UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

DIEGO GALINDO PECIN

Uso de rotas elementares no *CVRP*

Goiânia
2010

DIEGO GALINDO PECIN

Uso de rotas elementares no *CVRP*

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Computação.

Área de concentração: Otimização.

Orientador: Prof. Humberto José Longo

Goiânia
2010

DIEGO GALINDO PECIN

Uso de rotas elementares no *CVRP*

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Computação, aprovada em 23 de Fevereiro de 2010, pela Banca Examinadora constituída pelos professores:

Prof. Humberto José Longo
Instituto de Informática – UFG
Presidente da Banca

Prof. Cláudio Nogueira de Meneses
Instituto de Informática – UFG

Prof. Marcus Vinícius Soledade Poggi de Aragão
Departamento de Informática – PUC-Rio

Aos meus pais, João Pecin e Alexânia Dias Galindo, aos quais devo meu caráter e minha formação. Ao meu irmão Thiago Galindo Pecin, pela amizade e inspiração intelectual.

Agradecimentos

Sobretudo ao meu orientador Humberto Longo, pela oportunidade e confiança; ainda pela participação ativa em todas as fases deste trabalho, sempre com boa vontade.

À minha irmã Giselle, aos meus irmãos Guilherme e Fillipe e ao meu primo Alex Filho, pela convivência e amizade. Aos meus amigos Emerson Barros, Rafael Moreira, Renato Arcanjo e Rodson Silva, por me proporcionarem momentos agradáveis.

À minha avó Eni, pelo amor e consideração e aos meus tios Alexsimone e Alex.

Ao Instituto de Informática da Universidade Federal de Goiás, pelas condições de ensino e pesquisa. À CAPES, pelo apoio financeiro.

Resumo

Galindo Pecin, Diego. **Uso de rotas elementares no *CVRP***. Goiânia, 2010. 79p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Esta dissertação aborda o Problema do Caminho Elementar Mínimo com Restrição de Capacidade (*ESPPCC – Elementary Shortest Path Problem with a Capacity Constraint*) e descreve algoritmos para a sua resolução que fazem uso de conceitos tais como Correção de Rótulos, Programação Dinâmica Bidirecional e Relaxação Decrescente do Espaço de Estados. Esses algoritmos foram usados como geradores de rotas elementares no subproblema de geração de colunas de um algoritmo *BCP* robusto para o *CVRP*. Os resultados (limites inferiores, tempo de processamento e número de nós gerados) obtidos, para algumas instâncias de teste do *CVRP*, são comparados aos obtidos na versão original desse algoritmo *BCP*, que utiliza rotas não elementares sem 3-ciclos ou 4-ciclos.
Rotas elementares também são exploradas em um contexto de enumeração para o *CVRP*, a qual permite obter rotas (usando um critério baseado em limites e em custo reduzido) que possuem uma chance de pertencer a uma solução ótima. Se o número de rotas não for muito grande (na ordem de poucos de milhares), então todo o problema pode ser resolvido como um problema de particionamento de conjuntos contendo apenas tais rotas. Algumas vezes isso acelera o algoritmo *Branch-and-Bound* consideravelmente, quando comparado com estratégias tradicionais de particionamento (*branching*), já que muitos nós da árvore podem ser resolvidos sem a geração de novos nós.

Palavras–chave

CVRP, Enumeração, *ESPPCC*, *ESPPRC*, Geração de Colunas, Rotas Elementares.

Abstract

Galindo Pecin, Diego. **Using elementary routes to solve the CVRP**. Goiânia, 2010. 79p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

This dissertation addresses the optimization of the Elementary Shortest Path Problem with a Capacity Constraint (ESPPCC) and describes algorithms for its resolution that make use of concepts such as Label-Setting, Bidirectional Dynamic Programming and Decremental State Space Relaxation. These algorithms were used in a robust CVRP's Branch-and-Cut-and-Price framework as the column generation mechanism. The resulting BCP was used to obtain results (lower bounds, processing time and the number of branching nodes generated) to several CVRP's test instances. These results are compared with previous ones obtained with the original BCP, which is based on k-cycle elimination.

Elementary routes are also explored in a route enumeration context, which allows the enumeration of all possible relevant elementary routes, i.e., all routes that have a chance of being part of an optimal CVRP's solution. If the number of relevant routes is not too large (say, in the range of tenths of thousands), the overall problem may be solved by feeding a general MIP solver with a set-partition formulation containing only those routes. If this set-partition can be solved, the optimal solution will be found and no branch will be necessary. Sometimes this leads to very significant speedups when compared to traditional branch strategies.

Keywords

Column Generation, CVRP, Elementary Routes, ESPPCC, ESPPRC.

Sumário

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO 1

Introdução

Os Problemas de Roteamento de Veículos (*VRPs – Vehicle Routing Problems*) correspondem a uma classe de problemas de otimização combinatória que consistem em determinar um conjunto ótimo de rotas para uma frota de veículos, inicialmente localizados em um ou mais depósitos, a fim de atender as demandas de um conjunto de clientes. Em geral, a solução para um *VRP* requer encontrar rotas, que iniciem e terminem em um mesmo depósito, e que atendam todas as demandas dos clientes, respeitando as restrições operacionais impostas (algumas restrições típicas entre outras são: a capacidade limitada dos veículos; os clientes requererem entrega ou coleta de produtos ou, em alguns casos, ambos; os períodos do dia em que os clientes podem ser atendidos), tal que o custo (geralmente, uma função da distância percorrida ou do tempo gasto) total de transporte seja mínimo.

Introduzido na literatura por Dantzig e Ramser [15], ao estudarem um problema de distribuição de gasolina entre estações de abastecimento, o *VRP* surge naturalmente como um problema central nas áreas de transporte, distribuição e logística. Como um elevado percentual do valor agregado aos produtos deve-se aos custos despendidos na distribuição dos mesmos, justifica-se o uso de métodos informatizados para o planejamento do transporte, pois estes geralmente resultam em economias significativas (entre 5% a 20%) nos custos totais, conforme relatado por Toth e Vigo [56].

Esta dissertação se concentra na versão básica do *VRP*, o Problema de Roteamento de Veículo com Restrição de Capacidade (*CVRP – Capacitated Vehicle Routing Problem*), que pode ser descrito da seguinte forma: um conjunto de clientes, cada um com uma demanda, precisa ser atendido por uma frota de veículos (de idêntica capacidade). Todos os veículos estão inicialmente localizados em um mesmo depósito central e devem desenvolver rotas que iniciem e terminem neste depósito; cada cliente deve ser visitado exatamente uma vez por um veículo e a capacidade deste não pode ser excedida. O objetivo é atender todos os clientes perfazendo a menor distância total.

A natureza intrinsicamente combinatorial dos *VRPs* sugere que os mesmos sejam formulados e resolvidos como um problema de Programação Linear Inteira (PLI). Assim, antes de apresentar o foco e a organização desta dissertação (Seção 1.2), propriamente,

discorre-se, brevemente, na Seção 1.1, sobre os métodos exatos empregados na resolução de PLIs e a aplicação destes em problemas de roteamento de veículos. Além disso, assume-se, para um bom entendimento deste e dos próximos capítulos, que o leitor tenha familiaridade com conceitos e técnicas de Programação Linear Inteira (Capítulos 1 e 2) e Programação Dinâmica (Capítulos 3, 4 e 5).

1.1 Métodos exatos

Muitos métodos exatos para PLIs baseiam-se no paradigma *Branch-and-Bound*. Um algoritmo baseado neste paradigma estrutura a busca de uma solução ótima em uma árvore, onde cada nó representa uma parte do espaço de soluções. Começando no nó raiz, um problema pode ser dividido em dois, dividindo a região viável de uma variável inteira ao meio (procedimento referido como *branching*). Relaxações do problema podem ser usadas em cada nó da árvore *Branch-and-Bound* para o cálculo de estimativas (ou limites) inferiores para o valor de uma solução ótima (no caso da função objetivo ser de minimização). Assim, se o limite inferior para algum nó da árvore for maior que o menor limite superior conhecido (isto é, a melhor solução viável encontrada até o momento), então esse nó pode ser, seguramente, descartado da busca. Uma forma simples de calcular um limite inferior é usar a relaxação linear do PLI, ou seja, considerar o espaço de soluções aumentado, onde as variáveis inteiras são autorizadas a obter valores fracionários.

Uma forma de melhorar o cálculo do limite inferior é aplicar o método de *planos de corte* que, muitas vezes, é usado para encontrar soluções inteiras em PLIs. Para se resolver um PLI por este método, em geral, resolve-se sua relaxação linear e, se a solução ótima for fracionária, desigualdades violadas (as quais são encontradas usando-se os chamados algoritmos de *separação*, cuja função é encontrar, dentre todas as desigualdades de uma família de desigualdades válidas para o problema, pelo menos uma violada pela solução corrente, caso exista) são adicionados à relaxação. Este processo é repetido, até encontrar uma solução inteira (a solução ótima para o PLI) ou não existir desigualdades violadas conhecidas (neste caso, tem-se um limite inferior). Mesmo neste último caso, ainda é possível encontrar uma solução inteira ótima. Para tanto, basta proceder conforme um algoritmo *Branch-and-Bound* (para se obter melhores limites, pode-se resolver cada nó da árvore com um algoritmo de planos de corte). Dá-se o nome *Branch-and-Cut* ao algoritmo obtido pela combinação do paradigma *Branch-and-Bound* com o método de planos de cortes. Algoritmos *Branch-and-Cut* estão entre os melhores métodos para a resolução de PLIs arbitrários, pois planos de cortes gerais em PLIs têm sido amplamente estudado (veja, por exemplo, Nemhauser and Wolsey [45] e Wolsey [59]).

Quando a formulação para um PLI é bem estruturada, isto é, quando há determinados conjuntos de variáveis onde algumas restrições não são sobrepostas, é possível usar a técnica de decomposição de Dantzig-Wolfe para decompor o problema original em subproblemas menores (normalmente, fáceis de resolver), e combinar suas soluções em um problema *mestre* (veja Martin [42]). O problema mestre é, em geral, pequeno no número de restrições, mas pode ter um grande número de variáveis (exponencial, no pior caso). Uma relaxação linear do problema mestre é uma forma diferente de calcular o limite inferior e pode ser, em muitos casos, um limite melhor que o obtido com a relaxação linear do problema original. A desvantagem é que para calcular este limite, o subproblema deve ser resolvido, iterativamente, até encontrar o valor ótimo da função objetivo do problema mestre. Isso também é conhecido como *geração de colunas*, uma vez que a solução dos subproblemas dá origem a novas colunas no problema mestre. Quando a geração de colunas é inserida em um algoritmo *Branch-and-Bound*, tem-se um algoritmo *Branch-and-Price* (veja, por exemplo, Desaulniers, Desrosiers e Solomon [16], Lübbecke and Desrosiers [39]). Note-se que algoritmos *Branch-and-Price* não são tão genéricos quanto algoritmos *Branch-and-Cut*, pois uma decomposição válida do problema deve ser feita pelo usuário do método. Além disso, os algoritmos para os subproblemas são, em geral, específicos e se aplicam para poucos tipos de problemas.

É possível combinar as abordagens de planos de cortes, geração de colunas e *Branch-and-Bound* para obter um algoritmo *Branch-and-Cut-and-Price* (*BCP*). No entanto, adicionar cortes em algoritmos *BCPs* não é tão simples como em algoritmos *Branch-and-Cut* puros. O cenário de corte pode ser visto sob dois ângulos:

- Cortes expressos em termos de variáveis da formulação original;
- Cortes expressos em termos de variáveis da formulação do problema mestre.

O primeiro caso tem uma relação com o segundo, uma vez que cortes expressos em termos de variáveis da formulação original podem ser dinamicamente separados, transformados e adicionados ao problema mestre, sem que isso modifique a estrutura do subproblema de geração de colunas, conforme observado, independentemente, por vários pesquisadores no final dos anos 90 [5, 21, 32, 33, 35, 57]. Esta observação permitiu a construção de algoritmos *BCP* ditos *robustos*, conforme a denominação dada por Poggi de Aragão e Uchoa [47], ao designar um algoritmo *BCP* no qual a estrutura dos subproblemas de separação e de geração de colunas permanece inalterada durante a sua execução. De fato, estes autores propuseram novas técnicas de reformulação que estendem a aplicabilidade de algoritmos BCP robustos para, praticamente, qualquer problema de otimização combinatória.

O segundo caso trata de cortes derivados diretamente das variáveis da formulação do problema mestre. Estes cortes não têm uma representação bem definida no problema

original e pode ser difícil incorporar o impacto dos mesmos nos subproblemas. Diferentes abordagens têm sido sugeridas, como expressar a mudança como uma função objetivo não-linear ou adicionar outras variáveis nos subproblemas. Ambas as abordagens podem mudar a complexidade do subproblema e resultar em um esforço computacional elevado para resolvê-lo. Contudo, este caso é menos estudado tanto teórica quanto experimentalmente e foi o principal foco de estudos recentes (veja Spoorendonk [53]). Algoritmos *BCP* incluídos neste caso são ditos *não-robustos*, de acordo com a classificação dada em [47], uma vez que a complexidade do subproblema de geração de colunas pode aumentar quando tais cortes são adicionados ao problema mestre.

Aplicação em roteamento de veículos

A seguir descreve-se a aplicação de conceitos e técnicas de PLI em problemas de roteamento de veículos. Em particular, uma reformulação para *VRPs* de particionamento de conjuntos, onde as variáveis são associadas a rotas viáveis, é apresentada.

A estrutura dos *VRPs* sugerem reformulá-los como um Problema de Particionamento de Conjuntos (*SPP – Set Partitioning Problem*) e aplicar a técnica de geração de colunas, porque uma solução para o *VRP* é composta por um conjunto de rotas (interpretadas como colunas), uma para cada veículo da frota, que podem ser calculadas de forma independente, desde que elas cubram o conjunto de clientes a serem visitados. Em uma abordagem de geração de colunas, o problema mestre é um *SPP* conforme o modelo a seguir:

$$\text{VRP-SPP}\begin{cases} C = \min \sum_{r_k \in \Omega} c_k . x_k \\ \text{sujeito a} \\ \quad \sum_{r_k \in \Omega} a_{ik} . x_k = 1 \quad \forall\, v_i \in V \quad (1) \\ \quad \sum_{r_k \in \Omega} x_k \leq K \quad (2) \\ \quad x_k \in \{0,1\} \quad \forall\, r_k \in \Omega, \quad (3) \end{cases}$$

onde $V = \{v_1, \ldots, v_n\}$ é o conjunto de clientes, Ω é o conjunto de *rotas elementares*[1] viáveis, K é o número de veículos disponíveis, c_k é o custo da rota $r_k \in \Omega$ e a_{ik} é igual a 1 se a rota $r_k \in \Omega$ visita o cliente v_i ou igual a 0 caso contrário. As restrições (1) indicam que cada cliente $v_i \in V$ deve ser servido por exatamente uma rota $r_k \in \Omega$. A restrição (2) impõe um limite máximo de K rotas para uma solução, já que cada veículo está associado a uma única rota. Finalmente, as restrições (3) indicam se uma rota $r_k \in \Omega$ pertence ($x_k = 1$) ou não ($x_k = 0$) a uma solução.

[1] No *VRP* original, cada cliente tem de ser visitado uma vez. Assim, as rotas não devem incluir ciclos, as quais são referidas como *rotas elementares*.

A relaxação linear do modelo VRP-SPP (que pode ser obtida fazendo-se $x_k \geq 0$ nas restrições (3)) geralmente produz limites inferiores justos (i.e., próximos do custo de uma solução ótima). Contudo, como Ω contém um número exponencial de colunas, somente um subconjunto $\Omega^{'}$ é mantido em um problema mestre linear restrito e mais rotas viáveis devem ser geradas em tempo real, resolvendo-se o subproblema de geração de colunas iterativamente. Este subproblema consiste em encontrar colunas de custo reduzido negativo ou provar que tais colunas não existem. O custo reduzido de uma rota $r_k \in \Omega$ é dado por:

$$\bar{c}_k = c_k - \sum_{v_i \in V} a_{ik}.\lambda_i + \lambda_0\,, \tag{1-1}$$

onde (λ, λ_0) é o vetor de variáveis duais correspondentes às restrições (1) e (2), respectivamente, no problema mestre linear restrito.

É comum que as rotas dos veículos satisfaçam algumas restrições adicionais como, por exemplo, restrições de capacidade, precedência ou janelas de tempo. Tais restrições não alteram a estrutura do problema mestre, mas elas devem ser consideradas no subproblema de geração de colunas, isto é, elas restringem o conjunto Ω de rotas viáveis. O tipo de subproblema de geração de colunas que surge neste contexto é um problema de caminho[2] mais curto com algumas características especiais, e este pode ser formulado em um grafo com arestas de custo negativo (e, possivelmente, com ciclos de custo negativo), além de poder estar sujeito a uma série de restrições, como mencionado acima. Estas restrições são, geralmente, representadas como *restrições de recursos*, uma vez que distâncias, custos, tempo e capacidades podem ser interpretadas como recursos que são consumidos cada vez que um veículo percorre uma aresta ou visita um cliente. Assim, o subproblema pode ser formulado como um Problema do Caminho Elementar Mínimo com Restrições de Recursos (*ESPPRC – Elementary Shortest Path Problem with Resource Constraints*), o qual consiste em encontrar um caminho simples, de menor custo, entre dois vértices de um grafo, tal que nenhuma restrição de recurso seja violada ao longo do mesmo. Se for permitido ter ciclos de custo negativo no grafo subjacente, então pode-se provar que o *ESPPRC* é um problema fortemente $\mathcal{NP}$-Difícil (veja Dror [18]). Assim, a restrição de ciclo é, geralmente, relaxada, substituindo-se as restrições (1) do modelo VRP-SPP pelas restrições:

$$\sum_{r_k \in \Omega} a_{ik}.x_k \;\geq\; 1 \quad \forall\, v_i \in V \tag{4}$$

onde, agora, a_{ik} representa o número de vezes que a rota r_k visita o cliente v_i. A substituição de tais restrições permite obter um novo modelo para o *VRP*, onde o

[2] Uma rota pode ser vista como um caminho que inicia e termina em um mesmo ponto.

problema mestre torna-se um Problema de Cobertura de Conjuntos (*SCP – Set Covering Problem*) e o subproblema torna-se um Problema do Caminho Mínimo com Restrições de Recursos (*SPPRC – Shortest Path Problem with Resource Constraints*), o qual não restringe múltiplas visitas a um mesmo vértice, admitindo algoritmos psedo-polinomiais (veja Christofides, Mingozzi e Toth [11]). Este novo modelo (referido como VRP-SCP) não requer que rotas sejam elementares (o conjunto Ω é estendido). Porém, pode-se demonstrar que uma solução inteira ótima para o modelo VRP-SCP contém apenas rotas elementares, Chabrier [10]. A principal razão para se relaxar a restrição de ciclo do subproblema deve-se ao fato do *SPPRC* admitir algoritmos pseudo-polinomais. No entanto, este método tem uma clara desvantagem, que é um enfraquecimento do limite inferior (obtido com a relaxação linear do modelo VRP-SCP), conforme mencionado, por exemplo, por Gelinas et al. [25].

São raros (e recentes) os algoritmos *Branch-and-Price* (ou mesmo algoritmos *Branch-and-Cut-and-Price*) propostos na literatura para *VRPs* onde o subproblema de geração de colunas é interpretado como um *ESPPRC*. De fato, Feillet et al. [20] foram os primeiros a usar um algoritmo de correção de rótulos para o *ESPPRC* em um contexto de roteamento de veículos, mais precisamente para o Problema de Roteamento de Veículos com Janelas de Tempo (*VRPTW – Vehicle Routing Problem with Time Windows*)[3]. Posteriormente, algoritmos de correção de rótulos para o *ESPPRC* foram usados por Chabrier [10], Danna e Pape [14], Salani [51] e Jepsen et al. [30] para resolver com êxito instâncias do *VRPTW* até então não resolvidas (veja Solomon [52]). Até o início da escrita desta dissertação, devia-se a Baldacci, Christofides e Mingozzi [3] o único trabalho na literatura a aplicar um algoritmo de correção de rótulos para o *ESPPRC* na geração de colunas de um algoritmo para o *CVRP*.

1.2 Foco e estrutura da dissertação

Técnicas de geração de colunas e *Branch-and-Price* fazem parte do conjunto de métodos exatos para a resolução do *CVRP*. Em geral, quando o problema original é decomposto (usando-se, por exemplo, a decomposição de Dantzig-Wolfe), o subproblema de geração de colunas correspondente consiste em encontrar uma rota elementar (coluna) que satisfaça a restrição de capacidade dos veículos e que seja de custo reduzido mínimo. Este subproblema é um caso especial do *ESPPRC* que ocorre quando uma única restrição de recurso é imposta (a restrição de capacidade), e é conhecido na literatura como

[3] O *VRPTW* estende o *CVRP* impondo que cada cliente deve ser visitado dentro de uma determinada janela de tempo.

Problema do Caminho Elementar Mínimo com Restrição de Capacidade (*ESPPCC* – *Elementary Shortest Path Problem with a Capacity Constraint*).

Devido ao *ESPPCC* ser um problema fortemente $\mathcal{NP}$-Difícil (veja Kohl [34]), geralmente resolve-se uma relaxação deste subproblema na geração de colunas. Ao invés de se encontrar uma rota elementar de custo mínimo, procura-se por uma rota de menor custo que satisfaça a restrição de capacidade dos veículos, porém esta só pode conter ciclos com uma quantidade de arestas superior a um parâmetro δ, Irnich e Villeneuve [27]. Como um cliente pode ser visitado mais de uma vez nesta versão relaxada, a cada visita toda a sua demanda deve ser atendida. Embora resolver uma tal relaxação seja mais rápido que resolver o problema em si, os limites obtidos com a eliminação de δ-ciclos para valores pequenos de δ em geral são de qualidade inferior. Um dos mais bem sucedidos algoritmos propostos para o *CVRP*, Fukasawa et al. [23], resolve o subproblema de geração de colunas baseando-se na resolução do *SPPRC* apresentada em [27], onde δ-ciclos são eliminados para a obtenção de melhores limites.

O objetivo principal desta dissertação é explorar o uso de rotas elementares na geração de colunas do *CVRP*. Os algoritmos finais propostos para o *CVRP* são modificações daquele apresentado por Fukasawa et al. [23]. Agora, o subproblema de geração de colunas resume-se a um *ESPPCC*, o qual é resolvido com os recentes algoritmos propostos por Feillet et al. [20] e por Righini e Salani [49, 50]. A fim de acelerar a etapa de geração de colunas destes novos algoritmos para o *CVRP*, uma heurística para o *ESPPCC* é proposta. Assim, a versão exata para o *ESPPCC* só é executada quando esta heurística falha em encontrar rotas elementares (colunas) de custo reduzido negativo. Para testar estes novos algoritmos foram usadas algumas instâncias do *CVRP* conhecidas na literatura. Os resultados obtidos (limites inferiores, tempo de processamento e número de nós gerados) são comparados com a versão original de Fukasawa et al. [23], quando apenas δ-ciclos são eliminados, para $\delta = 3$ e $\delta = 4$.

Rotas elementares também são exploradas em um contexto de enumeração, que pode ser vista como uma alternativa ao tradicional particionamento (*branching*) do espaço de soluções viáveis em algoritmos *Branch-and-Bound*. Em geral, quando algoritmos *Branch-and-Price* (ou mesmo algoritmos *BCP*) são aplicados em *VRPs*, em cada nó de uma árvore *Branch-and-Bound* uma relaxação linear da reformulação de cobertura de conjuntos (veja a reformulação VRP-SCP) do problema é resolvida via geração de colunas (e cortes, no caso dos *BCPs*). Se uma solução fracionária (menor que a melhor solução inteira conhecida) é obtida ao final da geração de colunas, o espaço de soluções do nó corrente deve ser particionado em dois ou mais nós filhos.

