Este exemplar corresponde à redação final da Tese/Dissertação devidamente corrigida e defendida por: Ricardo Scachetti Pereira

e aprovada pela Banca Examinadora.

Campinas, 11 de Janeiro de 2010

COORDENADOR DE PÓS-GRADUAÇÃO
CPG-IC

Algoritmos Combinatórios para a Logística de Distribuição

Ricardo Scachetti Pereira

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Algoritmos Combinatórios para a Logística de Distribuição

Ricardo Scachetti Pereira[1]

Outubro de 1999

Banca Examinadora:

- Prof. Dr. Cid Carvalho de Souza
  Instituto de Computação - Unicamp (Orientador)
- Prof. Dr. Geraldo Robson Mateus
  Departamento de Ciência da Computação - UFMG
- Prof. Dr. Geovane Cayres Magalhães
  Fundação CPqD / Instituto de Computação - Unicamp
- Profa. Dra. Claudia M. Bauzer Medeiros
  Instituto de Computação - Unicamp (Suplente)

[1]Este trabalho foi financiado pelo CNPq, CAPES e FAPESP (processo 98/01183-0).

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Pereira, Ricardo Scachetti

P414a Algoritmos combinatórios para a logística de distribuição / Ricardo Scachetti Pereira -- Campinas, [S.P. :s.n.], 1999.

Orientador : Souza, Cid Carvalho de

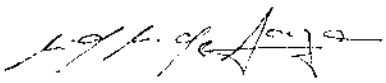
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Otimização combinatória. I. Souza, Cid Carvalho. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Algoritmos Combinatórios para a Logística de Distribuição

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Ricardo Scachetti Pereira e aprovada pela Banca Examinadora.

Campinas, 14 de outubro de 1999.

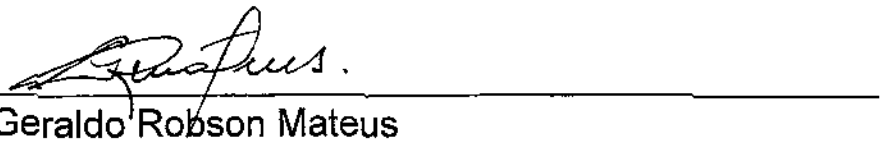


Prof. Dr. Cid Carvalho de Souza
Instituto de Computação - Unicamp (Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

TERMO DE APROVAÇÃO

Tese defendida e aprovada em 14 de setembro de 1999, pela Banca Examinadora composta pelos Professores Doutores:



Prof. Dr. Geraldo Robson Mateus
DCC - UFMG

Prof. Dr. Geovane Cayres Magalhães
IC - UNICAMP

Prof. Dr. Cid Carvalho de Souza
IC - UNICAMP

Para minha esposa, Larissa e para meus pais, Áurea e Idário.

Agradecimentos

Gostaria de expressar toda minha gratidão a todas as pessoas que, direta ou indiretamente, contribuiram de alguma forma para a realização deste trabalho. Meus sinceros agradecimentos:

- À Larissa, minha esposa.
- Aos meus pais, Áurea e Idário.
- Ao Cid, meu orientador e grande amigo.
- Ao pessoal do *Grupo de Otimização Aplicada*, em especial ao Tallys, pela grande contribuição, principalmente pela participação ativa na definição do *PE*, no início do trabalho.
- Ao pessoal do *Grupo de Pesquisas em Bancos de Dados* pela consultoria permantente em SIG, principalmente à Claudia, pela sua contribuição.
- Aos funcionários do *IC*, principalmente da secretaria, Vera e Daniel, e da administração de sistemas, Roland e Oliva.
- Às instituições que financiaram este projeto: CNPq, CAPES e FAPESP.

Resumo

Neste trabalho são estudados dois problemas combinatórios que ocorrem ao utilizar uma abordagem hierárquica para definir a estratégia a ser adotada na logística de distribuição de revistas. Tipicamente, a primeira fase da logística envolve a definição da região geográfica que será alocada a cada entregador. O problema de definir estas regiões é denominado **problema do distritamento** (PD). Na segunda fase da logística, para cada região de entrega, é preciso encontrar uma rota que minimize a distância percorrida pelo entregador. Esta rota deve satisfazer tanto a restrição de capacidade de carga do entregador quanto as restrições de fluxo de revistas, considerando-se as demandas dos pontos de entrega e o estoque nos depósitos. O problema combinatório referente a esta fase é denominado o **problema da entrega de revistas** (PE).

Neste trabalho propõe-se algoritmos heurísticos para ambos os problemas acima, que são modelados por meio de grafos. Para o **problema da entrega de revistas** é proposto ainda um algoritmo exato do tipo *branch-and-cut*. Este algoritmo está baseado em uma formulação de *Programação Linear Inteira* e em desigualdades válidas fortes adaptadas dos problemas de roteamento de veículos e de fluxo em redes com custos fixos.

Além disso, propõe-se um *Sistema Espacial de Apoio à Decisão* (SEAD) baseado em um *Sistema de Informação Geográfica* (SIG) para a Logística de Distribuição de revistas que pressupõe a integração das soluções dos problemas do distritamento e da entrega.

Todos algoritmos propostos são implementados e testados para um amplo conjunto de instâncias. Um protótipo do SEAD proposto é implementado através da integração das heurísticas ao SIG ArcView.

Abstract

In this work we study two combinatorial problems that arise when a hierarchical approach is used to define the strategy to be adopted in the logistics of magazine distribution. Typically, the first phase of the logistics involves the definition of the geographical region to be assigned to each deliverman. The problem of defining such regions is called **the district determination problem**. In the second phase of the logistics, to each deliver region, we have to find a route that minimizes the distance traversed by the deliverman. This route must satisfy both the deliverman capacity and the magazine flow constraints, given the demands in the delivery points and the stocks in the depots. The combinatorial problem related to this phase is called **the magazine delivery problem**.

In this work we propose heuristic algorithms for both problems above, which are modeled with graphs. For the **the magazine delivery problem** we also propose an exact *branch-and-cut* algorithm. This algorithm is based on an *Integer Programming* formulation and on strong valid inequalities adapted from the vehicle routing and fixed-charge network problems.

Besides, we propose a *Spatial Decision Support System* (SDSS) based on a *Geographical Information System* (GIS) for the logistics of magazine distribution that assumes the integration of the solutions of the district determination and magazine delivery problems.

All the algorithms proposed are implemented and tested over a wide set of instances. A prototype of the proposed SDSS is implemented via the integration of the heuristics to the GIS ArcView.

Conteúdo

Lista de Tabelas

Lista de Figuras

Capítulo 1

Introdução

A logística de distribuição é de importância vital para um grande conjunto de empresas. O modo com que estas atendem a seus clientes traz grande impacto sobre sua satisfação, além de influenciar de forma decisiva seus custos operacionais. Trata-se portanto de uma questão de estratégia competitiva manter os serviços de distribuição o mais eficientes possível.

No contexto da entrega de exemplares de revistas e jornais a assinantes, o processo de logística de distribuição consiste em resolver o seguinte problema: definido o produto que se pretende distribuir (uma determinada revista ou jornal), e a localização geográfica dos assinantes (em princípio denotada por seus endereços), deseja-se decidir como será realizada a distribuição de forma a racionalizar a alocação dos recursos disponíveis.

Há várias maneiras de se resolver este problema. Um exemplo trivial, muito utilizado pelas grandes editoras até os anos 80, é a entrega através do correio. Para realizar a entrega utilizando-se este método, era necessário apenas afixar uma etiqueta contendo o endereço do assinante em cada revista e levar os exemplares à agência postal mais próxima. Esse método é ainda hoje muito utilizado para distribuição de mala-direta.

Dado o grande número de assinantes que as editoras têm atualmente, hoje na ordem de grandeza de centenas de milhares até alguns milhões, a aplicação de métodos mais eficientes se faz necessária.

Nesta dissertação é apresentada uma abordagem computacional para a resolver o problema da logística de distribuição de exemplares de jornais e revistas a assinantes. É proposto um sistema baseado em um *Sistema de Informação Geográfica* que dispõe de ferramentas automáticas para a geração de soluções.

Nessa abordagem, o problema da logística de distribuição é resolvido de forma hierárquica sendo o processo composto de duas fases principais: o **distritamento** e o **roteamento**. Mais adiante será caracterizado o que compreende exatamente cada uma dessas fases. Além disso, será mostrado também que os problemas a serem resolvidos em ambas as fases da logística de distribuição podem ser modelados como problemas de otimização em grafos. A resolução computacional desses problemas é o tema central desta dissertação.

Na fase do **distritamento** (fase 1) deseja-se alocar um conjunto de entregadores de forma a

permitir que a distribuição seja realizada em toda a região de atendimento. Os dados necessários para a execução desta tarefa são: (1) os mapas da região atendida, (2) a localização de cada cliente na região considerada e (3) estatísticas sobre o tempo de distribuição gasto (ou estimado) durante a entrega em cada rua. Cada um desses componentes será melhor caracterizado a seguir.

A unidade geográfica básica sobre a qual se executa o distritamento é denominada **zona de distribuição**. Corresponde, em geral, à região pertencente a uma pequena cidade, um bairro ou parte dele.

O primeiro componente dos dados de entrada para a fase de distritamento compreende os mapas da zona de distribuição considerada. Estes mapas representam a rede de logradouros[1] através do paradigma de *centerlines*, isto é, um logradouro é abstraído por uma linha poligonal em que os segmentos de reta são geralmente determinados por intersecções entre ruas, ou suas extremidades e sua largura não é relevante. Cada segmento de reta da linha poligonal representa um **segmento de logradouro**.

No caso da entrega de revistas, a unidade básica de distribuição, a ser alocada a um único entregador é denominada **trecho**. Um trecho composto por um conjunto conexo de segmentos de logradouros que representam uma única rua, parte de uma longa avenida, ou até mesmo um dos lados de uma avenida.

A localização dos clientes é obtida através de um processo chamado *espacialização*, em que os clientes a serem atendidos (os assinantes) são posicionados na rede de ruas, através de seus endereços, em um dos lados da *centerline* que representa a rua em que estão localizados. O ponto no mapa que representa a localização de cada cliente denomina-se **ponto de entrega**. Vários clientes podem ser agrupados num mesmo *ponto de entrega* caso compartilhem do mesmo endereço, como por exemplo, os assinantes que residem num mesmo edifício.

Finalmente, as estatísticas sobre o tempo de entrega são expressas através do **tempo médio de entrega por trecho**, que corresponde ao tempo total, expresso em minutos, gasto por um entregador para distribuir os exemplares de revistas naquele trecho. Esse valor é composto por uma soma de quatro parcelas: (1) *tempo de percorrida* que corresponde ao tempo gasto para o entregador percorrer todo o trecho; (2) *tempo de parada* que corresponde ao tempo gasto em paradas em pontos de entrega; (3) *tempo de entrega* que corresponde ao tempo gasto para a entrega de cada exemplar; (4) *tempo improdutivo* que corresponde ao tempo que o entregador gasta para ordenar as revistas a serem entregues e se deslocar da empresa até o ponto onde este iniciará a distribuição das revistas. No contexto deste trabalho, a última parcela é considerada constante para todos os entregadores, independentemente do ponto de início dos seus trajetos.

Os *tempos médios de entrega por trecho* variam em função do veículo utilizado durante a distribuição. Por exemplo, o *tempo de percorrida* é significativamente maior na entrega a pé do que na entrega usando motocicleta, pois a velocidade deste último é muito superior à do primeiro. Por outro lado, o *tempo de parada* na entrega de motocicleta é maior que o da entrega a pé, pois a cada parada, o entregador tem que estacionar e desligar o veículo antes de fazer a entrega, tempo que não é gasto quando da entrega a pé.

[1] Um **logradouro** é a denominação genérica para uma rua, avenida, alameda, praça, etc.

Baseando-se nestes dados de entrada, a fase do *distritamento* consiste em determinar o conjunto conexo de trechos que será atendido por cada entregador alocado, de forma a atender toda a região considerada. Este conjunto conexo de trechos de responsabilidade de um único entregador, é denominado **distrito**. Cada distrito tem uma **carga de trabalho** que corresponde à soma dos tempos médios de entrega dos trechos naquele distrito.

Dois objetivos devem ser atingidos durante o *distritamento*: (1) minimização do número de distritos, e conseqüentemente o número de entregadores e (2) balanceamento das *cargas de trabalho* dos distritos gerados.

Duas restrições são consideradas neste problema. A primeira delas restringe a carga de trabalho dos distritos a um limite máximo pré-estabelecido, ou seja, nenhum distrito pode ter carga de trabalho superior a um valor pré-fixado. A segunda restrição diz respeito à conexidade dos distritos. Um distrito deve ser necessariamente conexo pois o entregador não deve ser obrigado a passar por ruas que não entrega nenhum exemplar (evitando aumentar seu tempo ocioso). Um distrito é dito *conexo* se é possível percorrer todos os trechos que o compõem, sem a necessidade de se utilizar trechos que não se encontram no próprio distrito.

Este problema, denominado *Problema do Distritamento* ou simplesmente *PD*, pode ser modelado como um problema de otimização em grafos, conforme apresentado na próxima seção.

A fase de *distritamento*, descrita acima, aplicada na logística de distribuição de revistas é uma adaptação de uma das etapas da logística de distribuição de correspondências utilizada na *ECT - Empresa Brasileira de Correios e Telégrafos*, que forneceu dados e informações sobre o distritamento. De uma forma geral, todos os resultados deste trabalho para o tratamento do *PD* pode ser aplicado, sem modificações, para a logística de distribuição da *ECT*. As diferenças entre estes dois processos de logística começam a surgir a partir da segunda fase do processo, o **roteamento**, que passará a ser descrito a seguir, apenas para o caso da distribuição de revistas.

Na fase do **roteamento** (fase 2) são considerados os distritos gerados na fase do distritamento (fase 1) individualmente. São dados: o mapa do distrito considerado, o conjunto de pontos de entrega (localização dos clientes), e o conjunto de depósitos localizados no distrito. No caso da particular da logística de distribuição de revistas, os depósitos são pequenos locais de comércio e portarias de prédios e, portanto, podem ser facilmente criados através de acordos com comerciantes e síndicos. Deste modo, após o distritamento, tipicamente a empresa distribuidora negocia estes acordos e, como resultado desta negociação, ficam estabelecidas as localizações exatas dos depósitos que poderão vir a ser utilizados durante o roteamento.

Vale observar que, caso os depósitos fossem escolhidos antes do distritamento, seria necessário levar em conta ainda mais uma restrição ao construir os distritos para garantir que o estoque nos depósitos de um distrito fosse suficiente para atender à demanda dos trechos contidos naquele distrito. Evidentemente, isto exigiria alterações no algoritmo do distritamento.

O objetivo do roteamento em um distrito é encontrar uma rota que permita ao entregador distribuir todas as revistas aos assinantes, minimizando a distância total percorrida. Ou seja, procura-se uma seqüência ordenada de pontos de entrega e de depósitos que será percorrida pelo entregador durante a distribuição dos exemplares.

Duas restrições importantes devem ser respeitadas. Em primeiro lugar, dado que o número

de assinantes num distrito é grande o suficiente para impedir que o entregador saia de sua empresa munido de todas as revistas que vai precisar, deve-se respeitar a capacidade de carga do entregador. Ou seja, em nenhum momento da distribuição, o entregador deve ser obrigado a carregar mais revistas do que a quantidade máxima pré-determinada. Esse valor depende do veículo utilizado para a entrega. No caso da entrega a pé, por exemplo, o entregador não deve carregar mais do que $8kg$, o que corresponde a cerca de 40 revistas. Na entrega de motocicleta, esse valor sobe para mais de $50kg$.

Conseqüentemente tornam-se necessárias uma ou mais paradas para "reabastecimento" nos depósitos. O estoque desses depósitos é previamente suprido por um veículo. Embora os depósitos sejam definidos *a priori*, não existe qualquer restrição quanto ao número de vezes que o entregador deve passar por cada um, fazendo com que, em uma solução válida, alguns depósitos possam ser utilizados uma ou mais vezes para o reabastecimento, enquanto que outros podem ser evitados durante a distribuição.

Outra restrição a ser respeitada é a restrição de capacidade nos depósitos. No caso da entrega de revistas, geralmente os depósitos têm dimensões limitadas o que impõe restrições à quantidade de revistas neles armazenadas.

Este problema, denominado *Problema da Entrega de Revistas* ou simplesmente *PE*, também pode ser modelado como um problema de otimização em grafos, conforme apresentado na seção a seguir.

Um diagrama funcional da abordagem proposta é mostrada na figura 1.1.

1.1 Formulação do *PD* como um problema de otimização em grafos

Mostra-se a seguir como modelar o *PD* usando um grafo não-direcionado $G = (V, E)$. Primeiro os trechos são identificados no mapa da cidade.

Em seguida, associa-se a cada trecho um nó em V. Para cada nó $u \in V$, é atribuído um peso t_u igual ao tempo médio de entrega do trecho correspondente. Resta agora definir quando uma aresta está presente em E. Para tanto, se dois trechos correspondendo a dois nós u e v em V são tais que suas linhas poligonais têm um ponto em comum (extremidade de um segmento de reta de ambas as linhas poligonais) então a aresta $(u, v) \in E$.

Se W denota a carga diária máxima de um carteiro, então a partição $(V_1, V_2, \ldots, V_k)$ dos nós de V é dita ser W-conexa se: (i) $w_j = \sum_{u \in V_j} t_u \leq W$ e (ii) o subgrafo induzido por V_j é conexo para todo $j \in 1, \ldots, k$.

Claramente, toda partição W-conexa de G corresponde a uma partição de distritos válida da região de distribuição considerada. Assim, objetivo a ser alcançado é encontrar uma partição W-conexa de G de cardinalidade mínima, tal que $\lambda = max(w_j) - min(w_i), i, j \in (1, \ldots, k)$ também seja mínimo.

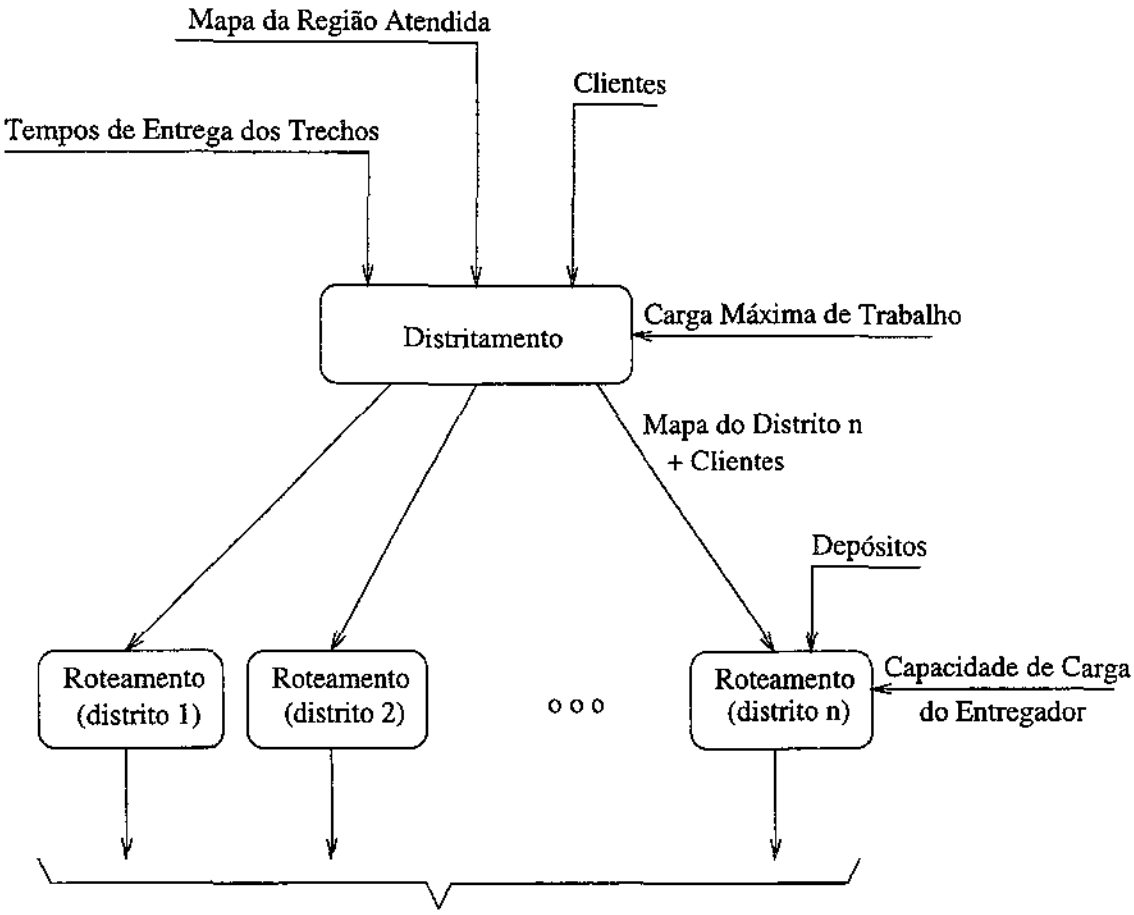


Figura 1.1: Diagrama funcional da logística de distribuição

1.2 Formulação do *PE* como um problema de otimização em grafos

Formalmente, o *PE* pode ser formulado como um problema de otimização em grafos como se segue.

Seja $G = (V, A)$ um grafo orientado, completo e sem laços. A cada vértice de V associa-se um ponto de entrega de revistas ou um depósito. Em seguida, atribui-se a cada arco $a = (i, j)$ ($a \in A$ e $i, j \in V$) um peso w_a que corresponde à distância entre pontos no mapa referentes aos vértices i e j em G. Os vértices de V estão particionados em dois subconjuntos P e D de modo que a cada vértice $p \in P$ associa-se um depósito, com estoque $d_p < 0$ e a cada vértice $d \in D$ um ponto de entrega, com demanda $d_d > 0$. Nos depósitos o estoque representa a quantidade máxima de revistas que podem ser armazenadas, enquanto que nos pontos de entrega, a demanda representa o número de exemplares a serem distribuídos. A quantidade de revistas transportada de um vértice i para outro vértice j é representada como um fluxo de i para j, denotado por y_{ij}.

A constante c representa a capacidade máxima de fluxo suportada por cada arco de G (número máximo de revistas transportadas pelo entregador). Deseja-se então, encontrar um ciclo C em G e os valores de fluxo y_{ij} que satisfaçam às seguintes condições:

- a conservação de fluxo nos vértices correspondentes aos pontos de entrega deve ser respeitada, ou seja, a soma do fluxo que entra num nó deve ser igual a soma do fluxo que sai, considerando-se os valores das demandas;
- o ciclo C deve conter todos os pontos de entrega;
- todas as capacidades de fluxo dos arcos devem ser respeitadas, ou seja, $y_{ij} \leq c, \forall (i,j) \in A$
- todo depósito $i \in P$ deve ter sua restrição de capacidade respeitada, ou seja, não podem ser coletadas mais revistas do que o estoque (dado por $|d_i|$) existente em cada depósito;
- $\sum_{a_i \in C} w_{a_i}$ deve ser mínima.

É interessante notar que, originalmente, o *PE* está definido sobre uma porção de um mapa que é conexa, ou seja, de cada ponto do mapa, é sempre possível chegar a um outro ponto do mapa seguindo os trechos nele representados. Além disso, o menor caminho entre dois pontos quaisquer do mapa bem como o seu comprimento podem ser obtidos facilmente a partir da localização dos mesmos.

