demandas das plataformas é maior do que a capacidade da embarcação. Tais circuitos também são eliminados pelas restrições abaixo:

$$x(E(W)) \leq |W| - r(W), \quad (3.3)$$

para todo $W \subseteq V \setminus \{v_0\}$, onde $r(W)$ é o número mínimo de viagens necessárias para fazer as transferências das plataformas em W . Note-se que no caso do PCV, tínhamos que $r(W)$ valia 1, para todo W .

Nossa função objetivo, como foi discutido no Capítulo 1, pode ser o número total de viagens necessárias para realizar todas as trocas de funcionários; a distância total percorrida; ou uma combinação linear desses valores. No caso geral essa função pode ser escrita como

$$\lambda t + \gamma \sum_{e \in E} d_e x_e \quad (3.4)$$

onde λ e γ são números reais representando respectivamente o custo de cada nova embarcação a ser adquirida e o custo do combustível por quilômetro, contado num período de tempo correspondente ao da vida útil de uma embarcação

Chamamos de *roteamento* a união das arestas pertencentes às viagens de uma instância viável do PRV, isto é, se (x, t) é uma solução do problema, então as arestas do grafo G_x (como definido na seção 0.2) formam um roteamento.

Apresentamos a seguinte formulação inteira do PRV que é a que escolhemos para a aplicação do método *branch-and-cut*. Observe que os valores C , c , T , ν e ρ aparecem implícitos no cálculo de $r(W)$.

Teorema 3.1 *Um vetor x representa uma solução ótima do PRV definido por $n, C, c, T, \nu, \rho, d, \lambda$ e γ se e somente se é solução ótima de*

$$\begin{aligned} & \text{minimize } \lambda t + \gamma \sum_{e \in E} d_e x_e \\ & \text{sujeito a } x(\delta(v_0)) = 2t, \\ & \quad x(\delta(v)) = 2, \quad \text{para todo } v \in V \setminus \{v_0\}, \\ & \quad x(E(W)) \leq |W| - r(W), \quad \text{para todo } W \subseteq V \setminus \{v_0\}, \\ & \quad x_{v_0 v} \in \{0, 1, 2\}, \quad \text{para todo } v \in V \setminus \{v_0\}, \\ & \quad x_{v_i v_j} \in \{0, 1\}, \quad \text{para } i, j = 1, \dots, n, i < j, \\ & \quad t \in \mathbb{N}. \end{aligned} \quad (3.5)$$

Prova: Vamos mostrar que um vetor x representa um roteamento se e somente se satisfaz as restrições da formulação.

Acima, ao apresentarmos as restrições, já mostramos que, se x é um roteamento, então elas deverão estar satisfeitas.

Vamos mostrar agora que se x satisfaz as restrições, então as arestas de G_x formam a união de conjuntos disjuntos de arestas correspondendo a viagens viáveis (i.e. viagens que obedecem às restrições do problema). Se, para alguma aresta $e = (v_0, v)$, temos que $x_e = 2$, então é claro que e é a única aresta que incide em v e que a viagem (v_0, v, v_0) é uma viagem viável. As restrições do tipo 3.1 implicam que as demais arestas de G_x formam um conjunto de circuitos disjuntos nos vértices (com exceção de v_0).

Consideremos um conjunto Y de vértices de um desses circuitos. Observamos que o vértice v_0 está em Y , caso contrário, como $x(E(Y)) = |Y|$, teríamos que uma restrição do tipo 3.3 seria violada. Logo, cada circuito corresponde a uma viagem partindo de v_0 e a ele retornando. E, como a restrição do tipo 3.3 está satisfeita por $Y \setminus \{v_0\}$, a viagem tem que ser viável. Portanto, x representa um roteamento.

□

O poliedro do PRV associado a essa formulação é definido de forma semelhante ao poliedro do PCV (veja Capítulo 2). Para um determinado PRV, definimos então o respectivo poliedro a partir do conjunto dos vetores x que representam roteamentos possíveis, que é determinado em função dos valores n , C , c , T , ν e ρ :

$$PRV(n, C, c, T, \nu, \rho) := \text{conv}\{x \in \{0, 1, 2\}^E \mid x \text{ representa um roteamento para o PRV definido por } n, C, c, T, \nu \text{ e } \rho\}.$$

Convém aqui lembrar o que significam as variáveis do problema:

C : capacidade de cada veículo;

c_i : demanda dos clientes;

T : limite de tempo para cada viagem;

ν : velocidade de cada veículo;

ρ : taxa de descarga (quantidade de mercadoria descarregada por unidade de tempo).

3.2 Estudo do poliedro do PRV

Os resultados até hoje obtidos para poliedros associados ao PRV são bem mais fracos do que os existentes para o PCV. Trata-se de estudos iniciados mais recentemente. Além disso, como há inúmeras versões do PRV, um resultado que vale para um poliedro correspondendo a uma das versões pode não valer para um poliedro de uma outra versão.

A primeira questão que surge é a de sabermos se as restrições 3.3 definem faceta do poliedro mencionado. Cornuéjols e Harche (1993) apresentaram condições em que isso ocorre para uma variante do PRV em que o número de viagens é fixo.

Para o poliedro que definimos, encontramos resultados semelhantes em Araque (1989) e Campos, Corberan e Mota (1991). Em ambos os casos, os clientes do PRV considerado apresentam demandas iguais, o que torna o estudo mais simples, porém restrito demais para que possa ser usado em nossa aplicação.

3.3 Utilização de método de plano de corte

A formulação aqui apresentada será utilizada em um método de plano de corte, assim como fizemos com o PCV. As restrições de integralidade são relaxadas para $0 \leq x_{v_0v} \leq 2$, $0 \leq x_{v_i v_j} \leq 1$ e $t \geq 0$. O programa linear é formado inicialmente pelas restrições do tipo 3.1 e pela restrição 3.2. As restrições do tipo (3.3) são introduzidas por separação, ou seja, dado um ponto $(\bar{x}, \bar{t})$, resposta do simplex, buscamos um conjunto W tal que a restrição $\bar{x}(E(W)) \leq |W| - r(W)$ esteja violada. A falta de conhecimento a respeito de outros tipos de restrições válidas nos leva a restringir a separação a cortes desse tipo.

Não foi possível em nosso caso utilizar o resultado de Cornuéjols e Harche. Em primeiro lugar, porque não estamos trabalhando com um número fixo de viagens. Além disso, mesmo que esse resultado pudesse ser adaptado para o nosso poliedro, verificar se uma restrição do tipo 3.3 para um conjunto W define faceta envolveria necessariamente o cálculo de $r(W)$, o que já é um problema difícil no caso de haver apenas restrições de capacidade (trata-se de um problema de “bin-packing” — veja Garey e Johnson (1979), que é NP-completo), e torna-se ainda mais intratável quando restrições de tempo também estão presentes.

Assim sendo, em vez de calcular $r(W)$, buscamos um limitante inferior $\bar{r}$ para esse valor. É claro que, nesse caso, a restrição $x(E(W)) \leq |W| - \bar{r}$ é válida. Notemos, no entanto, que estamos dessa forma abrindo mão não apenas de trabalhar com facetas, mas também de utilizarmos restrições de suporte, pois se $\bar{r} < r(W)$ então o hiperplano $x(E(W)) \leq |W| - \bar{r}$ pode nem mesmo “tocar” o poliedro.

3.4 Heurísticas de separação

Dada uma solução $\bar{x}$, devolvida pelo simplex como solução ótima do problema de programação linear para um certo nó da árvore de *branching*, o problema de separação que temos que resolver é o de buscar um conjunto $W \subseteq V \setminus \{v_0\}$ e um limitante inferior $\bar{r}$ para $r(W)$, tais que a restrição $\bar{x}(E(W)) \leq |W| - \bar{r}$ esteja violada.

Os autores Laporte, Nobert e Desrochers (1985) utilizaram duas heurísticas para resolver esse problema.