É razoável supor que o algoritmo *Branch-and-Price* tenha melhor desempenho se alguns de seus nós puderem ser resolvidos à otimalidade (rapidamente) sem o particionamento do espaço de soluções (o qual dá origem a novos nós). A observação seguinte

permite o descarte de alguns nós sem a realização de tal particionamento (*branching*). Suponha que o *gap* de integralidade, isto é, a diferença entre o custo da melhor solução viável conhecida e o valor da relaxação linear para o nó corrente seja suficientemente pequeno. Então, pode ser prático enumerar todas as rotas elementares que possuem alguma chance de pertencer a uma solução ótima. Se este número de rotas não for tão grande (na faixa de dezenas de milhares), pode-se resolver, alternativamente, um problema (reduzido) de particionamento de conjuntos contendo apenas tais rotas (veja a reformulação VRP-SPP). Caso uma solução ótima seja encontrada, então não é necessário particionar o espaço de soluções do nó em questão (veja, por exemplo, Baldacci, Bodin, e Mingozzi [2]).

Esta dissertação está organizada da seguinte forma: o Capítulo 2 define o *CVRP* formalmente, apresenta a formulação e as principais partes (geração de colunas e cortes, regra de *branching*, entre outras) do algoritmo *BCP* robusto proposto por Fukasawa et al. [23]. O Capítulo 3 define o problema *ESPPRC* (e os conceitos de rótulos, extensão e dominância), descreve o algoritmo de programação dinâmica básico para a sua resolução, o qual foi proposto por Feillet et al. [20], e propõe uma adaptação deste algoritmo para o caso particular *ESPPCC*, além da heurística supramencionada. O final deste capítulo apresenta resultados computacionais obtidos quando o algoritmo e a heurística para o *ESPPCC* são usados, conjuntamente, na geração de colunas do algoritmo de Fukasawa et al. [23], substituindo o mecanismo original. O Capítulo 4 propõe algoritmos aprimorados para o *ESPPCC*, os quais são adaptações dos recentes algoritmos de programação dinâmica propostos por Righini e Salani [49, 50] para o *ESPPRC*, que fazem uso de técnicas avançadas, tais como as técnicas de programação dinâmica bidirecional e relaxação decrescente do espaço de estados (*DSSR – Decremental State Space Relaxation*). De forma análoga ao capítulo anterior, estes novos algoritmos para o *ESPPCC* são testados na geração de colunas do algoritmo de Fukasawa et al. [23] conjuntamente com a referida heurística. O Capitulo 5 descreve um enumerador de rotas elementares, baseado em programação dinâmica bidirecional, e apresenta resultados computacionais quando este é usado no mecanismo de enumeração de um novo algoritmo para o *CVRP*, o qual segue a reformulação e as técnicas propostas por Pessoa, Poggi de Aragão e Uchoa [46] para a construção de algoritmos *BCP* robustos para *VRPs*. Finalmente, o Capítulo 6 apresenta conclusões e considera possíveis trabalhos futuros.

CAPÍTULO 2

O Problema de Roteamento de Veículo com Restrição de Capacidade

O Problema de Roteamento de Veículos com Restrição de Capacidade (*CVRP – Capacitated Vehicle Routing Problem*) refere-se à determinação de um conjunto de rotas ótimas para uma frota de veículos, a fim de atender as demandas de um conjunto de clientes. As restrições do problema são: todos os veículos têm a mesma capacidade de carga e inicialmente estão posicionados em um mesmo depósito central; todas as demandas são do mesmo tipo (todas de coleta ou todas de entrega); cada rota deve começar e terminar no depósito; a demanda de cada cliente deve ser integralmente atendida por apenas uma rota e a soma das demandas atendidas por uma rota deve ser inferior à capacidade do veículo que a percorre. O objetivo é determinar rotas que atendam essas restrições e tais que o custo total para se percorrer todas elas seja mínimo. Este problema, proposto por Dantzig e Ramser [15], pertence à classe $\mathcal{NP}$-Difícil de problemas de Otimização Combinatória, já que generaliza o Problema do Caixeiro Viajante (*TSP – Traveling Salesman Problem*), que por sua vez também pertence à classe $\mathcal{NP}$-Difícil, Garey e Johnson [24].

Um algoritmo exato para o *CVRP* considerado um marco foi o proposto por Christofides, Mingozzi e Toth [11] em 1981. Este algoritmo usa um limite Lagrangeano obtido a partir da solução do problema das q-rotas mínimas. Uma q-rota é um caminho que inicia no depósito, percorre uma sequência de clientes, limitando a demanda acumulada a um dado valor máximo, e retorna ao depósito. Este caminho não precisa ser, necessariamente, um caminho simples, isto é, um mesmo cliente pode ser visitado mais de uma vez e a cada vez toda a sua demanda é contabilizada. Portanto, o conjunto de rotas válidas para o *CVRP* está estritamente contido no conjunto de q-rotas. Este mesmo trabalho descreve outro limite inferior, baseado em árvores geradoras, em que o depósito tem grau $K \leq k \leq 2K$ e são utilizadas $2K - k$ arestas de menor custo possível, onde K é o número de veículos. O algoritmo *Branch-and-Bound* resultante foi capaz de resolver instâncias com até 25 clientes, um tamanho respeitável para a época.

A importância prática do *CVRP* justifica o esforço envolvido no desenvolvimento

de algoritmos heurísticos (veja Laporte e Semet [38] e Gendreau, Laporte e Potvin [26]). Algoritmos exatos são revisados nos trabalhos de Laporte [36] e Ficher [22]. Os capítulos de Toth e Vigo [54], Naddef e Rinaldi [44] e Bramel e Simchi-Levi [8] do livro editado por Toth e Vigo [55], revisam os melhores métodos exatos propostos na literatura até 2002. Uma revisão recente sobre algoritmos exatos e heurísticos para o *CVRP* pode ser encontrada no capítulo de Cordeau et al. [13] do livro editado por Barnhart e Laporte [4].

Os melhores métodos exatos atualmente disponíveis para o *CVRP* são os propostos por Fukasawa et al. [23] e Baldacci, Christofides e Mingozzi [3]. Fukasawa et al. propuseram um algoritmo *BCP* robusto que combina o algoritmo Branch-and-Cut de Lysgaard, Letchford e Eglese [41] com a abordagem de particionamento de conjuntos (*SPP – Set Partitioning Problem*), onde as colunas do *SPP* correspondem ao conjunto de q-rotas, o qual contém o conjunto de rotas válidas para *CVRP*. Os experimentos computacionais mostraram que este algoritmo conseguiu resolver, de forma consistente, todas as instâncias então disponíveis na literatura com até 135 clientes.

Baldacci, Christofides e Mingozzi propuseram um algoritmo *BCP* não-robusto que usa uma formulação de particionamento de conjuntos para o *CVRP* com cortes adicionais (clique e capacidade), expressos em termos das variáveis do problema mestre, para reforçar sua relaxação linear. O método de solução proposto por estes autores consiste, primeiramente, em eliminar um grande número de variáveis, usando um critério baseado em custo reduzido, antes de resolver o modelo resultante (um *SPP* que contém somente as rotas elementares cujo custo reduzido é menor que a diferença entre um limite superior e inferior alcançados) com um solucionador de problemas de programação inteira[1]. A aplicação deste critério requer a computação de uma solução para o dual da relaxação linear da formulação *SPP*. Para fazer isso, eles usam um procedimento para o cálculo de um limite inferior para o *CVRP* que resolve (por geração de colunas), na sua última etapa, a relaxação linear de um modelo *SPP* modificado. Baseado na observação de que os valores duais das desigualdades de clique são não-positivos, o subproblema de geração de colunas é resolvido por um esquema de enumeração exaustiva, baseada em programação dinâmica, que desconsidera esses valores duais mas garante que todas as colunas de custo reduzido negativo podem ser encontradas.

O uso de tais elementos (cortes sobre a formulação *SPP* e geração de rotas elementares)[2] permitiu Baldacci, Christofides e Mingozzi [3] resolverem quase todas as instâncias então disponíveis na literatura com até 135 clientes em um tempo abaixo daqueles apresentados em [23]. Contudo, alguns autores (veja Pessoa, Poggi de Aragão e Uchoa [46]) argumentam que este procedimento parece pouco escalável, pois o esquema

[1] CPLEX versão 9.0.

[2] Um algoritmo *BCP* não-robusto com geração de rotas elementares (subproblema) e separação de cortes de clique (sobre o problema mestre) já havia sido proposto para o *VRPTW* por Jepsen et al. [31].

enumerativo funciona bem para instâncias pequenas, mas tende a falhar quando a maior rota na solução ótima obtida contém muitos clientes (tipicamente, mais de 12 clientes).

Este capítulo descreve detalhadamente o algoritmo para o *CVRP* proposto por Fukasawa et al. [23], o qual foi a base para a construção dos novos algoritmos apresentados nos capítulos 3 e 4, que diferem do original apenas quanto ao mecanismo de geração de colunas. Assim, este capítulo está organizado conforme a seguir: a Seção 2.1 apresenta a formulação para o *CVRP* sobre a qual o algoritmo de Fukasawa et al. foi construído e a Seção 2.2 apresenta cada uma das partes que o compõem, tais como a geração de colunas e cortes, a regra de *branching*, a seleção de nós, entre outras.

2.1 Formulação do *CVRP*

O *CVRP* é definido em um grafo não orientado $G = (V,A)$, onde $V = \{v_0, v_1, \ldots, v_n\}$ e $A = \{(v_i, v_j) \mid v_i, v_j \in V,\ i < j\}$. O vértice v_0 representa o depósito e os demais representam os clientes ($V^+ = \{v_1, \ldots, v_n\}$). O depósito é a base de uma frota de $K \in \mathbb{N}^*$ veículos de idêntica capacidade $C \in \mathbb{N}^*$. A cada aresta $(v_i, v_j) \in A$ associa-se um custo $c_{ij} > 0$ de percorrê-la e a cada vértice $v_i \in V^+$ associa-se uma demanda $d_i > 0$ (para o depósito v_0, tem-se $d_0 = 0$). O objetivo é encontrar um conjunto de K rotas com custo mínimo e tais que:

- cada vértice $v_i \in V^+$ deve ser visitado exatamente uma vez e o veículo que o visita deve atender toda a sua demanda;
- todos os K veículos devem ser usados e cada veículo estar associado a uma única rota;
- a rota desenvolvida por um veículo deve iniciar e terminar no depósito v_0, passando por pelo menos um vértice $v_i \in V^+$; e
- a demanda total atendida por cada veículo não deve exceder a sua capacidade C.

Uma formulação clássica para o *CVRP* é aquela de Laporte e Norbert [37], a qual representa por x_e o número de vezes que a aresta $e = (v_i, v_j) \in A$ é visitada por um veículo. Seja $d(S)$ a soma das demandas de todos os vértices em S, onde $S \subseteq V^+$, e $\delta(S)$ o conjunto de arestas que possuem exatamente um extremo dentro do conjunto S. Define-se, então, o politopo P_1 em $\mathbb{R}^{|A|}$:

$$
P_1 = \begin{cases}
\sum\limits_{e \in \delta(\{v_i\})} x_e = 2 & \forall\, v_i \in V^+ & (1)\\
\sum\limits_{e \in \delta(\{v_0\})} x_e = 2.K & & (2)\\
\sum\limits_{e \in \delta(S)} x_e \geq 2.k(S) & \forall\, S \subseteq V^+ & (3)\\
x_e \leq 1 & \forall\, e \in A \setminus \delta(\{v_0\}) & (4)\\
x_e \leq 2 & \forall\, e \in \delta(\{v_0\}) & \\
x_e \geq 0 & \forall\, e \in A\ . &
\end{cases}
$$

As restrições (3) especificam que todo subconjunto de vértices S deve ser servido por uma quantidade suficiente de veículos (isto é, $k(S)$ veículos) e são os conhecidos cortes de capacidade (*capacity cuts*). A quantidade $k(S)$ é o custo de uma solução ótima para uma instância do *Bin-Packing Problem* e, portanto, encontrar $k(S)$ é um problema $\mathcal{NP}$-Difícil[3]. Porém, na prática, é usado o limite inferior dado por $\lceil d(S)/C \rceil$, pois este é simples de ser calculado. As demais restrições especificam que: (1) cada vértice (cliente) é visitado por um único veículo; (2) K veículos deixam e retornam ao depósito; (4) as arestas não incidentes ao depósito podem ser percorridas no máximo uma vez, já as incidentes no máximo duas (isto é, rotas com apenas um vértice são permitidas). Todas as soluções viáveis para o *CVRP* são definidas no conjunto $\mathcal{F}_1 = \{x \in P_1\}$. Logo, uma solução ótima, um vetor inteiro $x \in P_1$, para o *CVRP* é dada pelo valor:

$$L_1 = \min_{x \in P_1} \sum_{e \in A} c_e.x_e\,. \tag{2-1}$$

A formulação 2-1 tem um número polinomial de variáveis ($O(|A|)$) e exponencial de restrições, devido as desigualdades do tipo (3). Portanto, é necessário um algoritmo que gere iterativamente tais desigualdades para calcular a estimativa L_1, isto é, um algoritmo de planos de cortes.

Uma formulação alternativa, com um número exponencial de variáveis (colunas), pode ser obtida definindo-se variáveis λ correspondentes a q-rotas[4] que satisfazem a restrição de capacidade dos veículos. Embora exista um grande número de possíveis q-rotas, uma de menor custo pode ser encontrada usando-se um algoritmo de complexidade pseudo-polinomial ($O(|V|^2C)$ – veja a Seção .), ou seja, o subproblema de geração de colunas, o qual consiste em encontrar colunas (q-rotas) de custo reduzido negativo (ou provar que tais colunas não existem), em geral, pode ser resolvido rapidamente.

[3] O valor $k(S)$ é o número mínimo de compartimentos (*bins*) de volume igual a C necessário para empacotar os $|S|$ elementos de volumes d_i, $\forall\, v_i \in S$.

[4] O conceito de q-rota está definido no início deste capítulo.

Assim, seja p o número de q-rotas distintas e suponha que elas estejam enumeradas de $1 \dots p$. Seja λ_j a variável associada a j-ésima q-rota dessa enumeração e seja Q uma matriz $|A| \times p$, onde as colunas são vetores de incidência das arestas em cada uma das q-rotas. Além disso, seja q_j^e o coeficiente associado com a aresta $e \in A$ na j-ésima coluna de Q. Agora, considere o seguinte politopo P_2 em $\mathbb{R}^{|A|}$, definido como a projeção de um politopo em $\mathbb{R}^{p+|A|}$:

$$
P_2 = \text{proj}_x \left\{ \begin{array}{rcl}
\sum_{j=1}^{p} q_j^e.\lambda_j - x_e = 0 & \forall\, e \in A & (5) \\
\sum_{j=1}^{p} \lambda_j = K & & (6) \\
\sum_{e \in \delta(\{v_i\})} x_e = 2 & \forall\, v_i \in V^+ & (1) \\
\lambda_j \geq 0 & \forall\, j \in \{1, \dots, p\} & \\
x_e \leq 1 & \forall\, e \in A \setminus \delta(\{v_0\}) & (4) \\
x_e \leq 2 & \forall\, e \in \delta(\{v_0\}) & \\
x_e \geq 0 & \forall\, e \in A\ . &
\end{array} \right.
$$

As restrições (5) definem a relação entre as variáveis x e λ. A restrição (6) define o número de veículos a ser utilizado. Pode-se demonstrar que o conjunto de vetores inteiros em P_2 também define todas as soluções viáveis para o *CVRP*. Como o número de variáveis λ é exponencial, o limite inferior L_2, definido a seguir, pode ser calculado utilizando, por exemplo, geração de colunas ou relaxação Lagrangeana:

$$
L_2 = \min_{x \in P_2} \sum_{e \in A} c_e.x_e\,. \tag{2-2}
$$

Essa descrição de poliedros em termos de dois conjuntos de variáveis, λ e x, associado a geração de colunas ou relaxação Lagrangeana é chamada de *Mestre Explícita* por Poggi de Aragão e Uchoa [47]. A formulação Mestre Explícita para o *CVRP*, proposta por Fukasawa et al. [23], combina a formulação por geração de colunas (2-2) com a formulação de variáveis por arestas 2-1, obtendo uma formulação com um número exponencial de variáveis e restrições. Essa formulação utiliza o politopo P_3 (que corresponde à interseção dos politopos P_1 e P_2), cuja descrição no formato Mestre Explícita é dada a seguir:

$$
P_3 = \text{proj}_x \left\{ \begin{array}{rcl}
\displaystyle\sum_{e\in\delta(\{v_i\})} x_e & = & 2 \qquad \forall\, v_i \in V^+ \qquad (1)\\
\displaystyle\sum_{e\in\delta(\{v_0\})} x_e & = & 2.K \qquad (2)\\
\displaystyle\sum_{e\in\delta(S)} x_e & \geq & 2.k(S) \quad \forall\, S \subseteq V^+ \qquad (3)\\
x_e & \leq & 1 \qquad \forall\, e \in A \setminus \delta(\{v_0\}) \qquad (4)\\
x_e & \leq & 2 \qquad \forall\, e \in \delta(\{v_0\})\\
x_e & \geq & 0 \qquad \forall\, e \in A\\
\displaystyle\sum_{j=1}^{p} q_j^e.\lambda_j \; - \; x_e & = & 0 \qquad \forall\, e \in A \qquad (5)\\
\displaystyle\sum_{j=1}^{p} \lambda_j & = & K \qquad (6)\\
\lambda_j & \geq & 0 \qquad \forall\, j \in \{1,\ldots,p\}\ .
\end{array} \right.
$$

A restrição (6) pode ser descartada, pois ela é implicada pelas restrições (2) e (5). O cálculo do limite inferior L_3, dado pelo valor

$$L_3 = \min_{x\in P_3} \sum_{e\in A} c_e.x_e\,, \qquad (2\text{-}3)$$

requer a resolução do programa linear definido a seguir (e referido como ME-CVRP), o qual contém um número exponencial de variáveis e restrições:

$$
\text{ME-CVRP} \left\{ \begin{array}{l}
\text{Min} \displaystyle\sum_{e\in A} c_e.x_e\\
\text{sujeito a}\\
\begin{array}{rcl}
\displaystyle\sum_{e\in\delta(\{v_i\})} x_e & = & 2 \qquad \forall\, v_i \in V^+ \qquad (1)\\
\displaystyle\sum_{e\in\delta(\{v_0\})} x_e & \geq & 2.K \qquad (2)\\
\displaystyle\sum_{e\in\delta(S)} x_e & \geq & 2.k(S) \quad \forall\, S \subseteq V^+ \qquad (3)\\
x_e & \leq & 1 \qquad \forall\, e \in A \setminus \delta(\{v_0\}) \qquad (4)\\
x_e & \leq & 2 \qquad \forall\, e \in \delta(\{v_0\})\\
x_e & \geq & 0 \qquad \forall\, e \in A\\
\displaystyle\sum_{j=1}^{p} q_j^e.\lambda_j \; - \; x_e & = & 0 \qquad \forall\, e \in A \qquad (5)\\
\lambda_j & \geq & 0 \qquad \forall\, j \in \{1,\ldots,p\}\ .
\end{array}
\end{array} \right.
$$

Pode-se obter um programa linear mais compacto substituindo cada ocorrência da variável x_e em ME-CVRP por sua representação equivalente, dada pelas restrições (5).

O programa linear resultante é referido como o problema Dantzig-Wolfe Mestre para o *CVRP* (*DWM-CVRP*):

$$
\text{DWM-CVRP}
\begin{cases}
L_3 = \min \sum_{j=1}^{p} \sum_{e \in A} c_e . q_j^e . \lambda_j \\
\text{sujeito a} \\
\sum_{j=1}^{p} \sum_{e \in \delta(\{v_i\})} q_j^e . \lambda_j = 2 \qquad \forall\, v_i \in V^+ \qquad (7) \\
\sum_{j=1}^{p} \sum_{e \in \delta(\{v_0\})} q_j^e . \lambda_j = 2.K \qquad (8) \\
\sum_{j=1}^{p} \sum_{e \in \delta(S)} q_j^e . \lambda_j \geq 2.k(S) \qquad \forall\, S \subseteq V^+ \qquad (9) \\
\sum_{j=1}^{p} q_j^e . \lambda_j \leq 1 \qquad \forall\, e \in A \setminus \delta(\{v_0\}) \qquad (10) \\
\sum_{j=1}^{p} q_j^e . \lambda_j \leq 2 \qquad \forall\, e \in \delta(\{v_0\}) \\
\sum_{j=1}^{p} q_j^e . \lambda_j \geq 0 \qquad \forall\, e \in A \\
\lambda_j \geq 0 \qquad \forall\, j \in \{1, \ldots, p\} \ .
\end{cases}
$$

As desigualdades (9) (*capacity cuts*) não são as únicas que podem aparecer em *DWM-CVRP*. De fato, um corte genérico $\sum_{e \in A} a_e x_e \geq b$ pode ser incluído como $\sum_{j=1}^{p} (\sum_{e \in A} a_e q_j^e) . \lambda_j \geq b$.

2.2 Um algoritmo *BCP* robusto para o *CVRP*

O algoritmo *BCP* robusto proposto por Fukasawa et al. [23], o qual se baseia na formulação *DWM-CVRP*, é detalhado nesta Seção. Antes da descrição de suas principais partes, pode-se resumir o procedimento aplicado em cada nó da árvore *Branch-and-Bound* conforme o pseudocódigo a seguir:

1. Resolver o programa linear *DWM-CVRP* com um pequeno subconjunto do número total de colunas (referido como *DWM-CVRP* restrito).
2. Resolver o subproblema de geração de colunas obtendo colunas de custo reduzido negativo.
3. Adicionar as colunas obtidas em 2 ao *DWM-CVRP* restrito.
4. Se alguma coluna foi adicionada em 3, retornar para 1.
5. Aplicar os algoritmos de separação à solução obtendo as desigualdades violadas.

6. Adicionar as desigualdades encontradas em 5 ao *DWM-CVRP*.

7. Se algum corte foi adicionado em 6, retornar para 1. Caso contrário, parar.

2.2.1 Geração de colunas

O custo reduzido de uma coluna (variável λ) é a soma dos custos reduzidos das arestas pertencentes à q-rota correspondente. Sejam μ, ν, π, e ω os multiplicadores duais associados às restrições (7), (8), (9) e (10), respectivamente, de *DWM-CVRP*. O custo reduzido $\bar{c}_e$ de uma aresta é dado por:

$$\bar{c}_e = \begin{cases} c_e - \mu_i - \mu_j - \sum\limits_{S \mid \delta(S) \ni e} \pi_S - \omega_e & \text{se } e = \{v_i, v_j\} \in A \setminus \delta(\{v_0\}) \\ c_e - \nu - \mu_j - \sum\limits_{S \mid \delta(S) \ni e} \pi_S & \text{se } e = \{v_0, v_j\} \in \delta(\{v_0\}). \end{cases} \tag{2-4}$$

Um corte genérico adicional $\sum_{j=1}^{p} (\sum_{e \in A} a_e . q_j^e) . \lambda_j \geq b$ em *DWM-CVRP*, com multiplicador dual α, contribui com o valor $-a_e . \alpha$ no cálculo de $\bar{c}_e$.

O subproblema de geração de colunas consiste em encontrar q-rotas (colunas) de custo reduzido negativo e pode ser resolvido por um algoritmo de complexidade pseudo-polinomial ($O(|V|^2 C)$), conforme descrito a seguir.

O algoritmo básico

A estrutura de dados básica utilizada no algoritmo é uma matriz M de dimensão $C \times |V^+|$. Cada entrada $M(d,i)$ desta matriz representa um caminho mais curto (q-rota parcial) que parte do depósito v_0 e termina no vértice $v_i \in V^+$, tendo consumido uma demanda d, $d_i \leq d \leq C$. Cada entrada contém um *rótulo* que registra as seguintes informações: o custo do caminho (denotado por $\bar{c}(M(d,i))$), o vértice v_i, e um apontador para o rótulo que representa um caminho mais curto até o vértice anterior. Estas duas últimas informações são utilizadas para construir uma solução ótima, isto é, para obter a ordem em que os vértices são visitados nesta solução. Inicialmente, o único rótulo conhecido representa um caminho vazio e tem custo zero (entrada $M(0,0)$); todas as outras entradas da matriz M são inicializadas com rótulos que representam caminhos vazios com custos infinitos. Então, a matriz M é preenchida por programação dinâmica, começando com $d = 1$ e indo até $d = C$. Para cada entrada $M(d,i)$, $i \neq 0$, o algoritmo percorre cada vizinho v_j de v_i e avalia o custo do caminho representado por $M(d - d_i, j)$, $d - d_i \geq 0$, até v_i. Se $\bar{c}(M(d-d_i,j)) + \bar{c}_e < \bar{c}(M(d,i))$ ($e = (v_i, v_j)$), então $\bar{c}(M(d,i))$ é atualizado.