O que se pretende ao redefinir o problema sobre um grafo completo é a adaptação, sem necessidade de grandes alterações, de algoritmos existentes para outros problemas de roteamento ao caso particular do *PE*, De fato, como será visto no capítulo 2, vários dos algoritmos projetados nesta dissertação para tratar o *PE* baseiam-se em algoritmos para problemas similares, como o problema do caixeiro viajante e do roteamento de veículos, em que são utilizados grafos completos.

Além disso, como será mostrado no capítulo 3, o uso de um grafo completo torna bem mais simples o modelo do *PE* usando *Programação Linear Inteira*, sem que haja perdas relevantes em termos de utilização de memória ou tempo de processamento.

Tendo em vista o contexto apresentado anteriormente, os objetivos desta dissertação são:

1. Resolver computacionalmente os problemas da logística de distribuição descritos anteriormente, de forma eficiente através de algoritmos heurísticos;
2. Realizar um estudo teórico a respeito do *PE* baseado em *Programação Linear Inteira*, resultando no projeto e na implementação de um algoritmo exato para o problema[2];
3. Propor e implementar um sistema computacional que ofereça ao especialista na logística de distribuição, ferramentas que auxiliem o processo decisório, de modo a estabelecer um contexto para o uso dos algoritmos desenvolvidos neste trabalho.

[2] O algoritmo exato servirá também para orientar a análise da eficácia dos algoritmos heurísticos. Seus resultados serão tomados como referência na comparação dos valores obtidos por estes últimos.

O restante desta dissertação está estruturado da seguinte forma. O capítulo 2 apresenta os dois algoritmos heurísticos desenvolvidos para resolver ambos os problemas tratados neste trabalho (*PD* e *PE*). Nesse capítulo faz-se uma descrição geral da abordagem heurística para a resolução de problemas combinatórios, enfatizando a técnica *GRASP*, utilizada no desenvolvimento de ambos os algoritmos citados. O capítulo 3 traz a descrição de um algoritmo exato para o *PE* baseado numa formulação em *Programação Linear Inteira*. Nesse capítulo é feita uma introdução a respeito da teoria combinatória poliédrica, fundamental para a compreensão do estudo desenvolvido, e do algoritmo exato implementado. O capítulo 4 apresenta a proposta de um *Sistema Espacial de Apoio à Decisão* capaz de dar suporte ao usuário durante a logística de distribuição, em que as heurísticas descritas no capítulo 2 desempenham papel fundamental no processo decisório. Esse capítulo descreve também o protótipo do sistema implementado sobre um *Sistema de Informação Geográfica* comercial. O capítulo 5 descreve a metodologia empregada nos testes computacionais dos três algoritmos desenvolvidos neste trabalho e suas respectivas análises. O capítulo 6 traz as conclusões e considerações sobre os resultados do presente trabalho, apresentando suas contribuições e possíveis extensões. Finalmente, o apêndice A traz um glossário com os principais termos utilizados ao longo do trabalho.

Capítulo 2

Heurísticas

Neste capítulo, serão apresentados os algoritmos heurísticos desenvolvidos para a resolução dos problemas de otimização combinatória identificados na logística de distribuição de revistas, descritos no capítulo 1. Em primeiro lugar, será apresentada uma pequena introdução à metodologia de desenvolvimento de algoritmos heurísticos. A seção 2.1 traz algumas noções básicas a respeito do paradigma *GRASP*, utilizado no desenvolvimento dos algoritmos heurísticos apresentados neste trabalho. As seções 2.2 e 2.3 trazem a descrição dos algoritmos para o *problema do distritamento* (*PD*) e para o *problema da entrega de revistas* (*PE*), respectivamente.

Ao tratar problemas computacionais, procura-se desenvolver soluções eficientes, que possam resolver os primeiros de forma a utilizar recursos como memória e tempo de CPU de maneira mais racional possível. Por eficiente, entende-se aquele algoritmo que tem seu tempo de execução limitado superiormente por um polinômio no tamanho da entrada.

Entretanto, existem problemas para os quais acredita-se que não existam soluções "eficientes". Uma classe importante desses problemas é representada pelos problemas *NP-completos*. Para mais detalhes sobre a teoria de *NP-completude* refira-se a [Cor90, Man89].

No caso do presente trabalho, ambos os problemas tratados são considerados "difíceis".

O *PD* pode ser reduzido ao problema do empacotamento de latas (bin packing problem [GJ79]), que é $\mathcal{NP}$*-difícil*, através de uma simples redução polinomial.

O *PE* também é um problema $\mathcal{NP}$*-difícil*. A prova é apresentada a seguir.

Considere $G = (V, A)$ um grafo orientado e sem laços. Ao conjunto de arcos $a \in A$ estão associados pesos w_a. O problema de se encontrar um *caminho hamiltoniano* de custo mínimo em G, que é um problema $\mathcal{NP}$*-difícil*, pode ser reduzido ao *PE* sobre um grafo $G' = (V', A')$ através da seguinte redução polinomial.

Primeiro, os vértices de $v \in V$ são transformados em pontos de entrega e adicionados ao conjunto V'. Estes vértices recebem a demanda unitária $d_v = 1$. É criado um depósito u, que é também adicionado ao conjunto de vértices V'. Ao depósito é atribuído estoque $d_u \leq -|V|$ (o estoque negativo é uma convenção adotada para as instâncias do *PE*). O conjunto de arcos A' contém os mesmos arcos de A, mais os arcos (u, v) e (v, u), ligando o depósito aos pontos de entrega em V' e vice-versa, adicionando-se $2|V|$ arcos. Esses arcos que saem e entram no

depósito recebem custos $w_{uv} = w_{vu} = \sum_{a \in A} w_a + 1$. Os pesos w_a dos arcos $a \in A$ são mantidos em A'. A cada arco $a \in A'$ é atribuída a capacidade de fluxo $y_a = |V|$ que corresponde à soma das demandas unitárias dos pontos de coleta. A capacidade do entregador assume o mesmo valor $c = |V|$.

Resolver o problema do *PE* sobre o grafo $G' = (V', A')$ é equivalente a encontrar o *caminho hamiltoniano* de custo mínimo sobre G. É fácil ver que a redução apresentada acima tem complexidade polinomial linear $O(|V| + |A|)$. Assim, prova-se que o *PE* é um problema *$\mathcal{NP}$-difícil.*

Como ambos os problemas tratados nesta dissertação são *$\mathcal{NP}$-difíceis* há poucas esperanças de se encontrar algoritmos polinomiais para resolvê-los.

Entretanto, para grande parte dos problemas combinatórios "difíceis", as heurísticas apresentam uma solução de compromisso entre eficiência na utilização de recursos computacionais e a qualidade da solução.

De um modo geral, pode-se definir uma *heurística* como sendo um método de resolução de problemas que busca boas soluções (próximas à otimalidade) a um custo computacional razoável, sendo, entretanto, incapaz de garantir sua otimalidade, e possivelmente, nem sua viabilidade. Usando este método, pode ser impossível também determinar quão próxima do valor ótimo está uma solução heurística [RSORS96].

A princípio, pode-se pensar em dois tipos de algoritmos heurísticos. O primeiro tipo é constituído pelos algoritmos construtivos, como por exemplo os algoritmos gulosos, em que parte-se da solução vazia e, iterativamente, constrói-se a solução final, elemento por elemento. Nessa classe de heurísticas, a qualidade das soluções obtidas depende principalmente do método de escolha do elemento que é adicionado à solução corrente a cada iteração.

O segundo tipo são as heurísticas de busca local as quais se baseiam no conceito de *vizinhança*. A **vizinhança** de uma solução S é o conjunto de soluções que podem ser atingidas a partir de S pela aplicação de um conjunto bem definido de operações sobre S.

As heurísticas de busca local, trabalham iterativamente, trocando a solução corrente por uma solução melhor presente na vizinhança da solução original. O processo termina quando não é encontrada nenhuma solução melhor que a corrente. A chave para a criação de boas heurísticas de busca local reside na escolha do conjunto de operações que definem a vizinhança das soluções, em técnicas eficientes de busca de soluções na vizinhança e na criação da solução inicial.

Entretanto, ambos os tipos de heurísticas falham freqüentemente pois são propensas a retornarem soluções que são ótimas apenas localmente. Para transpor esse problema, faz-se necessário o uso de técnicas mais elaboradas.

Como principais técnicas nesse grupo pode-se citar três: *Simulated Annealing* [JAMS89, JAM91, RSORS96], *Tabu Search* [Glo89, Glo90, RSORS96] e *Greedy Randomized Adaptative Search Procedure* (*GRASP*).

Esta última técnica foi escolhida para ser utilizada no projeto dos algoritmos heurísticos para a resolução dos problemas propostos no presente trabalho. Entre as principais razões que fundamentaram a escolha, pode-se citar:

- A facilidade de implementação eficiente do *GRASP*, principalmente quando já se dispõe de um algoritmo heurístico guloso para o problema combinatório em questão;
- Como conseqüência do item anterior, a reutilização de algoritmos já disponíveis na literatura é facilitada;
- Poucos parâmetros precisam ser definidos e ajustados;
- Foram relatados na literatura, vários casos de sucesso da implementação de técnicas *GRASP* para problemas combinatórios difíceis, como por exemplo, [LFE94], e outros citados por [FR95].

Por essa razão, na próxima seção será apresentada uma descrição mais aprofundada sobre a técnica *GRASP*.

2.1 Greedy Randomized Adaptative Search Procedure (GRASP)

Essa seção consiste basicamente de uma compilação dos idéias discutidas em [FR95] e apresenta os conceitos fundamentais da técnica *GRASP* utilizados no presente trabalho.

O *GRASP* é um método iterativo em que cada iteração consiste de duas fases principais: a *fase construtiva* e a *fase de busca local*, que serão melhor definidas mais adiante. Durante as iterações, a melhor solução encontrada é armazenada e é retornada como resultado, ao final da execução, quando a condição de parada é atingida.

Na figura 2.1, é mostrado um pseudo-código de um algoritmo *GRASP* genérico.

procedimento $GRASP\_GENERICO()$

1. $S \leftarrow \emptyset$
2. $S_{melhor} \leftarrow \emptyset$
3. $LEITURA\_DA\_INSTANCIA()$
4. **enquanto** (critério de parada não satisfeito) **faça**
   - 4.1. $S \leftarrow ALGORITMO\_CONSTRUTIVO\_RANDOMIZADO()$
   - 4.2. $S \leftarrow BUSCA\_LOCAL(S)$
   - 4.3. $ATUALIZA\_SOLUCAO(S, S_{melhor})$;
5. **fim_enquanto**
6. **retorna** S_{melhor}

fim_procedimento

Figura 2.1: Algoritmo GRASP genérico

O algoritmo começa com inicialização das variáveis nas linhas 1 e 2 e com a leitura da instância na linha 3. A variável S armazena uma solução viável para o problema e S_{melhor} armazena a melhor solução encontrada até o momento pelo algoritmo. A iteração do *GRASP* se dá nas linhas 4 até a 5. A fase de construção corresponde ao procedimento chamado na linha 4.1 e a fase de busca local corresponde ao procedimento da linha 4.2. A linha 4.3 apenas armazena a solução encontrada, caso esta seja melhor que aquelas encontradas até o momento. Ao se atingir o critério de parada, que em geral é a execução de um número predefinido de iterações, o algoritmo é encerrado e é retornada a solução com melhor valor da função objetivo encontrada (linha 6). A seguir são apresentados respectivamente os algoritmos das fases construtiva e de busca local.

procedimento $ALGORITMO\_CONSTRUTIVO\_RANDOMIZADO()$

1. $S \leftarrow \emptyset$
2. **enquanto** (construção da solução não terminada) **faça**
   - 2.1. $RCL \leftarrow CRIA\_RCL(k)$
   - 2.2. $r \leftarrow SELECIONA\_ELEMENTO(RCL)$
   - 2.3. $S \leftarrow S \cup \{r\}$
   - 2.4. $ADAPTA\_FUNCAO\_GULOSA()$
3. **fim_enquanto**

fim_procedimento

Figura 2.2: Fase construtiva do algoritmo GRASP genérico

procedimento $BUSCA\_LOCAL(P, N(P), S)$

1. **enquanto** (S não é ótimo local) **faça**
   - 1.1. encontre uma solução $T \in N(S)$ melhor que S;
   - 1.2. **faça** $S \leftarrow T$;
2. **fim_enquanto**
3. **retorna** S como ótimo local de P

fim_procedimento

Figura 2.3: Fase de busca local do algoritmo GRASP genérico

Na fase construtiva, uma solução viável é construída iterativamente, um elemento de cada

vez, a partir da solução vazia. Em cada iteração, o próximo elemento a ser inserido na solução é escolhido dentre uma lista de candidatos, ordenada pela chamada *função gulosa adaptativa*. Esta função mede o benefício trazido pela escolha de cada elemento à função objetivo. A heurística é *adaptativa* porque o benefício associado à escolha de cada elemento é atualizado a cada iteração da fase construtiva para refletir as alterações feitas pela escolha do elemento anterior. A componente probabilística de um *GRASP* está na escolha aleatória de um dos candidatos da lista, não necessariamente o melhor. Essa lista dos k melhores candidatos denomina-se **lista restrita de candidatos** (ou *RCL*, *Restricted Candidate List* em inglês). Usando este método de escolha a cada iteração da fase construtiva, é possível obter soluções distintas a cada iteração do *GRASP* sem comprometer necessariamente a qualidade das soluções obtidas.

A figura 2.2 mostra o algoritmo genérico para a fase construtiva do *GRASP*. Na linha 1 a solução a ser construída é inicializada. As linha 2 até 3 são repetidas até que a solução esteja construída. Na linha 2.1 a lista restrita de candidatos *RCL* de tamanho k é construída. Um candidato da lista é escolhido aleatoriamente na linha 2.2 e adicionado à solução corrente na linha 2.3. O efeito da inserção do elemento na linha 2.3 sobre o valor da *função gulosa adaptativa* para cada elemento remanescente na lista de candidatos é levado em consideração na linha 2.4, onde a função é "adaptada". Note que, na linha 2.4, a *função gulosa adaptativa* não está sendo modificada. O que ocorre é que o valor da função varia de acordo com o conjunto de elementos já inseridos na solução corrente. Ao se inserir um novo elemento, o valor da função para os elementos não alocados muda, e essa atualização é considerada como uma adaptação da função.

A solução retornada pelo procedimento construtivo é então submetida ao procedimento de busca local, na tentativa de obtenção de uma nova solução de melhor qualidade. Na busca local, cujo algoritmo é mostrado na figura 2.3, novas soluções são visitadas de maneira iterativa, substituindo-se a solução corrente por outra solução melhor pertencente à sua vizinhança (linhas 1.1 e 1.2). Essa fase termina quando nenhuma solução melhor que a atual é encontrada na vizinhança. A melhor solução encontrada, que é dita localmente ótima com respeito à vizinhança definida, é retornada (linha 3). $N(S)$ denota a vizinhança da solução S.

O sucesso das técnicas de *GRASP* na resolução de problemas combinatórios difíceis é relatada em [FR95].

O método baseia-se na idéia de que soluções gulosas têm uma qualidade razoável e que, às vezes, a qualidade se deteriora por conta de poucas escolhas erradas ao longo das iterações. Ou seja, a escolha localmente ótima realizada em uma iteração pode desviar o algoritmo de uma solução de boa qualidade. Contudo, usualmente o critério guloso de escolha é relativamente bom e melhor do que se fossem feitas escolhas completamente aleatórias. Assim, a maneira encontrada no método *GRASP* para aumentar as chances de um algoritmo guloso construir uma boa solução tenta combinar as vantagens da escolha gulosa tornando esta escolha mais flexível. Isto foi implementado através da *RCL*.

A construção de várias soluções desta forma, sendo posteriormente dadas como a entrada de um procedimento de busca local, só aumentam as chances de se encontrar uma boa solução.

Na verdade, a idéia de gerar várias soluções iniciais seguidas de uma boa busca local é antiga [Pap73]. O método *GRASP* simplesmente define uma forma sistemática de gerar boas soluções

iniciais.

A figura 2.4 mostra o comportamento típico das soluções obtidas por um algoritmo *GRASP* para um problema combinatório. Este gráfico mostra o número de soluções obtidas para cada valor da função objetivo para um número fixo de iterações quando $RCL = 1$ e $RCL > 1$.

Por exemplo, num problema de minimização, se a cardinalidade do RCL é limitada a 1, então apenas uma solução é gerada, fazendo com que a variância seja 0 (linha sólida apresentada na figura 2.4). Nesse caso, todas as N_1 soluções obtidas terão o mesmo valor V_2 da função objetivo.

Por outro lado, com uma boa *função gulosa adaptativa*, se a cardinalidade da RCL for aumentada, várias soluções distintas serão geradas, aumentando a variância da amostra. A distribuição típica esperada das soluções é denotada pela linha pontilhada na figura 2.4 [FR95]. Nesse caso, a média deve sofrer alguma degradação (movendo-se para o valor V_3), visto que a função gulosa adaptativa é agora sub-ótima. Entretando, a melhor solução encontrada tende a ser melhor que a média (valor V_1 na figura), uma vez que as soluções foram geradas com uma componente aleatória [FR95].

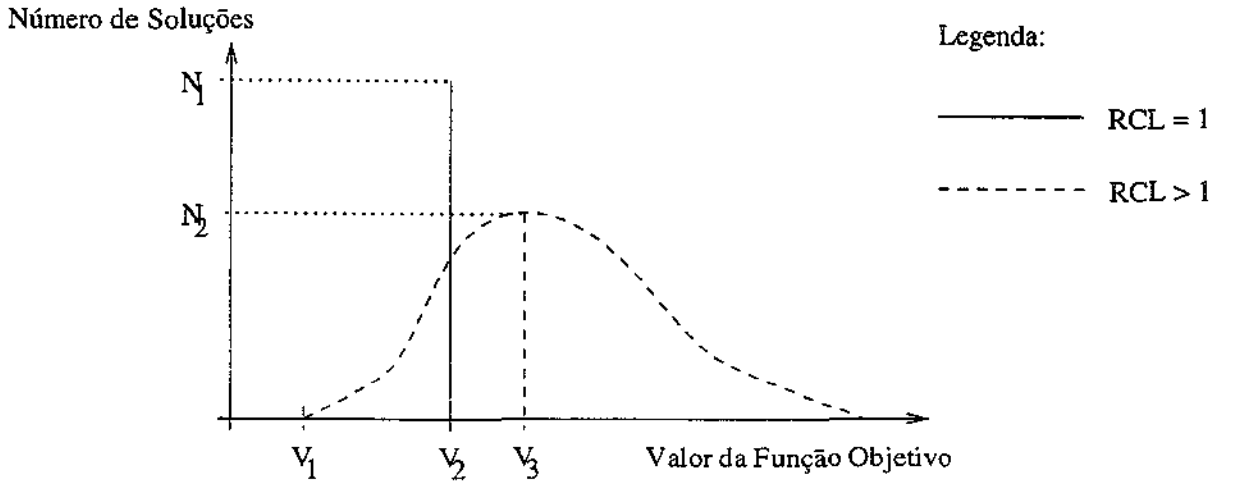


Figura 2.4: Distribuição esperada das soluções obtidas por um *GRASP* genérico para um problema de minimização

2.2 Algoritmo heurístico para o *PD*

Foi desenvolvido um algoritmo heurístico para resolver o *PD*, baseado no paradigma *GRASP*, mostrado na figura 2.5.

O algoritmo da figura 2.5 segue a estrutura apresentada pelo algoritmo *GRASP* genérico (figura 2.1). A fase construtiva corresponde ao procedimento chamado na linha 4.1. A fase de busca local é realizada no procedimento da linha 4.2.

Os algoritmos construtivo e de busca local são descritos nas seções a seguir.

procedimento $HEURISTICA\_PD()$

1. $LEITURA\_DA\_INSTANCIA()$
2. $S \leftarrow \emptyset$
3. $S_{melhor} \leftarrow \emptyset$
4. **enquanto** (número máximo de iterações não atingido) **faça**
   - 4.1. $S \leftarrow CONSTRUCAO\_DOS\_DISTRITOS()$
   - 4.2. $S \leftarrow BUSCA\_LOCAL(S)$
   - 4.3. $ATUALIZA\_SOLUCAO(S, S_{melhor})$;
5. **fim_enquanto**
6. **retorna** S_{melhor}

fim_procedimento

Figura 2.5: Algoritmo para resolução do *PD*

2.2.1 Algoritmo construtivo

A figura 2.6 apresenta o algoritmo construtivo que foi projetado para o *PD*.

O algoritmo construtivo funciona, de maneira geral, como descrito a seguir. Inicialmente, todos os vértices são marcados como não alocados a nenhum distrito. A partir da solução vazia, o algoritmo cria um distrito vazio e iterativamente vai inserindo vértices não alocados a esse distrito, até que não exista mais nenhum vértice que possa ser inserido sem violar a restrição de capacidade do distrito. Um novo distrito vazio é criado e são inseridos vértices não alocados como descrito acima. O processo termina quando todos os vértices foram alocados a algum distrito da solução.

A escolha do próximo vértice a ser inserido é feita dentre aqueles vértices não alocados que são adjacentes a algum vértice alocado ao distrito corrente, para que seja respeitada a restrição de conexidade dos distritos.

A *função gulosa adaptativa* que ordena os candidatos a serem inseridos no distrito na iteração corrente é dada pelo grau do vértice candidato no grafo induzido pelos vértices não alocados, ou seja, o valor da função para cada vértice candidato a entrar na solução é igual ao número de vértices adjacentes não alocados que este tem. A argumentação que fundamenta a escolha dessa *função gulosa adaptativa* é feita mais adiante, durante a explicação detalhada do algoritmo.

A componente aleatória do algoritmo está na escolha do vértice a ser inserido. Ao invés de se inserir o vértice com melhor valor da *função gulosa adaptativa*, é criada uma *RCL* com os k melhores candidatos e dela é escolhido um elemento aleatoriamente, não necessariamente o melhor.

A seguir é feita uma descrição detalhada de cada linha do algoritmo construtivo.

Na linha 1, todos os vértices são marcados como não visitados, e inicializa-se a variável i

procedimento *CONSTRUCAO_DOS_DISTRITOS*()

1. inicialize todos os vértices $u \in V$ como *NÃO_VISITADO* e **faça** $i \leftarrow 0$
2. **enquanto** ($\exists u \in V$ marcado como *NÃO_VISITADO*) **faça**
   - **2.1.** crie o distrito V_i vazio
   - **2.2.** $r \leftarrow ESCOLHE\_RAIZ(V_i)$ e marque r como *VISITADO*
   - **2.3.** Crie o conjunto Q de vértices v tais que $\exists u \in V_i$ com $(u, v) \in E$ (os trechos correspondentes aos vértices u e v são adjacentes no mapa)
   - **2.4.** Crie o conjunto Q' de vértices $v \in Q$ marcados como *NÃO_VISITADO* e tais que $\sum_{j \in V_i} t_j + t_v \leq W$ (a inclusão do vértice v não excede a capacidade do distrito V_i)
   - **2.5.** **se** $Q' = \emptyset$, **então**
     - **2.5.1.** **faça** $i \leftarrow i + 1$
   - **2.6.** **senão**
     - **2.6.1.** Seja $n(v)$ o número de vértices $u \in V$ adjacentes a v marcados como *NÃO_VISITADO*. Para todo $v \in Q'$ calcule $n(v)$. Crie a lista L de vértices em $v \in Q'$ e ordenada em ordem crescente de $n(v)$.
     - **2.6.2.** Crie o conjunto RCL dos k primeiros vértices da lista ordenada L
     - **2.6.3.** Escolha um elemento $w \in RCL$ aleatoriamente
     - **2.6.4.** $V_i = V_i \cup \{w\}$, marque w como *VISITADO* e volte para a linha 2.3.
   - **2.7.** **fim_se**
3. **fim_enquanto**

fim_procedimento

Figura 2.6: Algoritmo construtivo para o *PD*

como o contador que armazenará o identificador do distrito corrente. Nas linhas de 2 até 3, é construída a solução do distritamento a partir da solução vazia. Um distrito é criado de cada vez, e encontra-se inicialmente vazio (linha 2.1).