Dois tipos de limitantes inferiores de $r(W)$ serão úteis. O primeiro diz respeito às demandas:

$$r_C(W) = \lceil c(W)/C \rceil. \quad (3.6)$$

É claro que $r_C(W) \leq r(W)$. O outro limitante está relacionado com o limite de tempo de cada viagem. É fácil observar que se d é um limitante inferior da distância de um roteamento que visita W , então

$$r_T(W, d) = \lceil (d/\nu + c(W)/\rho)/T \rceil \quad (3.7)$$

é também um limitante inferior de $r(W)$. No caso em que d corresponde à distância mínima de um roteamento denotamos $r_T(W, d)$ simplesmente por $r_T(W)$.

A primeira heurística busca violações de capacidade, procurando conjuntos W para os quais temos que $\bar{x}(E(W)) > |W| - r_C(W)$.

Para tanto, consideramos o grafo $G''_{\bar{x}}$ formado pelas arestas que não estão ligadas à base e para as quais $\bar{x}_e > 0$. Os primeiros conjuntos a serem testados são aqueles que correspondem aos vértices das componentes de $G''_{\bar{x}}$. Se, para o conjunto de vértices W de uma componente de $G''_{\bar{x}}$, a restrição $\bar{x}(E(W)) \leq |W| - \lceil c(W)/C \rceil$ estiver satisfeita, são buscados então subconjuntos de W que violem a respectiva restrição através do procedimento abaixo. Aqui $z(W)$ denota a diferença entre os dois lados da restrição, isto é,

$$z(W) = |W| - r_C(W) - \bar{x}(E(W)).$$

A idéia é, dado W , obter $Y \subset W$ tal que $z(Y)$ é negativo.

Procedimento 3.2 Entrada: Conjunto de vértices W tal que $E(W)$ é uma componente de G'_x e $x(E(W)) \leq |W| - r_C(W)$.

faça $Y := W$;
 enquanto $|Y| \geq 2$
 seja v tal que $r_C(Y - v) = r_C(Y)$
 e que minimiza $z(Y - v)$;
 se tal v não existe ou se $z(Y - v) \geq z(Y)$
 pare;
 senão se $z(Y - v) < 0$
 guarde $Y - v$; /* restrição violada obtida */
 pare;
 faça $Y := Y - v$;

Note-se que se trata de uma busca gulosa e que não há garantia de que um subconjunto Y de W com $z(Y) < 0$ seja obtido sempre que tal subconjunto existir. No Capítulo 4, apresentamos um exemplo em que essa heurística falha.

A segunda heurística utilizada busca violações da restrição do tempo e baseia-se na seguinte proposição.

Proposição 3.3 Se $E(W)$ é uma componente conexa de G'_x , então

$$\sum_{e \in E(W+v_0)} d_e \bar{x}_e \leq d(W).$$

Prova: Seja x^* o vetor $\{0, 1, 2\}^E$ que representa um conjunto de viagens viáveis visitando os vértices de W e seja $\hat{x}$ o vetor dado por:

$$\tilde{x}_e = \begin{cases} x_e^*, & \text{se } e \in E(W + v_0), \\ \bar{x}_e, & \text{caso contrário.} \end{cases}$$

Observemos que, em G'_x , $\bar{x}(\delta(W)) = 0$, já que $E(W)$ é uma componente conexa de G'_x . Então, $\hat{x}$ é uma solução viável para o problema de programação linear do qual $\bar{x}$ é solução ótima. Da otimalidade de $\bar{x}$, temos que

$$\sum_{e \in E} d_e \bar{x}_e \leq \sum_{e \in E} d_e \tilde{x}_e.$$

Assim,

$$\sum_{e \in E(W+v_0)} d_e \bar{x}_e + \sum_{e \in E(V \setminus W)} d_e \bar{x}_e \leq \sum_{e \in E(W+v_0)} d_e x_e^* + \sum_{e \in E(V \setminus W)} d_e \bar{x}_e,$$

e, portanto,

$$\sum_{e \in E(W+v_0)} d_e \bar{x}_e \leq \sum_{e \in E(W+v_0)} d_e x_e^*,$$

como queríamos. □

A segunda heurística busca componentes $E(W)$ para as quais esteja valendo que $\bar{x}(E(W)) > |W| - r_T(W, \sum_{e \in E(W+v_0)} d_e \bar{x}_e)$.

Essa heurística apresenta o seguinte inconveniente: dependendo do nó da árvore de *branching* em que é aplicada, as restrições violadas por ela obtidas podem não ser globalmente válidas, isto é, podem não representar restrições válidas para o problema. Isso se deve ao fato de que, tomando arestas que no nó correspondente a $\bar{x}$ estão fixadas em 0 ou deixando de tomar arestas fixadas em 1, é possível chegar a um roteamento em que a distância total das viagens visitando os vértices de W seja menor do que $\sum_{e \in E(W+v_0)} d_e \bar{x}_e$. Dessa forma, ao executarmos um próximo nó, se este não for filho do primeiro, devemos eliminar essas restrições do conjunto de restrições que definem o programa linear dado como entrada para o simplex. Isso implica um esforço computacional maior quando ocorre um *branching*.

3.5 Outras Formulações

Apresentaremos nessa seção formulações alternativas para o PRV.

3.5.1 Formulação por viagens

Nessa formulação, devida a Fisher e Jaikumar (1978), as diferentes viagens compondo a solução do PRV são numeradas de 1 a t . São utilizados dois tipos de variáveis, um relacionando pares de cidades e viagem (x_{vuk}) e outro relacionando clientes e viagem (y_{vk}). Os autores não consideraram as restrições de tempo, somente as de capacidades. Note que aqui as arestas têm orientação (i.e. $x_{vuk} \neq x_{ukv}$), diferentemente da primeira Formulação (3.5).

Assim, temos

$$x_{vwk} = \begin{cases} 1, & \text{se o percurso de } v \text{ a } w \text{ é realizado na viagem } k \\ 0, & \text{caso contrário,} \end{cases}$$

$$y_{vk} = \begin{cases} 1, & \text{se o cliente } v \text{ é visitado na viagem } k \\ 0, & \text{caso contrário.} \end{cases}$$

O PRV pode ser formulado então como

$$\begin{aligned} & \text{minimize } \sum_{v,w} d_{vw} \sum_k x_{vwk} \\ & \text{sujeito a} \\ & \sum_k y_{v_0k} = t, \\ & \sum_k y_{vk} = 1, & v = 1, \dots, n, \\ & \sum_v c_v y_{vk} \leq C & k = 1, \dots, t, \\ & \sum_w x_{vwk} = \sum_w x_{wvk} = y_{vk}, & v = 1, \dots, n, & k = 1, \dots, t, \\ & \sum_{v,w \in W} x_{vwk} \leq |W| - 1, & \text{para todo } W \subseteq V \setminus \{v_0\}, & k = 1, \dots, t, \\ & x_{vwk} \in \{0, 1\}, & v, w = 1, \dots, n, & k = 1, \dots, t, \\ & y_{vk} \in \{0, 1\}, & v = 1, \dots, n, & k = 1, \dots, t. \end{aligned} \tag{3.8}$$

Assim como na formulação original apresentada (3.5), essa formulação pode ser utilizada como problema de programação linear relaxando-se as restrições de integralidade das variáveis.

A vantagem dessa formulação reside em que as restrições de capacidade estão explicitadas, de modo que violações de capacidade não necessitam do cálculo do $r(W)$. Sua desvantagem está em que o número de variáveis é muito maior, o que tende a tornar a solução dos problemas de programação linear mais lenta.

Observe que, nessa formulação, o número t de viagens é fixo. Esse inconveniente, no entanto, pode ser contornado. Uma vez que viagens “vazias” são admitidas, podemos atribuir a t um valor alto, que saibamos ser suficiente para a solução. Entretanto, quanto maior for o t que tomarmos, maior será o número de variáveis.

Essa formulação poderia ser generalizada envolvendo restrições de tempo, bastando incluir as restrições

$$(\sum_{i,j} d_{ij} x_{ijk})/\nu + (\sum_i c_i y_{ik})/\rho \leq T,$$

para $k = 1, \dots, t$. Essas restrições determinam que, para cada viagem, o tempo de percurso — $(\sum_{i,j} d_{ij} x_{ijk})/\nu$ — acrescido do tempo de descarga — $(\sum_i c_i y_{ik})/\rho$ — não pode ultrapassar o limite de tempo T .