Para armazenar os resultados, usa-se um vetor B de tamanho V^+, onde a entrada $B(i)$, para cada vértice $v_i \in V^+$, representa um caminho mais curto que parte do depósito

v_0 e alcança o vértice v_i, tendo consumido uma demanda total dentro do intervalo $[d_i, C]$. Para se obter uma q-rota ótima, basta estender os rótulos em B até o vértice v_0 e tomar a de menor custo. Como existem $|V^+|.C$ entradas na matriz M e cada uma é processada em tempo $O(|V|)$, a complexidade do algoritmo é $O(|V|^2C)$.

Eliminação de ciclos

Uma forma natural de obter melhores limites inferiores na geração de colunas, sem um aumento substancial de complexidade, é obter q-rotas que não contenham δ-ciclos, isto é, ciclos com δ ou menos arestas, para valores pequenos de δ (veja Irnich e Villeneuve [27]).

O algoritmo funciona como descrito anteriormente, usando programação dinâmica para preencher uma matriz M, de dimensão $C \times |V^+|$, com caminhos parciais. Os rótulos possuem o mesmo significado de antes, mas agora cada entrada $M(d,i)$ desta matriz contém um *repositório* de rótulos, e não um único rótulo. Assim, o repositório $M(d,i)$ não representa apenas o caminho mais curto sem δ-ciclos que parte do depósito v_0 e alcança o vértice v_i, tendo consumido uma demanda d, $d_i \leq d \leq C$, mas também caminhos alternativos que garantem que todas as possíveis extensões de v_i com exatamente δ vértices são consideradas.

Esta noção é formalizada com o conceito de *dominância*. Um rótulo ℓ é δ-*dominado* por um conjunto de rótulos $\mathcal{L}$ se não existir um rótulo em $\mathcal{L}$ com custo superior ao de ℓ e se cada caminho que estende ℓ legalmente (isto é, sem que esta extensão crie um δ-ciclo) também estende algum rótulo em $\mathcal{L}$.

Como o algoritmo procura por uma q-rota sem δ-ciclos de menor custo, somente rótulos que podem representar um segmento das (isto é, somente rótulos não-dominados) q-rotas ótimas precisam ser considerados. Para simplificar a apresentação, assume-se que existe apenas uma q-rota ótima, e denota-se esta por $\mathcal{R}$. Assim, o algoritmo requer a existência, para cada rótulo dominado ℓ, de uma q-rota (sem δ-ciclos) de menor custo que não contenha ℓ, conforme a argumentação seguinte. Suponha que a q-rota ótima contivesse um rótulo ℓ dominado por um conjunto $\mathcal{L}$. O rótulo ℓ representa uma q-rota parcial sem δ-ciclos que parte do depósito v_0 e alcança algum vértice v_i; seja υ o restante desta q-rota ótima. A definição de dominância implica que existe pelo menos um rótulo $\ell' \in \mathcal{L}$ que, ao ser concatenado com υ, cria uma q-rota $\mathcal{R}'$ de custo menor que $\mathcal{R}$ (observe-se que, por definição, ambos os rótulos ℓ' e υ não contêm δ-ciclos; dominância apenas garante que não se forma δ-ciclos na junção destes rótulos).

Inicialmente, todos os repositórios da matriz M estão vazios. Uma vez que um rótulo existente é estendido (em um processo idêntico ao algoritmo básico), um novo rótulo é criado, e o algoritmo tenta inseri-lo no repositório apropriado. Se o rótulo é dominado por outros rótulos no repositório, ele é descartado; do contrário, ele é inserido

(se outros rótulos neste repositório tornarem-se dominados após esta inserção, então eles são descartados).

Isso significa que a parte complicada do algoritmo é a verificação eficiente de dominância. Christofides, Mingozzi and Toth [11] mostraram que para $\delta = 2$ é necessário manter apenas os dois rótulos de menor custo, pois dados quaisquer três rótulos pertencentes a um mesmo repositório, aquele com maior custo será dominado pelos outros dois. Para $\delta \geq 3$, decidir quais rótulos manter é significativamente mais complicado, conforme mostrado por Irnich e Villeneuve [27]. No algoritmo *BCP* proposto por Fukasawa et al. [23], apenas q-rotas sem δ-ciclos (colunas) são consideradas na geração de colunas, usando uma abordagem semelhante a apresentada em [27], a qual requer repositórios capazes de armazenar, pelo menos, $\delta!$ rótulos. Por esta razão, apenas valores de $\delta \in \{2,3,4\}$ são testados em [23].

Aceleração heurística

Em geral, algoritmos baseados na técnica de geração de colunas devem gerar colunas favoráveis (isto é, colunas de custo reduzido negativo) iterativamente, até minimizar o valor da função objetivo da relaxação linear do problema mestre (supondo que a função objetivo seja de minimização). Nesses algoritmos, tais colunas são obtidas resolvendo-se subproblemas em cada iteração. Para que todo o mecanismo de geração de colunas funcione eficientemente, é interessante usar heurísticas capazes de obter colunas de custo reduzido negativo rapidamente, e somente usar o algoritmo exato quando tais heurísticas falham em encontrar colunas favoráveis. Esse mecanismo, geralmente, reduz a quantidade de chamadas ao algoritmo exato, permitindo acelerar o procedimento como um todo.

Para este fim, Fukasawa et al. [23] usam três heurísticas, que encontram apenas q-rotas negativas (isto é, q-rotas nas quais a soma dos custos das arestas é de valor negativo) e sem δ-ciclos. Observe-se que tais heurísticas não geram, necessariamente, q-rotas de custo mínimo. Assim, quando nenhuma q-rota negativa é encontrada pelas heurísticas, o algoritmo exato deve ser executado para encontrá-las ou atestar a inexistência das mesmas (este último caso é a condição de término da etapa de geração de colunas).

A primeira técnica de aceleração usada é a de redução de *granularidade*, a qual considera $g > 1$ unidades de demanda de cada vez. O algoritmo exato é executado com uma demanda modificada $d_i^{'} = \lceil d_i/g \rceil$, para todo vértice v_i, e uma capacidade modificada $C^{'} = \lceil C/g \rceil$ para os veículos. O tempo de processamento será em função de $C^{'}$ em vez de C, e as soluções encontradas ainda permanecerão válidas na configuração original.

A segunda técnica usada é conhecida como *esparcificação*. Intuitivamente, arestas de baixo custo no grafo original têm maior probabilidade de aparecerem na solução. Levando isso em conta, um conjunto de cinco florestas geradoras disjuntas é pré-computado, usando uma extensão do algoritmo de *Kruskal* similar a heurística gulosa

descrita em [58]. Ao considerar apenas arestas dessas árvores e arestas incidentes ao depósito, o algoritmo de programação dinâmica beneficia-se de um conjunto restrito de vizinhos quando cada vértice é processado (consequentemente, menos rótulos são gerados).

A terceira aceleração é a poda de repositórios, a qual consiste em armazenar somente δ (ao invés de $\delta!$) rótulos por repositório. Para diminuir o número de extensões válidas bloqueadas, ao se decidir quais rótulos serão mantidos, leva-se em conta não apenas os custos dos mesmos, mas também a diversidade entre eles.

O algoritmo começa utilizando todas as três heurísticas descritas. Quando falha em encontrar q-rotas negativas usando uma ou mais heurísticas, uma delas é desativada. Por outro lado, se uma q-rota negativa for encontrada usando-se apenas um subconjunto de heurísticas, então outras heurísticas são novamente ativadas. Eventualmente, o algoritmo irá falhar sem nenhuma aceleração, caso em que se pode seguramente afirmar que nenhuma q-rota negativa existe (e que a geração de colunas convergiu).

2.2.2 Geração de cortes

Inicialmente, a formulação *DWM-CVRP* inclui somente restrições de grau; outras desigualdades violadas são adicionadas durante a execução do algoritmo. Além dos cortes de capacidade (9) e dos cortes de limite (10) da formulação *DWM-CVRP*, o algoritmo *BCP* de Fukasawa et al. [23] inclui os cortes utilizados por Lysgaard, Letchford e Eglese [41] (mantendo-se os nomes originais, conforme [41]): *framed capacity*, *strengthened comb*, *multistar*, *partial multistar*, *generalized multistar* e *hypotour cuts*. Estes cortes são definidos sobre as variáveis x, portanto deve-se converter uma solução ótima λ^* de *DWM-CVRP* em uma correspondente x^*. Se esta solução em x^* é fracionária, então ela é passada como parâmetro de entrada para o pacote CVRPSEP[5] [40]. As desigualdades violadas encontradas são traduzidas em variáveis λ para serem adicionadas ao programa linear.

2.2.3 Representação de variáveis e restrições

Ao se implementar um algoritmo *Branch-and-Cut-and-Price* é preciso ter uma diferenciação clara entre variáveis e colunas, e também entre restrições e linhas. Cada variável λ é associada a uma q-rota, que é armazenada em um repositório de variáveis. Toda vez que um novo corte é adicionado, deve-se acessar todas as q-rotas correspondentes às colunas do programa linear corrente para calcular os coeficientes da linha a ser inserida. Similarmente, cada restrição expressa em termos de variáveis x é armazenada em um re-

[5] O pacote CVRPSEP implementa os algoritmos de separação utilizados no algoritmo *Branch-and-Cut* de Lysgaard, Letchford e Eglese [41].

positório de restrições. Toda vez que uma variável é gerada, deve-se acessar cada linha do programa linear corrente para determinar as restrições correspondentes e calcular os coeficientes da coluna a ser inserida.

Como muitas operações são executadas toda vez que uma nova coluna ou linha é adicionada, torna-se imprescindível o uso de estruturas de dados apropriadas para tratá-las eficientemente. Os repositórios de variáveis e restrições implementados no algoritmo *BCP* de Fukasawa et al. [23] são indexados utilizando tabelas auxiliares. Dada uma aresta e, pode-se determinar rapidamente quais as colunas do programa linear estão associados a q-rotas contendo e e quais as linhas estão associadas às restrições com coeficiente não-nulo em x_e.

2.2.4 Regra de *branching*

Em cada nó da árvore *Brach-and-Bound*, o algoritmo começa adicionando colunas ao programa linear até não ser possível encontrar alguma de custo reduzido negativo. Então, o algoritmo procura por todos os cortes violados. Esses passos são repetidos até que a geração de colunas e cortes falhem em encontrar colunas e cortes. Se nesse ponto o nó não puder ser descartado da árvore de busca[6], então o algoritmo faz uma divisão do espaço de soluções (*branching*).

A regra de *branching* usada no algoritmo *BCP* de Fukasawa et al. [23] é uma adaptação da regra usada em [41]: escolhe-se conjuntos candidatos S tais que $2 < x^*(\delta(S)) < 4$ e faz-se *branching* impondo a disjunção $(x(\delta(S)) = 2) \vee (x(\delta(S)) \geq 4)$. Esses conjuntos são escolhidos utilizando-se um procedimento disponível no pacote CVRPSEP [40], o qual retorna os conjuntos que satisfazem $2 < x^*(\delta(S)) < 4$.

Dados os conjuntos candidatos retornados, usa-se a técnica *strong branching* para selecionar aquele sobre o qual se fará *branching*. Como o cálculo exato do limite inferior para cada nó filho pode ser demorado, estima-se este valor executando um pequeno número de iterações de geração de colunas. Para limitar o tempo gasto fazendo-se *strong branching*, principalmente quando a profundidade f do nó avaliado é grande, somente r conjuntos candidados são considerados, onde $5 \leq r \leq \max\{10 - f, 5\}$. Na escolha dos conjuntos, dá-se prioridade àqueles com menores valores de $|x^*(\delta(S)) - 2.7|/d(S)$. Fukasawa et al. [23] usam o valor 2.7 porque a restrição $x(\delta(S)) = 2$ tende a ter um impacto maior na relaxação do programa linear que $x(\delta(S)) \geq 4$, podendo levar a um desbalanceamento na árvore *Branch-and-Bound*. Valores mais próximos de 2 que de 4 aumentam o impacto de impor a desigualdade $(x(\delta(S)) \geq 4)$.

[6] O nó da árvore *Brach-and-Bound* pode ser descartado da busca se o valor de sua solução for inteiro ou superior ao valor do melhor limite superior conhecido.

2.2.5 Seleção de nós e limites superiores

Fukasawa et al. [23] utiliza a busca em profundidade como estratégia de seleção de nós na árvore *Branch-and-Bound*, pois esta requer uma menor quantidade de memória se comparada com outras estratégias, tal como a busca em largura, por exemplo.

Não usam-se heurísticas para encontrar limites superiores (soluções viáveis) para o *CVRP*. O limite primal utilizado é o valor da melhor solução conhecida para a instância.

CAPÍTULO 3

O algoritmo básico para o ESPPCC

O Problema do Caminho Elementar Mínimo com Restrição de Capacidade (*ESPPCC – Elementary Shortest Path Problem with a Capacity Constraint*) consiste em encontrar, em um grafo vértice-valorado, um caminho elementar (isto é, um caminho simples), entre um vértice origem e um vértice destino, tal que a soma dos valores associados aos vértices visitados ao longo deste caminho não supere um dado limite superior.

O *ESPPCC* é um caso especial do Problema do Caminho Elementar Mínimo com Restrições de Recursos (*ESPPRC – Elementary Shortest Path Problem with Resource Constraints*). O caráter fortemente $\mathcal{NP}$-Difícil do *ESPPRC* e do *ESPPCC* foi provado, respectivamente, por Dror [18] e Kohl [34].

O primeiro método exato para a resolução do *ESPPRC* foi um método baseado em Relaxação Lagrangeana, proposto por Beasley e Christofides [6]. Recentemente, Carlyle, Royset e Wood [9] usaram uma abordagem parecida e obtiveram bons resultados para o caso em que os custos das arestas são não negativos.

Um algoritmo de correção de rótulos foi proposto por Dumitrescu [19] e Feillet et al. [20]. Baseando-se na ideia do algoritmo bidirecional de *Dijkstra* para o cálculo de um caminho mais curto [1], Righini e Salani [49] aprimoraram o algoritmo de correção de rótulos. Estes autores propuseram expandir um caminho em ambas direções, progressivamente (partindo do vértice origem) e regressivamente (partindo do vértice destino), e conectá-los no meio, potencialmente reduzindo o tempo de execução e o uso de memória. Além disso, Boland, Dethridge e Dumitrescu [7] e, independentemente, Righini e Salani [50] propuseram um algoritmo de correção de rótulos para o *ESPPRC* resolvendo, iterativamente, a sua versão não elementar (*SPPRC – Shortest Path Problem with Resource Constraints*), a qual admite algoritmos psedo-polinomiais (veja Christofides, Mingozzi e Toth [11]). A técnica usada foi o aumento gradual, ao longo de sucessivas iterações, do número de vértices que não podem ser visitados mais de uma vez. Righini e Salani [50] também propuseram um algoritmo para o *ESPPRC* que resolve a sua versão *SPPRC* e explora os limites inferiores obtidos em um algoritmo *Branch-and-Bound*.

Recentemente, Jepsen, Petersen e Spoorendonk [29] apresentaram um modelo

PLI e desigualdades válidas para o *ESPPCC* que os permitiram desenvolver um algoritmo de geração de cortes (*Branch-and-Cut*) para o mesmo. Resultados numéricos apresentados em [29] sugerem que algoritmos *Branch-and-Cut* podem ser consideravelmente melhores que algoritmos de correção de rótulos, em especial quando a instância do *ESPPCC* considerada contém muitas arestas com custo negativo.

O presente capítulo apresenta um algoritmo exato e uma heurística para o *ESPPCC*, ambos baseados em programação dinâmica, além de resultados computacionais obtidos para o *CVRP*, quando o mesmo é resolvido por uma abordagem de geração de colunas em que o *ESPPCC* é o subproblema a ser resolvido para a geração de novas colunas. Este capítulo está organizado da seguinte forma: a Seção 3.1 define o *ESPPRC* (o qual refere-se ao caso geral de caminhos mais curtos com restrições de recursos) formalmente, considera alguns conceitos relativos a algoritmos de correção de rótulos, para então apresentar uma revisão do algoritmo para o *ESPPRC* proposto por Feillet et al. [20]. A Seção 3.2 apresenta um algoritmo exato para o *ESPPCC*, adaptando o algoritmo de Feillet et al. para o caso em que a única restrição de recurso é a de capacidade. Uma heurística para o *ESPPCC* também é proposta nesta seção. Já a Seção 3.3 apresenta resultados computacionais obtidos para o *CVRP* quando o algoritmo exato e a heurística para o *ESPPCC* são usados, conjuntamente, em substituição ao mecanismo original de geração de colunas do algoritmo *Branch-and-Cut-and-Price* de Fukasawa et al. [23] (veja Capítulo 2).

3.1 O *ESPPRC*

O *ESPPRC* pode ser definido em um grafo $H = (N, E)$, onde E é o conjunto de arestas e $N = \{v_1, \ldots, v_n\}$ é o conjunto de vértices, incluindo um vértice origem s e um vértice destino t. Cada aresta $(v_i, v_j) \in E$ tem um custo $c_{ij} \in \mathbb{R}$ e uma quantidade $d_{ij}^{\ell} \geq 0$, referente ao consumo do recurso $\ell \in \{1, \ldots, L\}$, associados ao percurso da aresta. L é o total de recursos disponíveis. Assume-se que os valores d_{ij}^{ℓ} satisfazem a desigualdade triangular para cada recurso ℓ, ou seja, $d_{ij}^{\ell} < d_{ik}^{\ell} + d_{kj}^{\ell}$, $\forall\, v_i, v_j, v_k \in N$. Para cada vértice $v_i \in N$ e cada recurso ℓ, associam-se dois valores não negativos a_i^{ℓ} e b_i^{ℓ}, tal que o consumo do recurso ℓ ao longo de um caminho de s para v_i deve pertencer ao intervalo $[a_i^{\ell}, b_i^{\ell}]$. O objetivo é encontrar um caminho elementar, de custo mínimo, do vértice origem s ao vértice destino t que satisfaça todas as restrições de recursos ao longo de seu trajeto.

Quando a restrição de elementaridade é retirada (isto é, quando permite-se caminhos com ciclos) tem-se o Problema do Caminho Mínimo com Restrições de Recursos, ou *SPPRC* (veja Irnich e Desaulniers [28] para um levantamento recente sobre modelos e algoritmos para o *SPPRC*).

A seção 3.1.2 descreve o algoritmo proposto por Feillet et al. [20] para o *ESPPRC*, o qual é um algoritmo de programação dinâmica que considera somente rótulos não dominados. Antes da descrição desse algoritmo para o *ESPPRC* propriamente, algumas definições e teoremas são apresentados na seção 3.1.1.

3.1.1 Notação e princípios da correção de rótulos

Definição 3.1 *$R_i = (T_i^1, \ldots, T_i^L, V_i^1, \ldots, V_i^n, C_i)$ é um rótulo associado a um caminho X_{si} do vértice origem s ao vértice v_i. R_i é composto da quantidade de cada recurso T_i^ℓ, $1 \leq \ell \leq L$, usado ao longo do caminho, do vetor de visitação V_i^k, $1 \leq k \leq n$ ($V_i^k = 1$ se o caminho visita o vértice v_k, 0 caso contrário)[1], e de um custo C_i.*

Esta definição de rótulos é uma redefinição daquela apresentada por Desrochers [17] para o *SPPRC* e apenas acrescenta um vetor que registra os vértices já visitados pelo rótulo. A idéia de se considerar um tal vetor foi originalmente proposta por Beasley e Christofides [6], que sugeriram adicionar um recurso binário extra a cada vértice $v_i \in N$ que é consumido quando o vértice é visitado.

A partir desta definição de rótulos, definem-se, respectivamente, os conceitos de *extensão viável* e *dominância* entre rótulos:

Definição 3.2 *A extensão de um rótulo R_i para um vértice v_j é uma extensão viável se $V_i^j = 0$ e $T_i^\ell + d_{ij}^\ell \leq b_j^\ell$, para $\ell = 1, \ldots, L$ e, se realizada, dá origem ao novo rótulo $R_j = (T_i^1 + d_{ij}^1, \ldots, T_i^L + d_{ij}^L, V_i^1, \ldots, V_i^j = 1, \ldots, V_i^n, C_i + c_{ij})$.*

Definição 3.3 *Dados $X_{si}^{'}$ e $X_{si}^{''}$, dois caminhos distintos de s para v_i com rótulos associados $R_i^{'}$ e $R_i^{''}$, respectivamente, diz-se que o rótulo $R_i^{'}$ domina o rótulo $R_i^{''}$ se, e somente se, $R_i^{'} \neq R_i^{''}$, $T_i^{'\ell} \leq T_i^{''\ell}$, para $\ell = 1, \ldots, L$, $V_i^{'k} \leq V_i^{''k}$, para $k = 1, \ldots, n$, e $C_i^{'} \leq C_i^{''}$.*

A extensão viável de um rótulo R_i para um vértice v_j dá origem a um novo rótulo R_j, o qual deve ser descartado caso seja dominado por algum rótulo $R_j^{'}$ conhecido. Portanto, todos os rótulos não dominados são extensões de algum rótulo não dominado. De fato, a extensão de um rótulo dominado R_i ao vértice v_j resulta em um rótulo R_j que é dominado por algum rótulo $R_j^{'}$, obtido pela extensão de um rótulo $R_i^{'}$, que domina R_i, ao vértice v_j.

Quando o consumo de algum recurso é não negativo e obedece à desigualdade triangular, a regra de dominância entre rótulos pode ser reforçada, conforme observado por Feillet et al. [20]. Além dos vértices já visitados por um dado rótulo, pode-se

[1] Observe-se que um rótulo registra apenas os vértices já visitados e, portanto, não guarda nenhuma informação da ordem em que estes vértices foram visitados.

identificar aqueles que não podem ser visitados em qualquer extensão viável deste rótulo, devido aos limites impostos ao consumo de recursos. Estes vértices são chamados *inacessíveis*:

Definição 3.4 *Um vértice $v_j \in N$ é inacessível para um dado rótulo R_i se este rótulo já tiver visitado o vértice v_j ou se $T_i^\ell + d_{ij}^\ell > b_j^\ell$, para algum $\ell \in \{1,\ldots,L\}$.*

Este conceito leva à redefinição do conceito de rótulos:

Definição 3.5 *$R_i = (T_i^1,\ldots,T_i^L, V_i^1,\ldots,V_i^n, C_i)$ é um rótulo associado a um caminho X_{si} do vértice origem s ao vértice v_i. R_i é composto da quantidade de cada recurso T_i^ℓ, $1 \leq \ell \leq L$, usado ao longo do caminho, do vetor de vértices inacessíveis V_i^k, $1 \leq k \leq n$ ($V_i^k = 1$ se o vértice v_k é inacessível, 0 caso contrário)[2], e de um custo C_i.*

Esta nova definição de rótulos, que considera não apenas os vértices já visitados mas todos os vértices inacessíveis para um dado rótulo, potencialmente reduz o tempo de processamento do Algoritmo 3.1 (apresentado na seção 3.1.2), pois permite identificar um número maior de rótulos dominados, os quais podem ser descartados durante a execução do algoritmo, conforme o Lema 3.6 (cuja prova deve-se a Feillet et al. [20]).

Lema 3.6 *Apenas rótulos não dominados precisam ser considerados em algoritmos de correção de rótulos.*

Prova. Considere os rótulos $R_i^{'}$ e $R_i^{''}$, que representam caminhos parciais do vértice origem s ao vértice v_i, tal que $R_i^{'}$ domina $R_i^{''}$. Seja $R_j^{''}$ o rótulo obtido a partir da extensão viável do rótulo $R_i^{''}$ ao vértice v_j. A argumentação a seguir mostra que a extensão do rótulo $R_i^{'}$ ao vértice v_j é uma extensão viável e que o rótulo $R_j^{'}$, resultante desta extensão, domina o rótulo $R_j^{''}$.