Na linha 2.2 é escolhido um vértice r que é o primeiro vértice a ser inserido no distrito corrente. Tal vértice é denominado *vértice raiz* do distrito. A princípio, a escolha desse vértice pode ocorrer de duas formas: (1) pode-se escolher o vértice r, dentre todos os vértices $v \in V$ marcados como *NÃO_VISITADOS*, que tem melhor valor da *função gulosa adaptativa*, ou (2) pode-se escolher um vértice r no mesmo conjunto de vértices, aleatoriamente. Como apresentado em [SMP96], não foram identificadas diferenças significativas entre os métodos de escolha do vértice raiz do distrito. Por isso, no algoritmo apresentado aqui, a escolha do vértice r utilizada foi a aleatória.

Nas linhas 2.3 até 2.7, vértices são inseridos no distrito corrente até que não seja possível inserir mais vértices sem que a restrição de capacidade do distrito seja violada, ou até que todos

os vértices tenham sido visitados. Nas linhas 2.3 até 2.6.2 é criada a lista restrita de candidatos (*RCL*) de tamanho máximo k, conforme a metodologia *GRASP*. Essa lista é construída a partir da lista contendo todos os possíveis vértices a serem inseridos no distrito corrente. Esses vértices devem satisfazer as seguintes condições: (1) a sua inserção no distrito corrente não pode violar a restrição de capacidade do distrito, ou seja, a soma dos pesos dos vértices no distrito corrente, mais o peso do vértice considerado para a inserção no distrito deve ser menor que a capacidade máxima carga W dos distritos; e (2) a inserção do vértice não pode violar a restrição de conexidade dos distritos, ou seja, só podem ser considerados para a inserção no distrito aqueles vértices que têm vértices adjacentes dentro do distrito corrente, garantindo que o distrito permaneça conexo após a sua inserção.

Desta lista é retirado aleatoriamente (linha 2.6.3) o vértice a ser inserido no distrito corrente (linha 2.6.4).

A linha 2.5.1 trata o caso de não haver mais vértices candidatos à inserção no distrito corrente. Um novo distrito deve então ser criado, reiniciando a iteração do laço *enquanto*.

A escolha do vértice a ser inserido no distrito corrente é feita com base na seguinte idéia.

Considere H_i o grafo induzido pelos vértices não visitados (correspondentes aos trechos do mapa ainda não distritados) sobre o grafo original V numa dada iteração i do algoritmo. Na iteração j, cada um dos vértices $u \in H_j$ isolados (sem vértices adjacentes) vai gerar obrigatoriamente um distrito contendo apenas o próprio vértice u e com carga de trabalho t_u, que geralmente é muito menor que a capacidade de carga W. Dessa forma, chega-se a soluções que contém mais distritos compostos de um único trecho, que além de dificilmente satisfazerem o balanço de cargas, ainda podem conter um elevado número de distritos.

Na iteração j do algoritmo, ao se escolher um dentre os vértices candidatos à inserção no distrito corrente (aqueles da lista Q'), o vértice v com menor número de vértices adjacentes em H_j, pretende-se, minimizar o número de vértices que podem tornar-se isolados nas próximas iterações do algoritmo. Por outro lado, a escolha de qualquer outro vértice u que não o próprio v, pode aumentar as chances de deixar outros vértices isolados. Em especial se u é adjacente a v, então as chances de v ficar isolado aumentam.

De fato, basear a escolha do vértice a ser inserido no distrito corrente na idéia acima mostrou-se bastante efetiva na minimização do número de distritos. Tanto em [SMP96] quanto no capítulo 5 são apresentados resultados que comprovam tal afirmação.

Como será visto no capítulo 5, o algoritmo construtivo desenvolvido resolve de forma bastante satisfatória o problema da minimização dos distritos no *PD*, dado que este atinge o limite inferior para o número de distritos em várias das instâncias de testes.

Análise da complexidade do algoritmo construtivo

Sejam $n = |V|$, $m = |E|$, e d o número de distritos criados na solução do problema.

A operação na linha 1 tem complexidade $O(n)$. O laço iniciado na linha 2 (incluindo o número de iterações do laço entre as linhas 2.3 e 2.6.4) é executado $n + d$ vezes, pois a cada iteração, no máximo um vértice é adicionado ao distrito corrente totalizando n vértices a serem

inseridos até a conclusão do laço, mais d iterações em que nenhum vértice pôde ser inserido, fato que implica na criação de um novo distrito (linha 2.5 e 2.5.1). Vale lembrar que sempre $d \leq n$, para instâncias viáveis do problema, sendo que $d = n$ corresponde à solução trivial em que se cria um distrito para cada vértice do grafo. As linhas 2.1 e 2.2 são executada em tempo constante, ou seja, $O(1)$.

No laço que vai da linha 2.3 até a linha 2.7, três operações importantes são realizadas: (1) o cálculo de $n(v)$ na linha 2.6.1, (2) a criação do conjunto Q' na linha 2.4 e (3) a criação da RCL na linha 2.6.2. As complexidades destas três operações podem ser analisada globalmente, como é feito a seguir.

A chave para a análise da complexidade das três operações citadas acima é a constatação de que cada vértice é inserido uma única vez na solução. Quando o vértice $v \in V$ é inserido, os vértices u_i adjacentes a v são inseridos no conjunto Q' e têm seus valores de $n(u_i)$ decrementados em uma unidade. Assim, cada inserção de um vértice implica na visita às arestas a ele incidentes, totalizando $2m$ visitas.

Para cada vértice $v \in V$, o valor $n(v)$ será potencialmente alterado (durante todo o algoritmo) tantas vezes quanto for o grau de v. Como o custo de uma alteração é $O(1)$, o custo total sobre todos os vértices do grafo para atualizar os valores de $n(v)$ é $O(m)$.

Durante toda a execução do algoritmo, cada vértice $v \in V$ é analisado para entrar em Q' tantas vezes quanto for seu grau, resultando também numa complexidade $O(m)$.

A construção da RCL pode ser feita através de um algoritmo da k-ésima estatística de ordem [Cor90, Man89], que pode ser implementado com complexidade $O(n)$. Nesse caso, a cada inserção de vértice na solução corrente implica na criação de uma RCL, resultando numa complexidade de $O(n^2)$ no total.

Como as demais linhas são executadas em tempo constante, a complexidade do algoritmo todo resulta em $O(n^2)$.

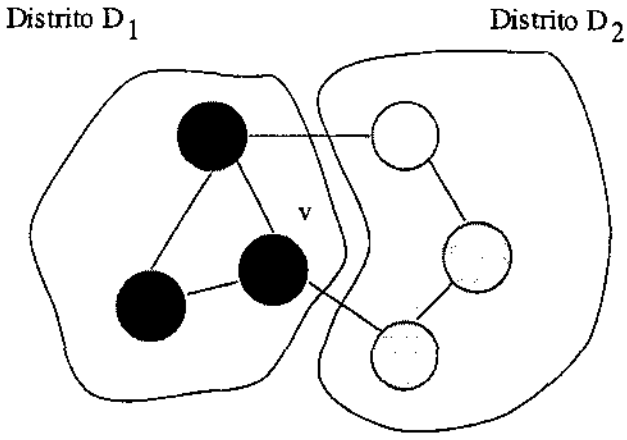
2.2.2 Busca local

Embora as soluções geradas pelo algoritmo construtivo sejam muito boas com relação ao seu número de distritos, estas em geral apresentam má qualidade quanto ao balanceamento de cargas. Delega-se então a tarefa de balancear as cargas de trabalho dos distritos ao algoritmo de busca local apresentado nesta seção.

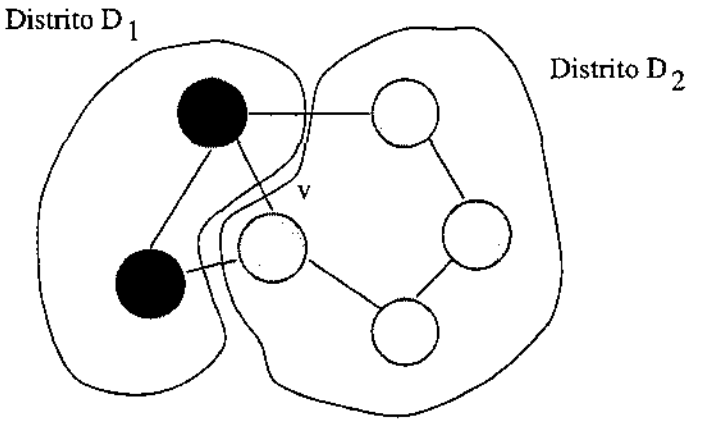
Para balancear as cargas de trabalho de uma solução, são utilizados algoritmos de busca local em que as operações que definem a vizinhança de uma solução são baseadas na troca de um vértice ou mais entre dois distritos adjacentes. Um distrito D_1 é adjacente a outro distrito D_2 se existe $u, v \in V$, $u \in D_1$ e $v \in D_2$ tal que $(u, v) \in E$, ou seja, os distritos D_1 e D_2 são adjacentes se existe um trecho em D_1 e outro em D_2 que são adjacentes entre si.

Duas operações de trocas de vértices entre distritos são consideradas: *troca simples* e *troca dupla*. Na troca simples, um vértice v é movido de um distrito D_1 para outro distrito adjacente D_2 (vide figura 2.7). Na troca dupla, um vértice $v_1 \in D_1$ é movido para um distrito adjacente

D_2 e outro vértice $v_2 \in D_2$ é movido para o distrito D_1 (vide figura 2.8).



(a) antes da troca simples



(b) depois da troca simples

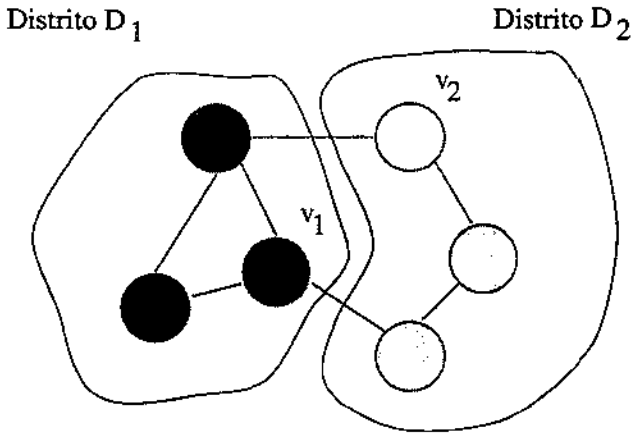
Figura 2.7: Exemplo de uma operação de troca simples de um vértice v de um distrito D_1 para outro distrito adjacente D_2

Entretanto, uma operação de troca simples ou dupla entre distritos pode tornar a solução inviável pois pode violar a restrição de conexidade dos distritos. Isso ocorre quando a remoção de um trecho desconecta o distrito original. Um exemplo seria a remoção do trecho u do distrito D_2 na figura 2.9.

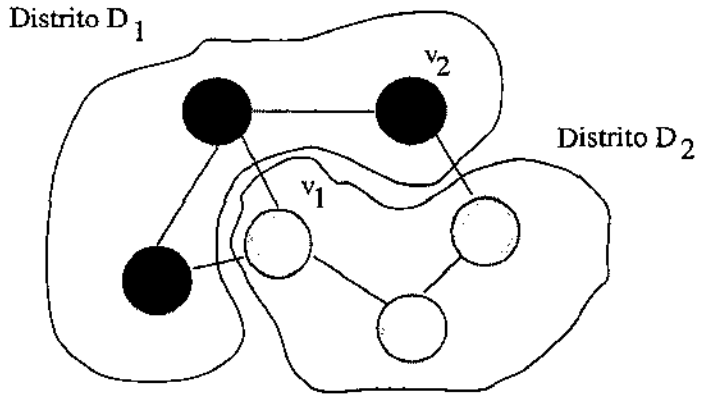
Para tratar esse caso, é necessário introduzir a noção de *ponto de articulação*. Num grafo conexo $H = (V, E)$, um vértice $v \in V$ é um ponto de articulação, se e somente se, sua remoção do conjunto V desconecta o grafo não orientado H, gerando duas ou mais componentes conexas. Na figura 2.9 o vértice u é um exemplo de *ponto de articulação*.

Dessa forma, os trechos representados por vértices que são pontos de articulação em seus distritos não devem participar de operações de troca.

Na realidade, nenhum ponto de articulação pode ser utilizado em trocas simples, sob pena



(a) antes da troca dupla



(b) depois da troca dupla

Figura 2.8: Exemplo de uma operação de troca dupla de um vértice v de um distrito D_1 para outro distrito adjacente D_2

de desconectar o seu distrito original. Entretanto, existem alguns casos em que pontos de articulação podem ser utilizados em trocas duplas. Isso ocorre quando as componentes conexas geradas pela remoção do ponto de articulação são novamente conectados devido à inserção do vértice trazido do outro distrito. Contudo, a detecção de tais casos torna o procedimento de identificação dos vértices candidatos a participar das trocas duplas significativamente mais complexo. Por essa razão, tais casos foram desconsiderados pelo algoritmo de busca local.

Assim, sempre que for necessário realizar uma operação de troca de trechos, deve-se identificar todos os pontos de articulação do distrito de onde um trecho será retirado. Esses pontos de articulação não devem ser removidos dos seus distritos e, portanto, são excluídos da lista de candidatos à troca de distritos.

Um algoritmo de identificação de pontos de articulação foi desenvolvido a partir dos algoritmos de busca em grafos descritos em livros texto como [Cor90, Man89] e tem complexidade

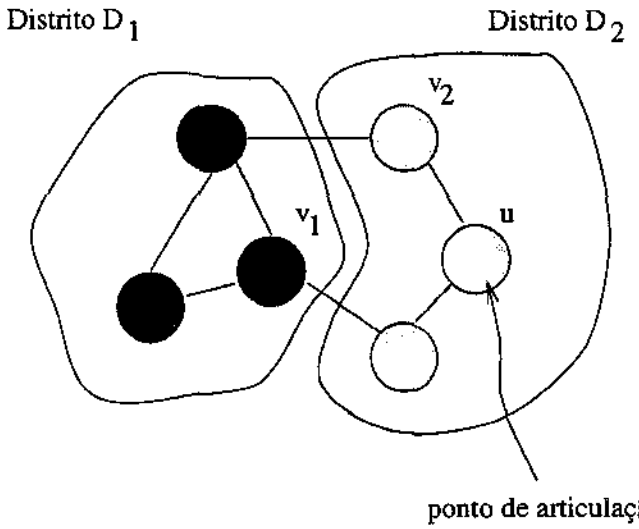


Figura 2.9: Exemplo de um ponto de articulação (vértice u) num distrito.

linear no número de arestas e vértices do grafo.

Para o balanceamento de cargas, a função objetivo adotada foi o desvio padrão da média dos tempos de entrega dos distritos. Ambos os algoritmos são apresentados a seguir.

O algoritmo de balanceamento por troca simples é dado na figura 2.10.

O algoritmo de busca local baseado em trocas simples funciona da seguinte forma. A cada iteração, é identificado um distrito com carga de trabalho inferior à media das cargas. São identificados os vértices que poderiam ser inseridos nesse distrito para elevar a sua carga de trabalho, diminuindo o desvio padrão da solução. Os vértice candidatos para a troca de distrito devem satisfazer às condições: (1) devem ser adjacentes a algum vértice do distrito escolhido (sem estar no próprio distrito); (2) a remoção do vértice escolhido de seu distrito original não deve desconectá-lo; (3) a troca do vértice entre os distritos deve acarretar na diminuição do desvio padrão das cargas de trabalho dos distritos (4) e também não pode violar a carga de trabalho do distrito em que está sendo inserido o vértice. As condições (1) e (2) asseguram que a restrição de conexidade não seja violada, a condição (4) mantém a restrição de capacidade nos distritos. A condição (3) assegura que o algoritmo termine num número finito de passos, com a solução corrente sendo localmente ótima, com respeito ao balanceamento de cargas.

O distrito onde será adicionado um vértice é aquele com menor carga, dentre aqueles que têm carga de trabalho inferior à media. Quando não é possível encontrar vértices candidatos à inserção nesse distrito (nenhum satisfaz todas as condições citadas acima), ele é removido da lista de distritos candidatos à escolha nas linhas 2 e 4. O algoritmo termina quando a lista de distritos candidatos está vazia. Quando alguma troca é realizada, todos os distritos são novamente inseridos na lista de distritos candidatos.

Note que o número de distritos na solução não é alterado durante a busca local.

A seguir é apresentada uma descrição mais detalhada do algoritmo.

procedimento $BUSCA\_LOCAL\_COM\_TROCA\_SIMPLES()$

1. $L \leftarrow CRIA\_LISTA\_DE\_DISTRITOS()$
2. $D \leftarrow ESCOLHE\_DISTRITO(L)$
3. **enquanto** $D \neq \emptyset$ **faça**
   - 3.1. $T \leftarrow VERTICES\_ADJACENTES(D)$
   - 3.2. $T \leftarrow REMOVE\_ARTICULACOES(T)$
   - 3.3. $t \leftarrow MELHOR\_VERTICE\_PARA\_TROCA(T)$
   - 3.4. **se** $t \neq \emptyset$ **então**
     - 3.4.1. $REMOVE\_VERTICE(t, distrito(t))$
     - 3.4.2. $ADICIONA\_VERTICE(t, D)$
     - 3.4.3. $L \leftarrow CRIA\_LISTA\_DE\_DISTRITOS()$
   - 3.5. **senão**
     - 3.5.1. $REMOVE\_DISTRITO\_DA\_LISTA(D, L)$
   - 3.6. **fim_se**
   - 3.7. $D \leftarrow ESCOLHE\_DISTRITO()$
4. **fim_enquanto**

fim_procedimento

Figura 2.10: Algoritmo de busca local para o *PD* baseado em troca simples de vértices entre distritos

Na figura 2.10. Na linha 1, é criada a lista de distritos candidatos para receberem um vértice. A linha 2 é encarregada de escolher um distrito que tenha tempo inferior à média de tempos dos distritos, para que este possa receber um vértice de um de seus distritos adjacentes, contribuindo para reduzir o valor da função objetivo. Após a primeira iteração, a escolha passa a ser feita na linha 4.

A linha 3.1 retorna o conjunto de vértices que são adjacentes ao distrito D passado como parâmetro, ou seja, retorna todos aqueles vértices cujos vértices correspondentes em V sejam adjacentes a pelo menos um vértice pertencente ao distrito D e não estejam contidos em D. Em outras palavras, essa função retorna todos os possíveis candidatos a serem inseridos no distrito D.

Na linha 3.2 são removidos todos os vértices do conjunto T passado como parâmetro cujos vértices correspondentes em V sejam pontos de articulação em seus respectivos distritos, ou seja, removem-se aqueles vértices contidos em T que não podem ser removidos de seus distritos, sob pena de torná-los desconexos.

Na linha 3.3 é selecionado o vértice dentre aqueles contidos no conjunto T cuja troca de distritos resulta na maior melhoria da função objetivo.

Escolhido tal vértice, este é removido do seu distrito de origem (linha 3.4.1) e inserido no

distrito D (linha 3.4.2). Na linha 3.4.3 todos os distritos removidos da lista L de candidatos, que porventura tenham sido removidos em outras iterações, são colocados novamente em L, pois a troca simples pode ter modificado a configuração dos distritos, permitindo novas trocas envolvendo os distritos outrora removidos.

Caso não exista nenhum vértice em condições de entrar no distrito D escolhido, tal distrito é removido da lista L e o laço é reiniciado para que outro distrito seja escolhido e uma outra troca seja realizada.

A condição de parada é atingida quando não há nenhum distrito em condições de realizar trocas simples.

O algoritmo de troca dupla também é bastante eficiente quanto ao balanceamento de cargas de trabalho porque permite que as cargas líquida transferida entre distritos adjacentes correspondem a frações dos tempos dos vértices, o que é impossível com as trocas simples.

O algoritmo de troca dupla, apresentado na figura 2.11, é bastante semelhante ao algoritmo de trocas simples, exceto pelo fato de que a troca é feita entre dois distritos (D_1 e D_2) definidos a *priori*. Além disso, a troca dos dois vértices é feita simultaneamente, aumentando o número de trocas possíveis com relação ao algoritmo de trocas simples, ou seja, a vizinhança de uma solução é ampliada.

2.2.3 Complexidade dos algoritmos de busca local

Analisando a complexidade do laço das linhas 3 até 4, percebe-se que cada iteração do algoritmo tem complexidade $O(m)$. Na linha 3.1 são identificados os vértices adjacentes aos vértices do distrito escolhido, operação que é realizada com complexidade $O(m)$. Na linha 3.2 os pontos de articulação são removidos com complexidade $O(n+m)$. Na linha 3.3 a escolha do vértice cuja troca de distritos implica na maior melhoria da função objetivo tem complexidade $O(n)$. A escolha do distrito para a troca de vértices é realizada também com complexidade $O(n)$. As demais operações do laço são executadas em tempo constante.

Como o grafo utilizado no *PE* é completo, a complexidade de cada iteração do algoritmo de busca local é $O(n^2)$.

Embora o algoritmo execute um número finito de iterações, (visto que as trocas sempre melhoram a função objetivo), esse número pode ser exponencial no número de vértices da instância. Pode-se resolver esse problema limitando-se o número máximo de iterações executadas a uma constante. Entretanto, nos testes realizados, descritos no capítulo 5 não foi necessário restringir o número de iterações.

O algoritmo de trocas duplas comporta-se da mesma forma, tendo a mesma complexidade ($O(n^2)$) para cada iteração do algoritmo. De maneira semelhante, o número de iterações é, no pior caso, exponencial no número de vértices da instância do problema.

procedimento $BUSCA\_LOCAL\_COM\_TROCA\_DUPLA()$

1. $ESCOLHE\_PAR\_DE\_DISTRITOS(D_1, D_2)$
2. **enquanto** $D_1 \neq \emptyset$ **faça**
   - 2.1. $T_1 \leftarrow VERTICES\_ADJACENTES(D_1, D_2)$
   - 2.2. $T_2 \leftarrow VERTICES\_ADJACENTES(D_2, D_1)$
   - 2.3. $T_1 \leftarrow REMOVE\_ARTICULACOES(T1)$
   - 2.4. $T_2 \leftarrow REMOVE\_ARTICULACOES(T2)$
   - 2.5. $t_1 \leftarrow MELHOR\_VERTICE\_PARA\_TROCA(T_1)$
   - 2.6. $t_2 \leftarrow MELHOR\_VERTICE\_PARA\_TROCA(T_2)$
   - 2.7. **se** $t_1 \neq \emptyset$ e $t_2 \neq \emptyset$ **então**
     - 2.7.1. $REMOVE\_VERTICE(t_1, D_1)$
     - 2.7.2. $REMOVE\_VERTICE(t_2, D_2)$
     - 2.7.3. $ADICIONA\_VERTICE(t_2, D_1)$
     - 2.7.4. $ADICIONA\_VERTICE(t_1, D_2)$
   - 2.8. **fim_se**
3. $ESCOLHE\_PAR\_DE\_DISTRITOS(D_1, D_2)$
4. **fim_enquanto**

fim_procedimento

Figura 2.11: Algoritmo de busca local para o PD baseado em troca dupla de vértices entre distritos.