3.5.2 Formulação por partição de conjuntos

Nessa formulação, apresentada pela primeira vez por Balinski e Quandt (1964), cada conjunto I de pontos correspondendo a uma possível viagem é associada a um vetor a^I , chamado de vetor coluna, e o custo dessas diversas viagens é guardado nas entradas do vetor c . Tendo em vista que cada vetor a^I representa apenas o conjunto de pontos pertencentes a uma viagem e não a ordem em que esses pontos são visitados, o valor de cada um dos custos c_I é dado pela solução de um PCV contendo a base e o conjunto I . Note que se está implicitamente assumindo que, para cada um dos conjuntos I , existe uma rota satisfazendo as restrições do problema.

Temos, assim, os vetores constantes

$$a_v^I = \begin{cases} 1, & \text{se o cliente } v \text{ pertence ao conjunto } I, \\ 0, & \text{caso contrário,} \end{cases}$$

$c_I = \text{valor do PCV aplicado ao conjunto contendo a base e o conjunto } I.$

e os vetor de variáveis

$$y_I = \begin{cases} 1, & \text{se a viagem correspondendo ao } I \text{ está incluída na solução,} \\ 0, & \text{caso contrário.} \end{cases}$$

A formulação é dada por

$$\begin{aligned} & \text{minimize } \sum_I c_I y_I \\ & \text{sujeito a } \sum_I a_v^I y_I = 1 \quad \text{para todo } v \in V \setminus \{v_0\} \\ & \quad y_I \in \{0, 1\}, \quad \text{para todo } I. \end{aligned} \tag{3.9}$$

O inconveniente dessa formulação está em que o número de conjuntos correspondendo a viagens viáveis é em geral muito grande. Para a utilização dessa formulação numa relaxação via programação linear, é necessário a utilizar técnicas de geração de coluna, em que são construídos programas lineares com uma parte dos vetores colunas e onde novos vetores colunas são inseridos, de acordo com a necessidade. Para a obtenção de soluções inteiras, há necessidade ainda de se recorrer a técnicas de enumeração tais como as de *branch-and-bound*.

Essa foi a abordagem adotada no trabalho de Agarwal, Malthur e Salkin (1989). Esses autores apontam para um atrativo dessa formulação que é o de admitir variados tipos de restrições para o PRV, como as de janela de tempo (veja a definição do PRV no Capítulo 1), restrições essas que não parecem poder ser tratadas facilmente por outras formulações, como aquela que adotamos.

3.5.3 Formulação por Partição em Caminhos

A diferença básica dessa formulação para a (3.5) está em considerar apenas as arestas que ligam os clientes, eliminando da formulação as arestas que ligam clientes ao depósito. As soluções obtidas para o problema são então caminhos, que, ligados pelos extremos à base, dão origem aos circuitos correspondentes às viagens. Dessa forma, as arestas de uma solução do problema correspondem a caminhos e os conjuntos dos vértices dos caminhos formam uma partição do conjunto dos vértices.

Por causa dessa diferença, a função objetivo dessa formulação não pode ser a mesma. Utiliza-se, então, a maximização das *economias* usadas na heurística de Clark e Wright (1964), que descrevemos na Seção 1.3.1. A economia correspondendo a uma aresta (v, w) é dada por $d_{v_0v} + d_{v_0w} - d_{vw}$.

Os demais itens são definidos como na Formulação 3.5.

$$\begin{aligned}
 &\text{maximize } \sum_{v,w \in V \setminus \{v_0\}} (d_{v_0v} + d_{v_0w} - d_{vw}) x_{vw} \\
 &\text{sujeito a} \\
 &\quad x(\delta(v)) \leq 2, \quad \text{para todo } v \in V \setminus \{v_0\} \\
 &\quad x(E(W)) \leq |W| - r(W), \quad \text{para todo } W \subseteq V \setminus \{v_0\} \\
 &\quad x_{v_i v_j} \in \{0, 1\}, \quad \text{para } i, j = 1, \dots, n, i < j
 \end{aligned} \tag{3.10}$$

A formulação por partição em caminhos foi introduzida por Araque et al. (1994). Com base em estudos de propriedades do poliedro induzido por essa formulação, os autores obtiveram a solução de PRVs com até 60 clientes. Esses PRVs, no entanto, não apresentam restrições de tempo e os clientes têm todos demandas de mesmo valor, o que, em particular, torna mais fácil o cálculo de $r(W)$.

3.6 Conclusão

Neste capítulo estudamos o PRV sob um ponto de vista poliédrico. Apresentamos a abordagem que utilizamos na nossa implementação de um algoritmo *branch-and-cut* para o problema. Mostramos também procedimentos heurísticos para separar inequações de eliminação de circuitos inviáveis (por violarem restrições de capacidade ou de tempo).

Além disso, outras formulações alternativas foram apresentadas. Discutimos vantagens e desvantagens de sua utilização.

Capítulo 4

Resultados obtidos

Neste capítulo descrevemos os resultados obtidos com a utilização do método *branch-and-cut* ao PRV, com dados da aplicação da Petrobrás que investigamos. Descrevemos cada passo da implementação do algoritmo, ilustrando o progresso resultante de cada aprimoramento. Apresentamos sugestões para o leitor interessado em implementar um algoritmo *branch-and-cut* e mostramos os resultados finais obtidos com o nosso algoritmo.

A maioria das definições, teoremas e procedimentos utilizados nesse capítulo estão expostos nos capítulos anteriores.

4.1 Otimizando distância

Decidimos adotar como função objetivo a distância total, sem nos ocuparmos de otimizar o número total de viagens. É mais fácil dessa forma verificar se uma alteração no algoritmo está de fato aumentando a sua eficiência ou não. Além disso, verificamos que o programa comporta-se melhor otimizando distância do que o número de veículos. Maiores comentários a respeito estão na Seção 1.2.2.

A menos que indiquemos o contrário, estaremos sempre usando a distância total como função objetivo.

4.2 Estrutura de branch-and-bound

O algoritmo foi implementado em diversas partes. Cada uma das partes, a partir da segunda, representa um aprimoramento do algoritmo. Os testes realizados após a implantação de cada módulo estão na Tabela 4.1 e 4.2 (veja explicações a respeito na seção 4.6).

A primeira parte do algoritmo implementada foi a estrutura de *branch-and-bound*, isto é, o algoritmo *branch-and-cut* sem a separação.

Foi implementada também uma heurística que é executada após o processamento de cada nó (veja descrição do método *branch-and-cut* na Seção 2.9). A heurística inicia o processamento com um grafo H correspondendo ao grafo G do problema, mas sem conter arestas. Em seguida, tenta inserir arestas em H seguindo a ordem decrescente do valor das coordenadas em $\bar{x}$, que é a solução ótima da atual relaxação. Uma aresta não pode ser inserida, se um de seus extremos já for extremo de duas arestas inseridas anteriormente. Além disso, uma aresta não pode ser inserida, se o novo caminho obtido com a sua inserção não formar uma viagem viável, ao ligarmos os extremos desse caminho à base.

Observemos que se $\bar{x}$ é um vetor inteiro representando um roteamento, então a heurística devolve esse roteamento como solução. Temos então a garantia de que uma solução ótima é sempre obtida pelo algoritmo.

Os testes realizados após essa implementação (veja a coluna TESTE1 da Tabela 4.1) mostram que esse algoritmo é muito ineficiente e revela-se inviável para resolver instâncias com mais do que 10 vértices (a partir de 30 vértices o programa consome toda a memória de máquina que usamos para testes).

É interessante, no entanto, que essa estrutura básica do programa, em especial a chamada ao simplex, a rotina de *branching* e criação de nós, as rotinas para o tratamento do limitante inferior e do superior estejam bem implementadas e depuradas antes de começarmos a desenvolver os procedimentos de separação.