O primeiro passo é mostrar que o rótulo $R_i^{'}$ pode ser estendido ao vértice v_j. Tem-se que $T_i^{'\ell} \leq T_i^{''\ell}$ e $T_j^{''\ell} \leq b_j^\ell$, para $\ell = 1,\ldots,L$. Visto que $T_j^{'\ell} = \max\{a_j^\ell, T_i^{'\ell} + d_{ij}^\ell\}$ e $T_j^{''\ell} = \max\{a_j^\ell, T_i^{''\ell} + d_{ij}^\ell\}$, então $T_j^{'\ell} \leq T_j^{''\ell} \leq b_j^\ell$, para $\ell = 1,\ldots,L$. Também tem-se que $V_i^{'k} \leq V_i^{''k}$, para $k = 1,\ldots,n$, e $V_i^{''j} = 0$, então $V_i^{'j} = 0$. Logo, a extensão do rótulo $R_i^{'}$ ao vértice v_j é uma extensão viável.

O segundo passo é mostrar que o rótulo $R_j^{'}$ domina o rótulo $R_j^{''}$. Já sabe-se que $T_j^{'\ell} \leq T_j^{''\ell}$, para $\ell = 1,\ldots,L$. A definição do vetor de vértices inacessíveis implica que $V_j^{'j} = V_j^{''j} = 1$. Também é óbvio que $C_j^{'} = C_i^{'} + c_{ij} \leq C_i^{''} + c_{ij} = C_j^{''}$. Deste modo, resta apenas verificar se $V_j^{'k} \leq V_j^{''k}$ para todos os vértices $v_k \neq v_j$. Suponha que $v_k \neq v_j$ seja um vértice inacessível para o rótulo $R_j^{'}$. Pela definição de vértices inacessíveis, v_k foi visitado por $R_j^{'}$ ou existe um recurso $\ell \in \{1,\ldots,L\}$, tal que $T_j^{'\ell} + d_{jk}^\ell > b_k^\ell$. Mas:

[2] A informação da quantidade de vértices inacessíveis $p_i = \sum_{k=1}^{n} V_i^k$ pode ser mantida em cada rótulo para a verificação eficiente de dominância.

i Se $R_j^{'}$ visitou v_k, então $R_i^{'}$ visitou v_k e $V_i^{'k} = 1$. Sabe-se também que $V_i^{'k} \leq V_i^{''k}$ e que $V_i^{''k} \leq V_j^{''k}$. Isto permite concluir que $V_j^{''k} = 1$.

ii Se existe um recurso $\ell \in \{1,\ldots,L\}$, tal que $T_j^{'\ell} + d_{jk}^{\ell} > b_k^{\ell}$, então $T_j^{''\ell} + d_{jk}^{\ell} > b_k^{\ell}$, pois $T_j^{'\ell} \leq T_j^{''\ell}$, para $\ell = 1,\ldots,L$.

Assim, se um vértice $v_k \neq v_j$ é inacessível para o rótulo $R_j^{'}$, ele também é inacessível para o rótulo $R_j^{''}$, isto é, $V_j^{'k} \leq V_j^{''k}$ para todo vértice $v_k \neq v_j$. Portanto, conclui-se que o rótulo $R^{'}$ domina o rótulo $R^{''}$. □

3.1.2 Um algoritmo de correção de rótulos para o *ESPPRC*

Em algoritmos de correção de rótulos, tais como aqueles de Desrochers [17] e Feillet et al. [20], rótulos são explorados de acordo com os vértices que eles estão associados. Todos os vértices são ciclicamente visitados e, para cada vértice, o algoritmo estende todos os rótulos que ainda não foram estendidos, gerando novos rótulos. Esta operação é repetida até que todos os possíveis rótulos tenham sido estendidos em todas as formas viáveis. Nesse processo, somente rótulos não dominados são mantidos. Observe-se que rótulos associados a um mesmo vértice podem ser classificados de acordo com um critério secundário como, por exemplo, de acordo com o custo ou com o consumo de um certo recurso. Classificar rótulos dessa forma é útil para a verificação eficiente de dominância.

O Algoritmo 3.1, proposto por Feillet et al. [20] e revisto a seguir, determina todos os rótulos não-dominados que representam caminhos que iniciam no vértice s e alcançam todos os vértices do grafo (um caminho $s \rightsquigarrow t$ ótimo é dado por um rótulo R_t de custo mínimo). A notação usada no algoritmo é:

Γ_i : Conjunto de rótulos associados ao vértice v_i;
Δ_i : Conjunto de vértices sucessores do vértice v_i;
Y : Conjunto de vértices ainda não tratados;
$Estende(R_i, v_j)$: Procedimento que retorna o rótulo R_j resultante da extensão do rótulo $R_i \in \Gamma_i$ ao vértice v_j, caso esta extensão seja viável. Esse procedimento também reconhece e registra os vértices inacessíveis ao novo rótulo gerado; e
$RND(\Gamma_i, R_i)$: Procedimento que insere o rótulo R_i no conjunto Γ_i aplicando as regras de dominância entre rótulos.

Algoritmo 3.1: $Espprc(H,L,s,t)-$ DP

Entrada: Instância do ESPPRC.

Saída: Todos os rótulos não-dominados.

```
/* Inicialização. */
Γ_s ← {(0,0,0)};
para cada v_i ∈ N \ {s} faça
    Γ_i ← ∅;
Y ← {s};

repita
    /* Explorando os sucessores de um vértice. */
    v_i ← v ∈ Y;
    para cada R_i = (T_i^1,...,T_i^L,V_i^1,...,V_i^n,C_i) ∈ Γ_i faça
        para cada v_j ∈ Δ_i faça
            se V_i^j = 0 então
                R_j ← Estende(R_i, v_j);
                Γ'_j ← Γ_j;
                Γ_j ← RND(Γ_j, R_j);
                se (Γ_j ≠ Γ'_j) então
                    Y ← Y ∪ {v_j};

    /* Redução de Y. */
    Y ← Y \ {v_i};
até Y = ∅ ;

retorna Γ;
```

O Algoritmo 3.1 foi originalmente apresentado em [20], porém uma análise de complexidade precisa para este algoritmo não foi feita no referido trabalho. Contudo, Feillet et al. ressaltam que esta complexidade é fortemente dependente da instância do *ESPPRC* que se pretende resolver, isto é, do número de vértices, da estrutura do grafo e das restrições de recursos associadas a esta instância. Observou-se, nos experimentos descritos na Seção 3.3, que algoritmos de correção de rótulos são sensíveis aos custos associados às arestas do grafo subjacente, em particular, quando tal grafo contém uma significativa quantidade de arestas com custo negativo associado.

3.2 O *ESPPCC*

O *ESPPCC* pode ser definido por um grafo não orientado $H = (N,E)$, onde E é o conjunto de arestas e $N = \{v_1,\ldots,v_n\}$ é o conjunto de vértices, incluindo um vértice

origem s e um vértice destino t. Cada aresta $(v_i, v_j) \in E$ tem um custo $c_{ij} \in \mathbb{R}$ associado e cada vértice $v_i \in N^+$, onde $N^+ = N \backslash \{s,t\}$, tem associado uma demanda d_i, $d_i \in \mathbb{N}^*$ (define-se $d_s = d_t = 0$). O objetivo é encontrar um caminho elementar de custo mínimo, do vértice origem s ao vértice destino t, tal que a soma das demandas dos vértices visitados ao longo deste caminho não supere um limite superior Q, $Q \in \mathbb{N}^*$, dado[3].

Suponha que uma instância do *ESPPCC* seja definida em um grafo $H = (N,E)$ tal como descrito. É fácil notar que a instância do *ESPPRC* correspondente pode ser definida sobre um grafo $H' = (N',E')$ idêntico ao grafo H. Neste caso, esta instância do *ESPPRC* possui uma única restrição de recurso, que pode ser representada definindo-se intervalos $[d_i, Q]$ em cada vértice $v_i \in N'$. A quantidade de demanda consumida ao se percorrer a aresta $(v_i, v_j) \in E'$ é a demanda d_j associada ao vértice $v_j \in N$. Note-se que são consumidas $d_k + d_j$ unidades de demanda ao se percorrer as arestas $(v_i, v_k) \in E'$ e $(v_k, v_j) \in E'$ em sequência. Portanto, a desigualdade triangular é satisfeita para este recurso, pois $d_j < d_k + d_j$, já que $d_k > 0, \forall\, v_k \in N^+$.

Sendo o *ESPPCC* um caso especial do *ESPPRC*, a Definição 3.5 pode ser adaptada de acordo:

Definição 3.7 *$R_i = (D_i, V_i^1, \ldots, V_i^n, C_i)$ é um rótulo associado a um caminho X_{si} do vértice origem s ao vértice v_i. R_i é composto da quantidade de demanda D_i consumida ao longo do caminho, do vetor de vértices inacessíveis V_i^k, $1 \leq k \leq n$ ($V_i^k = 1$ se o vértice v_k é inacessível, 0 caso contrário), e de um custo C_i.*

Então, os conceitos de extensão viável e dominância entre rótulos podem ser reescritos:

Definição 3.8 *A extensão de um rótulo R_i para um vértice v_j é uma extensão viável se $V_i^j = 0$ e $D_i + d_j \leq Q$ e, se realizada, dá origem ao novo rótulo $R_j = (D_i + d_j, V_i^1, \ldots, V_i^j = 1, \ldots, V_i^n, C_i + c_{ij})$.*

Definição 3.9 *Dados X_{si}' e X_{si}'', dois caminhos distintos de s para v_i com rótulos associados R_i' e R_i'', respectivamente, diz-se que o rótulo R_i' domina o rótulo R_i'' se, e somente se[4], $R_i' \neq R_i''$, $D_i' \leq D_i''$, $V_i'^k \leq V_i''^k$, para $k = 1, \ldots, n$, e $C_i' \leq C_i''$.*

Uma formulação PLI para o *ESPPCC*

Embora algoritmos de correção de rótulos[5] sejam o método de solução exato dominante para o *ESPPCC*, é interessante considerar o recente modelo PLI, proposto

[3] Quando a restrição de elementaridade é retirada, tem-se um *SPPCC – Shortest Path Problem with a Capacity Constraint.*

[4] Observe-se que é suficiente verificar se $V_i'^k \leq V_i''^k$, para $k = 1, \ldots, n$, e $C_i' \leq C_i''$.

[5] Todos os algoritmos apresentados para o *ESPPCC* nesta dissertação são algoritmos de correção de rótulos baseados em programação dinâmica.

por Jepsen, Petersen e Spoorendonk [29], sobre o qual um algoritmo *Branch-and-Cut* foi desenvolvido para o mesmo.

Na formulação seguinte para o *ESPPCC*, as variáveis y_i e x_e indicam, respectivamente, o uso do vértice $v_i \in N$ e da aresta $e = (v_i, v_j)$, $e \in E$. Para um conjunto de vértices $S \subseteq N$ e para um conjunto de arestas $T \subseteq E$, sejam $y(S) = \sum_{v_i \in S} y_i$ e $x(T) = \sum_{e \in T} x_e$. Além disso, seja $\delta(S)$ o conjunto de arestas que possuem exatamente um extremo dentro do conjunto S, isto é, $\delta(S) = \{(v_i, v_j) : v_i \in S \wedge v_j \in N \backslash S\}$ e $\lambda(S)$ o conjunto de arestas que possuem ambos os extremos dentro do conjunto S, isto é, $\lambda(S) = \{(v_i, v_j) : v_i \in S \wedge v_j \in S\}$. Define-se, então, o politopo P em $\mathbb{Z}^{|E|+|N-2|}$:

$$
P = \begin{cases}
x(\delta(s)) = 1 & & (1)\\
x(\delta(t)) = 1 & & (2)\\
\sum_{e \in \delta(v_i)} x_e = 2y_i & \forall\ v_i \in N^+ & (3)\\
\sum_{v_i \in N} d_i y_i \leq Q & & (4)\\
x(\lambda(S)) \leq y(S) - y_i & \forall\ v_i \in S, \forall\ S \subseteq N, |S| \geq 2 & (5)\\
x_e \in \{0,1\} & \forall\ e \in E & (6)\\
y_i \in \{0,1\} & \forall\ v_i \in N & (7)
\end{cases}
$$

As restrições (1) e (2) garantem que o caminho inicia no vértice s e termina no vértice t. As restrições (3) interligam as variáveis x e y. A restrição (4) impõe a restrição de capacidade. As restrições (5) são as restrições de eliminação de subrotas generalizada. As restrições (3), juntamente com as restrições (5), impõem conectividade ao caminho. Finalmente, as restrições (6) e (7) indicam o uso de arestas e vértices. Logo, uma solução ótima para o *ESPPCC* é dada pelo valor:

$$C^* = \min_{x \in P} \sum_{e \in E} c_e x_e \,.$$

3.2.1 Um algoritmo exato para o *ESPPCC*

O primeiro algoritmo proposto para a resolução exata do *ESPPCC* é uma adaptação do Algoritmo 3.1, o qual resolve o problema geral de caminhos mais curtos com restrições de recursos. Este algoritmo foi adaptado para o caso em que a única restrição de recurso é a de capacidade. Esta adaptação faz uso de uma aceleração que elimina rótulos sub-ótimos, isto é, rótulos não dominados que se sabe não fazer parte de uma solução ótima.

A estrutura de dados principal utilizada no algoritmo é uma matriz M de dimensão $(Q+1) \times (n-2)$. Cada entrada $M(d,i)$ desta matriz contém um repositório que armazena todos os possíveis rótulos não dominados dos caminhos elementares parciais que

partem de s e terminam no vértice $v_i \in N^+$, tendo consumido uma demanda d, $d_i \leq d \leq Q$. Cada rótulo do repositório $M(d,i)$ mantém um vetor que registra quais vértices são inacessíveis e o custo do caminho parcial que ele representa. Além disso, o rótulo contém agora duas informações adicionais: o vértice v_i ao qual ele está associado e um apontador para o rótulo que ele estendeu. Estas duas novas informações são utilizadas para construir uma solução ótima obtida pelo algoritmo, isto é, para obter a ordem em que os vértices são visitados nesta solução de menor custo.

A matriz M é preenchida, linha a linha, começando com $d = 0$ e indo até $d = Q$. Para cada rótulo R_i, da entrada $M(d,i)$, o algoritmo percorre cada vizinho $v_j \in N^+$ de v_i e avalia a extensão do rótulo R_i ao vértice v_j. Se a extensão do rótulo R_i ao vértice v_j for uma extensão viável (isto é, se $V_i^j = 0$ e $d + d_j \leq Q$) e se o novo rótulo R_j, resultante da extensão de R_i ao vértice v_j, não for dominado por nenhum rótulo pertencente a qualquer um dos repositórios $M(d_j, j), M(d_j + 1, j), \ldots, M(d + d_j, j)$, a extensão é realizada e o novo rótulo R_j é guardado no repositório $M(d + d_j, j)$ (neste caso, o vetor de vértices inacessíveis $\mathbf{V}_j$ deve ser atualizado). É possível que o rótulo R_j domine algum rótulo $R_j^{'}$ presente no repositório $M(d + d_j, j)$. Caso isto ocorra, $R_j^{'}$ deve ser descartado.

Quando a matrix M está preenchida, todos os rótulos[6] obtidos devem ser estendidos até o vértice destino t para se obter caminhos completos $s \rightsquigarrow t$. Uma solução ótima pode ser obtida por um rótulo R_t de menor custo.

Para facilitar o entendimento de como extensões de rótulos são realizadas, considere os seguintes exemplos:

i Sejam os rótulos R_j, armazenado no repositório $M(d + d_j, j)$, e R_i, armazenado no repositório $M(d,i)$, que representam os caminhos $X_{sj} : s \to u_1 \to v_i \to u_2 \to u_3 \to v_j$ e $X_{si} : s \to u_3 \to u_1 \to u_2 \to v_i$, respectivamente. Suponha que deseja-se estender o rótulo R_i ao vértice v_j, criando o novo rótulo $R_j^{'}$ para representar o novo caminho $X_{sj}^{'} : s \to u_3 \to u_1 \to u_2 \to v_i \to v_j$. O vetor de vértices inacessíveis deste novo rótulo $R_j^{'}$ será idêntico ao vetor de vértices inacessíveis do rótulo R_j. Portanto, essa extensão só será feita se $C_i + c_{ij} < C_j$ e então o novo rótulo $R_j^{'}$ substituirá o rótulo R_j no repositório $M(d + d_j, j)$.

ii Seja o rótulo R_i, armazenado no repositório $M(d,i)$, que representa o caminho $X_{si} : s \to u_1 \to u_2 \to u_3 \to u_4 \to u_5 \to u_6 \to u_7 \to v_i$ e suponha que deseja-se estender R_i ao vértice v_j, criando o novo rótulo $R_j^{'}$, a ser armazenado no repositório $M(d + d_j, j)$, para representar o novo caminho $X_{sj}^{'} : s \to u_1 \to u_2 \to u_3 \to u_4 \to u_5 \to u_6 \to u_7 \to v_i \to v_j$. Porém, suponha que já exista um rótulo R_j, que representa o caminho $X_{sj} : s \to u_4 \to u_2 \to u_3 \to v_j$, armazenado no repositório $M(q, j)$, onde $q = d_{u_4} + d_{u_2} + d_{u_3} + d_j$. Neste

[6] Lembre-se que tais rótulos representam caminhos elementares *parciais* que partem de s e terminam no vértice $v_i \in N^+$, tendo consumido uma demanda d, $d_i \leq d \leq Q$.

caso, todos os vértices visitados pelo rótulo R_j também terão sido visitados pelo novo rótulo $R_j^{'}$ (isso implicará que todos os vértices inacessíveis para R_j também serão inacessíveis para $R_j^{'}$). Portanto, essa extensão só será feita se $C_i + c_{ij} < C_j$ e então o novo rótulo $R_j^{'}$ será armazenado no repositório $M(d + d_j, j)$ (o rótulo R_j será mantido no repositório $M(q, j)$ pois não haverá relação de dominância entre ambos os rótulos R_j e $R_j^{'}$).

A aceleração

Suponha que a extensão de um rótulo não dominado R_i a um vértice $v_j \in N^+$ seja viável e resulta em um rótulo não dominado R_j. Sabe-se que uma solução ótima só pode ser obtida a partir de extensões de rótulos não dominados (veja o Lema 3.6). Portanto, o rótulo R_j deve ser considerado, a menos que fosse possível garantir que o mesmo não faz parte de uma solução procurada. Porém, suponha que uma demanda d, $d_i \leq d \leq Q$, tenha sido consumida ao longo do caminho X_{si}, representado pelo rótulo R_i, e que se conheça um limite inferior $\overline{C_{jt}}$ para o custo de um caminho elementar ótimo que inicia no vértice v_j e termina no vértice t, consumindo uma demanda total dentro do intervalo $[d_j, Q - d]$. Além disso, suponha que se conheça um limite superior U para o custo de uma solução ótima[7]. Então, R_i só será estendido ao vértice v_j se $C_i + c_{ij} + \overline{C_{jt}} \leq U$.

Os limites inferiores podem ser obtidos por um algoritmo que retorna, para cada vértice $v_i \in N^+$ e para cada demanda $d = d_i, \ldots, Q$, o custo de um caminho ótimo que parte do vértice v_i e alcança o vértice t, tendo consumido uma demanda dentro do intervalo $[d_i, d]$. Só que agora permite-se caminhos com ciclos, isto é, um vértice pode ser visitado mais de uma vez e cada vez toda a sua demanda é novamente consumida[8] (observe-se que a permissão de caminhos com ciclos é uma relaxação do problema em estudo, onde somente caminhos elementares são permitidos). Uma forma natural de obter bons limites inferiores é encontrar caminhos sem δ-ciclos, isto é, caminhos que não possuem ciclos com uma quantidade de arestas menor ou igual a δ. O caso geral da eliminação de δ-ciclos é descrito por Irnich e Villeneuve [27]. Nesta dissertação optou-se por eliminar apenas 2-ciclos.

[7] Quando o algoritmo para o *ESPPCC* é usado como um gerador de caminhos elementares $s \rightsquigarrow t$ (colunas) em um algoritmo de geração de colunas, em geral apenas colunas de custo reduzido negativo precisam ser consideradas. Neste contexto, define-se $U = 0$.

[8] Este problema pode ser resolvido em tempo pseudo-polinomial se todas as demandas forem inteiras (veja Christofides, Mingozzi e Toth [11]).

3.2.2 Uma heurística para o *ESPPCC*

Novamente, usa-se uma matriz M, de dimensão $(Q+1)\times(n-2)$, em que cada entrada $M(d,i)$ contém um repositório *limitado*, isto é, um repositório que armazena apenas uma quantidade fixa de rótulos de caminhos elementares parciais que partem de s e terminam no vértice $v_i \in N^+$, tendo consumido uma demanda igual a d, $d_i \leq d \leq Q$. A matriz M é preenchida, linha a linha, começando com $d = 0$ e indo até $d = Q$. Para cada entrada $M(d+d_j, j)$, escolhe-se rótulos pertencentes a qualquer um dos repositórios $M(d,i)$ que podem ser estendidos até v_j. Esses rótulos escolhidos devem ser os rótulos que, quando estendidos até v_j, resultem nos rótulos R_j de menor custo, dentre todos os rótulos que podem ser estendidos até v_j. Conforme descrito anteriormente, um rótulo R_i pertencente ao repositório $M(d,i)$ só pode ser estendido até um vértice v_j se esta extensão for viável (isto é, se $V_i^j = 0$ e $d + d_j \leq Q$) e se o rótulo R_j, resultante desta extensão, não for dominado por nenhum rótulo pertencente a qualquer um dos repositórios $M(d_j, j), M(d_j+1, j), \ldots, M(d+d_j, j)$ (na verdade, considerou-se o teste de dominância somente para o repositório $M(d+d_j, j)$). Note-se que pode haver perda de rótulos durante a execução do algoritmo pois cada entrada $M(d,i)$ armazena apenas uma pequena quantidade de rótulos.

É intuitiva a ideia de se escolher os rótulos de menor custo, dentre um conjunto de rótulos que podem ser estendidos até v_j, pois supõe-se que a probabilidade de um desses rótulos fazer parte de uma solução ótima no final do processo é maior que a dos demais rótulos de custo superior descartados. Um vetor B de tamanho $(n-2)$ é usado para armazenar os melhores resultados obtidos até o momento. No final, o elemento $B(i)$, para cada vértice $v_i \in N^+$, armazena o rótulo que representa o caminho mais curto, encontrado pelo algoritmo, que parte de s e alcança o vértice v_i, tendo consumido uma demanda total dentro do intervalo $[d_i, Q]$. Para se obter os caminhos completos $s \rightsquigarrow t$, todos os rótulos em B são estendidos até o vértice t (note-se que todas as soluções encontradas pela heurística proposta são viáveis, isto é, apenas caminhos elementares $s \rightsquigarrow t$ que consomem, no máximo, uma demanda total igual a Q, são obtidos). Na prática, verificou-se que este algoritmo obtém boas soluções e que o mesmo executa em um tempo razoavelmente baixo quando apenas uma pequena quantidade de rótulos (tal quantidade pode ser passada como parâmetro de entrada para a heurística) é armazenada em cada entrada $M(d,i)$ da matriz M (testou-se para quantidades menores ou iguais a 25).

3.3 Testes computacionais

Quando um algoritmo exato para o *ESPPCC* é usado como um gerador de colunas (rotas elementares) em um algoritmo *Branch-and-Price* (ou em um algoritmo

Branch-and-Cut-and-Price), o mesmo, geralmente, necessita ser executado repetidas vezes. Como o *ESPPCC* é um problema fortemente $\mathcal{NP}$-Difícil (veja Kohl [34]), é importante, por razões de desempenho, que o algoritmo exato seja executado poucas vezes durante o processo iterativo da geração de colunas (não há necessidade de se resolver o *ESPPCC* à otimalidade nas primeiras iterações da geração de colunas). Uma estratégia para diminuir o número de execuções do algoritmo exato é usar heurísticas que conseguem obter colunas de custo reduzido negativo rapidamente. Neste caso, o algoritmo exato só é executado quando a heurística falha em encontrar tais colunas.