2.3 Algoritmo heurístico para o PE

Relembrando a formulação apresentada na seção 1.2, resolver o PE significa encontrar um ciclo C no grafo orientado completo G que passe por todos os pontos de entrega respeitando a restrição de conservação de fluxo no caminho e as restrições de capacidade tanto do entregador quanto dos depósitos. Além disso, os depósitos podem ser usados mais de uma vez ou até mesmo evitados no caminho.

Para resolver este problema, foi desenvolvido um algoritmo também baseado no paradigma *GRASP*, mostrado na figura 2.12.

Na figura 2.12, as linhas 1 até 3 inicializam as variáveis usadas no algoritmo. Nas linhas 4 a 5 se dá a iteração *GRASP*, controlada pelo laço **enquanto**. Na fase construtiva do algoritmo (linhas 4.1 e 4.2), é construída a solução S, viável para o PE. Essa fase do algoritmo é descrita em detalhes na seção 2.3.1.

A solução inicial S é fornecida como ponto de partida para a fase de busca local (linha 4.3), em que se tenta melhorar o valor da função objetivo utilizando-se uma técnica conhecida como *busca em profundidade variável*, introduzida por [BLS93] e apresentada na seção 2.3.3.

procedimento *HEURISTICA_PE*()

1. *LEITURA_DA_INSTANCIA*()
2. $S \leftarrow \emptyset$
3. $S_{melhor} \leftarrow \emptyset$
4. **enquanto** (número máximo de iterações não atingido) **faça**
   - **4.1.** Construa uma rota C' apenas com os pontos de entrega (removendo-se os depósitos da instância) que minimize a distância percorrida e atribua a solução a S
   - **4.2.** Insira os depósitos na solução S de modo a tornar a rota viável quanto às restrições de capacidade do entregador e de conservação de fluxo nos vértices
   - **4.2.** $S \leftarrow BUSCA\_LOCAL(S)$
   - **4.3.** $ATUALIZA\_SOLUCAO(S, S_{melhor})$;
5. **fim_enquanto**
6. **retorna** S_{melhor}

fim_procedimento

Figura 2.12: Algoritmo para resolução do *PE*

2.3.1 Algoritmo construtivo

Na fase construtiva do algoritmo do *PE* (linhas 4.1 e 4.2 da figura 2.12 busca-se obter uma solução inicial viável para ser submetida ao algoritmo de busca local.

Inicialmente constrói-se um ciclo C' que passa por todos os pontos de entrega da instância, sem passar por nenhum depósito, que minimiza a soma dos custos das arestas utilizadas em C' (processo denotado pela linha 4.1 no algoritmo apresentado na figura 2.12). Atribui-se rótulos $p_0, \ldots, p_{n-m-1}$ aos pontos de entrega, seguindo sua ordem no ciclo C', onde n é o número total de vértices no grafo G e m é o número de depósitos.

A idéia do algoritmo construtivo é seguir o trajeto do entregador, inserindo depósitos à medida em que se fazem necessários (operação executada na linha 4.2 do figura 2.12).

Em primeiro lugar, coloca-se um depósito entre dois pontos de entrega de C', escolhidos arbitrariamente. Em seguida, "caminha-se" no ciclo C' atribuindo a carga do entregador, considerando-se que este pegou o máximo de revistas quanto possível naquele depósito inserido. Em cada ponto de entrega visitado, desconta-se da carga do entregador, o montante de revistas correspondente à demanda do ponto de entrega em questão. Numa determinada posição do trajeto, as revistas trazidas pelo entregador não são mais suficientes para atender à demanda do próximo ponto de entrega. Insere-se então um novo depósito na posição anterior ao ponto de entrega não suprido. Com o novo depósito no caminho, o entregador pode novamente pegar mais revistas e continuar o percurso, repetindo o processo até que seja atingido novamente o

primeiro depósito inserido.

A idéia expressa acima é formalizada no algoritmo descrito a seguir. Inicialmente, utilizando um algoritmo heurístico para o *TSP* (por exemplo, o $k - opt$ [LLRS85]), contrói-se um ciclo hamiltoniano passando por todos os pontos de entrega. Seja $C = \{p_0, \ldots, p_{n-1}\}$ de vértices nesse ciclo. A seguir, é sorteado aleatoriamente um ponto de entrega em C. Sem perda de generalidade, suponha que p_0 seja este ponto. Para satisfazer a restrição de fluxo, é preciso que ao chegar em p_0 o entregador tenha pelo menos tantas revistas quantas forem aquelas relativas à demanda em p_0. Assim, como neste momento o entregador está vindo de p_{n-1} para p_0, procura-se um depósito d_0 que será visitado pelo entregador antes de passar por p_0. Assume-se que ao passar em d_0 vindo de p_{n-1} o entregador está sem revistas e que em d_0 ele pega o máximo de revistas que puder. A partir daí, seja p_t o último ponto de entrega tal que foi inserido um depósito d_t entre p_{t-1} e p_t (índices módulo n). Um novo depósito deverá ser incluído no ciclo entre os pontos de entrega p_ℓ e $p_{\ell+1}$ se para algum $\ell \leq n$ a demanda acumulada de p_t a $p_{\ell+1}$ é maior que do que a carga do entregador ao sair do depósito d_t e a demanda acumulada de p_t até p_ℓ for menor do que esta carga.

Note que, um depósito d_t, inserido na iteração t do algoritmo acima, pode ser um depósito já inserido em outra iteração anterior, desde que não sejam utilizadas mais revistas do que o estoque máximo do depósito permite. Os depósitos duplicados são considerados como novos vértices (independentes) adicionados à instância do problema. Para possibilitar o teste de viabilidade, durante o processo, são mantidas informações sobre o estoque remanescente em cada depósito, levando em consideração o seu número de replicações durante o algoritmo.

Como será mostrado mais adiante, a randomização do algoritmo construtivo é de fundamental importância no desempenho geral do algoritmo pois determina o conjunto de depósitos e o número de vezes que cada um destes será utilizado na solução final. Isso decorre do fato de que o conjunto de operações que define a vizinhança das soluções na busca local não inclui a inserção ou remoção de depósitos da solução corrente. Assim, uma vez escolhidos os depósitos e o número de passagens em cada um deles, durante a fase construtiva do algoritmo, esse conjunto não é mais alterado durante a fase de busca local. Durante a busca local, apenas a ordem entre os pontos de entrega e os depósitos é alterada.

Na seção seguinte o método de randomização do algoritmo construtivo é discutida em maiores detalhes.

2.3.2 Randomização do algoritmo construtivo do *PE*

A randomização da fase construtiva se dá em duas fases. A primeira randomização diz respeito à escolha do depósito a ser inserido. A segunda randomização influencia a posição ℓ em que o depósito será inserido (não sendo mais inserido apenas no momento em que o entregador fica sem revistas).

A primeira randomização é realizada da seguinte forma. Suponha que numa dada iteração do algoritmo mostrado na seção anterior, se faz necessária a inserção de um depósito entre os vértices $\ell - 1$ e ℓ. Todos os m depósitos $d_0, \ldots, d_{m-1}$ disponíveis no momento são ordenados

numa lista de candidatos L, em ordem crescente da soma dos custos $w_{p_{\ell-1},d_j} + w_{d_j,p_\ell}$, com $j = 0,\ldots,m-1$, ou seja, pela ordem do custo do caminho $\{p_{\ell-1}, d_j, p_\ell\}$. Toma-se então a lista dos k primeiros elementos da lista L e forma-se a lista restrita de candidatos (RCL) da qual se escolhe um elemento aleatoriamente (conforme o paradigma *GRASP* descrito na seção 2.1).

A segunda randomização é baseada na posição em que um depósito é inserido. No algoritmo construtivo, mostrado na seção 2.3.1, enquanto é visitado o i-ésimo ponto de entrega do caminho C, a probabilidade λ de inserção de um novo depósito é definida por uma função exponencial definida por

$$\lambda(i) = e^{-\alpha \times y_{i-1,i}}, \tag{2.1}$$

onde α é uma constante conhecida (seu cálculo será detalhado mais adiante) e $y_{i-1,i}$ é o fluxo de revistas (quantidade trazida pelo entregador) entre os pontos de entrega $i-1$ e i.

Dessa forma, a cada iteração do algoritmo, há uma determinada probabilidade de se ter um depósito inserido naquela posição, mesmo que o fluxo de revistas naquele ponto seja suficiente para atender mais alguns pontos de entrega.

A função exponencial para a probabilidade é particularmente apropriada ao problema pois se adequa consideravelmente aos pontos extremos da função. Logo após a inserção de um depósito, o valor de $y_{i,i-1}$ é grande (pois acabou de ocorrer um reabastecimento no depósito inserido). Assim a função $\lambda(i)$ assume valor próximo a zero, tendo portanto poucas chances de haver nova inserção. Ao contrário, quando $y_{i,i-1}$ está próximo de zero (isto é, o entregador está a ponto de ficar sem revistas), a função exponencial assume um valor próximo a 1, tornando obrigatória a inserção do depósito naquele ponto.

O ajuste da função $\lambda(i)$ pode ser feito através da escolha da probabilidade λ_0 com que um depósito é inserido quando o fluxo $y_{i-1,i} = c$.

Com $y_{i-1,i} = c$, tem-se que $\lambda(i) = e^{-\alpha \times c} = \lambda_0$ e

$$\alpha = -\frac{ln\lambda_0}{c} \tag{2.2}$$

Nas figuras 2.13, 2.14 e 2.15 são mostradas três curvas exponenciais para $\lambda(i)$, para três escolhas diferentes de λ_0, para a mesma capacidade $c = 50$.

Na figura 2.13, $\lambda_0^a = 0,61$ e $\alpha^a = 0,01$ e na figura 2.14, $\lambda_0^b = 0,082$ e $\alpha^b = 0,05$, na figura 2.15, $\lambda_0^c = 0,0067$ e $\alpha^c = 0,1$. Note que a inserção dos depósitos ocorre com freqüência crescente nas figuras 2.13, 2.14 e 2.15, ou seja, o número de inserções decresce com o aumento de α. Esses três valores de α para os quais foram traçados os gráficos da probabilidade de inserção foram os valores utilizados nos testes computacionais apresentados no capítulo 5.

Embora a segunda randomização não siga exatamente aquela utilizada no paradigma *GRASP*, esta aumenta significativamente a diversidade das soluções geradas na fase construtiva, o que, como será visto no capítulo 5, é fundamental para a obtenção de soluções de boa qualidade.

Desconsiderando-se a complexidade da operação de obtenção da solução inicial através da resolução do *TSP* sobre os pontos de entrega, o algoritmo construtivo descrito acima tem complexidade $O(n^2)$ no pior caso, que pode ser obtida como se segue. Seja n o número de pontos de

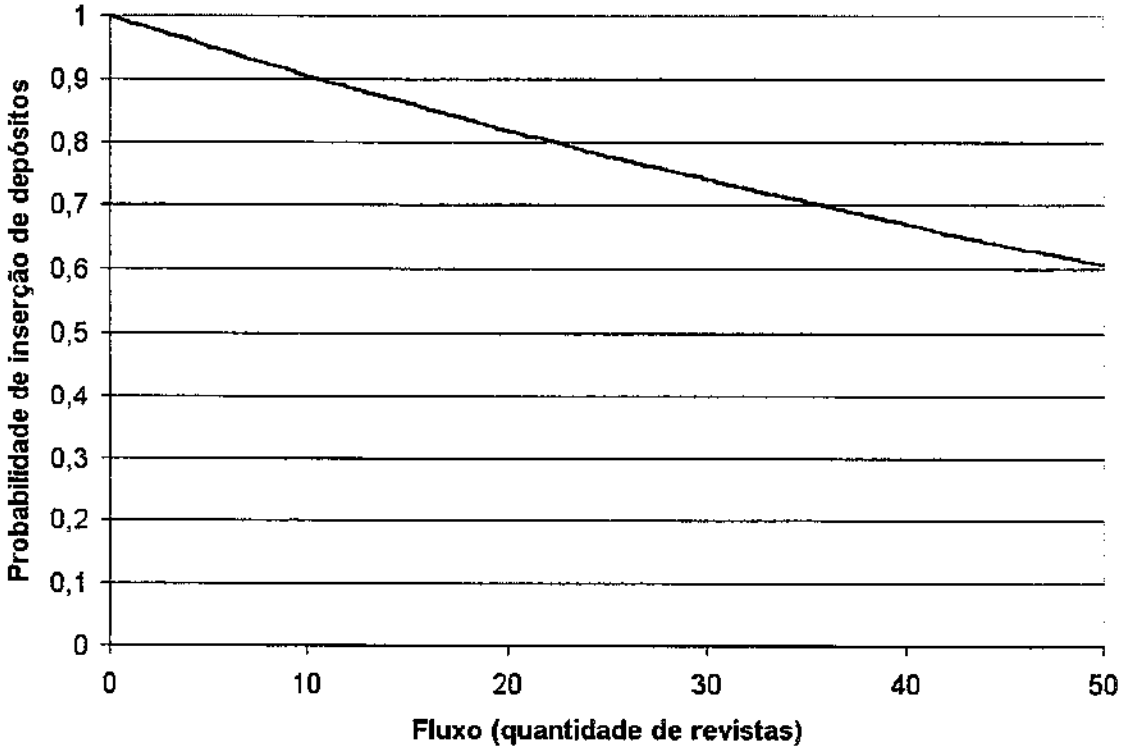


Figura 2.13: Gráfico da função exponencial da probabilidade de inserção de um depósito em função do fluxo (quantidade de revistas) num determinado ponto da rota de entrega, para $\lambda_0^a = 0,61$ e $\alpha^a = 0,01$.

entrega em V. Cada uma das n iterações do algoritmo pode, no pior caso, ter executado uma operação de inserção de depósito, que tem complexidade $O(n)$ correspondente à criação da lista de depósitos que podem ser inseridos no ciclo C. Dessa forma, cada iteração do algoritmo tem potencialmente complexidade $O(n)$. Como são executadas n iterações, o algoritmo construtivo tem complexidade $O(n^2)$.

Uma vez criada uma solução inicial válida para o PE, esta é submetida à heurística de busca local, descrita em detalhes na próxima seção.

2.3.3 Busca local

O algoritmo de busca local desenvolvido para o PE parte do princípio de que a decisão de quais e quantas vezes cada depósito deve ser utilizado já foi tomada na fase construtiva do algoritmo.

Por essa razão, durante a busca local, o número de vezes que um depósito é utilizado não é alterado, ou seja, dado o conjunto de depósitos e o número de vezes que cada um é utilizado, este se mantém até o final da busca local sem sofrer alterações.

O objetivo da busca local passa a ser então encontrar um ciclo passando por todos os pontos de entrega e depósitos uma única vez (considerando as replicações de um único depósito como

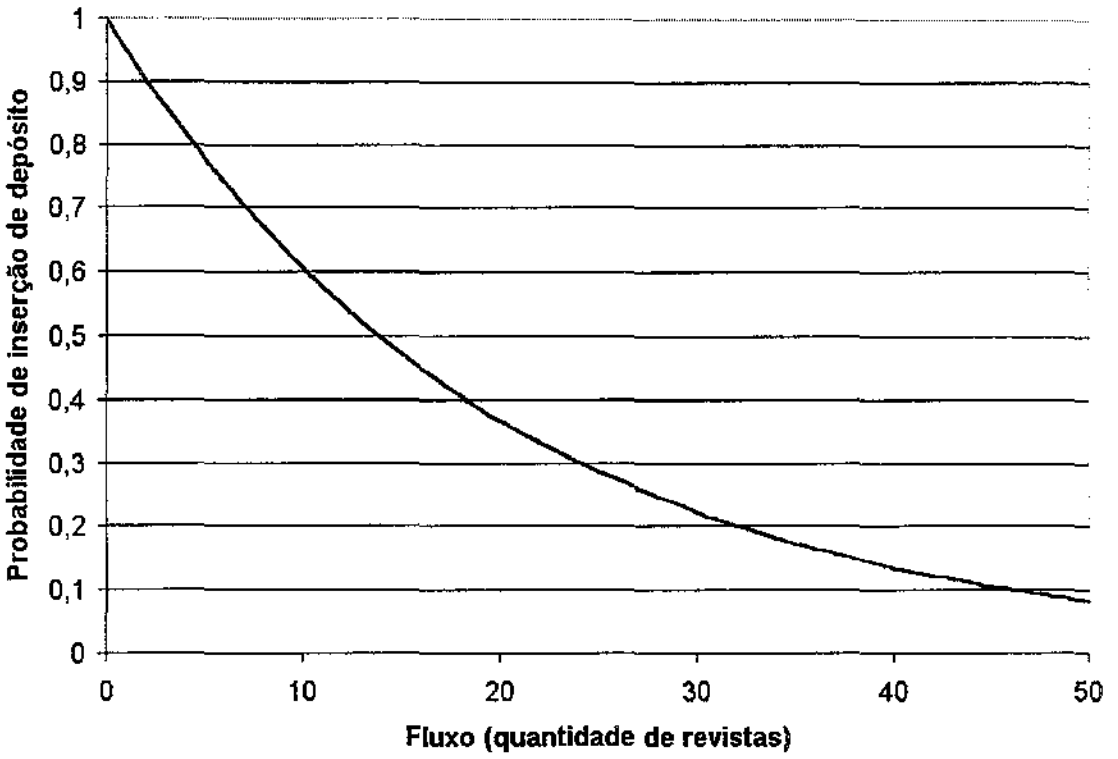


Figura 2.14: Gráfico da função exponencial da probabilidade de inserção de um depósito em função do fluxo (quantidade de revistas) num determinado ponto da rota de entrega, para $\lambda_0^b = 0,082$ e $\alpha^b = 0,05$.

depósitos distintos), que minimize a distância percorrida e respeite a capacidade do entregador e dos depósitos. O novo problema definido a partir da aplicação dessa consideração sobre o *PE* original passará a ser chamado de *PE fixo*.

Esse fato simplifica bastante o desenvolvimento do algoritmo de busca local, visto que não há mais a necessidade de criação de operações de inserção e remoção de depósitos na solução do *PE fixo*. Além disso, o problema resultante passa a ser agora, um problema semelhante ao *TSP*, o que permite a utilização de um dos vários algoritmos de busca local existentes para este problema na literatura.

Dentre os muitos algoritmos existentes, foi utilizado o algoritmo proposto em [BLS93], onde se apresenta um algoritmo heurístico de busca local para uma versão restrita do *TSP*, o *Problema da Entrega e Coleta com um Único Veículo com Janelas de Tempo*, ou *SVPDPTW* (em inglês, *Single-Vehicle Pickup and Delivery Problem with Time Windows*).

A seguir será apresentado um resumo do algoritmo proposto em [BLS93]. Logo em seguida serão descritas as alterações necessárias para adaptar tal algoritmo ao caso do *PE fixo*.

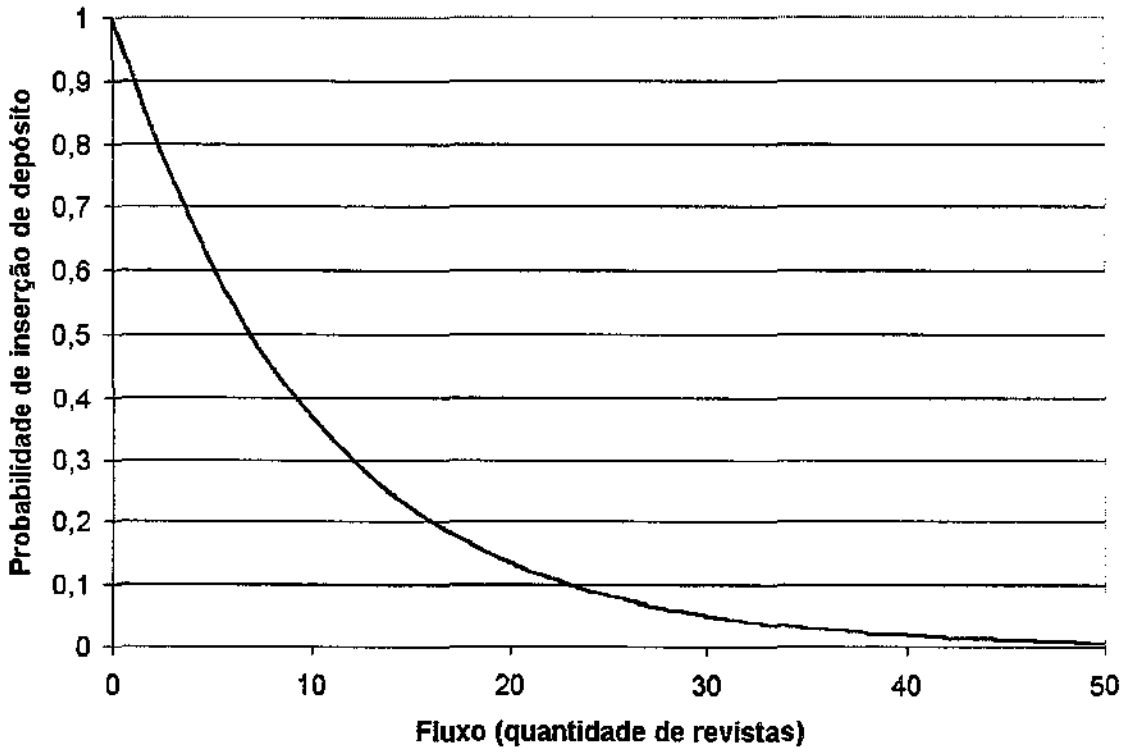


Figura 2.15: Gráfico da função exponencial da probabilidade de inserção de um depósito em função do fluxo (quantidade de revistas) num determinado ponto da rota de entrega, para $\lambda_0^c = 0,0067$ e $\alpha^c = 0,1$.

Busca de profundidade variável para o *SVPDPTW*

Esta seção traz uma compilação dos conceitos discutidos em [BLS93], onde é apresentado um algoritmo de busca de profundidade variável para o *SVPDPTW*, baseado na técnica de busca local introduzida em [LK73].

No *SVTSPTW* são dados: um único depósito, um veículo com capacidade limitada e um conjunto de n' clientes com demandas conhecidas. Cada cliente deve ser apanhado em seu ponto de origem (coleta) e levado até seu destino (entrega) no intervalo de tempo determinado por duas janelas de tempo (uma para a coleta e outra para a entrega). O problema é determinar uma rota para o veículo que minimize a distância a ser percorrida, respeitando a sua capacidade de carga.

O *SVPDPTW* é uma versão restrita do *TSP* definido sobre $n + 1$ vértices, onde $n = 2n'$. As restrições impostas sobre o *SVPDPTW* são: (1) as janelas de tempo, (2) a capacidade do veículo e (3) as relações de precedência entre a origem e o destino de cada cliente.

Um algoritmo de busca local tradicionalmente aplicado ao *TSP* é a *k-troca* (*k-exchange* em inglês). Nesta técnica, a vizinhança de uma solução do *TSP* é baseada na operação de troca de um conjunto de k arcos, por um outro conjunto de k arcos. Em geral, emprega-se a técnica com

$k = 2$ ou $k = 3$, pois o tempo computacional para se verificar a *k-otimalidade* de uma solução cresce rapidamente com o valor de k. Uma solução C é dita *k-ótima* se não existe nenhuma solução na vizinhança definida por k-trocas, cujo valor da função objetivo é melhor que a de C.