Os testes estão apresentados na entrada TESTE1 das Tabelas 4.1 e 4.2.

Lembramos que todos os testes foram executados em uma SUN SPARC 1000, com 196 MBytes de memória.

4.3 Primeira separação

Em seguida, implementamos a primeira parte da separação sugerida por Laporte *et al.* (1985) (veja Seção 3.4). Montamos o grafo $G''_{\bar{x}}$ composto por arestas e não ligadas à base com $\bar{x}_e > 0$. Em seguida, verificamos se alguma das componentes W desse grafo viola $x(E(W)) \leq |W| - r_C(W)$.

Podemos notar que, apesar de não estar implementada a busca de conjuntos menores do que as componentes, tal como no Procedimento 3.2, essa rotina já é capaz de separar todo vetor inteiro x contendo subcircuitos. Observando as soluções devolvidas pelo simplex sem a separação, percebemos que esses circuitos aparecem com muita frequência.

A influência dessa rotina de separação no desempenho do algoritmo pode ser vista na coluna TESTE2 da Tabela 4.1. Fica claro na Tabela 4.1 a evolução dos resultados, já que conseguimos resolver em tempo razoável instâncias de até 30 vértices. Há também uma melhora sensível de qualidade do limitante inferior e do limitante superior com relação aos resultados da coluna TESTE1.

4.4 Buscando violações em subconjuntos

O próximo passo consistiria em implementar o Procedimento 3.2 sugerido pelos autores, isto é, dado o conjunto de vértices W de uma componente de $G''_{\bar{x}}$, buscar conjuntos $Y \subseteq W$ tais que $z(Y) < 0$. Lembremos que $z(Y) = |Y| - r_C(Y) - x(E(Y))$.

No entanto, observando uma solução devolvida pelo simplex para a qual o algoritmo de separação não obteve nenhum corte, verificamos que esse procedimento não iria ajudar a obter novos cortes.

A solução está representada na Figura 4.1. Os números indicam as demandas dos vértices $v_1, \dots, v_8$. O valor de C é 250.

Observemos o resultado do Procedimento 3.2 de Laporte *et al.* O conjunto Y começa contendo os vértices e temos $c(Y) = 500$, $r_C(Y) = 2$ e $z(Y) = 0$. Observemos que os vértices v que minimizam $z(Y - v)$ são aqueles que minimizam $x([v : Y - v])$. Assim, os três primeiros vértices a serem retirados de Y são v_1 , v_2 e v_4 . Em qualquer ordem que eles forem retirados, temos que $z(Y)$ permanece igual a zero. Em seguida, é retirado v_5 e $z(Y)$ ainda vale zero. Para os vértices restantes observemos que a retirada de qualquer um

Figura 4.1: Ponto que não é separado pela heurística de Laporte *et al.*

deles resulta em $r_C(Y) = 1$ o que faz o procedimento parar.

Dessa forma, nenhuma restrição violada seria obtida. Mas, se tomarmos o conjunto $\bar{Y} = \{v_1, v_2, v_5, v_8\}$ percebemos que a restrição $x(E(\bar{Y})) \leq |\bar{Y}| - r_C(\bar{Y})$ não está satisfeita ($r_C(\bar{Y}) = 2$, $x(E(\bar{Y})) = 7/3$).

Esse exemplo levou-nos a escrever um procedimento diferente para buscar conjuntos Y tais que $z(Y) < 0$. Esse procedimento dispensa o pré-processamento das componentes de $G''_{\bar{x}}$, bastando ter como entrada o grafo $G''_{\bar{x}}$.

Procedimento 4.1 Entrada: $G''_{\bar{x}}$

tome H como o grafo $G''_{\bar{x}}$ sem arestas;
 enquanto houver arestas unindo componentes distintas de H
 tome e tal que e conecta componentes distintas de H e
 minimiza $z(Y_e)$;
 insira e em H ;
 se $z(Y) < 0$
 guarde Y ;

obs: Y_e denota o conjunto de vértices da componente de H que é obtida com a inserção de e .

Obtivemos então um aprimoramento muito acentuado da separação. Seus resultados estão na Tabela 4.1 na entrada TESTE3. Comparando esses resultados com o da coluna anterior, notamos que muito menos cortes tiveram que ser gerados para a obtenção de melhores limitantes inferiores, o que indica que os cortes gerados são de melhor qualidade.

4.5 Verificando violação de restrição de tempo

Até agora, a rotina de separação só se ocupava em obter conjuntos Y violando as restrições de capacidade. Acontece com frequência, no entanto, que o número de viagens necessárias para atender um conjunto W não é determinado pela demanda de suas plataformas, mas pelo tempo necessário para atendê-las, ou seja, $r_T(W) > r_C(W)$.

A segunda heurística de Laporte *et al.*, descrita na Seção 3.4, busca restrições de tempo violadas. Nós consideramos essa heurística insatisfatória, já que apenas componentes inteiras são verificadas. Assim, devido à importância das restrições de tempo para o nosso problema, decidimos escrever uma outra heurística. Estaremos utilizando o fato de que o PRV com que estamos trabalhando é euclidiano.

Definição 4.2 *Dado $Y \subseteq V$ dizemos que um subgrafo F de $E(Y)$ é uma k -floresta de Y se F não contém circuitos e é composto por k componentes conexas.*

Proposição 4.3 *Seja $Y \subseteq V \setminus \{v_0\}$ e k um número inteiro. Tomemos uma k -floresta F de Y , com $\sum_{e \in F} d_e$ mínimo, e um conjunto F' composto pelas k arestas de menor distância de $E([v_0 : F])$. Então,*

$$\bar{d}(Y) = \sum_{e \in F} d_e + 2 \sum_{e \in F'} d_e \quad (4.1)$$

é um limitante inferior para a distância total de um conjunto de k viagens atendendo Y .

Prova: Como o PRV é euclidiano, existe um roteamento atendendo aos clientes de Y que não contém clientes fora de Y . Assim, como esse roteamento é composto de k viagens, o conjunto de suas arestas corresponde à união de uma k -floresta de Y com um conjunto de arestas ligando Y ao depósito. O percurso entre o depósito e os clientes é realizado $2k$ vezes. Como é possível que tenhamos viagens com apenas um cliente, podemos usar até duas vezes cada aresta.

□

Motivados por esse resultado, desenvolvemos o seguinte procedimento. Tomamos o valor de k na saída do procedimento como um limitante inferior de $r(T)$. Observe que, se k chega a atingir o valor de $r(T)$, então, como nos indica a Proposição 4.3, o procedimento pára. Isso então garante que k é sempre menor ou igual a $r(T)$ e que o procedimento sempre pára.

Procedimento 4.4 Entrada: $Y \subseteq V$

faça $k:=0$;

repita

faça $k:=k+1$;

obtenha F e F' para Y e k tais como especificados
na Proposição 4.3;

calcule $\bar{d}(Y)$ com base em F e F' ;

até que $k \geq r_T(Y, \bar{d}(Y))$

O atrativo desse procedimento está em que os conjuntos F e F' podem ser obtidos rapidamente. Para tanto, criamos, antes do início da otimização, uma lista com as arestas ligando pares de clientes, ordenadas de acordo com a distância. Dessa forma podemos obter o conjunto F com complexidade de tempo de pior caso de $O(|E|)$, utilizando o algoritmo de Kruskal para árvore geradora de peso mínimo (veja, por exemplo, Nemhauser e Wolsey (1988)). Para F' , criamos uma lista semelhante com as arestas ligadas ao depósito e obtemos facilmente o conjunto com a mesma complexidade de tempo. Observe que, uma vez obtidos F e F' para $k = 1$, é fácil atualizar esses conjuntos em tempo constante para os próximos valores de k .

Utilizamos esse procedimento para todo Y gerado no *loop* do Procedimento 4.1.