Assim, o algoritmo exato e a heurística[9] propostos para o *ESPPCC* nas Seções 3.2.1 e 3.2.2, respectivamente, foram usados, conjuntamente, para a geração de colunas no algoritmo *Branch-and-Cut-and-Price* de Fukasawa et al. [23]. Os resultados obtidos (para algumas instâncias do *CVRP*, das séries A, B, E e P, disponíveis em [48]) por esse novo algoritmo *BCP* são fornecidos na Tabela A.1 e comparados com os obtidos pelo algoritmo *BCP* robusto original, onde rotas não elementares sem δ-ciclos (para $\delta = 3$ e $\delta = 4$) são usados na geração de colunas.

As colunas L_1, N_1 e T_1 apresentam, respectivamente, os limites inferiores obtidos no nó raiz, o número total de nós explorados na árvore *Branch-and-Bound*, e o tempo total (em segundos) para resolver a instância à otimalidade com o algoritmo *BCP* que implementa o subproblema de geração de colunas como um *ESPPCC*. Já as colunas L_4, N_4 e T_4 mostram os valores dos limites inferiores, o número de nós, e o tempo gasto para resolver a instância com o algoritmo *BCP* que implementa o subproblema de geração de colunas como um *SPPCC*, quando 3-ciclos são eliminados. Analogamente, as colunas L_5, N_5 e T_5, fornecem os valores quando 4-ciclos não são permitidos. Já a coluna C^* apresenta o valor ótimo das instâncias.

Os limites inferiores obtidos no nó raiz na versão que considera o subproblema de geração de colunas como um *ESPPCC*, em geral, são razoavelmente melhores que os obtidos pela versão relaxada. Tome-se, por exemplo, a instância A-n37-k6, onde chega a 5.287 unidades a diferença entre os limites obtidos com a geração de rotas elementares e com a eliminação de 4-ciclos. Neste caso o *gap* é reduzido em 37%. De fato, os *gaps* entre os primeiros limites e os valores das soluções ótimas são sempre inferiores (ou iguais) àqueles calculados com os limites obtidos com o algoritmo original. Além disso, em alguns casos uma solução ótima é encontrada no nó raiz apenas na nova versão, como ocorre para as instâncias B-n45-k5, B-n45-k6, B-n50-k7, B-n57-k7 e P-n40-k5. Contudo, ressalte-se que algumas instâncias de grau de dificuldade moderado, tais como as instâncias E-n76-k8, E-n76-k10, E-n76-k14 e P-n70-k10, a redução do *gap* com o uso

[9] Lembre-se que a quantidade de rótulos tratados pela heurística proposta na Seção 3.2.2 pode ser adaptada dinamicamente, tornando-a, progressivamente, mais eficaz na tarefa de encontrar colunas de custo reduzido negativo.

de rotas elementares é inferior a 6% (com relação a eliminação de 4-ciclos).

Com relação ao tempo de processamento para encontrar uma solução ótima, algumas instâncias obtiveram uma redução significativa quando rotas elementares são usadas na geração de colunas, destacando-se as instâncias A-n60-k9, A-n63-k10, A-n69-k9, B-n50-k8, E-n76-k14, E-n101-k14 e P-n70-k10. Isso se deve, em parte, ao aumento obtido no limite inferior e a consequente redução do número de nós explorados na árvore *Branch-and-Bound*. Note-se que ao usar rotas elementares, os ganhos nos limites inferiores refletem-se não apenas no nó raiz, mas em todos os nós, pois para se obter um limite inferior válido, o novo algoritmo *BCP* deve usar somente rotas elementares na geração de colunas de cada nó da árvore (isso não é, necessariamente, uma vantagem, já que um subproblema fortemente $\mathcal{NP}$-Difícil deve ser resolvido à otimalidade pelo menos uma vez em cada nó e, se o mesmo tornar-se intratável, o novo algoritmo *BCP* como um todo pode falhar). Em contrapartida, para algumas instâncias (em especial, as instâncias A-n62-k8, B-n56-k7, B-n67-k10, e B-n78-k10), o ganho obtido no limite inferior com o uso de rotas elementares não compensa o aumento no tempo da geração de colunas.

CAPÍTULO 4

Algoritmos aprimorados para o *ESPPCC*

O algoritmo de programação dinâmica para o *ESPPRC* proposto por Feillet et al. [20] (veja Capítulo 3) gera uma quantidade de rótulos (representação de caminhos parciais) que aumenta rapidamente com o tamanho da instância do problema em questão. Cada rótulo pode gerar uma quantidade de novos rótulos tão grande quanto a quantidade de sucessores do vértice ao qual ele está associado. Portanto, no pior caso, o número de rótulos cresce exponencialmente com o número de arestas no caminho (somente rótulos dominados são descartados).

Visando melhorar o desempenho do algoritmo de Feillet et al., Righini e Salani [49] propuseram um algoritmo de programação dinâmica que combina uma busca bidirecional com as restrições de recursos do *ESPPRC* (os autores chamaram esta técnica de *BBDP – Bounded Bi-directional Dynamic Programming*). Programação dinâmica bidirecional mostrou ser uma técnica capaz de acelerar o algoritmo de *Dijkstra*, no cálculo de um caminho mais curto, entre um par de vértices de um grafo direcionado com arcos de custo não negativo [1]. No caso do *ESPPRC*, os caminhos são estendidos tanto progressivamente, a partir do vértice origem, quanto regressivamente, a partir do vértice destino. Para evitar que um mesmo caminho seja gerado duas vezes, procura-se limitar o tamanho dos caminhos em ambas direções (progressiva e regressiva). Para tanto, elege-se um dado recurso, cujo consumo é monotônico ao longo das extensões, e caminhos que já tenham consumido pelo menos a metade deste recurso não são mais estendidos. Ao final, combinam-se os caminhos parciais para se obter caminhos completos. Como o número de rótulos cresce exponencialmente com o tamanho dos caminhos, reduzir o tamanho destes pela metade faz com que o número de rótulos gerados caia, potencialmente, por um fator exponencial.

Recentemente, Righini e Salani [50] aprimoraram ainda mais os resultados para o *ESPPRC* propondo um algoritmo iterativo que procura diminuir o espaço de relaxação para o *SPPRC* aumentando, ao longo das iterações, o número de vértices que não podem ser visitados mais de uma vez. Em uma dada iteração, uma solução ótima para o *SPPRC* não incluirá ciclos e será, portanto, uma solução ótima para o *ESPPRC* (os autores

chamaram esta técnica de *DSSR – Decremental State Space Relaxation*)[1]. Como a versão não elementar tem complexidade pseudo-polinomial, esta técnica mostrou-se satisfatória quando um caminho elementar pode ser encontrado restringindo-se múltiplas visitas para apenas uma pequena percentagem da quantidade de vértices da instância do *ESPPRC*.

Este capítulo apresenta recentes algoritmos de correção de rótulos propostos para o *ESPPRC* e os adaptam para resolver o *ESPPCC*, no qual há uma única restrição de recurso, a restrição de capacidade. Esses novos algoritmos para o *ESPPCC* são embutidos em um mecanismo de geração de colunas de um algoritmo *Branch-and-Cut-and-Price* (*BCP*) para o *CVRP*, tal como no Capítulo 3. A organização deste capítulo dá-se conforme a seguir: a Seção 4.1 apresenta um algoritmo para o *ESPPRC* baseado em programação dinâmica bidirecional. A Seção 4.2 discorre sobre a técnica de relaxação decrescente do espaço de estados e mostra como esta pode ser empregada na construção de um algoritmo para o *ESPPRC*. Já a seção 4.3 apresenta novos algoritmos de programação dinâmica para o *ESPPCC*, os quais são adaptações dos algoritmos apresentados nas seções anteriores. Finalmente, a Seção 4.4 apresenta resultados obtidos quando os novos algoritmos para o *ESPPCC* são usados para a geração de colunas do algoritmo *BCP* de Fukasawa et al. [23] (veja Capítulo 2), substituindo o mecanismo original.

O presente capítulo segue as mesmas definições usados no Capítulo 3. Mais especificamente, rótulos seguem a Definição 3.5, extensão viável a Definição 3.2, dominância entre rótulos a Definição 3.3 e vértices inacessíveis a Definição 3.4. Além disso, o *ESPPRC* segue a definição dada na Seção 3.1 e o *ESPPCC* a dada na Seção 3.2.

4.1 Programação dinâmica bidirecional

Em programação dinâmica bidirecional, rótulos são estendidos tanto progressivamente, do vértice origem s para seus sucessores, quanto regressivamente, do vértice destino t para seus predecessores (uma aplicação desta ideia para o *TSP* com restrições pode ser encontrada em [43]). No *ESPPRC*, para cada caminho X_{si} do vértice origem s para o vértice v_i associa-se um rótulo R_i^{pg}, assim como para cada caminho X_{it} do vértice v_i para o vértice destino t associa-se um rótulo R_i^{rg}. Um caminho de s para t é obtido cada vez que a união entre um rótulo progressivo R_i^{pg} e um rótulo regressivo R_j^{pg} for uma *união viável*, isto é, quando o conjunto de vértices visitados por R_i^{pg} for disjunto ao conjunto de vértices visitados por R_j^{rg} e o consumo total de cada recurso ao longo do caminho completo $s \rightsquigarrow t$, resultante desta união, não exceder a quantidade disponível para este recurso.

[1] Righini e Salani [50] mostram que as técnicas de programação dinâmica bidirecional e relaxação decrescente do espaço de estados podem ser aplicadas, conjuntamente, em um algoritmo para o *ESPPRC*.

Como rótulos progressivos e regressivos gerados por um algoritmo de programação dinâmica bidirecional são temporariamente unidos (a fim de se obter caminhos completos $s \rightsquigarrow t$), é crucial reduzir o número de uniões destes rótulos o quanto possível (sem essa limitação, o algoritmo bidirecional produziria o dobro de rótulos se comparado a um algoritmo mono-direcional, tal como o de Feillet et al. [20]). A ideia é que pode-se parar de estender um caminho em uma direção, quando tem-se a garantia de que a parte restante deste caminho será gerada na direção oposta e, portanto, nenhuma solução ótima será perdida. Para este propósito, Righini e Salani [49] sugerem a escolha de um *recurso crítico*, cujo consumo ao longo de caminhos é monotônico, e então rótulos não são estendidos se pelo menos metade da quantidade disponível deste recurso já tiver sido consumida (o critério de parada requer que um consumo positivo do recurso crítico esteja associado a cada aresta). Isto permite reduzir drasticamente o número de rótulos gerados e ainda garante que uma solução ótima será encontrada.

4.1.1 Um algoritmo de programação dinâmica bidirecional

O Algoritmo 4.1, proposto por Righini e Salani [49] para o *ESPPRC* e revisto a seguir, é um algoritmo de correção de rótulos, tal como o Algoritmo 3.1, mas agora ambos os rótulos progressivos e regressivos associados a um dado vértice são estendidos quando o mesmo é examinado. Novamente, todos os vértices são ciclicamente visitados e, para cada vértice, o algoritmo estende todos os rótulos que ainda não foram estendidos. Esta operação é repetida até que todos os possíveis rótulos tenham sido estendidos em todas as formas viáveis. Porém, para reduzir o número de extensões (e, portanto, a quantidade de rótulos gerados), antes do algoritmo iniciar a propagação de rótulos, um recurso crítico é escolhido e rótulos não são estendidos se metade deste recurso tiver sido consumido. No final, os rótulos obtidos são unidos um-a-um, permitindo-se obter caminhos completos $s \rightsquigarrow t$. Durante a execução do algoritmo, somente os rótulos não-dominados são considerados. A notação usada no algoritmo é:

Γ_i^{pg} e Γ_i^{rg} : Conjunto de rótulos progressivos e regressivos associados ao vértice v_i;
Δ_i^{+} e Δ_i^{-} : Conjunto de vértices sucessores e predecessores do vértice v_i.
Y : Conjunto de vértices ainda não tratados;
$Estende(R_i, v_j)$: Procedimento que retorna o rótulo R_j resultante da extensão do rótulo $R_i \in \Gamma_i$ ao vértice v_j, caso esta extensão seja viável (também reconhece e registra os vértices inacessíveis ao novo rótulo gerado);
$RND(\Gamma_i, R_i)$: Procedimento que insere o rótulo R_i no conjunto Γ_i aplicando as regras de dominância entre rótulos;
$Viavel(R_i, R_j)$: Procedimento que verifica se a união entre o rótulo R_i e o rótulo R_j é uma união viável; e
$Salva(R_i, R_j)$: Procedimento que registra a solução obtida ao unir os rótulos R_i e R_j.

Algoritmo 4.1: $Espprc(H,L,s,t)-$ BBDP

```
   /* Inicialização. */
1  Γ_s^pg ← Γ_t^rg ← {(0,0,0)};
2  para cada v_i ∈ N \ {s} faça
3      Γ_i^pg ← ∅;
4  para cada v_i ∈ N \ {t} faça
5      Γ_i^rg ← ∅;
6  Y ← {s,t};
7  repita
8      v_i ← v ∈ Y;
       /* Extensão progressiva. */
9      para cada R_i = (T_i,V_i,C_i) ∈ Γ_i^pg faça
10         para cada v_j ∈ Δ_i^+ faça
11             se V_i^j = 0 então
12                 R_j ← Estende(R_i,v_j);
13                 Γ_j^pg ← RND(Γ_j^pg,R_j);
14                 se Γ_j^pg mudou então
15                     Y ← Y ∪ {v_j};
       /* Extensão regressiva. */
16     para cada R_i = (T_i,V_i,C_i) ∈ Γ_i^rg faça
17         para cada v_k ∈ Δ_i^- faça
18             se V_i^k = 0 então
19                 R_k ← Estende(R_i,v_k);
20                 Γ_k^rg ← RND(Γ_k^rg,R_k);
21                 se Γ_k^rg mudou então
22                     Y ← Y ∪ {v_k};
23     Y ← Y \ {v_i};
24 até Y = ∅ ;
   /* União de rótulos. */
25 para cada v_i ∈ N faça
26     para cada R_i ∈ Γ_i^pg faça
27         para cada v_j ∈ N faça
28             para cada R_j ∈ Γ_j^rg faça
29                 se Viavel(R_i,R_j) então
30                     Salva(R_i,R_j);
```

Nas linhas de 1 a 6 têm-se a inicialização dos conjuntos de rótulos associados a cada um dos vértices $v_i \in N$; somente os vértices origem e destino são declarados como não tratados. Enquanto houver vértices a serem tratados (laço que vai da linha 7 a linha 24), escolhe-se um vértice qualquer $v_i \in Y$ e estende-se todos os rótulos (progressivos e regressivos) a ele associados (extensões progressivas, linhas 9 a 15, e regressivas, linhas 16 a 22, são análogas) para então declará-lo como tratado (linha 23). Extensões de rótulos criam novos rótulos, os quais devem ser inseridos no conjunto apropriado. Se um novo rótulo gerado (linhas 12 e 19) for um rótulo não dominado, o vértice ao qual ele está associado deve entrar na lista de vértices não tratados (linhas 15 e 22). Após a fase de extensão de rótulos, tem-se a fase de união (linhas 25 a 30), onde todos os rótulos progressivos são unidos (de forma viável) a todos os rótulos regressivos, a fim de se obter caminhos $s \rightsquigarrow t$. Uma solução ótima é dada por um caminho $s \rightsquigarrow t$ de menor custo.

Resultados computacionais apresentados em [49] sugerem que este novo algoritmo para o *ESPPRC* melhora significativamente o desempenho do algoritmo básico proposto por Feillet et al. [20].

4.2 Relaxação decrescente do espaço de estados

A técnica de relaxação do espaço de estados foi proposta por Christofides, Mingozzi e Toth [12] no contexto de problemas de roteamento. Para este fim, o espaço de estados $\mathcal{S}$, explorado pelo algoritmo de programação dinâmica, é projetado em um espaço $\mathcal{T}$, de dimensão inferior, tal que cada estado em $\mathcal{T}$ mantém o custo mínimo entre aqueles de seus estados correspondentes em $\mathcal{S}$ (assumindo que a função objetivo é de minimização). Dessa forma, o número de estados que devem ser explorados é reduzido drasticamente; a desvantagem é que algum estado original, correspondente a uma solução não viável em $\mathcal{S}$, pode ser projetado em um estado correspondente a uma solução viável em $\mathcal{T}$. Consequentemente, a busca em $\mathcal{T}$ não garante encontrar uma solução ótima em $\mathcal{S}$, e sim um limite inferior para o custo desta solução.

Baseando-se nessa técnica, Righini e Salani [50] propuseram identificar alguns vértices como *críticos*, de acordo com a estrutura da solução ótima para o *SPPRC*, obtida com a relaxação do espaço de estados. Seja Θ o conjunto de vértices críticos da iteração atual. Na iteração subsequente, o algoritmo de programação dinâmica previne múltiplas visitas aos vértices em Θ, mas ainda permite múltiplas visitas aos demais vértices. Para este efeito, o vetor de vértices inacessíveis da Definição 3.5 deve ser reduzido aos vértices em Θ. Se $\mathbf{V}_\Theta$ denota este vetor reduzido, então o tamanho de $\mathbf{V}_\Theta$ é igual a quantidade atual de vértices críticos. Como o tamanho do conjunto Θ aumenta ao longo das iterações, Righini e Salani [50] chamaram esta técnica de Relaxação Decrescente do Espaço de Estados (*DSSR – Decremental State Space Relaxation*).

Algoritmo 4.2: $Espprc(H,L,s,t)-$ DSSR

```
/* Inicialização. */
Ψ ← Θ ← ∅;
repita
    Θ ← Θ ∪ Ψ;
    Γ_s^pg ← Γ_t^rg ← {(0,0,0)};
    para cada v_i ∈ N \ {s} faça
        Γ_i^pg ← ∅;
    para cada v_i ∈ N \ {t} faça
        Γ_i^rg ← ∅;
    Y ← {s,t};
    repita
        /* Extensão progressiva. */
        v_i ← v ∈ Y;
        para cada R_i = (T_i, V_Θi, C_i) ∈ Γ_i^pg faça
            para cada v_j ∈ Δ_i^+ faça
                se v_j ∉ Θ ou V_i^j = 0 então
                    R_j ← Estende(R_i, v_j);
                    Γ_j^pg ← RND(Γ_j^pg, R_j);
                    se Γ_j^pg mudou então
                        Y ← Y ∪ {v_j};
        /* Extensão regressiva. */
        para cada R_i = (T_i, V_Θi, C_i) ∈ Γ_i^rg faça
            para cada v_k ∈ Δ_i^- faça
                se v_k ∉ Θ ou V_i^k = 0 então
                    R_k ← Estende(R_i, v_k);
                    Γ_k^rg ← RND(Γ_k^rg, R_k);
                    se Γ_k^rg mudou então
                        Y ← Y ∪ {v_k};
        Y ← Y \ {v_i};
    até Y = ∅ ;
    /* União entre rótulos. */
    para cada v_i ∈ N faça
        para cada R_i ∈ Γ_i^pg faça
            para cada v_j ∈ N faça
                para cada R_j ∈ Γ_j^rg faça
                    se Viavel(R_i, R_j) então
                        Salva(R_i, R_j);
    Ψ ← VisitasMultiplas();
até Ψ = ∅ ;
```

O Algoritmo 4.2, originalmente apresentado por Righini e Salani [50], apresenta o pseudo-código para o *DSSR*, onde $\mathbf{V}_\Theta$ é o vetor de vértices inacessíveis associado aos vértices críticos; o procedimento *VisitasMultiplas* retorna o conjunto Ψ de vértices visitados mais de uma vez no caminho $s \rightsquigarrow t$ ótimo atual. O algoritmo é iterativo: toda vez que ele produz uma solução com ciclos, os vértices visitados mais de uma vez são marcados como críticos e o algoritmo recomeça. Se Ψ não é vazio, então outra iteração é executada com um conjunto de vértices críticos igual a $\Theta' = \Psi \cup \Theta$. Assim, o conjunto de vértices críticos cresce em cada iteração e, eventualmente, o algoritmo *DSSR* encontra uma solução ótima para o *ESPPRC*.

4.3 Novos algoritmos para o *ESPPCC*

Baseando-se nas técnicas apresentadas nas Seções 4.1 e 4.2, dois novos algoritmos exatos são propostos para o *ESPPCC*[2]. Primeiramente, um algoritmo de programação dinâmica bidirecional é visto na Seção 4.3.1. Então, mostra-se como esse algoritmo pode ser facilmente adaptado para fazer uso da técnica de relaxação decrescente do espaço de estados (Seção 4.3.2). Ressalte-se que ambos os algoritmos apresentados nesta seção usam a aceleração heurística descrita na Seção 3.2.1.

4.3.1 Programação dinâmica bidirecional

O algoritmo de programação dinâmica bidirecional implementado para o *ESPPCC* divide-se em duas partes distintas. A primeira é uma etapa onde extensões progressivas e regressivas são realizadas, já a segunda é uma etapa de união de rótulos que permite obter caminhos completos $s \rightsquigarrow t$.

Antes da descrição do algoritmo propriamente, ressalte-se um detalhe fundamental para a sua corretude: extensões progressivas (ou regressivas) devem ser autorizadas a ir além de $Q/2$ em, pelo menos, uma extensão adicional (isto é, um único passo). Assuma, por exemplo, que a instância do *ESPPCC* é constituída pelos vértices v_1, v_2 e v_3, com demandas ε, $Q/2$ e ε, respectivamente, onde ε denota um valor arbitrariamente pequeno, porém superior a zero. Idealmente, espera-se que um algoritmo de programação dinâmica bidirecional produza caminhos progressivos e regressivos tão curtos quanto possível (pois a quantidade de rótulos é exponencial no tamanho dos caminhos), preservando a garantia de que uma solução ótima será encontrada. Agora, suponha que o caminho $s \rightarrow v_1 \rightarrow v_2 \rightarrow v_3 \rightarrow t$ represente a única solução ótima para a instância em questão. Para

[2]Lembre-se que este capítulo segue a notação usada para o *ESPPCC* dada na Seção 3.2.1.

que esta solução seja obtida pelo algoritmo bidirecional, é suficiente que uma das duas condições a seguir seja satisfeita:

1. O subcaminho $s \rightarrow v_1 \rightarrow v_2$, gerado com extensões progressivas, seja unido ao subcaminho $v_3 \rightarrow t$, gerado com extensões regressivas.

2. O subcaminho $s \rightarrow v_1$, gerado com extensões progressivas, seja unido ao subcaminho $v_2 \rightarrow v_3 \rightarrow t$, gerado com extensões regressivas.

Isto é, extensões progressivas (ou regressivas) devem ir além de $Q/2$ em, pelo menos, uma extensão adicional, para que o algoritmo garantidamente encontre uma solução ótima.

Assim, no algoritmo de programação dinâmica bidirecional implementado, rótulos progressivos são estendidos a menos que metade do recurso de capacidade disponível para este rótulo já tenha sido consumido, enquanto rótulos regressivos são estendidos somente se o rótulo resultante dessa extensão não exceder metade do recurso de capacidade.

A etapa de extensão de rótulos

Para as extensões progressivas, usa-se uma matriz M_{pg} de dimensão $(Q+1) \times (n-2)$ (permite-se uma extensão adicional além de $\lceil Q/2 \rceil$, o que justifica a dimensão de tal matriz). Cada entrada $M_{pg}(d,i)$ desta matriz contém um repositório que armazena rótulos progressivos não dominados dos caminhos parciais que partem de s e terminam no vértice $v_i \in N^+$, tendo consumido uma demanda d, $d_i \leq d \leq Q$. Cada rótulo do repositório $M_{pg}(d,i)$ mantém um vetor que registra quais vértices são inacessíveis e o custo do caminho parcial que ele representa. Além disso, o rótulo contém outras duas informações adicionais: o vértice v_i ao qual ele está associado e um apontador para o rótulo que ele estendeu (estas duas informações são utilizadas para reconstruir o subcaminho $s \rightsquigarrow v_i$, caso este faça parte de uma solução ótima).