No algoritmo apresentado em [BLS93] são consideradas diferentes estratégias de k-trocas. Além da 2-troca, considera-se também a *Or-troca* [Or76][1], que consiste num subconjunto das 3-trocas em que uma cadeia de um, dois ou três vértices consecutivos é removida de sua posição original e inserida entre outros dois vértices.

Como o *TSP* não tem restrições adicionais, o resultado de qualquer k-troca é sempre viável. No caso do *SVPDPTW*, faz-se necessária a execução de testes de viabilidade, uma vez que a k-troca pode resultar numa solução inviável. O mesmo ocorre com o *PE fixo*, em que testes de viabilidade, diferentes daqueles usados para o *SVPDPTW*, devem ser aplicados às soluções resultantes de k-trocas. Esta será uma das modificações fundamentais para a aplicação deste algoritmo ao caso do *PE fixo*.

Em [LK73] foi desenvolvido um procedimento de troca de produndidade variável para o *TSP*, onde o número de arcos a serem trocados numa k-troca é determinado dinamicamente. Dada a opção de se trocar s arcos, regras heurísticas são aplicadas para determinar se uma troca de $s + 1$ arcos deveria também ser considerada.

Este procedimento de troca de arcos, com as devidas modificações para o caso do *SVPDPTW*, é o ponto central do algoritmo proposto em [BLS93].

Procedimentos de trocas de arcos

Para que se possa descrever os procedimentos de troca de arcos utilizados pelo algoritmo antes é necessário relembrar o enunciado do *TSP*, como a seguir.

Considere um grafo sobre $n + 1$ vértices. Uma solução para o *TSP* é um *tour*, isto é, um ciclo que visita todos os vértices exatamente uma vez. Como convenção, um *tour* será denotado pela seqüência $(0, \ldots, i, \ldots, n + 1)$, onde i é o i-ésimo vértice visitado no *tour* e os vértices 0 e $n + 1$ denotam o depósito do *SVPDPTW*. A distância percorrida é dada pela matriz de custos (c_{ij}), com $c_{0,n+1} = 0$. Assume-se que as distâncias entre vértices são simétricas, ou seja, $c_{ij} = c_{ji}$, $\forall i, j \in (0, \ldots, n + 1)$. O objetivo é encontrar um *tour* que minimize a distância total percorrida $\sum_{0 \leq i \leq n} c_{i,i+1}$.

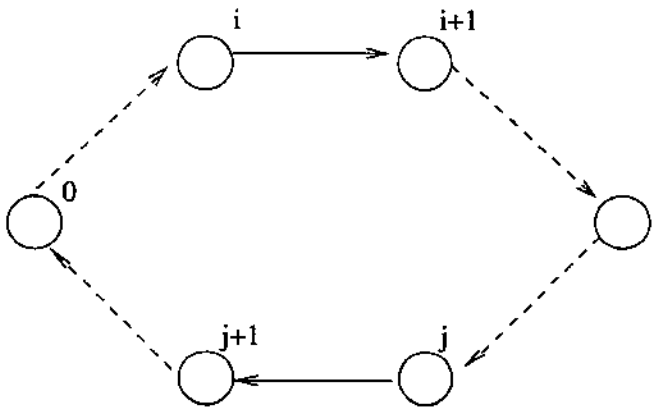
As k-trocas consideradas no algoritmo apresentado em [BLS93] são descritas a seguir.

O primeiro tipo de k-trocas usadas no algoritmo são as 2-trocas. A figura 2.16 dá um exemplo deste tipo de troca, em que os arcos $(i, i + 1)$ e $(j, j + 1)$ são trocados pelos arcos (i, j) e $(i + 1, j + 1)$. Note que no caso das 2-trocas, existe uma única maneira de trocar os dos arcos tal que o resultado seja um novo *tour* e que, no caso do problema direcionado, um dos caminhos $(i + 1, \ldots, j)$ ou $(j + 1, \ldots, i)$ devem ter sua direção invertida para reestabelecer um *tour* viável.

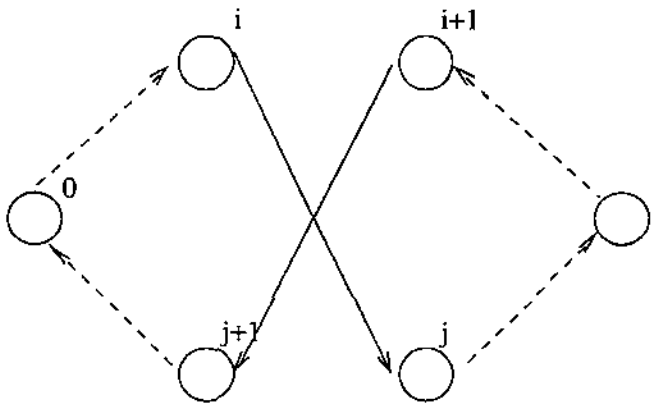
Como o *TSP* não tem restrições adicionais, toda k-troca resulta numa solução viável. Para computar o ganho obtido com a troca, basta se calcular a diferença de custos entre os arcos

[1] A vizinhança *Or-troca* é assim denominada pois foi apresentada pela primeira vez por *Or*, em sua tese de doutorado.

removidos e os arcos adicionados, no caso simétrico. Note que, no caso assimétrico, a reversão de uma dos caminhos do *tour* altera o valor da função objetivo naquele caminho, tornando mais complexo o cálculo, o que não ocorre com o caso simétrico.



(a) Antes de troca de arestas



(b) Depois da troca de arestas

Figura 2.16: Exemplo de uma operação de troca de arestas do tipo *2-troca*.

Outro tipo de k-troca considerada no algoritmo são aquelas propostas por *Or* em [Or76]. Como dito anteriormente, estas trocas, chamadas *Or-trocas* consistem na relocação de cadeias de um, dois ou três vértices consecutivos entre outros dois vértices do *tour*. São consideradas as trocas em que a cadeia de vértices é inserida numa posição anterior à original (*Or-exchange backward*), denominadas *Orb-trocas* e também aquelas que inserem a cadeia numa posição posterior à posição original (*Or-exchange foward*), denominadas *Orf-trocas*. A figura 2.17 dá exemplos de ambos os tipos de *Or*-trocas. Os arcos $(j, j+1)$, $(i, i+1)$ e $(i+m, i+m+1)$ (marcados como

setas pontilhadas na figura 2.17a) são trocados pelos arcos $(j, i+1)$, $(i+m, j+1)$ e $(i, i+m+1)$ (marcadas por setas em negrito nas figuras 2.17b e 2.17c, respectivamente para uma *Orb*-troca e para uma *Orf*-troca), tal que o caminho $(i+1, i+m)$ é removido de sua posição original, entre os vértices i e $i+m+1$, e inserido entre os vértices j e $j+1$, onde m é o tamanho da cadeia de vértices consecutivos relocados. No caso das *Or*-trocas, não há inversão de caminhos.

No caso do algoritmo apresentado em [BLS93], a vizinhança de uma solução é visitada de maneira sistemática, pela chamada *estratégia de busca lexicográfica da vizinhança*, em que se examina todas as possíveis trocas, considerando um tipo particular dentre as k-trocas descritas anteriormente, de tal forma que os testes de viabilidade (para o caso do *SVPDPTW*) podem ser feitos em tempo de execução constante, ou seja com complexidade $O(1)$.

Infelizmente, como será visto na próxima seção, não foi possível desenvolver um teste de viabilidade para o caso do *PE fixo* que pudesse ser executado com mesma eficiência (tempo de execução constante). Embora tenha sido utilizada a mesma estratégia de busca de vizinhança apresentada em [BLS93], esta não influencia o tempo de execução do teste de viabilidade. Por essa razão, tal técnica não será descrita no presente trabalho. Para maiores detalhes, refira-se ao artigo anteriormente citado.

Procedimento de busca de profundidade variável para o *SVPDPTW*

Considerando todas as as k-trocas, é óbvio que a qualidade de uma solução k-ótima não pode deteriorar à medida que k aumenta. Entretanto, é necessário estabelecer um compromisso entre a utilização de um tempo de execução razoável, obtido com menores valores de k, e a qualidade da solução, resultante de maiores valores de k.

A necessidade de se especificar o valor de k *a priori* pode comprometer o desempenho do algoritmo sendo projetado. Uma solução mais flexível seria a definição do número de arcos a serem trocados dinamicamente. Tal flexibilidade é obtida através do procedimento de busca em profundidade, descrito a seguir, na forma em que foi originalmente proposto por [LK73] e apresentado em [BLS93].

Seja G_s^* a melhora no valor da função objetivo na iteração s do algoritmo, onde s arcos da solução original devem ser trocados por outros s arcos, e a solução obtida resulta num novo ciclo. No algoritmo proposto por [LK73], pode existir um caso em que uma s-troca resulta numa deterioração da função objetivo ($G_s^* < 0$), mas leva a uma $(s+1)$-troca em que há uma melhoria da função objetivo ($G_{s+1}^* > 0$). O procedimento é descrito de forma genérica sem considerar o tipo de troca sendo realizada, na figura 2.18.

Note que ao final de uma iteração, é realizada a troca que resulta no maior ganho. A figura 2.20 descreve o comportamento típico de G_s^*. Nesse exemplo, a troca realizada é aquela associada a $s = 5$. Esta troca teria sido difícil de ser encontrada executando-se apenas uma 2-troca de cada vez. As trocas para $s = 3$ e $s = 4$ levariam à deterioração do custo, isto é, $G_2^* > G_3^* > G_4^*$, e assim seriam desfavoráveis. Essas pioras no custo são permitidas para que se possa obter ganhos maiores nas iterações seguintes. Dessa forma são realizadas k-trocas favoráveis ($k \geq 2$) sem que seja necessário varrer exaustivamente todas as possíveis k-trocas.

O algoritmo completo para o *SVPDPTW* é apresentado na figura 2.21.

Note que, para completar o algoritmo heurístico para o *SVPDPTW*, falta a descrição dos testes de viabilidade que devem ser executados durante a busca na vizinhança da solução corrente. Estes testes levam em consideração as restrições de precedência, de capacidade do veículo e das janelas de tempo, que segundo [BLS93] podem ser realizados em tempo constante com relação ao tamanho da entrada.

Como foi dito anteriormente, já que os testes de viabilidade para o *PE fixo* são diferentes dos testes do *SVPDPTW*, estes últimos não serão apresentados aqui.

Aplicação do algoritmo do *SVPDPTW* ao caso do *PE fixo*

Para adaptar o algoritmo de busca local do *SVPDPTW* para o caso do *PE fixo*, são necessárias as seguintes modificações:

1. Desconsidera-se as restrições de janelas de tempo e precedência do problema original;
2. Desconsidera-se a existência do depósito central, substituindo-o por qualquer outro vértice do problema;
3. Embora seja mantida a restrição de capacidade do veículo, o teste de viabilidade para o caso do *PE fixo* é diferente, pois deve levar em conta a flexibilidade no uso dos estoques dos depósitos, respeitando suas capacidades. O algoritmo que faz este teste de viabilidade é apresentado mais adiante, no final desta seção. Vale lembrar que, no caso particular do *PE fixo*, o teste de viabilidade tem um fator complicador adicional. Uma vez que podem existir vértices representando replicações do mesmo depósitos da instância original, é necessário manter registro da quantidade de revistas utilizada por cada depósito na solução corrente.

Para que o algoritmo, com as modificações descritas acima possa tratar o caso do *PE fixo*, é necessário ainda fazer uma transformação nos vértices deste último, como se segue. Os depósitos do *PE fixo* são transformados em vértices onde ocorreria a coleta do cliente no *SVPDPTW* (pontos de origem) e os pontos de entrega são transformados em pontos de destino dos clientes, lembrando que a restrição de precedência entre pontos de origem e destino não tem mais efeito. Dessa forma, não haveria mais relação entre a demanda dos clientes nos pontos de origem e destino.

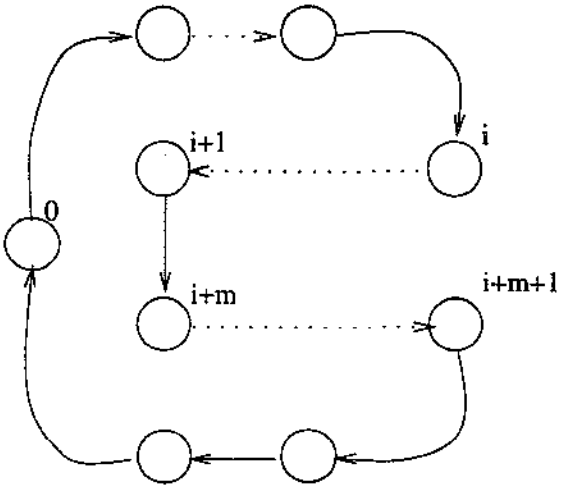
A seguir é feita a descrição do algoritmo para o teste de viabilidade de uma solução para o *PE fixo*.

Considere C o ciclo que está sendo testado pelo algoritmo. Este ciclo é formado por uma seqüência ordenada dos vértices do problema, ou seja, $C = (0, \ldots, m-1)$, onde m é o número total de vértices do problema, contando as replicações de um mesmo depósito. As variáveis d_i denotam a demanda de cada vértice do problema, sendo que $d_i > 0$ para os pontos de entrega e $d_i < 0$ para os depósitos, como na definição do *PE*. Sejam e_i o valor do estoque de um depósito numa dada iteração do algoritmo. A variável $e_i = 0$ para $i \in D$ (quando i é ponto de entrega). Considera-se durante o algoritmo que os índices são todos módulo m.

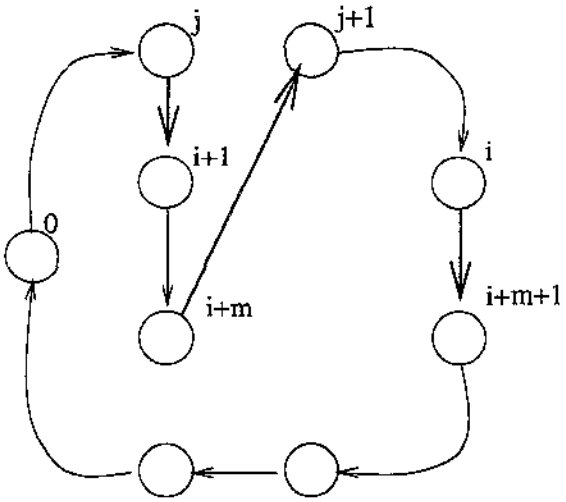
De uma maneira geral, o algoritmo funciona tentando estabelecer um fluxo máximo válido nos arcos do ciclo, correspondente à carga do entregador, começando por um de seus depósitos. Quando é detectado um fluxo negativo, tem-se uma indicação de que não é possível fazer a entrega partindo-se do depósito inicialmente escolhido. Escolhe-se outro depósito e é repetido o procedimento. Ao final do algoritmo, se foi possível estabelecer um fluxo válido passando por todos os vértices, conclui-se que o ciclo sendo testado é válido. Caso nenhum fluxo válido possa ser estabelecido, o ciclo em questão não representa uma solução válida para o *PE fixo*.

O algoritmo descrito acima é apresentado na figura 2.22.

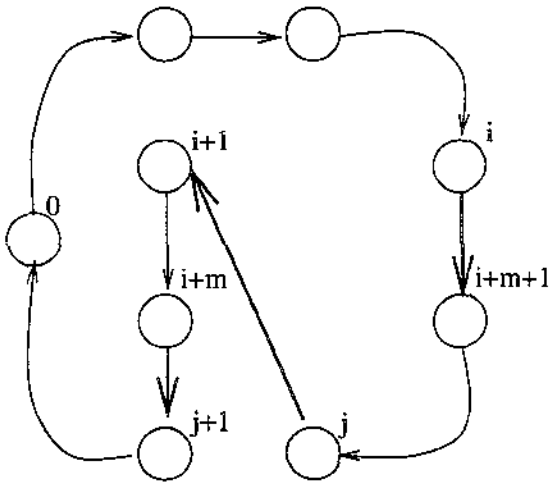
O algoritmo apresentado acima tem complexidade $O(n)$, pois é necessário visitar no máximo $2m$ vértices para verificar se o ciclo C é ou não válido, e não existe nenhuma operação dentro dos laços que não possa ser executada em tempo constante. Note que, a reinicialização dos valores dos e_k na linha 5.6, apesar de aparentar ter complexidade $O(n)$, pode ser feita em tempo constante. Basta marcar os depósitos já visitados e ao executar a linha 5.4.1 for visitado um depósito k marcado, o valor de e_k deve ser atualizado para d_k.



(a) Antes da troca de arestas



(b) Depois da Orb-troca



(c) Depois da Orf-troca

Figura 2.17: Exemplo de uma operação de troca de arestas do tipo *Or-troca* em que m denota o número de vértices consecutivos sendo relocados.

procedimento *HEURISTICA_VAR_DEPTH_SEARCH_TSP*(S)

1. $G_0^* \leftarrow 0$
2. Escolha qualquer vértice como o vértice inicial e considere um arco do ciclo que é incidente a este vértice. Na figura 2.19a o vértice e o arco escolhidos foram i e (i, j), respectivamente.
3. $s \leftarrow 1$
4. A partir da outra extremidade do arco escolhido, isto é, o vértice j, escolha um arco, por exemplo, (j, q), que não encontra-se no ciclo S corrente e tal que o ganho $g_1 = c_{ij} - c_{jq} > 0$ é maximizado. Se tal arco não existe, volte ao passo 1 e tente outro vértice. Veja a figura 2.19b.
5. Escolhido o arco (i, j) a ser removido e o arco (i, q) a ser inserido no ciclo S, tem-se univocamente determinado o arco incidente a q, que deve ser removido na iteração s para que seja reconstruído o ciclo da solução do *TSP*. Nesse caso, remove-se o arco (q, p) da solução corrente S. Adicionando-se o arco (p, i), obtém-se como resultado um ciclo onde $G_1^* = g_1 + c_{qp} - c_{pi}$. Veja a figura 2.19c.
6. $s \leftarrow s + 1$
7. Procure então um vértice w tal que $g_s = c_{pq} - c_{pw}$ é maximizado. Compute o ganho $G_s = \sum_{1 \leq k \leq s} g_k$. Escolha o arco (p, w) para entrar na solução nesta iteração. Durante esse processo, os conjuntos de arcos que irão entrar e sair da solução são mantidos disjuntos. Se $w = i$ significa que a solução corrente já é um ciclo, então vá para o passo 8. Caso contrário, complete este passo da seguinte forma. Remova o arco (w, v) da solução corrente. Adicionando-se o arco (v, i) obtém-se um ciclo tal que $G_s^* = G_s + c_{wv} - c_{vi}$. Veja a figura 2.19d. Compute $G^* = max\{G_0^*, G_1^*, \ldots, G_s^*\}$.
8. $s \leftarrow s + 1$
9. volte ao passo 7 a menos que (a) não há mais trocas a serem verificadas; ou (b) $G_s \leq G^*$. Note que no caso (b), o ganho associado à solução corrente não é maior que o ganho associado ao melhor ciclo obtido até aqui. Portanto, (b) pode ser visto como um critério de parada para evitar buscas infrutíferas.
10. Realize a troca associada ao melhor dentre os valores do conjunto $\{G_0^*, G_1^*, \ldots, G_s^*\}$.

fim_procedimento

Figura 2.18: Algoritmo de busca em profundidade variável para o *TSP*

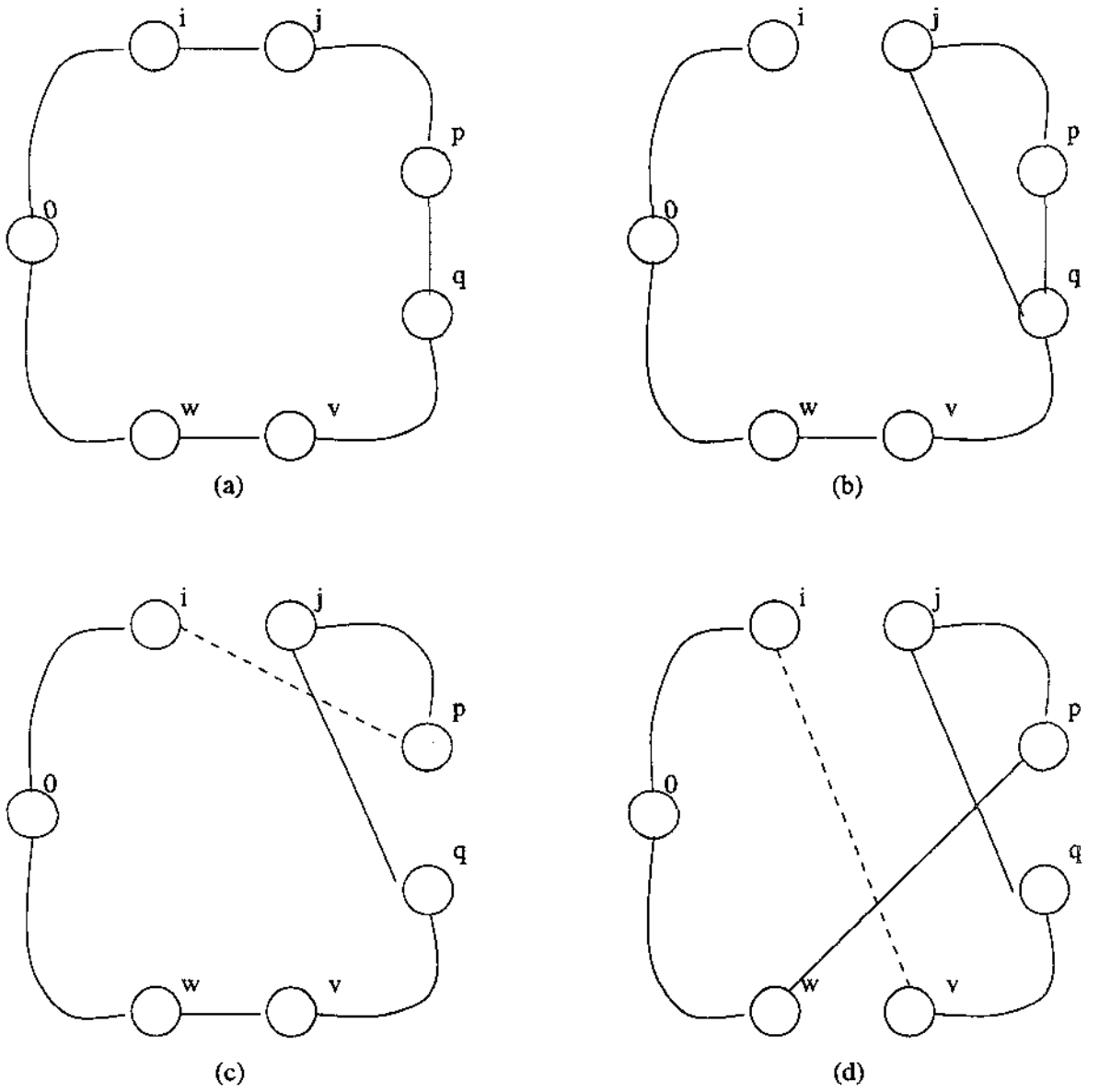


Figura 2.19: Exemplo de construção de uma nova rota a partir de uma rota pré-existente, na busca de profundidade variável.