No caso de termos um conjunto Y tal que $r_C(Y) = 1$ e tal que $r = k = 1$ na saída do Procedimento 4.4, julgamos interessante resolver o PCV para $Y + v_0$, obtendo a distância total da solução ótima, que chamaremos de $\tilde{d}(Y)$. Como $\tilde{d}(Y)$ é o valor exato do mínimo de distância de uma viagem que visita o conjunto Y , se tivermos $r_T(W, \tilde{d}(Y)) = 1$, então $r(Y) = 1$, caso contrário, $r(Y) > 1$.

Esse teste melhora bastante o desempenho do algoritmo, pois muitas vezes o limitante fornecido pelo Procedimento 4.4 não é muito bom. A desvantagem desse teste é que o cálculo do PCV pode tomar muito tempo, dependendo do tamanho de Y . Decidimos, assim, aplicá-lo somente para conjuntos Y de tamanho até 6, casos em que o PCV pode ser calculado rapidamente por meio de um procedimento enumerativo. Verificamos, em seguida, que as nossas instâncias utilizadas estabelecem as restrições que não permitem viagens com número de plataformas maior do que 6, de forma que o PCV pôde ser calculado sempre que necessário.

Os resultados obtidos após a separação envolvendo restrições de tempo estão indicados com rótulo TESTE4. Nas instâncias de 40 e 50 vértices nota-se um aumento pequeno do limitante inferior com a introdução das restrições de tempo. Isso se deve ao fato de que na instância F as restrições de tempo. Isso se deve ao fato de que na instância F as restrições de tempo não são muito restritivas. Na tabela 4.2 mostramos os resultados para a instância A, onde o aumento é mais sensível.

4.6 Comparação de diversos estágios do algoritmo

A tabela abaixo aponta os resultados obtidos em diversos estágios do desenvolvimento do programa. Foi utilizada a instância A para esses testes (veja Tabela 1.1 no Capítulo 1). Para resolver a relaxação linear utilizamos o código CPLEX 3.0. As instâncias de tamanho menor do que 50 foram geradas tomando-se as n primeiras plataformas da lista.

Na entrada “resposta” indicamos se uma solução comprovadamente ótima foi obtida (“ótimo”), se o tempo de processamento ultrapassou 10 minutos (“tempo”) ou se o limite de memória de 100 Megabytes foi alcançado (“memória”).

O limite inferior “L” e o limite superior “U” para o valor ótimo da

		TESTE1	TESTE2	TESTE3	TESTE4
n=10	resposta	ótimo	ótimo	ótimo	ótimo
	L	663,685	663,685	663,685	663,685
	U	663,685	663,685	663,685	663,685
	tempo	37"	0"	0"	0"
	nós	9259	2	2	2
	lps	9259	7	7	7
	cortes	0	22	5	5
n=20	resposta	memória	ótimo	ótimo	ótimo
	L	1126,056	1185,270	1185,270	1185,270
	U	1823,225	1185,270	1185,270	1185,270
	tempo	9'37"	19'	5"	7"
	nós	18999	297	33	57
	lps	18999	451	119	131
	cortes	0	1360	211	168
n=30	resposta	memória	ótimo	ótimo	ótimo
	L	1832,354	2051,507	2051,507	2051,507
	U	3857,902	2051,507	2051,507	2051,507
	tempo	5'40"	4'12"	1'08"	14"
	nós	11699	787	182	30
	lps	11699	1605	311	125
	cortes	0	7816	487	891
n=40	resposta	memória	tempo	tempo	tempo
	L	2641,551	3013,068	3020,343	3021,594
	U	4358,264	3287,652	3242,693	3246,335
	tempo	5'42"	10'01"	10'02"	10'01"
	nós	8099	1556	339	264
	lps	8099	1797	582	580
	cortes	0	4080	815	1161
n=50	resposta	memória	tempo	tempo	tempo
	L	3152,163	3628,281	3660,958	3666,135
	U	5782,665	4189,503	4174,297	3963,718
	tempo	3'56"	10'01"	10'08"	10'02"
	nós	5761	882	190	137
	lps	5761	1039	371	319
	cortes	0	1968	843	966

Tabela 4.1: Resultados obtidos em diversas etapas de desenvolvimento do programa para a instância F.

		TESTE1	TESTE2	TESTE3	TESTE4
n=10	resposta	ótimo	ótimo	ótimo	ótimo
	L	663,685	663,685	663,685	663,685
	U	663,685	663,685	663,685	663,685
	tempo	41"	0"	0"	0"
	nós	9259	2	2	2
	lps	9259	7	7	7
	cortes	0	22	5	5
n=20	resposta	tempo	ótimo	ótimo	ótimo
	L	1126,056	1185,270	1185,270	1185,270
	U	1823,225	1185,270	1185,270	1185,270
	tempo	10'01"	7'31"	1'59"	7"
	nós	17245	297	33	57
	lps	17245	451	119	131
	cortes	0	1360	211	168
n=30	resposta	memória	tempo	tempo	tempo
	L	1835,585	2055,470	2064,409	2074,297
	U	3625,337	2322,350	2319,469	2322,392
	tempo	5'56"	10'03"	10'03"	10'05"
	nós	11699	1672	501	520
	lps	11699	1986	855	903
	cortes	0	5153	891	854
n=40	resposta	memória	tempo	tempo	tempo
	L	2646,657	3014,677	3020,892	3097,619
	U	5226,762	4003,792	3968,536	3911,581
	tempo	5'04"	10'01"	10'01"	10'04"
	nós	8099	1272	330	175
	lps	8099	1746	593	484
	cortes	0	5513	925	891
n=50	resposta	memória	tempo	tempo	tempo
	L	3158,370	3644,738	3668,969	3762,384
	U	6758,867	5005,439	5145,428	5185,942
	tempo	3'16"	10'01"	10'03"	11'56"
	nós	4275	536	116	8
	lps	4275	762	261	221
	cortes	0	2008	664	1595

Tabela 4.2: Resultados obtidos em diversas etapas de desenvolvimento do programa para a instância A.

distância é indicado. Observe que quando uma solução comprovadamente ótima é obtida, o valor de “L” coincide com o de “U”.

O tempo de execução está indicado em minutos e segundos. Estabelecemos o limite tempo em apenas 10 minutos, já que o interesse desses testes estava na comparação entre diversos estágios do algoritmo e não na obtenção de respostas finais. Note que quando houve estouro de tempo, o tempo indicado um pouco superior a 10 minutos. Isso ocorre porque o limite de tempo foi verificado apenas após o tratamento de um nó da árvore de *branching*.

São indicados também o número de nós que a árvore de *branching* tomou, o número de problemas de programação linear (“lps”) resolvidos e o número de cortes gerados pelos procedimentos de separação.

A linha da tabela 4.1 que melhor ilustra o aprimoramento do algoritmo é a do tempo para $n=30$. Na tabela 4.2 verificamos esse mesmo fenômeno para a linha $n=20$.

4.7 Considerações práticas

Gostaríamos de apresentar aqui algumas sugestões práticas para o leitor interessado em implementar um algoritmo *branch-and-cut*. Em particular gostaríamos de mostrar as tentativas que se revelaram infrutíferas.

1. Observe as respostas devolvidas pelo simplex. Elas são o melhor indicador de como a separação deve ser aprimorada. Nós supusemos que o Procedimento 4.1, por se tratar de uma busca gulosa, poderia não estar achando conjuntos $Y \subseteq W$ tais que $z(Y) < 0$. Decidimos escrever um procedimento “backtrack” para fazer uma busca mais exaustiva em subconjuntos de W . Esse “backtrack” trouxe muito pouca ou quase nenhuma vantagem para o funcionamento do algoritmo, de forma que resolvemos eliminá-lo da separação na versão final. Se tivéssemos observado melhor as respostas do simplex que não estavam sendo separadas, poderíamos constatar que tais subconjuntos não existiam, e que, dessa forma, uma busca mais aprimorada seria inútil.
2. É bom ter uma noção de quanto tempo o programa gasta em cada função. Por exemplo, de nada adianta desenvolver uma estrutura de dados sofisticada para baixar o tempo médio da separação de 10 para 5 milissegundos, se o tempo médio gasto por uma chamada do simplex

é de 1000 milissegundos. Nós levamos tempo até darmos conta de que resolver exaustivamente PCVs com até 7 vértices (6 plataformas e o depósito) tomava muito pouco tempo em relação ao resto do algoritmo, e que o tempo gasto a mais com esse processamento era insignificante, se comparado com a melhora da separação por ele apresentada.