A matriz M_{pg} é preenchida, linha a linha, começando com $d = 0$ e indo até $d = \lceil Q/2 \rceil$. Para cada rótulo R_i, da entrada $M_{pg}(d,i)$, o algoritmo percorre cada vizinho $v_j \in N^+$ de v_i e avalia a extensão do rótulo R_i ao vértice v_j. Se a extensão do rótulo R_i ao vértice v_j for uma extensão viável (isto é, se $V_i^j = 0$ e $d + d_j \leq Q$) e se o novo rótulo R_j, resultante da extensão[3] de R_i ao vértice v_j, não for dominado por nenhum rótulo pertencente a qualquer um dos repositórios $M_{pg}(1,j), M_{pg}(2,j), \ldots, M_{pg}(d+d_j,j)$, a extensão é realizada e o novo rótulo R_j é guardado no repositório $M_{pg}(d+d_j,j)$ (neste caso, o vetor de vértices inacessíveis $\mathbf{V}_j$ deve ser atualizado). É possível que o rótulo R_j

[3] Observe-se que $d \leq \lceil Q/2 \rceil$, mas $d + d_j$ pode ser maior que $\lceil Q/2 \rceil$. Isto permite que a extensão adicional seja realizada.

domine algum rótulo R'_j presente no repositório $M_{pg}(d+d_j, j)$. Caso isto ocorra, R'_j deve ser descartado.

As extensões regressivas diferenciam-se das progressivas por usarem uma matriz M_{rg} de dimensão $(\lfloor Q/2 \rfloor + 1) \times (n-2)$ (isto é, as extensões regressivas não vão além de $\lfloor Q/2 \rfloor$). Além disso, cada entrada $M_{rg}(d,i)$ dessa matriz contém um repositório que armazena rótulos regressivos não dominados dos caminhos parciais que partem de $v_i \in N^+$ e terminam no vértice t, tendo consumido uma demanda d, $d_i \leq d \leq \lfloor Q/2 \rfloor$. Novamente, cada rótulo guarda as informações adicionais necessárias para a reconstrução de uma solução (úteis para obter o subcaminho $v_i \rightsquigarrow t$, caso o mesmo faça parte de uma solução ótima).

De forma análoga às extensões progressivas, a matriz M_{rg} é preenchida, linha a linha, começando com $d = 0$ e indo até $d = \lfloor Q/2 \rfloor$. Porém, ao avaliar a extensão de um rótulo R_i (do repositório $M_{rg}(d,i)$) ao vértice v_j, verifica-se, adicionalmente, se $d + d_j \leq \lfloor Q/2 \rfloor$ (isto é, rótulos regressivos consomem no máximo $\lfloor Q/2 \rfloor$ de demanda) e realiza-se a extensão somente em caso afirmativo.

A etapa de junção de rótulos

Esta etapa toma todos os pares de rótulos progressivos e regressivos, gerados na etapa anterior, e verifica se os mesmos podem ser unidos[4] para formar caminhos completos $s \rightsquigarrow t$ (observe-se que todos os rótulos progressivos devem, inclusive, ser estendidos até o vértice destino t). Um rótulo progressivo R_i^{pg}, pertencente ao repositório $M_{pg}(d,i)$, só pode ser unido a um rótulo regressivo R_j^{rg}, pertencente ao repositório $M_{rg}(q,i)$, se $d + q \leq Q$ e se $V_i^k + V_j^k \leq 1, 1 \leq k \leq n$ (isto é, o caminho completo $s \rightsquigarrow t$, resultante da união dos rótulos R_i^{pg} e R_j^{rg}, deve ser um caminho elementar que consome uma demanda total máxima igual a Q).

4.3.2 Relaxação decrescente do espaço de estados

O algoritmo de programação dinâmica bidirecional descrito na Seção 4.3.1 pode ser facilmente modificado para fazer uso da técnica de relaxação decrescente do espaço de estados. Para tanto, usa-se um vetor S, de tamanho n, tal que $S(i) = 1$ se o vértice $v_i \in N$ é um vértice crítico, e $S(i) = 0$ caso contrário (inicialmente, $S(i) = 0$, para todo vértice $v_i \in N$).

O novo algoritmo é iterativo, sendo que cada iteração corresponde a uma execução do algoritmo descrito na Seção 4.3.1, na qual o vetor $\mathbf{V}$ de vértices inacessíveis

[4] Lembre-se que um rótulo progressivo R_i^{pg} só pode ser unido a um rótulo regressivo R_j^{rg} se esta for uma união viável, de acordo com a definição dada na Seção 4.1.

(originalmente definido sobre todo vértice $v_i \in N$, de acordo com a Definição 3.5) é redefinido para apenas os vértices $v_i \in N$ tais que $S(i) = 1$ (note-se que isso reduz o espaço de rótulos para apenas tais vértices). Portanto, nesta execução, permite-se ciclos, tanto na etapa de extensão quanto na etapa de junção de rótulos, desde que tais ciclos ocorram sobre vértices $v_i \in N$ tais que $S(i) = 0$. Ao final da iteração, toma-se um caminho $s \rightsquigarrow t$ de menor custo e verifica-se quais vértices foram visitados mais de uma vez neste caminho, para que então o vetor S seja atualizado para a próxima iteração. Caso não haja ciclos neste caminho $s \rightsquigarrow t$, tem-se uma solução ótima para o *ESPPCC*.

Observe-se que no pior caso tem-se uma última iteração onde todos os vértices $v_i \in N$ estão definidos como críticos (isto é, $S(i) = 1$). Uma tal iteração equivalente a execução do algoritmo descrito na Seção 4.3.1.

4.4 Testes computacionais

De forma análoga ao descrito no Capítulo 3, os novos algoritmos para o *ESPPCC* (apresentados na Seção 4.3) foram usados para a geração de colunas (rotas elementares) do algoritmo *Branch-and-Cut-and-Price* de Fukasawa et al. [23]. Para acelerar o mecanismo de geração de colunas, os dois novos algoritmos *BCPs* também fazem uso da heurística descrita na Seção 3.2.2, isto é, em ambos os algoritmos o subproblema de geração de colunas é resolvido com exatidão apenas quando esta heurística falha em encontrar colunas de custo reduzido negativo. Os algoritmos *BCPs* foram testados utilizando-se o mesmo conjunto de instâncias do *CVRP* usado nos experimentos computacionais da Seção 3.3.

Os experimentos apresentados nesta seção tiveram como objetivo principal comparar os desempenhos dos três algoritmos de programação dinâmica descritos nesta dissertação (Seções 3.2.1, 4.3.1 e 4.3.2), quando os mesmos são usados para a geração de colunas do algoritmo *BCP*. Os novos resultados para o *CVRP* são apresentados na Tabela A.1. As colunas L_2, N_2 e T_2 apresentam, respectivamente, os limites inferiores obtidos no nó raiz, o número total de nós explorados na árvore *Branch-and-Bound*, e o tempo total (em segundos) para se resolver a instância à otimalidade com o algoritmo *BCP* que resolve o subproblema usando o algoritmo descrito na Seção 4.3.1. Analogamente, as colunas L_3, N_3 e T_3 fornecem os valores quando o algoritmo *BCP* resolve o subproblema usando o algoritmo descrito na Seção 4.3.2. Já a coluna M_r apresenta o comprimento da maior rota na solução ótima obtida pelo algoritmo de rotas elementares[5].

[5] Optou-se por apresentar apenas os comprimentos obtidos com o algoritmo de rotas elementares descrito na Seção 4.3.1

Esta seção também apresenta, adicionalmente, resultados para o *CVRP* quando os algoritmos para rotas elementares são aplicados em instâncias difíceis. Cada instância do *CVRP* na Tabela A.2 contém o melhor resultado obtido com o uso dos três algoritmos para rotas elementares implementados nesta dissertação (isto é, os dois algoritmos deste capítulo e o algoritmo de programação dinâmica "mono-direcional" descrito na Seção 3.2.1). As colunas L, T_{raiz} e T_{total} apresentam, respectivamente, os limites inferiores obtidos no nó raiz, o tempo total (em segundos) para se resolver o nó raiz, e o tempo total (também em segundos) para se resolver a instância à otimalidade. Entradas com hífen na Tabela A.2 indicam que a instância correspondente não pôde ser resolvida (à otimalidade ou até mesmo no nó raiz) com nenhum dos algoritmos para rotas elementares (obteve-se, em geral, problemas de memória na resolução de tais instâncias).

Note-se que não se obteve sempre os mesmos limites inferiores (L_1, L_2 e L_3) com a geração de rotas elementares. Por exemplo, nas instâncias A-n54-k7 e B-n50-k8, os três limites obtidos diferem-se uns dos outros. Já a instância B-n64-k9 foi resolvida no nó raiz somente com o uso do algoritmo descrito na Seção 4.3.1. Isso deve-se, provavelmente, à separação de cortes do algoritmo *BCP*. Lembre-se que no algoritmo de Fukasawa et al. [23], após se resolver a relaxação linear do problema mestre restrito, aplica-se os algoritmos de separação à solução, a fim de se obter as desigualdades violadas. Contudo, ressalte-se que podem existir várias soluções ótimas (diferentes algoritmos aplicados ao subproblema podem obter diferentes soluções), as quais não violam, necessariamente, as mesmas desigualdades.

Com relação ao tempo de processamento, o algoritmo de programação dinâmica bidirecional, descrito na Seção 4.3.1, apresentou, de uma forma geral, os melhores resultados, sendo seguido pelo algoritmo de programação dinâmica ("mono-direcional"), descrito na Seção 3.2.1. Isso contraria, em parte, os resultados computacionais apresentados para o *ESPPCC* por Righini e Salani [50], os quais sugerem que a técnica de relaxação decrescente do espaço de estados melhora significativamente o algoritmo de programação dinâmica. Os resultados apresentados nesta seção mostram que o algoritmo descrito na Seção 4.3.2 (quando usado para a geração de colunas do algoritmo *BCP*) melhorou os tempos para poucas instâncias, destacando-se as instâncias A-n62-k8, B-n45-k6, B-n66-k9, B-n67-k10 e B-n78-k10.

Observou-se que o algoritmo descrito na Seção 4.3.2 apresentou problemas de convergência em muitos casos, isto é, uma solução ótima (para o subproblema da geração de colunas) só pôde ser obtida depois de várias iterações[6] deste algoritmo. Por consequência, o conjunto de vértices críticos aumentava gradualmente. Isso é indesejável,

[6]Note-se que não se refere a iterações da geração de colunas, e sim iterações do algoritmo descrito na Seção 4.3.2.

pois quanto maior este conjunto, maior a quantidade de rótulos a serem tratados, o que torna as iterações progressivamente mais lentas. Lembre-se que no pior caso do algoritmo da Seção 4.3.2 tem-se uma última iteração com um conjunto de vértices críticos correspondente a todos os vértices do grafo (executar uma tal iteração é tão complicado quanto executar o algoritmo descrito na Seção 4.3.1).

Note-se que o tempo total (isto é, o tempo necessário para se resolver todas as instâncias à otimalidade) obtido com o algoritmo de programação dinâmica básico ("mono-direcional"), descrito na Seção 3.2.1, não é tão superior ao obtido com o algoritmo de programação dinâmica bidirecional (Seção 4.3.1). De fato, a técnica de bidirecionalidade potencialmente reduz a quantidade de caminhos parciais (gerados nas extensões progressivas e nas regressivas) por um fator exponencial (quando comparado com a quantidade de caminhos parciais gerados pelo algoritmo mono-direcional). Contudo, ao unir-se tais caminhos no algoritmo bidirecional (o que é custoso, pois os mesmos devem ser unidos um-a-um e, nesta união, deve-se verificar a ocorrência de ciclos), obtém-se, no final, uma quantidade de caminhos completos tão grande quanto a obtida com o algoritmo mono-direcional.

CAPÍTULO 5

Enumeração de rotas elementares

Enumeração é uma técnica que pode ser vista como uma alternativa ao tradicional particionamento do espaço de soluções viáveis (referido como *branching*) em algoritmos *Branch-and-Bound*. No caso dos *VRPs*, se é usada uma reformulação de cobertura de conjuntos (tal como a descrição dada na Seção 1.1), onde as variáveis da reformulação correspondem a rotas (elementares ou não), e se o *gap* de integralidade, isto é, a diferença entre a melhor solução viável conhecida e o limite inferior corrente for suficientemente pequeno, pode ser prático enumerar todas as rotas elementares relevantes, isto é, todas as rotas que possuem alguma chance de fazer parte de uma solução ótima. Uma rota pode ser considerada não-relevante se pelo menos uma das duas condições dadas a seguir ocorrer:

i Seu custo reduzido (com relação aos valores atuais de $\bar{c}_k$ na equação 1-1) for maior que o *gap* de integralidade, ou

ii Se houver outra rota visitando o mesmo conjunto de clientes com um custo menor (com relação aos custos originais dos arcos – valor c_k na equação 1-1).

Se o número de rotas relevantes não for muito grande (na ordem de poucos milhares), então todo o problema pode ser resolvido como um problema de particionamento de conjuntos contendo apenas tais rotas (usando-se, por exemplo, um solucionador de problemas de programação inteira genérico). Se este problema de particionamento puder ser resolvido, uma solução ótima será encontrada e não será necessário particionar o espaço de soluções (*branching*). Algumas vezes isso acelera o algoritmo *Branch-and-Bound* consideravelmente, quando comparado com estratégias tradicionais de particionamento, já que muitos nós da árvore podem ser resolvidos sem a geração de novos nós. Entretanto, deve-se notar que tal enumeração de rotas é um processo inerentemente exponencial. Seu desempenho prático depende crucialmente do valor do *gap* de integralidade e das características da instância do *VRP* a ser resolvida. Não há nenhuma garantia que uma explosão combinatorial não ocorrerá, mesmo para instâncias de tamanho pequeno.

Este capítulo descreve um algoritmo para a enumeração de rotas elementares, baseado em programação dinâmica bidirecional, e apresenta resultados computacionais de

seu uso no mecanismo de enumeração de um novo algoritmo para o *CVRP*, o qual segue a reformulação e as técnicas propostas por Pessoa, Poggi de Aragão e Uchoa [46] para a construção de algoritmos *BCP* robustos para Problemas de Roteamento de Veículos. Portanto, o enunciado do problema de enumeração acompanha a referida reformulação. Este capítulo está organizado da seguinte forma: a Seção 5.1 define um problema de enumeração de caminhos. A Seção 5.2 discorre sobre conceitos e princípios utilizados pelo algoritmo de enumeração. A Seção 5.3 apresenta o enumerador, propriamente. Já a Seção 5.4 apresenta resultados computacionais obtidos para o *CVRP* quando o algoritmo *BCP* utiliza o enumerador proposto.

5.1 Um problema de enumeração de caminhos

Seja $H = (N,E)$ um grafo orientado, onde E é o conjunto de arcos e $N = \{v_1,\ldots,v_n\}$ é o conjunto de vértices, incluindo um vértice origem s e um vértice destino t. Cada arco $(v_i,v_j) \in E$ tem um custo $c_{ij} \in \mathbb{R}^*_+$ associado e cada vértice $v_i \in N^+$, onde $N^+ = N\backslash\{s,t\}$, tem associado uma demanda d_i, $d_i \in \mathbb{N}^*$ (define-se $d_s = d_t = 0$). Seja Q, $Q \in \mathbb{N}^*$, um valor dado, tal que $Q \geq \max\{d_i\}$.

A partir do grafo H, define-se um multigrafo orientado $H_Q = (N,E_Q)$, onde E_Q contém arcos $(v_i,v_j)^d$, para cada $(v_i,v_j) \in E$, $v_j \neq t$, $d = 1,\ldots,Q-d_i$, e arcos $(v_i,t)^0$, para cada $(v_i,t) \in E$. Cada arco $(v_i,v_j)^d \in E_Q$ tem um custo $\bar{c}^d_{ij} \in \mathbb{R}$ associado.

Define-se um caminho elementar $s \rightsquigarrow t$ viável no multigrafo H_Q como uma sequência de arcos "capacitados" conforme a seguir. A sequência inicia com o arco $(s,v_k)^D$ (para algum valor D, $D \in \mathbb{N}^*$, tal que $\min\{d_i\} \leq D \leq Q$), onde v_k é o primeiro vértice visitado[1] depois de se partir do vértice origem s. Então, cada arco $(v_i,v_j)^d$ é seguido pelo arco $(v_j,v_\ell)^{d-d_\ell}$, onde v_ℓ é o vértice visitado imediatamente depois de v_j. A sequência termina necessariamente com o arco $(v_m,t)^0$, onde v_m é o último vértice visitado antes de se chegar ao vértice destino t.

Tome-se, por exemplo, a situação em que $D = 18$ e os vértices v_1, v_2, v_3 e v_4 têm demandas associadas $d_1 = 5$, $d_2 = 3$, $d_3 = 6$ e $d_4 = 4$. O caminho elementar $s \rightsquigarrow t$ víável que visita os vértices v_2, v_4, v_3 e v_1, nesta ordem (denotado por $s \to v_2 \to v_4 \to v_3 \to v_1 \to t$), corresponderia a sequência de arcos $((s,v_2)^{18}, (v_2,v_4)^{15}, (v_4,v_3)^{11}, (v_3,v_1)^5, (v_1,t)^0)$.

O problema de enumeração consiste em listar todos os caminhos elementares $s \rightsquigarrow t$ viáveis e *relevantes* do multigrafo H_Q, que partem do vértice origem s com capacidade de saída $D = \min\{d_i\},\ldots,Q$. Um caminho elementar $s \rightsquigarrow t$ é *não-relevante* se pelo menos uma das seguintes condições ocorrer:

i Seu custo no multigrafo H_Q for maior que um dado intervalo $\Delta \in \mathbb{R}^*_+$, ou

[1] O valor D é referido como capacidade de saída do caminho $s \rightsquigarrow t$ e o arco $(s,v_k)^D$ como arco de saída.

ii Se houver outro caminho visitando o mesmo conjunto de vértices com um custo menor (em relação ao custo no grafo H).

5.2 Conceitos e princípios para a enumeração

Definição 5.1 *$R_i = (D_i, V_i^1, \ldots, V_i^n, C_i, \overline{C_i})$ é um rótulo associado a um caminho elementar viável X_{si} do vértice origem s ao vértice v_i (ou X_{it} do vértice v_i ao vértice destino t) no multigrafo H_Q. R_i é composto da quantidade de demanda D_i consumida ao longo do caminho (a demanda d_i é contabilizada), do vetor de vértices visitados V_i^k, $1 \leq k \leq n$ ($V_i^k = 1$ se o vértice v_k foi visitado, 0 caso contrário), de um custo C_i, correspondente a soma dos custos c_{ij} da sequência de arcos do caminho no grafo H e, finalmente, de um custo $\overline{C_i}$, correspondente a soma dos custos $\bar{c}_{ij}^{d}$ da sequência de arcos do caminho no multigrafo H_Q.*

Em particular, se um rótulo representa um caminho X_{si}, então tem-se um rótulo progressivo R_i^{pg}. Analogamente, se um rótulo representa um caminho X_{it}, então tem-se um rótulo regressivo R_i^{rg}. A partir desta definição de rótulos, definem-se a seguir os conceitos de *extensão viável*, *união viável* e *dominância* entre rótulos, respectivamente.

Definição 5.2 *A extensão de um rótulo progressivo R_i^{pg} para um vértice $v_j \in N$ é uma extensão viável se $V_i^j = 0$ e $D_i + d_j \leq D$ e, se realizada, dá origem ao novo rótulo $R_j^{pg} = (D_i + d_j, V_i^1, \ldots, V_i^j = 1, \ldots, V_i^n, C_i + c_{ij}, \overline{C_i} + \bar{c}_{ij}^{D-D_i})$.*

Observe-se que a extensão de um rótulo progressivo R_i^{pg} para um vértice $v_j \in N$ se dá pelo arco $(v_i, v_j)^{D-D_i} \in E_Q$, onde o valor D representa a capacidade de saída do caminho X_{si} representado por este rótulo R_i^{pg}.

Definição 5.3 *A extensão de um rótulo regressivo R_i^{rg} para um vértice $v_j \in N$ é uma extensão viável se $V_i^j = 0$ e $D_i + d_j \leq Q$ e, se realizada, dá origem ao novo rótulo $R_j^{rg} = (D_i + d_j, V_i^1, \ldots, V_i^j = 1, \ldots, V_i^n, C_i + c_{ji}, \overline{C_i} + \bar{c}_{ji}^{D_i})$.*

De forma análoga às extensões de rótulos progressivos, note-se que a extensão de um rótulo regressivo R_i^{rg} para um vértice $v_j \in N$ se dá pelo arco $(v_j, v_i)^{D_i} \in E_Q$.

Definição 5.4 *A união de um rótulo progressivo R_i^{pg} a um rótulo regressivo R_j^{rg} é uma união viável se, e somente se, $V_i^k + V_j^k \leq 1, 1 \leq k \leq n$, e $D_i + D_j = D$ (onde D é a capacidade de saída do caminho X_{si} representado pelo rótulo R_i^{pg}) e, se realizada, dá origem ao novo rótulo $R_t = (D_i + D_j, V_i^1 + V_j^1, \ldots, V_i^n + V_j^n, C_i + c_{ij} + C_j, \overline{C_i} + \bar{c}_{ij}^{D_j} + \overline{C_j})$.*

Portanto, ao unir um rótulo progressivo R_i^{pg} a um rótulo regressivo R_j^{rg}, nenhum ciclo pode ser obtido para o caminho resultante $s \rightsquigarrow t$ (representado pelo rótulo R_t), isto é, o conjunto de vértices visitados pelo rótulo progressivo deve ser disjunto ao conjunto de vértice visitado pelo rótulo regressivo. Além da restrição de elementaridade, note-se também que o caminho $s \rightsquigarrow t$ obtido deve ser um caminho viável.

Definição 5.5 *Dados dois rótulos (progressivos ou regressivos) $R_i^{'}$ e $R_i^{''}$, diz-se que o rótulo $R_i^{'}$ domina o rótulo $R_i^{''}$ se, e somente se, $V_i^{'k} = V_i^{''k}$, para $k = 1, \ldots, n$, e $C_i^{'} \leq C_i^{''}$.*

Isto é, o rótulo $R_i^{'}$ domina o rótulo $R_i^{''}$ se o caminho representado pelo rótulo $R_i^{'}$ visitar o mesmo conjunto de vértices que o caminho representado pelo rótulo $R_i^{''}$, com um custo menor (em relação ao custo no grafo H).

5.3 Um enumerador de caminhos

O algoritmo para a enumeração de caminhos proposto nesta seção é baseado em programação dinâmica bidirecional. O algoritmo é semenhante ao descrito na Seção 4.3.1 e difere daquele basicamente em dois pontos. Primeiramente, o problema de enumeração de caminhos não é um problema de otimização tradicional onde, em geral, pretende-se encontrar uma única solução viável, de custo ótimo, dentre um conjunto de soluções viáveis (tal como o problema de caminho mais curto tratado na referida seção). No problema de enumeração, pretende-se encontrar um conjunto de soluções cujos custos associados estejam dentro de uma determinada faixa de valores. Em segundo, os caminhos explorados pelo enumerador são definidos sobre um multigrafo direcionado (nesse multigrafo, nem todo caminho elementar é viável, tal como definido na Seção 5.1), enquanto na Seção 4.3.1 os caminhos são definidos sobre um grafo não orientado.

Para acelerar a execução do enumerador, assume-se a existência de limites inferiores pré-computados para se evitar subcaminhos de caminhos não relevantes (veja a Seção 3.2.1). Assim, suponha que a extensão de um rótulo progressivo[2] não dominado R_i^{pg} a um vértice $v_j \in N$ seja viável e resulta em um rótulo não dominado R_j^{pg}. Suponha também que se conheça um limite inferior $\overline{C_{jt}}$ para o custo de um caminho elementar viável ótimo que inicia no vértice v_j e termina no vértice t, consumindo uma demanda total igual a $D - D_i$ (incluindo o consumo da demanda d_j). Então, R_i^{pg} só será estendido ao vértice v_j se $\overline{C_i} + \bar{c}_{ij}^{D-D_i} + \overline{C_{jt}} \leq \Delta$. Agora, suponha que a extensão de um rótulo regressivo[3] não dominado R_i^{rg} a um vértice $v_j \in N$ seja viável e resulta em um rótulo não dominado

[2] Lembre-se que esta extensão se dá pelo arco $(v_i, v_j)^{D-D_i} \in E_Q$, onde D é a capacidade de saída do caminho elementar viável $s \rightsquigarrow v_i$ representado pelo rótulo R_i^{pg}.

[3] Lembre-se que esta extensão se dá pelo arco $(v_j, v_i)^{D_i} \in E_Q$.