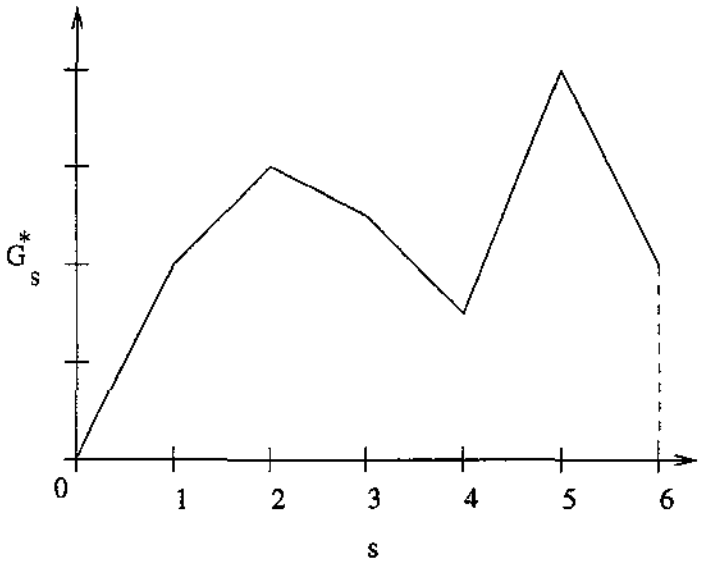


Figura 2.20: Comportamento de G^*_s em função da iteração s do algoritmo de trocas de profundidade variável.

procedimento $HEURISTICA\_SVTSPTW(S)$

1. **repita**
   - 1.1. $S \leftarrow 2\_EXCHANGE(S)$
   - 1.2. $S \leftarrow 1\_OR\_FORWARD\_EXCHANGE(S)$
   - 1.3. $S \leftarrow 1\_OR\_BACKWARD\_EXCHANGE(S)$
2. **até que** (S não pode mais ser melhorada)
3. $S \leftarrow 2\_OR\_FORWARD\_EXCHANGE(S)$
4. **se** S foi melhorada **então** vá para a linha 1
5. $S \leftarrow 2\_OR\_BACKWARD\_EXCHANGE(S)$
6. **se** S foi melhorada **então** vá para a linha 1
7. $S \leftarrow 3\_OR\_FORWARD\_EXCHANGE(S)$
8. **se** S foi melhorada **então** vá para a linha 1
9. $S \leftarrow 3\_OR\_BACKWARD\_EXCHANGE(S)$
10. **se** S foi melhorada **então** vá para a linha 1
11. **retorna** S

fim_procedimento

Figura 2.21: Algoritmo para resolução do $SVTSPTW$

procedimento *TESTE_DE_VIABILIDADE_PE_FIXO*(C)

1. $k \leftarrow i$, $\forall i \in P$
2. $k_0 \leftarrow k$
3. $k_{inicial} \leftarrow k$
4. Inicializa os valores dos e_k com as capacidades dos depósitos (d_k)
5. **enquanto** não foi achado nenhum fluxo válido e ainda existem depósitos para iniciar o teste, **faça**
   - **5.1.** $f \leftarrow min(c, e_k)$
   - **5.2.** $e_k \leftarrow e_k - f$
   - **5.3.** $k \leftarrow k + 1$ (módulo m)
   - **5.4.** **enquanto** $f \geq 0$ **ou** $k \neq k_0$ **faça**
     - **5.4.1.** **se** $k \in D$ **então** $f \leftarrow f - d_k$
     - **5.4.2.** **senão** $q \leftarrow min(c - f, e_k)$; $f \leftarrow f + q$; $e_k \leftarrow e_k - q$
     - **5.4.3.** $k \leftarrow k + 1$ (módulo m)
   - **5.5.** se $f \geq 0$ **então** C é válido. Pare.
   - **5.6.** se $f < 0$ **então** Seja ℓ o índice do próximo depósito no ciclo C. Reinicialize os e_k e **faça** $k \leftarrow \ell$. Repita o laço iniciado no passo 4.
   - **5.7.** se $k = k_{inicial}$ **então** pare. Não é possível encontrar mais nenhum fluxo válido em C. Este ciclo não é válido.

fim_procedimento

Figura 2.22: Algoritmo para o teste de viabilidade de uma solução para o *PE fixo*

Capítulo 3

Algoritmo Exato para o PE

Neste capítulo, será apresentado o algoritmo exato para o *PE*, baseado em *Programação Linear Inteira*. Antes disso, faz-se necessária uma introdução teórica a respeito das técnicas utilizadas no desenvolvimento do algoritmo. Em seguida, será apresentada a formulação em *Programação Linear Inteira* para o *PE* e o algoritmo propriamente dito.

3.1 Conceitos Básicos de *Programação Linear Inteira* e *Otimização Combinatória*

O objetivo dessa introdução à *Otimização Combinatória* é fornecer a base teórica necessária para a compreensão do modelo matemático formulado para o *PE* e o algoritmo exato desenvolvido para sua resolução. Não se pretende discutir o assunto em toda sua extensão. Para um tratamento completo do tema, refira-se a [Pap73, NW88, Wol98]. O restante desta seção é uma compilação das principais idéias apresentadas em [NW88, Wol98].

Um problema de *Otimização Combinatória* é composto basicamente por uma função que se deseja maximizar ou minimizar, com variáveis sujeitas a um conjunto de igualdades ou desigualdades, e um conjunto restrições de integralidade. Dada a sua generalidade, uma grande variedade de problemas pode ser representada por modelos discretos de otimização.

Tipicamente um problema de *Otimização Combinatória* pode ser modelado através de um *Problema de Programação Linear Inteira Mista*, ou *PLM* como

$$max\{cx + hy | Ax + Gy \leq b, x \in \mathbb{Z}_+^n, y \in \mathbb{R}_+^p\}, \tag{3.1}$$

onde $\mathbb{Z}_+^n$ é o conjunto dos vetores de dimensão n, inteiros e não negativos, $\mathbb{R}_+^p$ é o conjunto de vetores de dimensão p, reais e não negativos, e $x = (x_1, \ldots, x_n)$ e $y = (y_1, \ldots, y_p)$ são as variáveis. Uma *instância* do problema é definida pelos *dados* (c, h, A, G, b), sendo $c \in \mathbb{R}^n$, $h \in \mathbb{R}^p$, A uma matriz $m \times n$, G uma matriz $m \times p$ e $b \in \mathbb{R}^m$. O problema (3.1) é denominado misto devido à presença de variáveis inteiras e contínuas simultaneamente no modelo.

Problemas de otimização combinatória podem ser de maximização ou minimização e podem conter restrições sob forma tanto de igualdades como de desigualdades. Sem perda de genera-

lidade, problemas de otimização serão apresentados como problemas de maximização, sujeitos a desigualdades lineares. Note que minimizar uma função corresponde a maximizar a mesma função multiplicada por -1. Além disso, qualquer igualdade pode ser escrita como duas desigualdades. Em geral, as variáveis do problema são não-negativas.

O conjunto $S = \{x \in \mathbb{Z}_+^n, y \in \mathbb{R}_+^p, Ax + Gy \leq b\}$ é denominado *região viável*, e um par $(x, y) \in S$ é denominado *solução viável*. Em outras palavras, a *região viável* do problema é o espaço geométrico dos pontos que representam todas as suas soluções viáveis. Uma instância é considerada viável se $S \neq \emptyset$. A função $z = cx + hy$ é denominada *função objetivo* e a solução viável (x^0, y^0) em que a função objetivo assume o melhor valor possível (valor máximo nos problemas de maximização e valor mínimo nos de minimização) é denominada *solução ótima*, cujo *valor ótimo* é dado por $z^0 = cx^0 + hy^0$.

Resolver uma instância de um problema de *PLM* consiste em determinar uma solução ótima para o problema, ou mostrar que o problema é ilimitado (quando o valor ótimo é igual a $+\infty$ ou $-\infty$) ou inviável (região de viabilidade vazia, ou seja, $S = \emptyset$).

O *Problema de Programação Linear Inteira (Pura)*, ou *PLI*

$$max\{cx | Ax \leq b, x \in \mathbb{Z}_+^n\}, \tag{3.2}$$

é um caso especial de (3.1) em que não há variáveis contínuas.

O *Problema de Programação Linear*, ou *PL*

$$max\{hy | Gy \leq b, y \in \mathbb{R}_+^p\}, \tag{3.3}$$

é um caso especial de (3.1) em que não há variáveis inteiras.

Em muitos problemas, as variáveis inteiras representam relacionamentos lógicos ou de contingência, e em geral encontram-se restritos aos valores 0 ou 1. Quando um *PLI* contém variáveis deste tipo, este é denominado *PLI 0-1*. O conjunto $\mathbb{B}^n$ representa o conjunto dos vetores de dimensão n cujos coeficientes são 0 ou 1.

Como foi dito anteriormente, é possível modelar uma quantidade muito grande de problemas usando modelos de otimização combinatória. Entre eles, pode-se citar alguns bastante importantes e amplamente estudados, como por exemplo, o problema da mochila, os problemas de cobertura, empacotamento e particionamento de conjuntos, o problema da localização de facilidades, o problema do escalonamento de tripulação, de máquinas ou tarefas, o problema do caixeiro viajante, o problema do roteamento de veículos, o problema do fluxo em redes, o problema do fluxo em redes com custos fixos, entre outros.

De uma maneira geral, dado um problema combinatório definido por

$$max\{cx | x \in S \subset \mathbb{Z}_+^n\}, \tag{3.4}$$

onde S representa o conjunto de pontos viáveis em $\mathbb{Z}_+^n$, uma *formulação inteira* dada por

$$max\{cx | Ax \leq b, x \in \mathbb{Z}_+^n\}, \tag{3.5}$$

é dita *válida* para o problema em questão se $S = \{x \in \mathbb{Z}_+^n | Ax \leq b\}$.

3.1.1 Relação entre *PLs* e *PLIs*

Embora os problemas de programação linear (*PL*) sejam um caso particular dos problemas de programação linear inteira mista (*PLM*), resolver estes últimos pode ser uma tarefa muito mais complexa.

Um bom entendimento da teoria e dos algoritmos para programação linear é essencial para a compreensão da programação linear inteira por várias razões. Como será visto com mais detalhes, existe uma conexão muito forte entre as duas classes de problemas que é o fato de que para todo *PLI* existe um *PL* correspondente que tem a mesma solução ótima que o *PLI*.

Um *limitante superior* para um *PLI* de maximização é um valor maior ou igual ao da solução ótima do problema. Analogamente, um limitante inferior é um valor menor ou igual ao da solução ótima.

É possível encontrar um limitante superior resolvendo-se a *relaxação linear* de um *PLI*, que é o *PL* resultante da remoção das restrições de integralidade da formulação do *PLI*. Por essa razão, freqüentemente a resolução de *PLs* é parte de subrotinas de algoritmos de resolução de *PLIs*.

Problemas de programação linear inteira são freqüentemente muito mais difíceis de se resolver que os de programação linear. Uma vez que vários problemas *$\mathcal{NP}$-difíceis* podem ser modelados como *PLIs*, *programação linear inteira* também é *$\mathcal{NP}$-difícil*. Por outro lado, os problemas de *PL* são polinomiais.

Para entender a diferença de complexidade entre os dois problemas é necessário conhecer mais a respeito das estruturas poliédricas de ambos os problemas.

Para os propósitos do presente trabalho, será apresentada nesta seção uma idéia intuitiva do relacionamento entre a estrutura poliédrica da regiões viáveis de *PLs* e *PLMs*.

Na seção 3.1.2 discute-se as características das boas formulações de *PLI*. Por fim, na seção 3.1.3 é apresentada a teoria poliédrica básica necessária para o bom entendimento do restante deste trabalho. Para uma explanação completa a respeito deste tema refira-se a [NW88].

Um *poliedro* $P \subseteq \mathbb{R}^n$ é o conjunto de pontos que satisfazem um número finito m de desigualdades lineares, isto é, $P = \{x \in \mathbb{R}^n | Ax \leq b\}$, onde (A, b) é uma matriz $m \times (n+1)$. Um poliedro é dito *racional* se existe a matriz (A', b') de dimensões $m' \times (n+1)$ com coeficientes racionais tal que $P = \{x \in \mathbb{R}^n | A'x \leq b'\}$.

É possível provar que existe uma solução ótima de um *PL* viável e limitado que se encontra em um dos vértices do poliedro [BJ90, NW88]. Existem dois algoritmos que resolvem problemas de *PL*. O primeiro algoritmo é o chamado *algoritmo de pontos interiores* que tem tempo de execução limitado por um polinômio que é função do tamanho da instância.

O outro algoritmo é o *Simplex*. A sua idéia básica é visitar os pontos viáveis que são vértices do poliedro do *PL* sempre na direção que melhora (ou pelo menos mantém igual) o valor da função objetivo. Embora este algoritmo tenha tempo de execução limitado superiormente por uma função exponencial (pois no pior caso podem ser visitados todos os vértices do poliedro, que são em número exponencial em função do tamanho da instância), na maioria dos casos, o algoritmo é bastante eficiente. Para maiores detalhes a respeito da teoria e dos algoritmos da

programação linear, refira-se a [BJ90, Pap73].

A título de ilustração, considere o *PLI* definido pelo conjunto de restrições (3.6) abaixo.

$$\begin{aligned} 2x_1 + x_2 &\leq 13/2 \\ -3/4x_1 + x_2 &\leq 3/2 \\ x_1 + x_2 &\geq 3/2 \\ x_1 &\geq 0 \\ x_2 &\geq 1/2 \\ x_1, x_2 &\in \mathbb{Z} \end{aligned} \tag{3.6}$$

As regiões viáveis do *PLI* e da relaxação linear correspondente são apresentadas nas figuras 3.1 e 3.2, respectivamente.

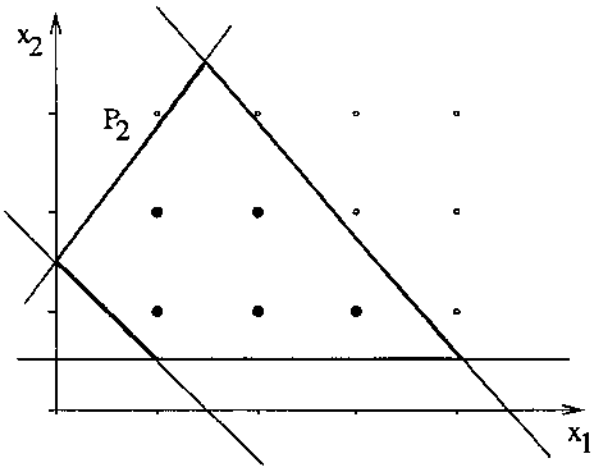


Figura 3.1: Exemplo da região viável de um *PLI*

Pode-se perceber que, neste caso, não há nenhum ponto viável para o *PLI* sobre as arestas do poliedro definido pelas desigualdades lineares. Portanto, não é possível utilizar os algoritmos da programação linear para resolver o caso inteiro.

Em geral, uma formulação válida para um problema de programação inteira define o conjunto $S \subseteq \mathbb{Z}_+^n$ de pontos viáveis em que S é descrito implicitamente, pela interseção do poliedro racional definido pelo sistema de desigualdades lineares $P = \{x \in \mathbb{R}_+^n | Ax \leq b\}$ com variáveis reais e o conjunto dos pontos inteiros contidos nesse poliedro. Ou seja, o conjunto $X = P \cap \mathbb{Z}_+^n$ define o conjunto de pontos viáveis do *PLI*.

Entretanto, existe um poliedro da forma de P para o qual a resolução do *PLI* se reduz à resolução do *PL* relaxado. Esse poliedro corresponde à *envoltória convexa* de P que é definida formalmente a seguir.

Dado um conjunto $S \subseteq \mathbb{R}^n$, um ponto $x \in \mathbb{R}^n$ é uma *combinação convexa* dos pontos de S se existe um conjunto finito de pontos $\{x^i\}_{i=1}^t \in S$ e $\lambda \in \mathbb{R}_+^t$ com $\sum_{i=1}^t \lambda_i = 1$ e $x = \sum_{i=1}^t \lambda_i x^i$.

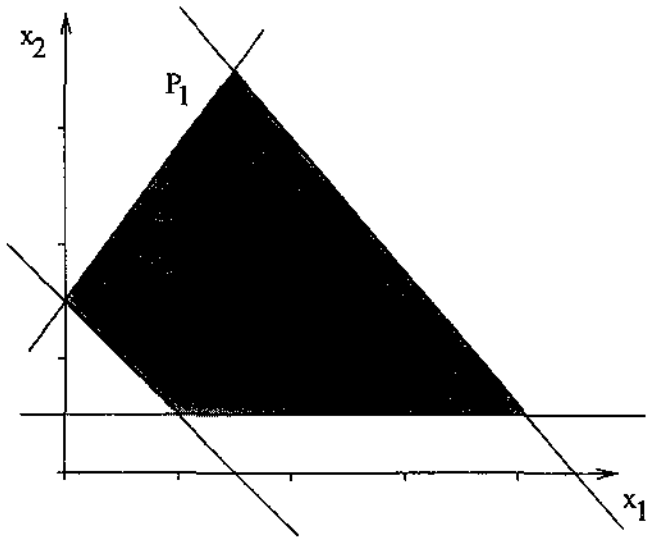


Figura 3.2: Exemplo da região viável de um *PL*

A *envoltória convexa* de S, denotada por $conv(S)$, é o conjunto de todos os pontos que são combinações convexas dos pontos de S.

Todos os vértices da envoltória convexa de um *PLI* são pontos inteiros viáveis. Portanto, a solução ótima do *PL* relaxado é inteira e é também solução ótima do *PLI*.

Entretanto, para a grande maioria dos problemas combinatórios formulados como problemas de programação linear inteira, descrever a envoltória convexa de seus pontos viáveis através de desigualdades lineares pode ser uma tarefa muito difícil. Além disso, em geral, o número de desigualdades cresce exponencialmente com o número de variáveis. O que se pode fazer é encontrar boas formulações para o problema que mais se aproximem da sua envoltória convexa.

A discussão sobre o que define a qualidade de uma formulação e como estas são obtidas segue nas próximas seções.

3.1.2 Boas formulações para *PLIs*

Considerando um *PLI* específico, existem infinitas formulações válidas que representam o mesmo conjunto de pontos viáveis para o problema. Em geral, é relativamente fácil encontrar um conjunto de desigualdades que represente uma formulação válida. Entretanto, uma escolha óbvia pode não apresentar bons resultados na resolução do problema. Encontrar uma boa formulação para o problema, como será mostrado a seguir, é fundamental para resolvê-lo eficientemente.

Pode-se comparar duas formulações diferentes para o mesmo problema através da análise dos limitantes superiores (problema de maximização) que podem ser obtidos da relaxação linear de cada formulação. Um limitante superior pode ser determinado resolvendo-se o seguinte *PL*:

$$z_{LP} = \{max\ cx :\ Ax \leq b,\ x \in I\!R^n_+\} \tag{3.7}$$

onde $P = \{x \in \mathbb{R}^n_+ : Ax \leq b\} \supseteq S$. Dadas duas formulações válidas, definidas por $i = 1, 2$ e $P^i = \{x \in \mathbb{R}^n_+ : A^i x \leq b^i\}$ e $z^i_{LP} = \{max\ cx : x \in P^i\}$. Note que se $P^1 \subseteq P^2$ então $z^1_{LP} \leq z^2_{LP}$, e neste caso, a formulação (A^1, b^1) é considerada uma formulação melhor que (A^2, b^2).

A figura 3.3 exemplifica a definição acima.

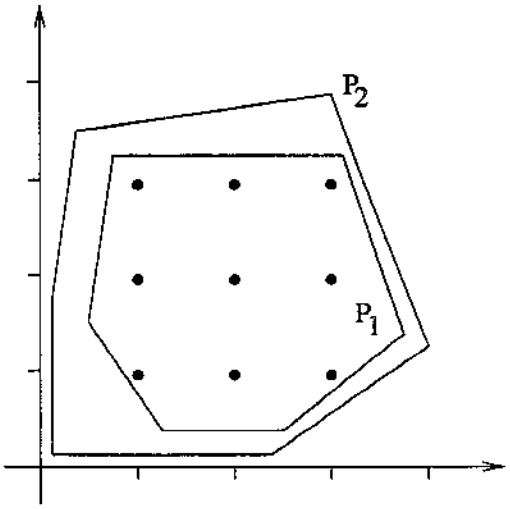


Figura 3.3: Exemplo de duas formulações em que a formulação que define o poliedro P_1 é melhor que a formulação que define o poliedro P_2.

Claramente, a melhor formulação possível para um problema é aquela em que a envoltória convexa é totalmente descrita explicitamente. Como foi visto anteriormente, existe uma representação de um *PLI* por um *PL* tal que ambos têm a mesma solução ótima. Tal representação é dada por

$$max\{cx|x \in S\} = max\{cx|x \in conv(S)\} \tag{3.8}$$

Embora seja difícil representar $conv(S)$ na maioria dos casos, pode-se obter boas formulações para o problema através de uma análise de suas desigualdades segundo a teoria de combinatória poliédrica apresentada a seguir.

3.1.3 Combinatória poliédrica

Como mostrado anteriormente, em teoria, para se resolver um *PLI*, cuja região viável é representada pelo conjunto S, é necessário apenas conhecer a descrição de $conv(S)$. Assim, é possível estudar a estrutura poliédrica de um problema em particular, a fim de se descobrir quais desigualdades são importantes para que se tenha uma boa formulação para o problema. Nesta seção será apresentado um conjunto mínimo de conceitos da *combinatória poliédrica* que servirá de base para a identificação das boas desigualdades lineares para a formulação dos problemas tratados neste trabalho.

Inicialmente, são relembrados algumas definições básicas de *Algebra Linear*:

- Os pontos $\{x_1, \ldots, x_n\}$ são *linearmente independentes* se $\sum_{i=1}^{n} \lambda_i x_i = 0$ implica em $\lambda_i = 0$, $\forall i = 1, \ldots, n$.
- O *posto* de uma matriz $A_{m \times n}$ é igual ao número máximo de linhas ou colunas linearmente independentes de A. O posto de A é denotado por $\rho(A)$.
- Os pontos $\{x_1, \ldots, x_n\}$ são *afim independentes* se $\sum_{i=1}^{n} \lambda_i x_i = 0$ e $\sum_{i=1}^{n} \lambda_i = 0$ implica em $\lambda_i = 0$, $\forall i = 1, \ldots, n$.
- O *posto afim* de $S \subseteq \mathbb{R}^n$ é igual ao número máximo de vetores afim independentes em S, denotado por $\rho_A(S)$. Se S é não vazio e dado por $S = \{x \in \mathbb{R}^n | Ax = b\}$, o número máximo de vetores afim independentes em S é igual a $n + 1 - \rho(A)$.
- A *dimensão* de $S \subseteq \mathbb{R}^n$ é dada por $dim(S) = \rho_A(S) - 1$. Além disso, S é dito ser de *dimensão cheia* se, e somente se, $dim(S) = dim(\mathbb{R}^n) = n$.
- A desigualdade $\pi x \leq \pi_0$ é uma *desigualdade válida* para $P \subseteq \mathbb{R}^n$ se, e somente se, $\pi y \leq \pi_0$ para todo $y \in P$.

Esta última definição é fundamental.

Nesse ponto, já é possível verificar que qualquer desigualdade presente na formulação deve ser válida para o *PLI* em consideração. Entretanto, apenas um subconjunto dessas desigualdades válidas são importantes para a descrição da envoltória convexa desse problema. Estas desigualdades são caracterizadas a partir das seguintes definições.

Sejam $\pi x \leq \pi_0$ e $\mu x \leq \mu_0$ duas desigualdades válidas para $P \subseteq \mathbb{R}^n_+$. A desigualdade $\pi x \leq \pi_0$ *domina* a desigualdade $\mu x \leq \mu_0$ se existe $\alpha > 0$ tal que $\pi \geq \alpha\mu$ e $\pi_0 \leq \alpha\mu_0$.