3. Ao fazer um teste, recomendamos que se use mais do que uma instância. Às vezes, uma alteração faz com que uma instância seja resolvida mais rapidamente, mas torna a resolução de muitas outras mais lenta. Também é interessante testar o algoritmo com diversos limites de tempo de execução. Uma rotina que nos primeiros minutos de execução parece piorar o algoritmo pode mostrar-se eficiente após uma hora. O ideal seria uma bateria padrão de testes.
4. É interessante testar cada inovação no programa, não apenas para saber o que foi mais decisivo no aprimoramento do algoritmo, mas também porque algumas inovações, que esperávamos que viriam a melhorar a eficiência do programa, apenas gastam tempo ou chegam até mesmo a piorá-la.
5. Para poder fazer um histórico, assim como fizemos nesse capítulo, é interessante guardar os programas tais como estavam em cada época, pois, em geral, não é simples reproduzirmos versões anteriores do algoritmo inibindo rotinas posteriormente implementadas.

4.8 Otimalidade do número de viagens

Lembremos que o principal objetivo da aplicação da Petrobrás era o de minimizar o número de viagens e que, como observamos na Seção 1.2.2, uma solução que minimiza distância total pode não minimizar o número de veículos.

No entanto, alterando a função objetivo para minimizar o número de viagens ao invés da distância total, o nosso algoritmo demonstrou, como dissemos, pior desempenho. As soluções obtidas eram piores não apenas em distância total como também em número de viagens. Além disso, para todas as instâncias de teste pequenas para as quais conseguimos obter soluções ótimas com relação às duas funções objetivo, observamos que as soluções que minimizam distância minimizam também o número de viagens.

Diante disso, pareceu-nos razoável procurar soluções tão boas quanto possível em termos de distância, e, em seguida, procurar demonstrar a sua otimalidade em termos de número de viagens.

Para tanto, supondo que uma solução apresente $\bar{r}$ viagens, basta-nos limitar o número de viagens em $\bar{r} - 1$ e executar novamente o algoritmo. Caso o problema se mostre inviável, concluímos então que $\bar{r}$ é de fato o número mínimo de viagens.

Para instâncias pequenas, essa estratégia revelou-se eficiente. Para as instâncias da Petrobrás, entretanto, não foi possível demonstrar a otimalidade das soluções obtidas em termos de número de viagens. As melhores soluções que obtivemos (apresentadas na próxima seção) coincidem em número de viagem com as soluções apresentadas por Brinati *et al.*

4.9 Resultados heurísticos

Mesmo com os aprimoramentos descritos, o algoritmo não se revelou capaz de abordar o PRV com 40 ou mais pontos, ainda que executando o programa por mais de 10 horas. Após muitas horas de execução, observamos que o avanço do algoritmo (medido pelo crescimento do limitante inferior) torna-se muito lento, de modo que, quando o limitante inferior está distante do limitante superior, poucas esperanças temos de que a execução do algoritmo por mais algumas horas possa nos fornecer uma solução comprovadamente ótima. Dessa forma, vimos que não seria possível apresentarmos soluções comprovadamente ótimas para as instâncias da Petrobrás, que apresentam 50 pontos. De fato, executando o algoritmo para os 50 pontos, obtivemos, após um período de tempo de aproximadamente 10 horas, soluções viáveis de qualidade relativamente baixa (os limitantes inferiores obtidos nessa computação aparecem na entrada “L global” do último quadro da Tabela 4.3).

Decidimos então adotar o seguinte procedimento heurístico. Em primeiro lugar, executamos o programa com a instância inteira obtendo uma solução heurística. Em seguida escolhemos uma ordenação $(w_1, \dots, w_k)$ para as plataformas tal que as viagens dessa solução correspondam a plataformas consecutivas. Aplicamos então o método *branch-and-cut* a $(w_1, \dots, w_i)$ para vários valores de i e guardamos o maior valor de i , digamos p , para o qual conseguimos obter uma solução S_p^1 ótima ou próxima do ótimo. Fazemos o mesmo tomando as plataformas do fim da ordenação para o início obtendo q como

o menor índice é obtida uma solução S_q^1 de boa qualidade para $(w_q, \dots, w_k)$. Assumindo que p é maior do que q , obtemos então com facilidade as soluções ótimas S_p^2 para $(w_{p+1}, \dots, w_k)$ e S_q^2 para $(w_1, \dots, w_{q-1})$. Obtemos assim duas soluções heurísticas S_p e S_q originadas, respectivamente, da união de S_p^1 com S_p^2 e de S_q^1 com S_q^2 . Guardamos, então, a melhor dentre as duas soluções.

Aplicando esse método às instâncias A, C e F obtivemos os resultados descritos na Tabela 4.3. Os limitantes inferiores que aparecem no último bloco foram obtidos executando-se o algoritmo para as instâncias com todos os 50 pontos.

As Tabelas 4.4 a 4.9 exibem as melhores soluções obtidas para as instâncias A, C e F e as soluções encontradas por Brinati *et al.* para as mesmas instâncias.

Os possíveis resultados que poderíamos obter para uma determinada instância, comparando-se o que o nosso algoritmo *branch-and-cut* foi capaz de gerar com as soluções heurísticas apresentadas por Brinati *et al.* são as seguintes:

1. Os resultados não são aprimorados. Nesse caso o algoritmo *branch-and-cut* pode eventualmente mostrar que os resultados são ótimos.
2. A distância total é diminuída, mas o número de viagens continua o mesmo.
3. O número de viagens é diminuído, mas o número de embarcações necessárias permanece igual.
4. O número de embarcações é diminuído.

Desde o início consideramos pouco provável que o último dos casos ocorresse. Na verdade, em muitos casos o número de embarcações é igual a $\lceil r_C(V)/7 \rceil$, o que significa que é ótimo (lembremos que cada embarcação pode ser utilizada para 7 viagens, uma para cada dia da semana). Para as três instâncias para as quais fizemos testes, o número de viagens também não foi diminuído.

Ainda que não tenhamos conseguido reduzir o número de embarcações na solução ótima (objetivo maior na aplicação da Petrobrás), para todas as instâncias testadas pudemos reduzir (ainda que em menos de 1 %) a distância total do roteamento. Devemos ainda ressaltar que os limitantes inferiores

que obtivemos fornecem uma garantia de qualidade para as soluções dos problemas que estudamos. Resultados desse tipo não eram conhecidos.

instância	A	C	F
p	30	34	39
S_p^1	2332,844	2594,778	2947,325
L	2310,987	2386,597	ótimo
tempo	7h16'02"	6h15'02"	1h45'12"
S_p^2	2566,255	1393,839	737,236
L	ótimo	1392,139	ótimo
tempo	4h39'09"	5h52'46"	0"
S_p	4899,099	3988,617	3684,561
q	28	33	21
S_q^1	2017,209	2343,406	1192,124
L	ótimo	ótimo	ótimo
tempo	7'16"	2'20"	2"
S_q^2	2911,909	1659,252	2492,437
L	2861,337	1658,758	ótimo
tempo	7h25'57"	16h42'36"	1h19'17"
S_q	4929,118	4002,658	3684,561
melhor	4899,099	3988,617	3684,561
L global	3955,126	3695,533	3666,135
Brinati	4940,020	4018,248	3696,142
melhora (em %)	0,82	0,73	0,31

Tabela 4.3: Resultados heurísticos obtidos (distâncias em km)

viagem	trajeto	demanda	distância (km)	tempo
1	01 16 17 18 20 19	250	264,723	9h53'
2	30 21	220	278,384	9h40'
3	04 06 09	230	226,143	8h43'
4	05 08 07 03	230	230,267	8h48'
5	02 12 11 10 13	240	238,867	9h 9'
6	15 14	215	232,298	8h36'
7	22 25 26	225	280,259	9h48'
8	23 24	205	268,167	9h12'
9	27 34	155	313,731	9h21'
10	28 29 31	225	276,962	9h43'
11	32 33	145	278,426	8h25'
12	38 37	105	374,415	9h50'
13	40 39	90	387,382	9h52'
14	41 36 35	130	361,101	9h57'
15	45 43 44 42	180	312,308	9h44'
16	46 47 48	190	298,644	9h37'
17	49	30	277,013	6h29'
distância total: 4899,099 km				
tempo total: 156h54'				