R_j^{rg}. Suponha ainda que se conheça um limite inferior $\overline{C_{sj}}$ para o custo de um caminho elementar viável ótimo que inicia no vértice s e termina no vértice v_j, consumindo uma demanda total dentro do intervalo[4] $[d_j, Q - D_i]$. Então, R_i^{rg} só será estendido ao vértice v_j se $\overline{C_{sj}} + \bar{c}_{ji}^{D_i} + \overline{C_i} \leq \Delta$.

De forma análoga ao descrito na Seção 4.3.1, para o correto uso de programação dinâmica bidirecional, observa-se que uma extensão adicional deve ser feita nas extensões progressivas ou nas regressivas. Por conveniência, optou-se por considerá-la nas extensões regressivas (as extensões progressivas devem considerar todas as capacidades de saída possíveis).

Para as extensões regressivas, usa-se uma matriz M_{rg} de dimensão $(Q+1) \times (n-2)$ (permite-se uma extensão adicional além de $\lceil Q/2 \rceil$, o que justifica a dimensão de tal matriz). Cada entrada $M_{rg}(d,i)$ desta matriz contém um repositório que armazena rótulos regressivos não dominados dos caminhos elementares (viáveis) parciais que partem de $v_i \in N^+$ e terminam no vértice t, tendo consumido uma demanda d, $d_i \leq d \leq Q$. Cada rótulo do repositório $M_{rg}(d,i)$ mantém um vetor que registra quais vértices foram visitados e o custo do caminho parcial no grafo H (e no multigrafo H_Q) que ele representa. Além disso, o rótulo contém outras duas informações adicionais: o vértice v_i ao qual ele está associado e um apontador para o rótulo que ele estendeu (estas duas informações são utilizadas para reconstruir o subcaminho $v_i \rightsquigarrow t$).

A matriz M_{rg} é preenchida, linha a linha, começando com $d = 0$ e indo até $d = \lceil Q/2 \rceil$. Para cada rótulo R_i, da entrada $M_{rg}(d,i)$, o algoritmo percorre cada vizinho $v_j \in N^+$ de v_i (isto é, cada arco $(v_j, v_i) \in E$) e avalia a extensão do rótulo R_i ao vértice v_j. Se a extensão do rótulo R_i ao vértice v_j for uma extensão viável (isto é, se $V_i^j = 0$ e $d + d_j \leq Q$) e se o novo rótulo R_j, resultante da extensão[5] de R_i ao vértice v_j, não for dominado por nenhum rótulo pertencente ao repositório $M_{rg}(d+d_j, j)$, a extensão é realizada e o novo rótulo R_j é guardado no repositório $M_{rg}(d+d_j, j)$. É possível que o rótulo R_j domine algum rótulo $R_j^{'}$ presente no repositório $M_{rg}(d+d_j, j)$. Caso isto ocorra, $R_j^{'}$ deve ser descartado.

As extensões regressivas garantem, por si só, a enumeração de todos os caminhos elementares viáveis (e relevantes) que possuem demanda total D, $\min\{d_i\} \leq D \leq \lceil Q/2 \rceil$ (para se obter tais caminhos, basta estender todos os rótulos regressivos obtidos ao vértice origem s). Contudo, ainda resta obter os caminhos elementares viáveis que possuem demanda total D, $\lceil Q/2 \rceil + 1 \leq D \leq Q$.

Para enumerar tais caminhos, procede-se com as extensões progressivas, con-

[4] O cálculo do limite inferior para a decisão da extensão do rótulo regressivo R_i^{rg} ao vértice v_j deve considerar todas as capacidades de saída possíveis D, $D_i + d_j \leq D \leq Q$, dos caminhos elementares $s \rightsquigarrow v_j$.

[5] Conforme no algoritmo bidirecional da Seção 4.3.1, observe-se que $d \leq \lceil Q/2 \rceil$, mas $d + d_j$ pode ser maior que $\lceil Q/2 \rceil$. Isto permite que a extensão adicional seja realizada.

forme a seguir. Para cada valor de capacidade de saída D, $\lceil Q/2 \rceil + 1 \leq D \leq Q$, toma-se uma matriz M_{pg} de dimensão $(D - \lceil Q/2 \rceil + 1) \times (n-2)$ (observe-se que as extensões progressivas não vão além de $D - \lceil Q/2 \rceil$). Cada entrada $M_{pg}(d,i)$ desta matriz contém um repositório que armazena rótulos progressivos não dominados dos caminhos elementares (viáveis) parciais que partem[6] de s e terminam no vértice $v_i \in N^+$, tendo consumido uma demanda d, $d_i \leq d \leq D - \lceil Q/2 \rceil$. Cada rótulo guarda as informações mencionadas anteriormente, inclusive aquelas necessárias para a reconstrução do subcaminho $s \rightsquigarrow v_i$.

De forma semelhante às extensões regressivas, nas progressivas a matriz M_{pg} é preenchida, linha a linha, começando com $d = 0$ e indo até $d = D - \lceil Q/2 \rceil$. Porém, ao avaliar a extensão de um rótulo R_i (do repositório $M_{pg}(d,i)$) ao vértice v_j, verifica-se, adicionalmente, se $d + d_j \leq D - \lceil Q/2 \rceil$ (isto é, rótulos progressivos consomem no máximo $D - \lceil Q/2 \rceil$ de demanda) e realiza-se a extensão somente em caso afirmativo.

Quando a matriz M_{pg} está preenchida para o valor de capacidade D corrente, todos os rótulos progressivos da matriz M_{pg} são unidos a todos os rótulos regressivos da matriz M_{rg} a fim de se obter todos os caminhos elementares $s \rightsquigarrow t$ viáveis (e relevantes) que possuem demanda total igual a D.

5.4 Testes computacionais

O algoritmo descrito na Seção 5.3 foi utilizado na enumeração de rotas em um algoritmo para o *CVRP*, o qual segue a reformulação e as técnicas propostas por Pessoa, Poggi de Aragão e Uchoa [46] para a construção de algoritmos *BCP* robustos para Problemas de Roteamento de Veículos. Uma dessas técnicas inspira-se naquela descrita por Baldacci, Bodin, e Mingozzi [2], que consiste na combinação do tradicional particionamento do espaço de soluções viáveis (*branching*) com a enumeração de rotas elementares. Nessa estratégia de particionamento híbrida, a enumeração de rotas se dá em cada nó da árvore *Branch-and-Bound*, após o mesmo ser resolvido via geração de cortes e colunas[7]. Contudo, a fim de se obter um conjunto de rotas relevantes[8] de tamanho razoável (sobre o qual é prático resolver um problema de particionamento de conjuntos com um solucionador de problemas de programação linear inteira), Pessoa, Poggi de Aragão e Uchoa [46] atentam-se em abortar o procedimento caso os números de rotas relevantes ou estados (representação de subcaminhos relevantes) atualmente mantidos pelo enumerador sejam suficientemente elevados.

[6] Tais caminhos partem do vértice origem s por algum arco de saída $(s, v_j)^D \in E_Q$.

[7] Lembre-se que quando algoritmos *BCP* robustos são aplicados em *VRPs*, geralmente, em cada nó de uma árvore *Branch-and-Bound*, uma relaxação linear da reformulação de cobertura de conjuntos do problema é resolvida via geração de cortes e colunas.

[8] A introdução deste capítulo discorre sobre o conceito de rotas elementares relevantes.

Seguindo as ideias em [46], configurou-se o referido algoritmo *BCP* para abortar a enumeração de rotas caso o número de rótulos não dominados gerados pelo algoritmo de programção dinâmica (descrito na Seção 5.3) seja maior que 120.000. Se este limite não for alcançado, um problema de particionamento de conjuntos (veja a reformulação VRP-SPP apresentada na Seção 1.1) contendo todas as rotas relevantes geradas é passado para um solucionador de PLIs. Depois do problema de particionamento ser resolvido, o nó corrente é declarado como *resolvido* e, portanto, o seu espaço de soluções viáveis não precisará ser particionado. Caso contrário, se a enumeração falhar (isto é, se mais de 120.000 rótulos tiverem sido gerados), então o espaço de soluções do nó é particionado em dois novos nós (e essa estratégia híbrida repete-se nos nós filhos). Naturalmente, nós com profundidade elevada terão *gaps* menores, então espera-se que a enumeração funcione em algum momento.

O algoritmo *BCP* usado nos testes também utiliza a enumeração de rotas como uma heurística no nó raiz, seguindo a técnica descrita em [46]. Se o *gap* g atual deste nó ainda for muito grande e mais de 120.000 rótulos já tiverem sido gerados pelo enumerador, então tenta-se a enumeração com um *gap* $g' = g/2$. Se ainda não for suficiente, tenta-se com $g' = g/4$ e assim por diante. Se agora a enumeração tiver sucesso, tenta-se com um *gap* $g' = (g/2 + g/4)/2$. Em suma, realiza-se uma pesquisa binária para determinar um valor para o *gap* que irá produzir um problema de particionamento de conjuntos de tamanho razoável. Observe-se que esta técnica heurística pode acelerar o algoritmo *BCP*, pois evita-se a enumeração naqueles nós cujos *gaps* são maiores que g'.

A fim de avaliar a eficiência da técnica de enumeração, testou-se o algoritmo *BCP* com algumas instâncias do *CVRP*, das séries A e E, disponíveis em [48]. Os resultados obtidos estão sumarizados na Tabela A.3, onde as colunas N_1 e T_1 apresentam, respectivamente, o número total de nós explorados na árvore *Branch-and-Bound*, e o tempo total (em segundos) gasto para se resolver a instância à otimalidade com o algoritmo *BCP* que realiza apenas o tradicional particionamento do espaço de soluções. Analogamente, as colunas N_2 e T_2, fornecem os valores quando o algoritmo *BCP* faz uso da estratégia híbrida supramencionada.

Note-se a significativa redução da árvore *Branch-and-Bound* (e, consequentemente, do tempo total de processamento) obtido com o uso da estratégia de particionamento híbrida. De fato, com a enumeração de rotas elementares, todas as instâncias da classe A testadas (além da instância E-n76-k14) puderam ser resolvidas à otimalidade no nó raiz, gastando-se um tempo total menor que 10% daquele requerido pelo algoritmo *BCP* que não se utiliza da técnica de enumeração. Embora as outras instâncias testadas não puderam ser resolvidas no nó raiz com o uso da estratégia híbrida, em geral, a quantidade de nós explorados foi inferior a 15% se comparado com a versão que utiliza apenas uma estratégia de particionamento (*branching*) tradicional.

Já na segunda etapa de testes computacionais, teve-se o intuito de avaliar a eficiência do algoritmo de programação dinâmica descrito na Seção 5.3. Para tanto, configurou-se o algoritmo *BCP* para abortar a enumeração de rotas se mais de 15.000.000 rótulos já tiverem sido gerados. Obviamente, é impraticável resolver um problema de particionamento de conjuntos (utilizando-se apenas um solucionador de PLIs) com uma quantidade rotas nesta ordem de grandeza. Contudo, ressalte-se a importância de se ter um algoritmo capaz de enumerar rotas rapidamente, pois o mesmo pode ser executado em cada nó da árvore *Branch-and-Bound*.

As colunas G, G_{max}, R e T da Tabela A.4 apresentam, respectivamente, o *gap* de integraliade (diferença entre o valor de uma solução ótima e o limite inferior obtido pelo algoritmo *BCP* ao resolver o nó raiz), o *gap* máximo obtido para a enumeração (isto é, o *gap* g' retornado pela heurística supramencionada), o número de rotas elementares relevantes enumeradas no nó raiz dentro deste *gap* máximo, e o tempo (em segundos) total gasto nesta enumeração. Os resultados mostram que o enumerador de rotas proposto na Seção 5.3 é capaz de enumerar uma grande quantidade de rotas em um tempo relativamente pequeno. Para todas as instâncias testadas, excetuando-se a instância E-n101-k8, pôde-se enumerar todas as rotas relevantes dentro do *gap* de integralidade, gastando-se poucos segundos. Em particular, na instância E-n101-k8, mais de 10.000.000 de rotas puderam ser enumeradas consumindo-se menos de 250 segundos.

CAPÍTULO 6

Conclusões

A principal contribuição desta dissertação foi a descrição do uso de algoritmos para o *ESPPCC*, como geradores de rotas elementares (subproblema de geração de colunas), em um algoritmo *BCP* robusto para o *CVRP*. Os resultados obtidos, para algumas instâncias de teste do *CVRP*, foram comparados aos obtidos na versão original desse algoritmo *BCP*, que utiliza rotas sem 3-ciclos ou 4-ciclos. Até então, devia-se ao recente trabalho de Baldacci, Christofides e Mingozzi [3] o único na literatura[1] a usar rotas elementares na geração de colunas para o *CVRP*. Contudo, esta dissertação apresentou um experimento comparativo entre ganhos (limites inferiores) e perdas (tempo de processamento) com um algoritmo *BCP* para o *CVRP* utilizando rotas elementares que não haviam sido relatados na literatura. Os resutados obtidos dão uma clara dimensão do quanto o uso de rotas elementares na geração de colunas pode acrescentar em termos de limites inferiores para o *CVRP*. Ressalte-se ainda que tais resultados são incipientes e que outros trabalhos recentemente propostos para o *ESPPCC* podem estender, naturalmente, o trabalho aqui apresentado, conforme discutido adiante.

Outra contribuição relevante desta dissertação foi a implementação de um enumerador de rotas elementares capaz de enumerar rapidamente uma grande quantidade de rotas (cujo custo reduzido encontra-se dentro de um pequeno intervalo), além de apresentar resultados que atestam que a técnica de enumeração pode melhorar o desempenho do algoritmo para o *CVRP*.

O primeiro desses algoritmos para o *ESPPCC*, apresentado no Capítulo 3, é uma adaptação do algoritmo de programação dinâmica básico (“mono-direcional”) de Feillet et al. [20], o qual resolve o caso geral de caminhos mais curtos com restrições de recursos (*ESPPRC*), para o caso em que a única restrição de recurso é a de capacidade. Este algoritmo para o *ESPPCC* foi usado para a resolução exata do subproblema de geração de colunas do algoritmo *BCP* de Fukasawa et al. [23], substituindo o mecanismo original,

[1] Esses autores propuseram um algoritmo *BCP* complexo onde o subproblema de geração de colunas é resolvido por um esquema de enumeração exaustiva, baseada em programação dinâmica, que garante encontrar todas as colunas de custo reduzido negativo.

onde uma relaxação do *ESPPCC* é resolvida com exatidão. Os resultados computacionais, apresentados no final do Capítulo 3, mostram que a redução do *gap* (com relação a eliminação de 4-ciclos) para algumas instâncias com mais de 70 vértices não foi tão expressiva (abaixo de 10%). Por outro lado, obteve-se considerável redução no tempo gasto para se resolver algumas instâncias de grau de dificuldade moderado, tais como as instâncias B-n50-k8, E-n76-k14 e E-n101-k14.

Outros dois novos algoritmos para o *ESPPCC*, apresentados no Capítulo 4, também foram utilizados para a resolução exata do subproblema de geração de colunas do algoritmo de Fukasawa et al. [23]. Estes novos algoritmos, que são adaptações dos algoritmos de programação dinâmica aprimorados propostos para o *ESPPRC* por Righini e Salani [49, 50], se baseiam em técnicas avançadas, tais como as técnicas de programação dinâmica bidirecional e relaxação decrescente do espaço de estados. Os resultados apresentados no final do referido capítulo mostram que o algoritmo de programação dinâmica bidirecional obteve melhor desempenho para a geração de colunas do algoritmo *BCP*, seguido pelo algoritmo de programação dinâmica básico (mono-direcional), apresentado no Capítulo 3. Tais resultados contrariam, em parte, aqueles apresentados por Righini e Salani [50], os quais sugerem que a técnica de relaxação decrescente do espaço de estados sobrepõe-se a estas duas anteriores.

Diferentemente dos dois capítulos anteriores, nos quais as rotas elementares são exploradas no mecanismo de geração de colunas de um algoritmo *BCP*, o Capítulo 5 apresentou o uso de rotas elementares em um contexto de enumeração. Os resultados computacionais mostram que o enumerador de rotas proposto, o qual é baseado em programação dinâmica bidirecional, é capaz de enumerar alguns milhões de rotas em poucos minutos. Além disso, os resultados atestam que a enumeração é uma técnica capaz de acelerar o algoritmo *Branch-and-Bound* para o *CVRP*, pois quando o número de rotas relevantes (isto é, rotas que possuem alguma chance de pertencer a uma solução ótima) no nó corrente não é tão grande (tipicamente, menor que 80.000 rotas), em geral, é mais rápido resolver um problema de particionamento de conjunto contendo tais rotas[2], a fim de se encontrar uma solução inteira ótima para este nó, que particionar (*branching*) o seu espaço de soluções com estratégias tradicionais.

Todos os algoritmos apresentados para o *ESPPCC* nesta dissertação são algoritmos de correção de rótulos baseados em programação dinâmica. Optou-se, no início deste trabalho, pelo estudo e implementação de tais algoritmos pelo fato deles constituirem o método de solução exato dominante na literatura para o *ESPPRC*. Contudo, observou-se que a aplicação de algoritmos de correção de rótulos para o *ESPPCC* tendem a apresentar falhas quando:

[2]Usando-se, por exemplo, um solucionador de problemas de programação inteira tal como o CPLEX.

- O grafo subjacente apresenta muitas arestas com custo negativo e, possivelmente, ciclos de custo negativo. Neste último caso, pode-se provar ser o *ESPPCC* um problema fortemente $\mathcal{NP}$-Difícil, Kohl [34].

Lembre-se que no algoritmo *BCP* robusto de Fukasawa et al. [23], ao final de cada iteração da geração de colunas, aquelas de custo reduzido negativo obtidas são adicionadas à relaxação linear do problema mestre restrito, e então resolve-se essa relaxação com as novas colunas adicionadas. Esse processo iterativo repete-se até se provar que não existem mais colunas a serem adicionadas (esta prova só pode ser assegurada com a resolução exata do subproblema de geração de colunas). Após a geração de colunas, inicia-se a geração de cortes: os algoritmos de separação são aplicados à solução fracionária corrente, e as desigualdades obtidas são adicionadas à relaxação linear do problema mestre. Novamente, o processo inteiro se repete (isto é, a geração de colunas é novamente ativada e, após esta, a geração de cortes). Quando ambas as fases de geração de colunas e cortes falham (isto é, quando não há mais colunas de custo reduzido negativo e nem desigualdades violadas conhecidas) e a solução para o nó corrente é fracionária (e inferior ao melhor limite superior conhecido), divide-se o espaço de solução deste nó (*branching*) em dois novos nós filhos e repete-se todo o processo em cada um destes nós filhos.

Portanto, durante a resolução de uma instância do *CVRP* com o algoritmo *BCP* robusto, os valores das variáveis duais são constantemente alterados e, por consequência, algumas arestas tornam-se mais atrativas (isto é, com um custo reduzido negativo associado a si). Logo, embora tenha-se um tipo de informação estática (estrutura do grafo e demandas), tem-se, por outro lado, uma informação dinâmica (os valores dos custos reduzidos associados às arestas do grafo). Nesse contexto dinâmico (geração de cortes e colunas e *branching* de nós), torna-se delicado o uso de rotas elementares na geração de colunas, especialmente se o *ESPPCC* for resolvido por algoritmos de correção de rótulos, pois estes são sensíveis aos custos associados às arestas do grafo subjacente. Como, em geral, o subproblema de geração de colunas deve ser resolvido à otimalidade pelo menos uma vez em cada nó da árvore *Branch-and-Bound* de um algoritmo *BCP* robusto, a fim de se obter um limite válido, se o subproblema tornar-se intratável, o algoritmo *BCP* como um todo pode falhar.

Observou-se que, para algumas instâncias de teste do *CVRP*, embora a solução para o nó raiz tenha sido obtida rapidamente, ao fazer-se *branching* em nós, o tempo gasto na geração de colunas dos nós filhos aumentava consideravelmente (para algumas dessas instâncias, este aumento tornou inviável a resolução das mesmas).

Uma outra possibilidade de falha de algoritmos de correção de rótulos para o *ESPPCC* ocorre quando:

- A solução pode incluir caminhos longos no número de vértices visitados (já que a quantidade de rótulos gerada é exponencial no tamanho dos caminhos).

Em geral, algoritmos de correção de rótulos apresentam problemas quando é possível produzir caminhos longos no número de nós visitados, por exemplo, quando as restrições de recurso não são tão restritivas. Isso normalmente resulta em uma grande quantidade de rótulos (representação de caminhos parciais) a ser processada, o que pode demandar tempo. De fato, o subproblema de geração de colunas não pôde ser resolvido à otimalidade uma única vez para várias instâncias de teste do *CVRP* com pouco mais de 100 vértices. Isso é indesejável, pois o custo ótimo para algumas dessas instâncias já é conhecido.

Uma abordagem alternativa (a de correção de rótulos) recentemente apresentada por Jepsen, Petersen e Spoorendonk [29] descreve um algoritmo *Branch-and-Cut* para o *ESPPCC*, o qual se baseia no modelo PLI revisado na Seção 3.2. Os resultados númericos apresentados no referido trabalho sugerem que algoritmos *Branch-and-Cut* podem ser consideravelmente melhores que algoritmos de correção de rótulos, especialmente quando a instância do *ESPPCC* em questão contém muitas arestas com custo negativo (os resultados em [29] mostram que o algoritmo *Branch-and-Cut* proposto é capaz de resolver várias instâncias do *ESPPCC* em apenas uma fração do tempo requerido por um algoritmo de correção de rótulos). Além disso, instâncias de teste para o *ESPPCC* com mais de 250 vértices puderam ser resolvidas à otimalidade em poucas dezenas de segundos com o novo algoritmo. Ressalte-se que várias dessas instâncias de teste foram derivadas de instâncias do *CVRP* conhecidas na literatura. Tais instâncias correspondem a uma certa iteração da geração de colunas de um simples algoritmo para o *CVRP*. Contudo, Jepsen, Petersen e Spoorendonk [29] não incluiram resultados para o *CVRP*.

Portanto, um possível trabalho imediato ao desta dissertação consiste na implementação de algoritmos *Branch-and-Cut* (como, por exemplo, o proposto em [29]) para o *ESPPCC* e utilização dos mesmos para a geração de colunas no algoritmo de Fukasawa et al. [23], tal como feito com algoritmos de correção de rótulos nos Capítulos 3 e 4.

Os resultados apresentados nesta dissertação mostram que o ganho em termos de limites inferiores, quando apenas rotas elementares são usadas na geração de colunas do algoritmo *BCP* robusto, não é tão significativo para algumas das instâncias testadas. Portanto, a fim de se alcançar resultados verdadeiramente competitivos para o *CVRP*, tais como os recentes resultados (limites inferiores e tempo de processamento) reportados por Baldacci, Christofides e Mingozzi [3], sugere-se um algoritmo *BCP* complexo que explore, em determinados momentos, elementos tais como:

- Cortes “fortes” (ou cortes não-robustos, conforme a definição dada na Seção 1.1), isto é, cortes expressos em termos das variáveis do problema mestre. Geralmente,

tais cortes aumentam a complexidade do subproblema de geração de colunas. No entanto, um trabalho recente esclarece como tratar os custos adicionais (novas variáveis e restrições) no subproblema (veja capítulo 6 de Spoorendonk [53]).

- Colunas "fortes", isto é, rotas elementares. Isto requer a investigação de novas abordagens exatas para a resolução do *ESPPCC*, além das exploradas nesta dissertação como, por exemplo, o uso de algoritmos *Branch-and-Cut* (tal como em [29]).
- Enumeração de rotas elementares, tal como apresentado no Capítulo 5.

Referências Bibliográficas

[1] AHUJA, R. K.; MAGNANTI, T. L.; ORLIN, J. B. **Network Flows: Theory, Algorithms, and Applications**. Prentice Hall, 1993.

[2] BALDACCI, R.; BODIN, L.; MINGOZZI, A. **The Multiple Disposal Facilities and Multiple Inventory Locations Rollon-Rolloff Vehicle Routing Problem**. *Computers & Operations Research*, 33(9):2667–2702, 2006.

[3] BALDACCI, R.; CHRISTOFIDES, N.; MINGOZZI, A. **An Exact Algorithm for the Vehicle Routing Problem Based on the Set Partitioning Formulation with Additional Cuts**. *Mathematical Programming*, 115(2):351–385, 2008.