Se $\pi x \leq \pi_0$ é uma desigualdade válida para $P \subset \mathbb{R}^n$, $F = \{x \in P | \pi x = \pi_0\}$ é a *face* definida em P por esta desigualdades. Diz-se que $\pi x = \pi_0$ *representa* F. Ainda, se $F \neq \emptyset$ e $F \neq P$, então F é denominada uma *face própria* de P.

Seja F uma face de P, F é uma *faceta* de P se, e somente se, $dim(F) = dim(P) - 1$.

As *facetas* de um poliedro P não são dominadas por nenhuma outra desigualdade. Além disso, o conjunto de todas as facetas de um poliedro P descrevem $conv(P)$.

Idealmente, procura-se uma formulação para um *PLI* que contenha todas as facetas que definem o poliedro inteiro deste problema. Entretanto, pode ser difícil encontrar todas essas desigualdades. Nesse caso, deve-se tentar incluir na formulação desigualdades que não são dominadas por outras desigualdades conhecidas.

Para identificar se uma desigualdade é uma faceta do poliedro do *PLI*, segundo a definição de faceta, primeiro é necessário conhecer a dimensão do poliedro do problema, visto que as facetas têm dimensão uma unidade a menos que a dimensão do poliedro.

Sejam $P = \{Ax \leq b\}$ um poliedro, onde A é uma matriz $m \times n$ e b um vetor de dimensão m. O conjunto M representa o conjunto dos índices das linhas de A. Este é particionado em dois subconjuntos: $M^=$ e $M^<$. O conjunto $M^= \in M$ contém os índices das linhas de A que são

equações e $M^{<} \in M$ contém os índices das linhas correspondentes às inequações. Note que se $i \in M^{<}$, então (a^i, b_i) não pode ser escrita como uma combinação linear das linhas de $(A^{=}, b^{=})$.

Seja $(A^{=}, b^{=})$ o conjunto de equações de $P \subseteq I\!R^n$ e $F = \{x \in P | \pi x = \pi_0\}$ uma face própria de P. As duas afirmações abaixo são equivalentes:

- A face F é uma faceta de P.
- Se $\lambda x = \lambda_0$, $\forall x \in F$ então $(\lambda, \lambda_0) = (\alpha\pi + uA^{=}, \alpha\pi_0 + ub^{=})$ para algum $\alpha \in I\!R$ e para algum $u \in I\!R^{|M^{=}|}$.

Ou seja, uma face própria F de P é uma faceta de P se esta pode ser escrita como uma combinação linear das equações $(A^{=}, b^{=})$ com a desigualdade (π, π_0). Se $P \subseteq I\!R^n$, então $dim(P) + \rho(A^{=}, b^{=}) = n$.

O processo de se determinar se uma dada desigualdade válida é uma faceta é muito mais simples quando o poliedro do problema em questão tem *dimensão cheia*. Relembrando a definição feita anteriormente, um poliedro $P \in I\!R^n$ tem *dimensão cheia* quando $dim(P) = n$. Isso decorre do fato de que quando $dim(P) = n$ significa que o $posto(A^{=}, b^{=}) = 0$, ou seja, não existem igualdades (implícitas ou explícitas) na representação de P. Nesse caso, as desigualdades que determinam facetas em P são únicas (a menos de multiplicação por escalar positivo), tornando simples a identificação de desigualdades redundantes.

Assim, um poliedro de dimensão cheia tem uma única representação minimal (a menos da multiplicação por escalar positivo) formada por um conjunto finito de desigualdades lineares. Especificamente, para cada faceta F_i de P existe uma desigualdade $a^i x \leq b_i$ única (a menos da multiplicação por escalar positivo) representando F_i e $P = \{x \in I\!R^n | a^i x \leq b_i, \text{para } i = 1, \ldots, m\}$.

No caso de poliedros de dimensão não cheia ($dim(P) < n$), é mais difícil identificar se duas desigualdades são ou não redundantes. Uma vez que o conjunto de equações em $(A^{=}, b^{=})$ não é vazio, as desigualdades redundantes podem envolver combinações lineares dessas equações. Na verdade, quanto maior o número de equações em $(A^{=}, b^{=})$, mais complexa se torna a tarefa de determinar desigualdades redundantes. Essa foi a principal razão pela qual não foi realizada a análise da dimensão do poliedro do *PE*. Como será visto mais adiante, a formulação aqui proposta para o problema contém muitas equações.

Existe ainda uma técnica para obtenção de desigualdades fortes, usada neste trabalho, conhecida como *lifting*. O *lifting* de uma desigualdade consiste em alterar o valor original dos seus coeficientes de forma a obter uma nova desigualdade válida que domine a desigualdade original.

Existem vários algoritmos para a resolução de *PLIs* baseados nos resultados apresentados anteriormente nessa seção. A seguir são apresentados exemplos desses algoritmos que foram utilizados para resolver o problema tratado na nesta dissertação.

3.1.4 Algoritmo de *Branch and Bound*

O algoritmo de *Branch and Bound* é baseado numa estratégia de divisão e conquista em que a região viável de um *PLI* é sucessivamente decomposta em subregiões menores correspondendo a subproblemas de *PLI* mais simples.

Seja o problema $z = max\{cx|x \in S\}$. Considere a decomposição do conjunto S em subconjuntos $S_1, \ldots, S_k$. Cada subconjunto S_i define a região viável do conjunto de subproblemas $z_i = max\{cx|x \in S_i\}$. Para cada $i = 1, \ldots, k$, são conhecidos os limitantes superiores e inferiores de z_i, $\overline{z_i}$ e $\underline{z_i}$, respectivamente. Pode-se então obter um limitante superior $\overline{z}$ e um limitante inferior $\underline{z}$, globais para o problema original, a partir das expressões

$$\overline{z} = max_{i=1,\ldots,k}\{\overline{z_i}\} \tag{3.9}$$

e

$$\underline{z} = max_{i=1,\ldots,k}\{\underline{z_i}\}. \tag{3.10}$$

O processo de decomposição sucessiva da região S pode ser visualizado através de uma árvore. Para isso, seja $T = (V, E)$ o grafo não-direcionado onde cada vértice de V corresponde a um subconjunto S_j da decomposição de S. Existe uma aresta entre dois vértices $u, v \in V$, representando os conjuntos $S_i e S_j$, respectivamente, se o conjunto S_i foi gerado por uma decomposição do conjunto S_j ou vice-versa. Esse grafo representa a *árvore de enumeração implícita* do algoritmo de *Branch and Bound* e descreve o conjunto de regiões S_i exploradas pelo algoritmo.

Durante a execução do algoritmo, três fases são executadas. São elas:

1. *branching* (ou ramificação);
2. *bounding* (ou obtenção dos limitantes globais);
3. *pruning* (ou amadurecimento);

Na fase de *branching* é feita a decomposição de um subconjunto S_i nos subconjuntos $S_{i1} \cup S_{i2} \cup \ldots \cup S_{ik}$. O subproblema para o qual o conjunto S_i será decomposto é escolhido entre todos os subproblemas representados por folhas da árvore de enumeração que ainda não tenham sido amadurecidos(veja descrição de amadurecimento a seguir).

Na fase de *bounding*, uma vez determinados os limitantes inferiores e superiores locais, são calculados os limitantes globais pelas expressões (3.9) e (3.10). Os limitantes locais são obtidos, em geral, resolvendo-se a relaxação linear do subproblema em questão. Outros métodos também podem ser utilizados, dentre os quais podem ser citados a *Relaxação Lagrangeana* e o uso de *Heurísticas Primais.*

Na fase de *prunning*, os vértices da árvore de enumeração são *amadurecidos* de acordo com os valores dos limitantes obtidos na fase de *bounding.* O amadurecimento de um vértice u ocorre quando fica provado que não é mais necessário explorar a sua subárvore de enumeração (vértices que seriam criados a partir da decomposição da região viável do subproblema correspondente ao vértice u da árvore de enumeração). Um vértice pode ser amadurecido por três razões: por limitantes, por inviabilidade ou por otimalidade.

Um vértice u é amadurecido por limitantes quando é possível concluir, através desses valores, que nenhuma solução viável potencialmente existente na subárvore cujo vértice u é raiz pode ser a solução ótima do problema original. Pode-se chegar a esta conclusão analisando o valor dos limitantes locais com os limitantes globais. Para o caso de um problema de maximização, isso ocorre quando $\overline{z_u} \leq \underline{z}$.

O amadurecimento por otimalidade de um vértice u ocorre quando ambos os limitantes inferior $\underline{z_u}$ e superior $\overline{z_u}$ são iguais. Nesse caso, nenhum vértice i descendente de u na árvore de enumeração pode ter melhores limitantes que u.

Um vértice é amadurecido por inviabilidade quando sua região viável é vazia. Qualquer partição dessa região resultará em um conjunto de regiões igualmente inviáveis, tornando desnecessária a exploração destes vértices da árvore de enumeração.

A execução do algoritmo termina quando a diferença (absoluta ou relativa) α entre os limitantes superior e inferior globais do problema é menor que uma certa constante ϵ. Alternativamente, pode-se limitar a execução do algoritmo a um certo número de iterações, ou a um certo tempo de utilização de CPU. A melhor solução encontrada durante a execução é retornada, caso alguma tenha sido encontrada.

3.1.5 Algoritmos de planos de corte

Os algoritmos de planos de cortes são apropriados para resolver problemas de programação linear inteira quando se dispõe de relaxações do *PLI* $max\{cx|x \in S\}$, onde $S = \{x \in Z_+^n | Ax \leq b\}$ e um conjunto de desigualdades válidas "fortes" $\mathcal{F}$ para S. Pode-se utilizar esses algoritmos também para resolver problemas de programação linear com um grande número de restrições.

Este algoritmo é particularmente eficiente nos casos em que é impraticável adicionar todas as desigualdades de $\mathcal{F}$ *a priori* quando o número de desigualdades nesse conjunto é muito grande. Além disso, definida a função objetivo, apenas um subconjunto muito reduzido da família de desigualdades $\mathcal{F}$ é necessária para a resolução do *PLI*. No algoritmo de planos de corte, apenas as desigualdades de $\mathcal{F}$ com chances de estarem ativas na solução ótima são geradas.

O algoritmo geral é mostrado na figura 3.4.

Dada a solução x^t para a relaxação $LP(\mathcal{F})^t$, é necessário mostrar que x^t é uma solução viável do *PLI* ou encontrar uma desigualdade válida $(\pi, \pi_0) \in \mathcal{F}$ para a qual $\pi x^t > \pi_0$. Este problema é chamado de *problema da separação*.

A vantagem de se separar desigualdades válidas à medida em que são violadas pelo ponto ótimo fracionário do *PL*, em contraste à adição de todas as desigualdades dessa família *a priori*, é que são consideradas apenas aquelas desigualdades que são relevantes para a descrição do poliedro inteiro na direção da função objetivo. Isso é especialmente importante no caso em que o número de desigualdades de uma família é muito grande. Como será visto mais adiante, esse é o caso das desigualdades de eliminação de subciclos do *problema do caixeiro viajante* (*traveling salesman problem*, ou *TSP*, em inglês).

Os primeiros algoritmos de planos de cortes foram criados por Gomory [Gom58, Gom60, Gom63] que em princípio seriam capazes de gerar, após um número finito de passos, todas as

Inicialização: $S_R^1 = \{x \in \mathbb{R}_+^n | Ax \leq b\}$; $t \leftarrow 1$.

Iteração t:

Passo 1: Resolva a relaxação $LP(\mathcal{F})$ do *PLI* $z_R^t = max\{cx | x \in S_R^t\}$, com solução ótima x^t.

Passo 2: Teste de Otimalidade. Se $\pi x^t \leq \pi_0$, $\forall (\pi, \pi_0) \in \mathcal{F}$, então pare.

Passo 3: Refinamento. Seja $\mathcal{F}^t \subset \mathcal{F}$ o conjunto de uma ou mais desigualdades (π, π_0) com $\pi x^t > \pi_0$ e $S_R^{t+1} = S_R^t \cap \{x \in \mathbb{R}_+^n | \pi x \leq \pi_0$, para $(\pi, \pi_0) \in \mathcal{F}^t\}$.

Passo 4: $t \leftarrow t + 1$.

Figura 3.4: Algoritmo de planos de corte para as desigualdades da família $\mathcal{F}$

facetas dos poliedros de qualquer *PLI*. Esses algoritmos, embora tenham influenciado de forma muito importante a teoria da programação inteira, não foram muito utilizados nas últimas décadas, devido ao fato de que os cortes gerados raramente geram desigualdades fortes. O número finito de iterações para se obter todas as facetas de um poliedro, usando este método, é em geral muito grande, na maioria das vezes, é intratável. Entretanto, vêm sendo reportados, nos últimos anos, alguns casos de sucesso com a utilização destes algoritmos de planos de cortes genéricos, como por exemplo, em [BCCN96].

Algoritmos mais bem sucedidos começaram a ser desenvolvidos para problemas específicos, cujas desigualdades da família $\mathcal{F}$ são obtidas a partir do estudo teórico do poliedro do problema, usando a *combinatória poliédrica* como base para seleção das desigualdades mais apropriadas.

3.1.6 Algoritmos de *Branch and Cut*

O algoritmo de *Branch and Cut* é a aplicação combinada dos algortimos de *Branch and Bound* e de planos de corte.

A idéia é, antes de se executar a fase de *branching* do algoritmo de *Branch and Bound*, sejam adicionados novas desigualdades fortes da família $\mathcal{F}$, válidas para o problema, através de um algoritmo de planos de corte. Dessa forma, é possível contornar os problemas existentes quando se aplicam os algoritmos de *Branch and Bound* e de planos de corte isoladamente. Com o ganho nos limitantes obtido com a adição de desigualdades na fase de inserção de planos de corte, é possível diminuir o número de vértices explorados da árvore de enumeração. Por outro lado, quando nenhuma desigualdade é encontrada durante a fase de planos de corte, é possível passar à fase de *branching*, continuando a execução do algoritmo (o que não era possível no algoritmo de plano de cortes puro).

Há aqui o compromisso entre a eficiência na separação dos cortes com relação aos ganhos obtidos nos limitantes de cada vértice da árvore de enumeração.

3.2 Formulação em *PLI* para o *PE*

O conhecimento da estrutura poliédrica de problemas como *TSP* e o problema da mochila, por exemplo, traz algumas vantagens na modelagem de problemas mais complexos.

Considere dois conjuntos S_1 e S_2. Desigualdades válidas para S_1 e S_2 são também válidas para um problema $S = S_1 \cap S_2$. Infelizmente, nesse processo, algumas propriedades das desigualdades dos problemas iniciais são perdidas quando aplicadas aos problemas mais complexos. Por exemplo, aquelas desigualdades que definem facetas do $conv(S_1)$ ou do $conv(S_2)$, não necessariamente definem facetas em $conv(S)$. Mesmo assim, essas desigualdades podem ajudar a melhorar os valores dos limitantes do problema.

Essa abordagem é utilizada neste trabalho, no tratamento do *PE*. Sua semelhança com três problemas conhecidos e bastante estudados levou a uma formulação híbrida, combinando desigualdades válidas para os três problemas. Esses problemas, que serão discutidos a seguir são: o *problema do caixeiro viajante* (*TSP*), o *problema do roteamento de veículos* (*vehicle routing problem*, ou *VRP*, em inglês) e o *problema do fluxo em redes com custos fixos* (*fixed-charge network flow problem*, ou *FCN*, em inglês).

3.2.1 O problema do caixeiro viajante (*TSP*)

No *TSP*, um caixeiro viajante deve visitar um conjunto de cidades, onde deseja vender seus produtos, de maneira a percorrê-las no menor tempo possível. Formalmente, são dados um conjunto de vértices $V = \{1, \ldots, n\}$ e um conjunto de arcos A, formando o grafo direcionado $G = (V, A)$. Cada cidade é representada por um vértice em V, e uma aresta direcionada $(u, v) \in A$, $u, v \in V$ se existe a possibilidade do caixeiro viajar da cidade representada pelo vértice u diretamente para o vértice correspondente à cidade v. A cada aresta $(u, v) \in A$ é atribuído um peso c_{uv} que representa o tempo de viagem da cidade u diretamente para a cidade v. No caso do problema simétrico, tem-se $c_{uv} = c_{vu}$. O problema é encontrar um *ciclo* C no grafo G tal que cada vértice de V seja visitado exatamente uma vez e que o tempo total de viagem (soma dos pesos das arestas no ciclo C) seja minimizado.

Durante muitos anos, o *TSP* tem despertado o interesse de muitos pesquisadores em diferentes áreas de pesquisa (matemática, pesquisa operacional, física, biologia ou inteligência artificial) no mundo todo. Há uma vasta coleção de trabalhos e estudos a seu respeito na literatura. Isso se deve em grande parte ao fato de que, embora seja facilmente enunciado, é um problema de resolução extremamente difícil. Além disso, o *TSP* exibe todos os principais aspectos da otimização combinatória, sendo portanto um dos exemplos mais proeminentes deste rico conjunto de problemas.

A importância dada ao *TSP* é devida também ao fato de que o modelo matemático em que se baseia serve para modelar uma grande variedade de outros problemas combinatórios.

Apesar de ser um problema *$\mathcal{NP}$-difícil* (como pode-se verificar a partir de uma redução simples do problema do ciclo hamiltoniano em grafos), enormes avanços em sua resolução foram alcançados nas últimas duas décadas. Desde o início da década de 80, o número de vértices da maior instância resolvida aumentou de algumas centenas para os milhares, ou seja, foi acrescida

em uma ordem de grandeza. Como citado por Reinelt [Rei94], os grandes marcos na busca de resultados ótimos para o *TSP* foram 120 cidades [Grö80], 318 cidades [CP80], 532 cidades [PR87], 666 cidades [Grö91], 2392 cidades [PR91], 3038 cidades [ABCC91] e 4461 cidades [ABCC93].

Embora parte desse sucesso tenha sido devido ao aumento do poder computacional dos computadores, o desenvolvimento da teoria matemática, em especial da combinatória poliédrica, e de novos algoritmos contribuiu com maior importância para que esses resultados fossem alcançados.

Entretanto, a despeito desses resultados, o *TSP* está longe de ser resolvido eficientemente, visto que é um problema *$\mathcal{NP}$-difícil.* Vale lembrar que os resultados apresentados acima foram aqueles obtidos para as maiores instâncias do problema já resolvidas, o que não significa que qualquer instância de tamanho comparável a estas seja facilmente resolvida.

Dada sua dificuldade de resolução, foram desenvolvidos muitos algoritmos heurísticos para o *TSP*. Entre as principais classes, pode-se citar os algoritmos gulosos, de inserção, e os de $k-$trocas (vide seção 2.3.3).

Muitas das soluções exatas para o *TSP* são baseadas em *programação linear inteira.* Uma dessas formulações, válida para o caso assimétrico do *TSP*, é apresentada a seguir.

$$min \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{3.11}$$

$$s.a. \quad \sum_{i:(i,j)\in A} x_{ij} = 1,\ \forall j \in V \tag{3.12}$$

$$\sum_{j:(i,j)\in A} x_{ij} = 1,\ \forall i \in V \tag{3.13}$$

$$\sum_{(i,j)\in A: i\in U,\ j\notin U} x_{ij} \leq |U| - 1,\ \forall\, U \subset V\ com\ 2 \leq |U| \leq |V| - 2 \tag{3.14}$$

$$x \in \mathbb{B}^{|A|} \tag{3.15}$$

As equações representadas por (3.12) e (3.13) são as restrições de conectividade, usualmente chamadas de *restrições de grau*, pois garantem que o grau de entrada e de saída de um vértice é sempre igual a 1, ou seja, exatamente uma aresta do ciclo entra e exatamente uma aresta sai de cada vértice do grafo.

As inequações em (3.14) garantem que nenhum subciclo fará parte da solução. Estas restrições são as conhecidas *restrições de eliminação de subciclos* do *TSP*. Estas definem facetas do politopo do problema (vide [GP85]).

Em (3.14) a desigualdade é expressa sob forma de limitação de arestas em conjuntos. Equivalentemente, pode-se representá-las sob forma de corte como na desigualdade abaixo.

$$\sum_{i\in S, j\in S\setminus U} x_{ij} \geq 1,\ com\ 2 \leq |U| \leq \lfloor |V|/2 \rfloor \tag{3.16}$$

A equivalência entre as desigualdades (3.14) e (3.16) deve-se à existência das desigualdades de grau (3.12) e (3.13).

Vale lembrar que o número de desigualdades representadas por (3.14) ou (3.16) é exponencial com relação ao número de vértices da instância do problema. Dessa forma, é praticamente impossível adicioná-las todas *a priori* na formulação. Entretanto, essas desigualdades podem ser separadas e adicionadas à formulação do problema em algoritmos de planos de cortes ou de *Branch and Cut* sem problemas, visto que o problema de separação para essas desigualdades pode ser resolvido por algoritmos polinomiais no tamanho da entrada [PG85, NOI94].

Este fato é especialmente importante pois existe um resultado fundamental da otimização combinatória que garante que se o problema de separação de desigualdades é polinomial, então o problema de otimização usando a separação também é polinomial, independentemente do número de desigualdades ser exponencial. No caso do *TSP*, isso garante que a relaxação linear do problema pode ser resolvida em tempo polinomial no tamanho da entrada.

Na seção 3.3 as desigualdades de eliminação de subciclos, inclusive sua separação, serão discutidas em mais detalhes.

Grande parte dos algoritmos exatos para o *TSP* baseados em *PLI*, utilizam relaxações de algumas das restrições da formulação dada por (3.15). Em [NW88, GP85] são encontrados exemplos de algoritmos baseados na relaxação das restrições de eliminação de subciclos, chamadas relaxações $1-tree$ e $2-matching$. Em [Rei94] é apresentado um algoritmo que utiliza a *relaxação lagrangeana*.

Além das desigualdades de eliminação de subciclos, outras facetas do *TSP* são conhecidas, tais como as desigualdades *comb* e *clique-tree* [NW88, GP85]. Muitos algoritmos bem sucedidos utilizam tais desigualdades na fase de planos de cortes de algoritmos de *Branch and Cut*. Em [GP85] são reportados bons resultados utilizando-se a separação de *cortes fracionários de Gomory* no vértice raiz da árvore de enumeração em algoritmos de *Branch and Cut*.

3.2.2 O problema do roteamento de veículos (*VRP*)

A versão mais popular do *VRP* consiste em alocar uma frota de veículos idênticos que devem fazer entregas a um conjunto de clientes partindo de um depósito central. O modelo é definido pelos seguintes parâmetros

- $m \rightarrow$ número de veículos na frota;
- $n \rightarrow$ número de clientes onde devem ser feitas as entregas. Os clientes são indexados de 2 até n e o índice 1 denota o depósito central;
- $c \rightarrow$ capacidade (de peso ou volume) de cada veículo;
- $d_i \rightarrow$ peso ou volume dos produtos entregues no cliente i (sempre medido na mesma unidade da capacidade dos veículos);
- $w_{ij} \rightarrow$ custo da viagem direta entre os clientes i e j (igual ao custo de i para j).

O problema é determinar m rotas para os veículos, onde uma rota é um ciclo que inicia no depósito, passando por um subconjunto de clientes numa seqüência especificada e ao final

retornando ao depósito central novamente. Cada cliente deve ser servido exatamente uma vez e por um único veículo e cada veículo não pode ter sua capacidade c excedida. As rotas obtidas como solução do problema devem minimizar a soma dos tempos gastos pelos veículos.