Tabela 4.4: Solução heurística para a instância A.

viagem	trajeto	demanda	distância (km)	tempo
1	01 30	195	282,300	9h20'
2	06 07	240	225,930	8h52'
3	04 05 08 09 02	230	233,363	8h52'
4	15 16 10 11 12	230	243,520	9h05'
5	13 14 03	250	235,205	9h14'
6	17 18 19 20 21	240	272,990	9h53'
7	22 23	205	268,447	9h12'
8	24 25 26	225	279,979	9h47'
9	27 34	155	313,731	9h21'
10	31 28 29	225	276,962	9h43'
11	33 48	165	291,234	9h02'
12	35 37	115	352,035	9h31'
13	32 36 38	110	378,036	9h59'
14	39 40	90	387,382	9h52'
15	43 41 42 44	170	312,179	9h34'
16	47 45 46	190	309,705	9h51'
17	49	30	277,013	6h29'
distância total: 4940,020 km				
tempo total: 157h47'				

Tabela 4.5: Solução de Brinati *et al.* para a instância A.

viagem	trajeto	demanda	distância (km)	tempo
1	04 05 09 07	250	229,764	9h33'
2	02 03 08 06	250	230,547	9h34'
3	12 11 10 17 13	250	246,024	9h56'
4	01	30	204,384	5h17'
5	14 15 16	250	240,923	9h49'
6	18 19 20 21 22	230	282,093	10h27'
7	23 24 31	245	270,627	10h26'
8	25 27	245	294,510	10h59'
9	47 34 28	245	315,722	11h29'
10	26 29 30	245	280,177	10h39'
11	32 33 46	245	304,176	11h13'
12	35 36 37 48	210	356,801	11h52'
13	38 39 40 41	160	390,523	11h50'
14	45 43 44 42 49	210	342,337	11h32'
distância total: 3988,617 km				
tempo total: 144h43'				

Tabela 4.6: Solução heurística para a instância C.

viagem	trajeto	demanda	distância (km)	tempo
1	06 07	240	225,930	9h18'
2	05 08 09 04	190	228,751	8h32'
3	02 12 10 11 13	240	238,870	9h36'
4	01 14 15	245	232,984	9h33'
5	17 19 20 18 16 03	250	267,869	10h27'
6	21 23	220	276,829	10h 9'
7	24 25 31	225	280,177	10h19'
8	26 27 34 47	240	318,815	11h29'
9	28 29 22	225	278,162	10h16'
10	30	165	265,413	8h58'
11	32 33 48 49	220	340,028	11h38'
12	35 36 37 42	210	361,164	11h58'
13	38 39 40 41	160	390,523	11h50'
14	43 44 45 46	235	312,726	11h15'
distância total: 4018,248 km				
tempo total: 145h25'				

Tabela 4.7: Solução de Brinati *et al.* para a instância C.

viagem	trajeto	demanda	distância (km)	tempo
1	05 06 09	240	228,538	6h50'
2	07 08 04	190	226,143	6h38'
3	01 14 15	245	232,984	6h58'
4	03 16 17 18 20 19	250	265,590	7h51'
5	13 10 11 12 02	240	238,867	7h 6'
6	23 24 22	245	268,447	7h55'
7	21 29 28	240	286,155	8h23'
8	25 27	245	294,510	8h37'
9	32 33 47 48	235	295,500	8h37'
10	30 31	205	265,413	7h44'
11	46 34 35 26	240	345,172	9h59'
12	37 36 38 39 40	245	394,898	11h20'
13	45 43 44 41 42 49	245	342,337	9h55'
distância total: 3684,461 km				
tempo total: 108h00'				

Tabela 4.8: Solução heurística para a instância F.

viagem	trajeto	demanda	distância (km)	tempo
1	06 07	240	225,930	6h46'
2	01 04 03 08 09 05	250	234,421	7h 1'
3	10 11 12 13 02	240	239,637	7h 8'
4	14 15	215	232,298	6h52'
5	16 17 18 19 20	220	264,894	7h45'
6	22 23 24	245	268,447	7h55'
7	21 25 26	240	287,255	8h25'
8	27 33	220	296,939	8h37'
9	48 28 29	230	297,473	8h40'
10	32 30 31	230	266,689	7h50'
11	47 46 34 35	245	344,919	9h59'
12	37 36 38 39 40	245	394,898	11h20'
13	45 43 44 41 42 49	245	342,337	9h55'
distância total: 3696,142 km				
tempo total: 108h18'				

Tabela 4.9: Solução de Brinati *et al.* para a instância F.

Conclusão e comentários finais

Nossos resultados computacionais, ainda que tenham sido superiores aos existentes na literatura, podem certamente ser aprimorados no futuro. Gostaríamos de apontar aqui direções em que esse trabalho pode prosseguir de forma que o algoritmo *branch-and-cut* se torne mais eficiente.

Uma possível direção seria incorporar técnicas — apontadas na Seção 2.10 — utilizadas nos sofisticados algoritmos *branch-and-cut* para o Problema do Caixeiro Viajante que não chegamos a utilizar. Dentre elas acreditamos que a utilização de um *pool* de restrições seria particularmente interessante para o nosso algoritmo.

Além disso, seria interessante investigar outras estratégias de *branching*. Aqui, ao invés de transpor técnicas utilizadas para o PCV, parece-nos mais interessante escrever procedimentos observando o que o PRV tem de específico.

Não resta dúvida, no entanto, de que a principal limitação que os algoritmos *branch-and-cut* para o PRV apresentam hoje em dia está no pouco conhecimento existente da estrutura poliédrica do problema. Uma melhor caracterização de facetas de poliedros correspondentes ao problema e o desenvolvimento de rotinas de separação eficientes são, no nosso entender, o maior desafio para que o método *branch-and-cut* tenha o seu potencial explorado de um modo mais pleno para o PRV, assim como ocorre para o PCV.

Cabe lembrar que as maiores instâncias do PRV resolvidas na literatura têm por volta de 50 a 60 vértices, e em alguns casos restrições de tempo não são consideradas, o que torna o problema mais fácil (compare a tabela 4.1 com a tabela 4.2).

Uma lição clara desse trabalho é que, ainda que muito esforço deva ser colocado para se resolver instâncias práticas do PRV por *branch-and-cut*, o método certamente compete com outros utilizados na literatura (como *branch-and-bound*). Se, por um lado, não é possível hoje achar a solução ótima para

problemas práticos, bons limitantes inferiores e superiores podem ser obtidos em tempo razoável, o que, para fins práticos, já pode ser suficiente.

Bibliografia

Y. Agarwal, K. Mathur, H. Salkin, "A set-partitioning-based exact algorithm for the vehicle routing problem", *Networks* **19** (1989) 731–749.

D. Applegate, R. Bixby, V. Chvátal, W. Cook, "Finding cuts in the TSP, a preliminary report", Technical Report, AT&T, 1994.

J.R. Araque, "Solution of a 48-city vehicle routing problem by branch-and-cut", Working paper, department of Applied Mathematics and Statistics, SUNY (Stony Brook, NY, 1989).

J.R. Araque, G. Kudva, T.L. Morin, J.F. Pekny, "A branch-and-cut algorithm for vehicle routing problems", *Annals of Operations Research* **50** (1994) 37–59.

E. Balas, P. Toth, "Branch and bound methods", in E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, D.B. Shmoys (eds.), *The Traveling Salesman Problem*, Wiley, New York, 1985.

M.L. Balinski, R.E. Quandt, "On integer program for a delivery problem", *Operations Research* **12** (1964) 300–304.