[4] BARNHART, C.; LAPORTE, G. **Transportation**, volume 14. Handbooks in Operations Research and Management Science, 2007.

[5] BARNHART, C.; HANE, C. A.; VANCE, P. H. **Using Branch-and-Price-and-Cut to Solve Origin-Destination Integer Multicommodity Flow Problems**. *Operations Research*, 48(2):318–326, 2000.

[6] BEASLEY, J.; CHRISTOFIDES, C. **An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraints: Application to some Vehicle Routing Problems**. *Networks*, 19:379–394, 1989.

[7] BOLAND, N.; DETHRIDGE, J.; DUMITRESCU, I. **Accelerated Label Setting Algorithms for the Elementary Resource Constrained Shortest Path Problem**. *Operations Research Letters*, 34(1):58–68, 2006.

[8] BRAMEL, J.; SIMCHI-LEVI, D. **Set-covering-based Algorithms for the Capacitated VRP**. In: Toth, P.; Vigo, D., editors, *The Vehicle Routing Problem*, volume 9, p. 85–108. SIAM Monographs on Discrete Mathematics and Applications, 2002.

[9] CARLYLE, W. M.; ROYSET, J. O.; KEVIN WOOD, R. **Lagrangian Relaxation and Enumeration for Solving Constrained Shortest-Path Problems**. *Networks*, 52(4):256–270, 2008.

[10] CHABRIER, A. **Vehicle Routing Problem with Elementary Shortest Path Based Column Generation**. *Computer & Operations Research*, 33(10):2972–2990, 2006.

[11] CHRISTOFIDES, N.; MINGOZZI, A.; TOTH, P. **Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relaxations**. *Mathematical Programming*, 20:255–282, 1981.

[12] CHRISTOFIDES, N.; MINGOZZI, A.; TOTH, P. **State-space Relaxation Procedures for the Computation of Bounds to Routing Problems**. *Networks*, 11(2):145–164, 1981.

[13] CORDEAU, J.-F.; LAPORTE, G.; SAVELSBERGH, M.; VIGO, D. **Vehicle Routing**. In: Barnhart, C.; Laporte, G., editors, *Transportation*, volume 14, p. 367–428. Handbooks in Operations Research and Management Science, 2007.

[14] DANNA, E.; PAPE, C. L. **Accelerating Branch-and-Price with Local Search: A Case Study on the Vehicle Routing Problem with Time Windows**. In: Desaulniers, G.; Desrosiers, J.; Solomon, M. M., editors, *Column Generation*, chapter 3, p. 30–130. Springer, 2005.

[15] DANTZIG, G. B.; RAMSER, J. H. **The Truck Dispatching Problem**. *Management Science*, 6(1):80–91, 1959.

[16] DESAULNIERS, G.; DESROSIERS, J.; SOLOMON, M. M. **Column Generation**. Springer, 2005.

[17] DESROCHERS, M. **An Algorithm for the Shortest Path Problem with Resource Constraints**. Technical Report G-88-27, GERAD, Ecole des HEC, Canada, 1988.

[18] DROR, M. **Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW**. *Operations Research*, 42:977–978, 1994.

[19] DUMITRESCU, I. **Constrained Path and Cycle Problems**. PhD thesis, Department of Mathematics and Statistics, University of Melbourne, Australia, 2002.

[20] FEILLET, D.; DEJAX, P.; GENDREAU, M.; GUEGUEN, C. **An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraints: Application to some Vehicle Routing Problems**. *Networks*, 44(3):216–229, 2004.

[21] FELICI, G.; GENTILE, C.; RINALDI, G. **Solving Large MIP Models in Supply Chain Management by Branch & Cut**. Technical report, Istituto di Analisi dei Sistemi ed Informatica del CNR, Italy, 2000.

[22] FISHER, M. L. **Vehicle Routing**. In: Ball, M.; Magnanti, T.; Monma, C.; Nemhauser, G., editors, *Handbooks in Operations Research and Management Science*, volume 8, p. 1–33. Network Routing, 1995.

[23] FUKASAWA, R.; LONGO, H.; LYSGAARD, J.; DE ARAGÃO, M. P.; REIS, M.; UCHOA, E.; WERNECK, R. F. **Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem**. *Mathematical Programming*, 106(3):491–511, 2006.

[24] GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability. A Guide to the Theory of NP-Completeness**. W. H. Freeman and Company, New York, 1979.

[25] GELINAS, S.; DESROCHERS, M.; DESROSIERS, J.; SOLOMON, M. **A New Branching Strategy for Time Constrained Routing Problems with Application to Backhauling**. *Annals of Operations Research*, 61:91–109, 1995.

[26] GENDREAU, M.; LAPORTE, G.; POTVIN, J.-Y. **Metaheuristics for the Capacitated VRP**. In: Toth, P.; Vigo, D., editors, *The Vehicle Routing Problem*, volume 9, p. 129–154. SIAM Monographs on Discrete Mathematics and Applications, 2002.

[27] IRNICH, S.; VILLENEUVE, D. **The Shortest-Path Problem with Resource Constraints and k-Cycle Elimination for k $\geq$ 3**. *INFORMS Journal on Computing*, 18(3):391–406, 2006.

[28] IRNICH, S.; DESAULNIERS, G. **Shortest Path Problems with Resource Constraints**. In: Desaulniers, G.; Desrosiers, J.; Solomon, M. M., editors, *Column Generation*, chapter 2, p. 33–65. Springer, 2005.

[29] JEPSEN, M.; PETERSEN, B.; SPOORENDONK, S. **A Branch-and-Cut Algorithm for the Elementary Shortest Path Problem with a Capacity Constraint**. Technical Report 08-01, DIKU, University of Copenhagen, Denmark, 2008.

[30] JEPSEN, M.; PETERSEN, B.; SPOORENDONK, S.; PISINGER, D. **Subset-Row Inequalities Applied to the Vehicle Routing Problem with Time Windows**. *Operations Research*, 56(2):497–511, 2008.

[31] JEPSEN, M.; SPOORENDONK, S.; PETERSEN, B.; PISINGER, D. **A Non-robust Branch-and-Cut-and-Price for the Vehicle Routing Problem with Time Windows**. Technical Report 06/03, University of Copenhagen, 2006.

[32] KATOK, E.; LEWIS, H. S.; HARRISON, T. P.; VANDERBECK, F. **Lot-Sizing with Start-Up Times**. *Management Science*, 44(10):1409–1425, 1998.

[33] KIM, D.; BARNHART, C.; WARE, K.; REINHARDT, G. **Multimodal Express Package Delivery: A Service Network Design Application**. *Transportation Science*, 33(4):391–407, 1999.

[34] KOHL, N. **Exact Methods for Time Constrained Routing and Related Scheduling Problems**. PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 1995.

[35] KOHL, N.; DESROSIERS, J.; MADSEN, O. B. G.; SOLOMON, M. M.; SOUMIS, F. **2-Path Cuts for the Vehicle Routing Problem with Time Windows**. *Transportation Science*, 33(1):101–116, 1999.

[36] LAPORTE, G. **The Vehicle Routing Problem: An Overview of Exact and Aproximate Algorithms**. *European Journal of Operational Research*, 59:345–358, 1992.

[37] LAPORTE, G.; NORBERT, Y. **A Branch-and-Bound Algorithm for the Capacitated Vehicle Routing Problem**. *Operations Research Spektrum*, 5:77–85, 1983.

[38] LAPORTE, G.; SEMET, F. **Classical Heuristics for the Capacitated VRP**. In: Toth, P.; Vigo, D., editors, *The Vehicle Routing Problem*, volume 9, p. 109–128. SIAM Monographs on Discrete Mathematics and Applications, 2002.

[39] LÜBBECKE, M. E.; DESROSIERS, J. **Selected Topics in Column Generation**. *Operations Research*, 53(6):1007–1023, 2005.

[40] LYSGAARD, J. **CVRPSEP: A Package of Separation Routines for the Capacitated Vehicle Routing Problem**. Working paper 03-04, Department of Management, Science and Logistics, Aarhus School of Business, 2003.

[41] LYSGAARD, J.; LETCHFORD, A. N.; EGLESE, R. W. **A New Branch-and-Cut Algorithm for the Capacitated Vehicle Routing Problem**. *Mathematical Programming*, 100(2):423–445, 2004.

[42] MARTIN, R. K. **Large Scale Linear and Integer Optimization: A Unified Approach**. Kluwer academic Publisher, 1999.

[43] MINGOZZI, A.; BIANCO, L.; RICCIARDELLI, S. **Dynamic Programming Strategies for the Traveling Salesman Problem with Time Window and Precedence Constraints**. *Operations Research*, 45(3):365–377, 1997.

[44] NADDEF, D.; RINALDI, G. **Branch-and-Cut Algorithms for the Capacitated VRP**. In: Toth, P.; Vigo, D., editors, *The Vehicle Routing Problem*, volume 9, p. 53–81. SIAM Monographs on Discrete Mathematics and Applications, 2002.

[45] NEMHAUSER, G. L.; WOLSEY, L. A. **Integer and Combinatorial Optimization**. John Wiley & Sons, Inc., 1988.

[46] PESSOA, A.; POGGI DE ARAGÃO, M.; UCHOA, E. **Robust Branch-Cut-and-Price Algorithms for Vehicle Routing Problems**. In: Golden, B.; Raghavan, S.; Wasil, E., editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43, p. 297–325. Springer, 2008.

[47] POGGI DE ARAGÃO.; UCHOA, E. **Integer Program Reformulation for Robust Branch-and-Cut-and-Price Algorithms**. In: *In Proceedings of the Conference Mathematical Program in Rio: A Conference in Honour of Nelson Maculan*, p. 56–61, 2003.

[48] RALPHS, T. **Branch Cut and Price Resource Web**, 2009. www.branchandcut.org, último acesso em 02/02/2010.

[49] RIGHINI, G.; SALANI, M. **Symmetry Helps: Bounded Bi-Directional Dynamic Programming for the Elementary Shortest Path Problem with Resource Constraints**. *Discrete Optimization*, 3(3):255–273, 2006.

[50] RIGHINI, G.; SALANI, M. **New Dynamic Programming Algorithms for the Resource Constrained Elementary Shortest Path Problem**. *Networks*, 51(3):155–170, 2008.

[51] SALANI, M. **Branch-and-Price Algorithms for Vehicle Routing Problems**. PhD thesis, Università degli Studi di Milano, 2006.

[52] SOLOMON, M. M. **Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints**. *Operations Research*, 35:234–265, 1987.

[53] SPOORENDONK, S. **Cut and Column Generation**. PhD thesis, DIKU, University of Copenhagen, Denmark, 2008.

[54] TOTH, P.; VIGO, D. **Branch-and-Bound Algorithms for the Capacitated VRP**. In: Toth, P.; Vigo, D., editors, *The Vehicle Routing Problem*, volume 9, p. 29–51. SIAM Monographs on Discrete Mathematics and Applications, 2002.

[55] TOTH, P.; VIGO, D. **The Vehicle Routing Problem**, volume 9. SIAM Monographs on Discrete Mathematics and Applications, 2002.

[56] Toth, P.; Vigo, D., editors. **The Vehicle Routing Problem**. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[57] VAN DEN AKKER, J.; HURKENS, C.; SAVELSBERGH, M. **Time-Indexed Formulations for Machine Scheduling Problems: Column Generation**. *INFORMS Journal on Computing*, 12(2):111–124, 2000.

[58] WERNECK, R.; SETUBAL, J.; DA CONCEICÃO, A. **Finding Minimum Congestion Spanning Trees**. *ACM Journal of Experimental Algorithmics*, 5:11, 2000.

[59] WOLSEY, L. A. **Integer Programming**. John Wiley & Sons, Inc., 1998.

APÊNDICE A

Resultados Computacionais

Todos os experimentos computacionais realizados nesta dissertação foram executados em um Intel Core 2 Duo, 2.8 GHz, com 4 GB de RAM, usando-se Windows XP. Usou-se CPLEX 9.0 para a resolução de programas lineares nos testes descritos nos Capítulos 3 e 4. Já no Capítulo 5, usou-se CPLEX 10.1 para a resolução de programas lineares e problemas de particionamento de conjuntos.

Ressalte-se que ao testar a implementação original do algoritmo de Fukasawa et al. [23] para a eliminação de 4-ciclos na geração de colunas, obteve-se um mesmo erro para algumas das instâncias, o qual indica que o limite inferior obtido para um nó filho é menor que o obtido para seu nó pai (entradas com hífen na Tabela A.1 indicam que a instância correspondente não pôde ser resolvida devido a ocorrência desse erro).

Para cada instância do *CVRP* testada nos Capítulos 3 e 4, usou-se um limite superior (passado como parâmetro de entrada para o algoritmo *BCP*) igual a uma unidade acima do valor da melhor solução conhecida. Já nos testes realizados no Capítulo 5, usou-se um limite superior igual ao valor da melhor solução conhecida.

A descrição dos resutados apresentados na Tabela A.1 encontra-se na Seção 3.3 e na Seção 4.4 (esta última seção também descreve as entradas para a Tabela A.2). Já a Seção 5.4 discorre sobre os resultados das Tabelas A.3 e A.4.

Tabela A.1: *Geração de rotas elementares para o* CVRP

	Alg1. Cap. 3			Alg2. Cap. 4			Alg3. Cap. 4			$\delta = 3$			$\delta = 4$				
Instância	L_1	N_1	T_1	L_2	N_2	T_2	L_3	N_3	T_3	L_4	N_4	T_4	L_5	N_5	T_5	M_r	C^*
A-n37-k5	667.500	7	10.26	667.166	5	7.90	667.166	6	103.65	667.156	7	4.60	667.104	7	7.75	11	**669**
A-n37-k6	940.144	21	19.29	940.144	21	18.14	940.144	21	22.82	932.575	197	40.14	934.857	64	36.93	8	**949**
A-n38-k5	723.421	11	14.32	723.421	7	10.90	723.421	11	15.70	719.313	35	14.60	720.500	31	28.07	11	**730**
A-n39-k5	819.035	5	11.31	819.035	8	14.79	819.035	9	15.76	816.699	25	12.25	817.808	17	17.09	9	**822**
A-n39-k6	825.151	7	9.23	825.151	7	8.93	825.151	11	13.15	822.800	21	6.34	824.392	13	8.64	10	**831**
A-n44-k6	936.800	2	6.65	936.800	2	6.87	936.800	2	7.12	934.852	7	2.96	934.900	6	6.53	8	**937**
A-n45-k6	942.518	3	5.87	942.518	3	6.04	942.518	3	6.81	938.112	14	7.54	940.097	13	18.14	9	**944**
A-n45-k7	1142.288	10	16.73	1142.288	10	17.18	1142.288	10	18.14	1139.292	30	11.81	1141.440	24	24.32	9	**1146**
A-n46-k7	914.	1	3.10	914.	1	3.15	914.	1	3.29	914.	1	2.00	914.	1	2.92	10	**914**
A-n48-k7	1072.027	2	6.87	1072.027	2	6.92	1072.027	2	7.46	1069.032	11	6.64	1071.338	4	7.70	9	**1073**
A-n53-k7	1004.433	9	120.07	1004.433	9	109.51	1004.433	9	369.62	1003.838	16	18.34	1004.060	25	44.73	11	**1010**
A-n54-k7	1156.591	26	101.42	1156.927	29	91.48	1156.596	65	300.82	1153.180	151	148.01	1155.455	117	224.50	10	**1167**
A-n55-k9	1069.340	6	18.18	1069.340	6	18.29	1069.340	6	20.20	1067.332	9	6.48	1067.613	9	13.43	8	**1073**
A-n60-k9	1345.680	41	196.62	1345.680	41	202.67	1345.680	61	301.59	1344.408	584	378.53	1345.254	646	971.10	8	**1354**
A-n61-k9	1024.588	116	434.84	1024.604	119	228.98	1024.588	105	372.35	1022.362	386	292.07	1023.284	-	-.-	10	**1034**
A-n62-k8	1281.998	45	2815.95	1281.986	59	589.54	1282.068	41	488.92	1280.326	186	248.78	1280.875	-	-.-	14	**1288**
A-n63-k9	1610.813	6	45.00	1610.813	6	45.85	1610.813	6	60.34	1606.438	161	134.62	1608.584	23	46.59	9	**1616**
A-n63-k10	1303.870	38	107.40	1303.870	36	102.25	1303.870	28	102.39	1299.080	1305	920.00	1302.773	-	-.-	10	**1314**
A-n65-k9	1169.006	8	34.21	1169.006	8	34.23	1168.357	22	72.37	1166.637	43	41.79	1168.324	15	35.00	9	**1174**
A-n69-k9	1145.224	235	934.14	1145.224	199	712.29	1145.264	171	780.15	1140.720	1229	1456.44	1143.529	709	1459.81	9	**1159**
B-n39-k5	549.	1	36.23	549.	1	29.34	549.	1	84.70	549.	1	1.68	549.	1	4.00	11	**549**
B-n41-k6	828.833	3	4.81	828.833	5	4.92	828.833	3	5.73	828.350	3	1.93	828.586	3	4.79	8	**829**
B-n43-k6	737.600	12	24.75	737.600	12	23.53	737.600	12	27.28	736.957	43	17.65	737.145	37	33.87	8	**742**
B-n44-k7	909.	1	2.57	909.	1	2.59	909.	1	3.10	909.	1	1.54	909.	1	2.34	9	**909**
B-n45-k5	751.	1	42.25	751.	1	42.90	751.	1	54.93	749.923	5	8.92	750.003	3	9.42	10	**751**
B-n45-k6	678.	1	35.03	678.	1	30.35	678.	1	5.50	677.502	3	4.54	677.512	6	11.21	10	**678**
B-n50-k7	741.	1	8.09	741.	1	7.54	741.	1	9.56	741.	4	2.56	741.	2	5.06	11	**741**
B-n50-k8	1303.773	40	158.78	1303.800	62	173.04	1303.766	74	242.90	1294.944	92571	51564.60	1302.007	528	676.09	9	**1312**
B-n51-k7	1027.244	39	101.37	1027.244	31	82.15	1027.244	25	90.29	1026.367	53	42.48	1026.609	71	102.95	9	**1032**
B-n52-k7	746.500	4	59.60	746.500	5	36.32	746.500	3	27.68	746.411	4	7.48	746.375	5	12.76	11	**747**
B-n56-k7	705.000	6	274.85	705.000	7	349.14	705.000	12	575.50	705.000	8	9.59	705.000	9	28.50	11	**707**

Continua na próxima página...

Tabela A.1: *Geração de rotas elementares para o* CVRP

	Alg1. Cap. 3			Alg2. Cap. 4			Alg3. Cap. 4			$\delta = 3$			$\delta = 4$				
Instância	L_1	N_1	T_1	L_2	N_2	T_2	L_3	N_3	T_3	L_4	N_4	T_4	L_5	N_5	T_5	M_r	C^*
B-n57-k7	1153.	1	7.32	1153.	1	7.32	1153.	1	10.56	1151.647	7	44.93	1152.279	2	42.67	10	**1153**
B-n57-k9	1596.170	4	28.87	1596.170	4	29.50	1596.170	4	34.31	1595.975	17	15.53	1596.018	9	17.81	7	**1598**
B-n64-k9	860.482	3	34.73	861.	1	22.92	860.482	3	45.89	860.391	4	11.78	860.416	3	20.50	9	**861**
B-n66-k9	1309.671	34	4180.05	1309.602	132	4215.14	1309.302	51	1045.72	1307.330	1593	1889.42	1308.410	-	-.-	11	**1316**
B-n67-k10	1028.049	38	1293.03	1028.049	39	900.62	1028.049	31	243.25	1026.806	485	441.09	1027.167	-	-.-	9	**1032**
B-n78-k10	1217.574	45	4673.88	1217.413	59	4190.89	1217.544	49	1338.94	1214.734	254	404.53	1215.824	264	933.60	11	**1221**
E-n51-k5	519.468	3	31.07	519.468	3	30.85	519.468	3	36.54	519.058	4	7.57	519.118	6	25.75	11	**521**
E-n76-k8	727.120	301	11205.20	727.149	376	12094.50	727.121	355	23993.50	726.460	1504	4267.89	726.669	-	-.-	12	**735**
E-n76-k10	818.008	1461	11593.30	818.008	1545	9693.89	818.008	1195	25152.50	816.848	6082	6953.23	817.308	-	-.-	10	**830**
E-n76-k14	1007.501	3205	6695.72	1007.501	2904	5931.47	1007.501	3360	10534.20	1006.519	19853	9457.38	1007.440	9144	10902.30	8	**1021**
E-n101-k14	1055.009	3142	49398.10	1055.009	3470	41039.30	1055.005	4500	55456.50	1053.542	47779	84296.40	1053.842	-	-.-	11	**1067**
P-n40-k5	458.	1	5.62	458.	1	5.70	458.	1	5.76	457.666	2	1.40	457.500	2	2.71	9	**458**
P-n45-k5	507.315	8	26.96	507.315	8	27.56	507.315	8	30.00	506.510	26	14.71	506.949	13	23.65	11	**510**
P-n50-k7	551.678	8	23.98	551.678	8	24.09	551.678	8	25.04	551.410	8	4.29	551.552	16	17.85	9	**554**
P-n50-k8	618.029	234	358.40	618.029	214	301.98	618.029	210	416.95	616.228	878	332.70	616.873	-	-.-	8	**631**
P-n50-k10	689.580	41	29.59	689.580	41	29.56	689.580	41	32.96	689.322	104	15.67	689.580	51	19.25	7	**696**
P-n51-k10	736.236	15	14.29	736.236	15	14.32	736.236	15	15.51	735.063	29	6.17	735.299	20	10.51	6	**741**
P-n55-k7	559.840	129	375.70	559.840	197	541.09	559.840	202	738.45	558.305	531	360.92	558.688	-	-.-	11	**568**
P-n55-k8	580.970	117	253.18	580.970	91	217.79	580.970	84	251.48	580.048	254	123.09	580.483	211	275.43	10	**588**
P-n55-k10	681.835	609	872.96	681.835	609	874.28	681.835	543	918.37	681.299	1786	471.82	681.312	2630	1715.25	7	**694**
P-n55-k15	972.845	187	124.56	972.845	187	124.26	972.845	187	134.45	972.844	407	64.87	972.679	609	211.20	5	**989**
P-n60-k10	739.693	16	26.60	739.693	16	26.57	739.693	16	28.17	738.674	54	18.89	739.010	66	56.60	7	**744**
P-n60-k15	963.585	18	15.09	963.585	18	15.06	963.585	18	16.48	962.844	56	8.59	963.472	31	11.76	5	**968**
P-n65-k10	788.279	15	29.53	788.279	15	29.56	788.252	12	30.00	785.898	72	31.17	786.931	31	42.68	9	**792**
P-n70-k10	815.400	1351	4312.25	815.400	1521	4629.00	815.400	1547	6536.44	814.373	6802	4962.28	814.775	-	-.-	9	**827**

Tabela A.2: *Geração de rotas elementares para o* CVRP

Instância	L	T_{raiz}	T_{total}	C^*
A-n64-k9	1390.016	14.62	18677.80	1401
A-n80-k10	1756.692	149.03	580628.00	1763
B-n63-k10	1487.425	12.54	39391.40	1496
B-n68-k9	1263.919	59.37	-	1272
E-n76-k7	671.937	97.73	208187.13	682
E-n101-k8	804.988	210.73	-	815
M-n121-k7	-	-	-	1034

Tabela A.3: *Branching com enumeração de rotas*

Instância	N_1	T_1	N_2	T_2
A-n63-k10	565	6608	1	463
A-n69-k9	651	9470	1	578
A-n64-k9	773	10786	1	631
A-n80-k10	101	2732	1	683
B-n50-k8	1839	11750	131	2103
E-n76-k7	3883	91930	605	24947
E-n76-k8	483	7744	39	1528
E-n76-k10	4399	65487	73	3475
E-n76-k14	7405	65019	1	211

Tabela A.4: *Enumeração de rotas elementares*

Instância	G	G_{max}	R	T
B-n50-k8	7.722	7.722	330682	22
B-n68-k9	5.854	5.854	287139	16
E-n76-k7	9.271	9.271	4315884	125
E-n76-k8	5.532	5.532	216091	9
E-n76-k10	9.017	9.017	200258	7
E-n101-k8	7.149	5.361	10690286	231
E-n101-k14	9.160	9.160	961018	24