B. Bollobás, "Graph Theory: an introductory course", Graduate Texts in Mathematics 63, Springer Verlag, New York, 1979.

L.D. Bodin, B.L. Golden, A. Assad, M. Ball, "Routing and scheduling of vehicles and crews, the state of art", *Computers and Operations Research* **10** (1983) 69–211.

J.A. Bondy, U. R. Murty, "Graph Theory with Applications", McMillan,

Londres, 1976.

K.H. Borgwardt, "The average number of pivot steps required by the simplex method is polynomial", *Zeitschrift für Operations Research* **26** (1982) 157–177.

M.A. Brinati, R.C. Botter, J.R.R. Roque, M.T. de Gouvea, R.L. de Souza, P.R.B. Carneiro, "Aplicação de heurísticas de roteamento para o dimensionamento de uma frota de transporte marítimo", *Anais do Congresso Latino-Ibero-Americano de Pesquisa Operacional e Engenharia de Sistemas* **21** (1988) 349–363.

P.M. Camerini, L. Fratta, F. Maffioli, "On improving relaxation methods by modified gradient techniques", *Mathematical Programming Study* **3** (1975) 26–34.

V. Campos, A. Corberan, E. Mota, "Polyhedral results for a vehicle routing problem", *European Journal of Operations Research* **52** (1991) 75–85.

N. Christofides, "Vehicle routing", in E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, D.B. Shmoys (eds.), *The Traveling Salesman Problem*, Wiley, New York, 1985.

N. Christofides, A. Mingozzi, P. Toth, "Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations", *Mathematical Programming* **20** (1981) 255–282.

V. Chvátal, "Edmonds polytopes and weakly hamiltonian graphs", *Mathematical Programming* **5** (1973), 29–40.

G. Clark, J. Wright, "Scheduling of vehicles from a central depot to a number of delivery points", *Operations Research* **12** (1964) 568–581.

G. Cornuéjols, F. Harche, "Polyhedral study of the capacited vehicle routing problem", *Mathematical Programming* **60** (1993) 21–52.

"The CPLEX callable library", Cplex Optimization Inc., (1994).

H. Crowder, M.W. Padberg, "Solving large-scale symmetric traveling sales-

man problems to optimality”, *Management Science* **26** (1980) 495–509.

G.B. Dantzig, D.R. Fulkerson, S.M. Johnson, “Solution of a large-scale traveling salesman problem”, *Operations Research* **2** (1954) 393–410.

M.L. Fisher, R. Jaikumar, “A decomposition algorithm for large-scale vehicle routing”, Report 78-11-05, Dept. of Decision Sciences, the Wharton School, University of Pennsylvania, Philadelphia, 1978.

M.L. Fisher, R. Jaikumar, “A generalized assignment Heuristic for Vehicle Routing”, *Networks* **11** (1981) 109–124.

B. Fleischmann, “A new class of cutting planes for the symmetric traveling salesman problem”, *Mathematical Programming* **40** (1988) 225–246.

M.R. Garey, D.S. Johnson, “Computer and Intractability: A guide to the theory of NP-completeness”, Freeman, NY, 1979.

M. Goemans, “Worst-case comparison of valid inequalities for the TSP”, *Mathematical Programming* **69** (1995) 335–349.

B. Golden, A. Assad, L. Levy, F. Gheysens, “The fleet size and mix routing problem”, *Computers and Operations Research*, **11** (1984), 49–65.

R.E. Gomory, “Outline of an algorithm for integer solutions to linear programs”, *Bulletin of American Mathematical Society* **64** (1958), 275–278.

R.E. Gomory, T.C. Hu, “Multi-terminal network flows”, *SIAM Journal of Applied Mathematics* **9** (1961) 551–570.

M. Grötschel, “Polyedrische Charakterisierungen kombinatorischer Optimierungsprobleme”, Hain, Meisenheim am Glan, 1977.

M. Grötschel, M. Jünger, G. Reinelt, “A cutting plane algorithm for the linear ordering problem”, *Operations Research* **32** (1984) 1195–1220.

M. Grötschel, O. Holland, “Solution of large-scale symmetric traveling salesman problems”, *Mathematical Programming* **51** (1991), 141–202.

M. Grötschel, L. Lovász, A. Schrijver, “Geometric Algorithms and Combina-

torial Optimization”, Springer, Heidelberg, 1988.

M. Grötschel, M.W. Padberg, “On the symmetric travelling salesman problem I: Inequalities”, *Mathematical Programming* **16** (1979) 265–280.

M. Grötschel, M.W. Padberg, “Polyhedral theory”, in E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, D.B. Shmoys (eds.), *The Traveling Salesman Problem*, Wiley, New York, 1985.

M. Grötschel, M.W. Pulleyblank, “Clique tree inequalities and the traveling salesman problem”, *Mathematics of Operations Research* **11** (1986) 537–569.

K. Helbig Hansen, J. Krarup, “Improvements of the Held-Karp algorithm for the symmetric travelling-salesman problem”, *Mathematical Programming* **7** (1974) 87–96.

M. Jünger, G. Reinelt, S. Thienel, “Practical problem solving with cutting plane algorithms in combinatorial optimization”, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* **20** (1995) 111–152.

R.M. Karp, E. Lawler, R. Tarjan, “Reducibility among combinatorial problems”, in R.E. Miller, J.W. Thatcher (eds.), “Complexity of Computer Computations”, Plenum Press, 1972.

R.M. Karp, C.H. Papadimitriou, “On linear characterizations of combinatorial optimization problems”, *SIAM Journal on Computing* **11** (1982) 620–632.

L.G. Khachian, “A polynomial algorithm in linear programming”, *Doklady Akademiy Nauk SSSR* **224** (1979), 1093–1096, traduzido em *Soviet Mathematics Doklady* **20** (1979) 191–194.

V. Klee, G.J. Minty, “How good is the simplex algorithm?”, in O. Shima (ed.), *Inequalities III*, Academic Press, 1972, 159–175.

G. Laporte, Y. Nobert, “Comb inequalities for the vehicle routing problem”, *Methods of Operations Research* **51** (1984), 271–276.

G. Laporte, Y. Nobert, "Exact algorithms for the vehicle routing problem", *Annals of Discrete Mathematics* **31** (1987) 147–184.

G. Laporte, Y. Nobert, M. Desrochers, "Optimal routing under capacity and distance restrictions", *Operations Research* **33** (1985) 1050–1073.

E.L. Lawler, J.K. Lenstra, A.H.G. Rinnoy Kan, D.B. Shmoys (eds.), "The Traveling Salesman Problem: A guided tour of Combinatorial Optimization", Wiley, 1985.

S. Lin, B.W. Kernighan, "An effective heuristic algorithm for the traveling salesman problem", *Operations Research* **21** (1973) 498–516.

J.D.C. Little, K.G. Murty, D.W. Sweeney, C. Karel, "An algorithm for the traveling salesman problem", *Operations Research* **11** (1963) 972–989.

G.L. Nemhauser, L.A. Wolsey, "Integer and Combinatorial Optimization", Wiley, 1988.

M.W. Padberg, S. Hong, "On the symmetric traveling salesman problem: a computational study", *Mathematical Programming Study* **12** (1980) 78–107.

M.W. Padberg, M.R. Rao, "Odd minimum cut-sets and b -matchings", *Mathematics of Operations Research* **7** (1982) 67–80.

M.W. Padberg, G. Rinaldi, "Optimization of a 532 city symmetric traveling salesman problem by branch and cut", *Operations Research Letters* **6** (1987), 1–7.

M.W. Padberg, G. Rinaldi, "An efficient algorithm for the minimum capacity cut problem", *Mathematical Programming* **47** (1990a), 19–36.

M.W. Padberg, G. Rinaldi, "Facet identification for the symmetric traveling salesman polytope", *Mathematical Programming* **47** (1990b), 219–257.

M.W. Padberg, G. Rinaldi, "A branch and cut algorithm for the resolution of large-scale symmetric traveling salesman problems", *SIAM Review* **33** (1991), 60–100.

A. Schrijver, "Linear and Integer Programming", Wiley, 1986.