Abaixo é dada um exemplo de formulação válida para o caso simétrico do *VRP*.

$$min \quad \sum_{i<j} w_{ij}x_{ij} \tag{3.17}$$

$$sujeito\ a \quad \sum_{j=2}^{n} x_{1j} = 2m, \tag{3.18}$$

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2,\ (k = 2, \ldots, n), \tag{3.19}$$

$$\sum_{i,j\in S} x_{ij} \leq |S| - v(S),\ \forall\, S \subset V\ com\ 2 \leq |S| \leq |V| - 2\} \tag{3.20}$$

$$x_{1j} \in \{0, 1, 2\},\ (j = 2, \ldots, n), \tag{3.21}$$

$$x_{ij} \in \{0, 1\},\ (i, j = 2, \ldots, n), \tag{3.22}$$

Na formulação do *VRP* dada pelas expressões (3.17) até (3.22), a função objetivo (3.17) minimiza o custo da rota de entrega. A equação (3.18) garante que o grau do vértice correspondente ao depósito central é sempre igual a 2 vezes o número de veículos, pois cada veículo entra e sai do depósito uma única vez. As equações (3.19) garantem que todos os demais vértices têm grau 2, pois devem ser visitados por um único veículo durante a entrega, que entra e sai do vértice uma única vez. As desigualdades (3.20) são as desigualdades de eliminação de subciclos. No caso do *VRP*, estas desigualdades também garantem que a solução respeite a capacidade de carga do veículo. Isto é obtido através de $v(S)$, que é um limitante inferior para o número de veículos necessários para atender os clientes do conjunto S. Detalhes destas desigualdades serão apresentados mais adiante. Finalmente as equações (3.21) e (3.22) são as restrições de integralidade e não negatividade das variáveis.

Existem muitos trabalhos sobre o *VRP* na literatura. Em [BMMN95], [BGAB83], [GA86] e [Chr85] são apresentadas excelentes revisões bibliográficas a respeito.

Os principais algoritmos exatos para o *VRP*, cujos trabalhos são citados em [LN87] e [Lap92], podem ser classificados em quatro grupos principais: os algoritmos de *Branch and Bound*, programação dinâmica, algoritmos de geração de colunas e, finalmente, algoritmos de *Branch and Cut*.

Em [LN86] é apresentado um algoritmo de *Branch and Bound*, em que se utiliza a relaxação das restrições de eliminação de subciclos (3.20) da formulação do *VRP*. Com isso, os subproblemas resultantes tornam-se problemas de *assignment*[1], de onde são obtidos os limitantes inferiores para o problema. Nesse caso, a eliminação dos subciclos fica a cargo das regras de *branching*.

[1] Problemas de *assignment* ou de alocação consistem em associar cada elemento de um conjunto A um elemento de outro conjunto B, tal que cada elemento de A seja associado a no máximo um elemento de B e o custo total da associação seja minimizado.

Como exemplo de um algoritmo baseado em programação dinâmica pode-se citar [EWGC71].

Os algoritmos de geração de colunas são muito utilizados para resolver formulações do *VRP* baseadas em particionamento de conjuntos. Em geral, nessas formulações, considera-se o conjunto de todas as rotas viáveis e obtém-se como resposta um conjunto de rotas que contenha cada um dos vértices da instância exatamente uma vez (exceto o próprio depósito). Para ilustrar o modelo em discussão, considere a formulação válida para o *VRP* dada a seguir.

$$min \qquad \sum_{j \in J} w_j^* x_j \tag{3.23}$$

$$sujeito\ a: \quad \sum_{i \in J} a_{ij} x_j = 1,\ (i \in V \setminus \{1\}), \tag{3.24}$$

$$x_j \in I\!B^{|J|}, \tag{3.25}$$

onde J é o conjunto de todas as rotas viáveis j, os coeficientes binários a_{ij} indicam se o vértice $i > 1$ aparece na rota viável j, w_j^* é o custo da rota ótima que passa pelos vértices de j e x_j é a variável binária que indica se a rota j é usada na solução ótima do problema.

Essa formulação apresenta vários problemas. Primeiro, o número de rotas viáveis é exponencial com relação ao número de vértices da instância, o que significa que o número de elementos em J e o número de variáveis x_j pode passar da casa dos milhões em problemas reais de roteamento. Segundo, o cálculo de w_j^* é complexo, pois cada rota j corresponde a um conjunto de vértices S_j satisfazendo $\sum_{i \in S_j} d_i \leq c$, onde d_i é a demanda do vértice i e c é a capacidade dos veículos. O valor de w_j^* é então obtido resolvendo-se um *TSP* em S_j.

Uma maneira de se contornar parcialmente os problemas mencionados é usar um algoritmo de geração de colunas. Bons resultados da aplicação desta técnica ao *VRP* são relatadas em [RZ68], [FR76], [Orl76], [DSD84], [AMS89], [DLS90] e [DDS91]. Esse método obtém melhores resultados em problemas mais restritos, onde o número de rotas viáveis é mais reduzido.

Finalmente, algoritmos de *Branch and Cut* podem ser utilizados também para resolver o *VRP*, separando desigualdades válidas, como (3.20) por exemplo.

Vários algoritmos exatos de programação linear inteira utilizam formulações baseadas em fluxo em redes. Essas formulações de fluxos podem ser divididas em dois tipos: formulações dois ou três índices.

Em ambas os tipos de formulação, as variáveis binárias são modeladas de forma a indicar a passagem de um veículo pela aresta (i, j). No caso das formulações de três índices, as variáveis binárias x_{ijk} indicam se a aresta (i, j) é percorrida pelo veículo k. No caso de dois índices, as variáveis binárias x_{ij} não indicam qual veículo percorre a aresta (i, j), indicando apenas que tal aresta é utilizada na solução do *VRP*. Exemplos de algoritmos baseados na formulação de três índices podem ser encontrados em [FJ78], [FJ81], [MT90] e [DDS91].

Claramente, as formulações de dois índices são mais compactas que as de três índices. Em [LND85] é apresentado um exemplo de algoritmo de *Branch and Cut* utilizando a formulação de dois índices. A formulação dada pelas equações (3.17) a (3.22), assim como a formulação do próprio *PE*, assunto do presente trabalho, também são exemplos de formulações de dois índices.

Desigualdades válidas para o *VRP*

Existem diversas desigualdades válidas para o *VRP* conhecidas. Várias delas são apresentadas e separadas num algoritmo de *Branch and Cut* em [Mar99].

As desigualdades de eliminação de subciclos, no *VRP* dadas por (3.20), além de eliminar subciclos, servem também para garantir que a capacidade de carga dos veículos seja respeitada.

Assim como no *TSP*, essas desigualdades podem ser representadas equivalentemente sob forma de corte, como na desigualdade (3.26) a seguir.

$$\sum_{i \in S, j \in V \setminus S} x_{ij} \geq 2v(S),\ \forall\ S \subset V\ com\ 2 \leq |S| \leq |V| - 2\} \tag{3.26}$$

Nesse caso, a equivalência entre as duas formas também é baseada nas restrições de grau dos vértices.

Nessas desigualdades, como foi dito anteriormente, o valor de $v(S)$ é um limitante inferior para o número de veículos necessários para suprir a demanda do conjunto de vértices em S. Note que, no caso do *TSP*, as desigualdades (3.14) e (3.16) são um caso particular de (3.20) e (3.26), respectivamente, onde $v(S) = 1$, pois no *TSP* a capacidade de carga do caixeiro é ilimitada, sendo necessário apenas um veículo para fazer a entrega.

No caso do *VRP*, como descrito em [Lap92], [LN86] e [LN87], e também para o *PE*, o valor de $v(S)$ pode ser obtido de diversas maneiras. O melhor limitante inferior para o número de veículos necessários para cobrir o conjunto S ($v(S)$) seria obtido resolvendo-se o problema do empacotamento ([GJ79]) para o conjunto S, com capacidade c. Entretanto, o problema do empacotamento é $\mathcal{NP}$-*difícil*, o que nos deixa com poucas chances de resolvê-lo eficientemente.

Outro valor para $v(S)$ é dado pela expressão:

$$v(S) = \left\lceil \frac{\sum_{i \in S} d_i}{c} \right\rceil, \tag{3.27}$$

onde d_i é a demanda do vértice i e c é a capacidade do veículo (entregador no caso do *PE*).

Embora este limitante seja mais fraco que o dado pela solução do problema do empacotamento, ele apresenta um bom compromisso entre qualidade e rapidez de cálculo.

O caso do *VRP* para múltiplos depósitos (*MDVRP*)

O *Problema do Roteamento de Veículos com Múltiplos Depósitos*, ou em inglês, *Multi-Depot Vehicle Routing Problem* (*MDVRP*), é uma generalização para o *VRP* em que os veículos encontram-se inicialmente num dado conjunto de depósitos. Da mesma forma que no *VRP*, os clientes e os depósitos são representados por um conjunto de vértices num grafo $G = (V, E)$ (direcionado ou não). Os veículos podem iniciar seus trajetos de qualquer um dos depóstitos. Segundo [BMMN95], existem duas variantes do problema: na primeira, cada veículo deve terminar seu trajeto no mesmo depósito de origem, sem passar por nenhum outro depósito. Esta versão é denominada *MDVRP fixo*. Na segunda variante, denominada *MDVRP não-fixo*, os veículos podem passar por quaisquer depósitos, durante seu trajeto, desde que no final retornem ao mesmo

depósito de origem. Esta última versão é a que mais se assemelha ao *PE*, pois permite que os depósitos sejam utilizados indistintamente pelos veículos (que no caso do *PE* é único). A única diferença entre estes problemas é que no *PE* existem restrições de capacidade nos estoques dos depósitos, fato que não ocorre no caso do *MDVRP não-fixo.*

Diferentemente do *VRP*, o *MDVRP* é um problema bem menos estudado, devido às dificuldades adicionais impostas pela utilização de mais de um depósito. Por essa razão, a maior parte dos estudos sobre este problema propõem soluções heurísticas para o problema, como é o caso de [BMMN95].

3.2.3 O problema de fluxo em redes com custos fixos (*FCN*)

No *FCN* são dados, uma rede com um conjunto V de vértices (facilidades) e um conjunto de arcos A. Um arco $e = (i, j)$ indo do vértice i para o vértice j representa a existência de uma rota direta entre as facilidades correspondentes aos vértices i e j. Associado a cada vértice i há uma demanda d_i. O vértice i é um cliente, um depósito, ou um ponto de passagem, se d_i é respectivamente, positivo, negativo ou zero. Assume-se que a demanda líquida da rede é zero, ou seja, $\sum_{i \in V} b_i = 0$. Cada arco (i, j) tem capacidade máxima de fluxo igual a u_{ij}, custo unitário de h_{ij} por unidade de fluxo e custo fixo c_{ij} caso exista um fluxo positivo no arco.

Uma formulação válida em *PLI* para o problema enunciado acima é dada a seguir.

$$
\begin{aligned}
\textit{min} \quad & \sum_{(i,j) \in A} (c_{ij} x_{ij} + h_{ij} y_{ij}) \\
\textit{sujeito a} \quad & x \in \mathbb{B}^{|A|},\ y \in \mathbb{R}^{|A|}, \\
& \textstyle\sum_{j \in V} y_{ji} - \sum_{j \in V} y_{ij} = b_i,\ \forall i \in V, \\
& y_{ij} - u_{ij} x_{ij} \leq 0,\ \forall (i,j) \in A.
\end{aligned}
\tag{3.28}
$$

O modelo do *FCN* é muito útil no projeto de problemas que envolvem fluxo de materiais em redes. No caso do presente trabalho, esse modelo surge na definição de desigualdades que modelam as restrições de capacidade do entregador e de capacidade dos depósitos. Além disso, um dos conjuntos de desigualdades válidas para o *FCN* (seção 3.4) é separado no algoritmo de *Branch and Cut* desenvolvido para o *PE.*

Várias desigualdades válidas fortes são conhecidas para esses problema. As mais conhecidas, e mais estudadas são as desigualdades de cobertura de fluxo (em inglês, *Flow Cover Inequalities* ou simplesmente *FCI*), que serão apresentadas na seção 3.4. Outras desigualdades conhecidas são as desigualdades de cobertura de fluxo em seqüências de vértices [RW87].

3.2.4 A formulação do *PE*

A seguir é apresentada uma formulação baseada em *Programação Linear Inteira Mista* para o *PE.*

Os dados de entrada do problema são:

- Grafo orientado $G = (V, A)$, onde os vértices de V estão particionados em dois conjuntos P e D. Os vértices do conjunto P correspondem a depósitos e os do conjunto D aos clientes. Assume-se que $|V| = n$ e $|A| = m$;
- Pesos inteiros w_{ij} para cada arco $a = (i, j) \in A$, representando a distância entre os nós $i, j \in V$.
- Demandas $d_i > 0$ dos clientes para cada vértice $i \in D$ e estoques $d_i < 0$ para os depósitos $i \in P$.
- Uma constante c, representando a capacidade dos arcos de G.

As variáveis do problema dividem-se em dois tipos:

- Variáveis de decisão $x_{ij} \in \mathbb{B}^{|A|}$ associadas a cada arco de G que indicam se o referido arco pertence ($x_{ij} = 1$) ou não ($x_{ij} = 0$) ao caminho C.
- Variáveis de fluxo $y_{ij} \in \mathbb{Z}$ associadas a cada arco de G que indicam a quantidade de fluxo (isto é, número de revistas) que passa pelo arco.

A formulação abaixo é válida para o *PE*:

$$min \quad \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} w_{ij} x_{ij} \tag{3.29}$$

$$sujeito\ a: \quad \sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = 1, \ \forall\, j \in D \tag{3.30}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} x_{ij} = 1, \ \forall\, i \in D \tag{3.31}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} x_{ij} = \sum_{\substack{k=1 \\ k \neq j}}^{n} x_{jk}, \ \forall\, j \in P \tag{3.32}$$

$$\sum_{i \in U} \sum_{\substack{j \in U \\ j \neq i}} x_{ij} \leq |U| - v(U), \ \forall\, U \subseteq D, \ 2 \leq |U| \leq |D| \tag{3.33}$$

$$\sum_{i \in U} \sum_{j \in V \setminus U} x_{ij} \geq v(U), \ \forall\, U \subset V, \ 2 \leq |U| \leq |V|/2, \ \emptyset \neq U \cap D \neq U \tag{3.34}$$

$$y_{ij} \leq c x_{ij}, \ \forall\, (i, j) \in A, \ \forall\, i \in P \tag{3.35}$$

$$y_{ij} \leq (c - d_i) x_{ij}, \ \forall\, (i, j) \in A, \ \forall\, i \in D \tag{3.36}$$

$$\sum_{\substack{i=1 \\ i \neq j}}^{n} y_{ij} = d_j + \sum_{\substack{k=1 \\ k \neq j}}^{n} y_{jk}, \ \forall\, j \in D \tag{3.37}$$

$$\sum_{\substack{k=1 \\ k \neq j}}^{n} y_{jk} \leq \sum_{\substack{i=1 \\ i \neq j}}^{n} y_{ij} - d_j, \ \forall\, j \in P \tag{3.38}$$

$$x_{ij} \in \{0, 1\}, \ \forall\, (i, j) \in A \tag{3.39}$$

$$0 \leq y_{ij}, \ \forall\, (i, j) \in A \tag{3.40}$$

A função objetivo (3.29) indica que se deseja minimizar o custo do ciclo encontrado, isto é, a soma dos pesos das arestas deste ciclo. As igualdades em (3.30) e (3.31) obrigam a existência

de apenas um arco entrando e de outro arco saindo do pontos de entrega, respectivamente. São equivalentes às desigualdades de grau do *TSP* e do *VRP*.

A igualdade em (3.32) garante que o número de arestas que entram é igual ao número de arestas que saem de cada depósito, possibilitando que o entregador passe quantas vezes forem necessárias por cada um, podendo inclusive descartar os depósitos desnecessários.

As desigualdades em (3.33) e (3.34) são as desigualdades de eliminação de subciclos do *TSP* e do *VRP*. Pode-se perceber que essas desigualdades aparecem com algumas modificações na formulação do *PE*, com relação às desigualdades originais (3.14) e (3.20). Uma discussão detalhada a respeito do uso dessas restrições e das adaptações feitas para o caso do *PE* é apresentada na seção 3.3.

No que diz respeito ao fluxo, tem-se as desigualdades em (3.35) e (3.36), que se referem à restrição de capacidade nos arcos de G que estão sendo usados ($x_{ij} = 1$). A equação (3.36) é válida para o *PE* pois dado que todo vértice $i \in D$ possui demanda positiva, se o fluxo y_{ij} saindo de i é tal que $y_{ij} > (c - d_i)$ significa que o fluxo y_{ki} entrando em i excede a capacidade de fluxo c, o que não é permitido. Assim, (3.36) leva a uma formulação mais justa.

As igualdades em (3.37) garantem que a solução respeita o princípio da conservação de fluxo em cada vértice do conjunto D (pontos de entrega). As desigualdades em (3.38) funcionam como restrições de capacidade para os depósitos. Estas impedem que sejam utilizadas mais revistas que as disponíveis nesses pontos, mas não obrigam a utilização de todo o estoque.

Por fim, as desigualdades em (3.39) garantem a integralidade e a não negatividade das variáveis x enquanto que as desigualdades em (3.40) garantem a não negatividade das variáveis y.

Como já foi dito anteriormente, a formulação acima combina restrições de três problemas combinatórios bastante conhecidos (*TSP*, *VRP* e *FCN*). O primeiro conjunto de restrições ((3.30) a (3.34)) garante que a solução ótima é constituída por um ciclo que passa uma única vez por todos os pontos de entrega e quantas vezes forem necessárias pelos depósitos mais apropriados. O outro conjunto de restrições ((3.35) a (3.38)) garante a viabilidade do fluxo na rota de entrega, garantindo que a capacidade de carga tanto do entregador quanto dos depósitos sejam respeitadas.

Como a formulação do *PE* inclui uma combinação das restrições de problemas conhecidos pode-se utilizar o fato de que desigualdades (π^1, π_0^1) e (π^2, π_0^2), válidas para os poliedros P_1 e P_2, respectivamente, são também válidas para $P \subseteq P_1 \cap P_2$. Portanto, se P é a envoltória convexa das soluções viáveis do *PE* e P_1 e P_2 denotam respectivamente os politopos do *VRP* e do *FCN*, então como $P \subseteq P_1 \cap P_2$, toda desigualdade válida para P_1 ou P_2 também é válida para P. Este fato é utilizado neste trabalho para reforçar a formulação do *PE*.

Nas seções seguintes (3.3 e 3.4) são discutidas duas classes de desigualdades válidas para o *PE* que são separadas no algoritmo de *Branch and Cut* apresentado na seção 3.5.

3.3 Desigualdades de eliminação de subciclos

As desigualdades de eliminação de subciclos, na forma original com que são encontradas no *TSP* e no *VRP* (restrições (3.14) e (3.20), respectivamente), não são válidas para o *PE*. Isso se deve ao fato das desigualdades que limitam o grau de entrada e de saída dos vértices do grafo terem sido modificadas para os depósitos (desigualdades (3.32)). Vale lembrar também que a equivalência tanto entre (3.14) e (3.16), quanto entre (3.20) e (3.26) existem apenas para os pontos de entrega, pela mesma razão.

Portanto, há a necessidade de se adaptar essas desigualdades ao caso particular do problema tratado no presente trabalho. Dessa forma, foram obtidas as três desigualdades válidas para o *PE*, baseadas nas desigualdades de eliminação de subciclos originais.

A primeira desigualdade válida para o *PE* é obtida a partir de (3.20). Essas desigualdades, mesmo com a modificação em (3.32) continuam sendo válidas para conjuntos de vértices que não contém depósitos. Dessa forma, é possível identificar as desigualdades dessa classe que estão violadas, apenas considerando os conjuntos de vértices que só contenham pontos de entrega.

A separação dessas desigualdades é feita usando o algoritmo heurístico descrito em [LND85].

O algoritmo usa como entrada o grafo construído da seguinte forma. Seja x^* a solução ótima da relaxação do *PE* numa dada iteração do algoritmo de *Branch and Cut*, e seja $G^* = (V, A^*)$ o grafo formado pelo conjunto V de vértices da instância (incluídos os pontos de entrega e depósitos) e um conjunto de arestas A^*, em que uma aresta $(i,j) \in A$, $i,j \in V$ se $x^*_{ij} > 0$. São atribuídos pesos $w_{ij} = x^*_{ij}$ a cada aresta $(i,j) \in A$. Em seguida, toma-se o grafo H^* induzido pelos vértices de V que são pontos de entrega. Em outras palavras, H^* é resultante da remoção dos vértices de V que são depósitos e os arcos neles incidentes.

Para cada uma das k componentes conexas S_k do grafo H^*, o algoritmo tenta obter uma desigualdade violada por um subconjunto dos vértices de S_k.

Define-se então o valor da violação da restrição (3.20) através da expressão

$$z(T) = |S_k \setminus T| - \lceil v(S_k \setminus T) \rceil - \sum_{i,j \in S_k \setminus T} x_{ij}, \tag{3.41}$$

onde $v(U) = \lceil \sum_{i \in U} d_i / c \rceil$.

A idéia do algoritmo de separação é, a cada iteração, adicionar vértices de S_k ao conjunto T (o que é equivalente a remover um vértice do conjunto $S_k \setminus T$), e calcular o valor da violação dado por (3.41). Uma vez detectado que $z(T) < 0$, é identificada uma desigualdade (3.20) violada sobre o conjunto de vértices em $S_k \setminus T$.

A descrição do algoritmo de separação é mostrado na figura 3.5.A linha 1 inicializa o conjunto T, que irá conter os vértices removidos do conjunto $S_k \setminus T$. A linha 2 representa a condição de parada do algoritmo quando não é mais possível encontrar desigualdades violadas com apenas um vértice em $S_k \setminus T$.

Na linha 3, é identificado um vértice $r \in S_k \setminus T$ tal que sua remoção deste conjunto implique na maior diminuição do valor do lado direito da expressão (3.41). Observando as parcelas de tal expressão pode-se avaliar a diferença entre $z(T)$ e $z(T \setminus \{r\})$. Na primeira parcela, ($|S_k \setminus T|$),

procedimento *Separa_Subciclos_VRP*()

1. $T \leftarrow \emptyset$;
2. **Se** $|T| = |S_k| - 1$ **então** pare a busca. Não é possível encontrar mais nenhuma desigualdade violada na componente conexa S_k.
3. Seja $r \in S_k \setminus T$ tal que $v(S_k \setminus T) = v(S_k \setminus (T \cup \{r\}))$ e $z(T \cup \{r\})$ é minimizado. Faça $T' \leftarrow T \cup \{r\}$.
4. **Se** $z(T') \geq z(T)$, então há poucas chances de que uma desigualdade violada seja encontrada. Pare a busca.
5. **Se** $z(T') < 0$, **então** adicione a restrição 3.20 à formulação corrente, com $S = S_k - T'$. Pare a busca.
6. **Se** $0 \leq z(T') < z(T)$, **então faça** $T \leftarrow T'$ e vá para o passo 2.

fim_procedimento

Figura 3.5: Algoritmo de separação de desigualdades de eliminação de subciclos para o *VRP*.

a remoção de qualquer elemento do conjunto $S_k \setminus T$ implica na diminuição de uma unidade em $z(T \setminus \{r\})$. A remoção de um vértice r tal que $v(S_k \setminus T) = v(S_k \setminus (T \cup \{r\}))$ mantém igual o valor segunda da parcela em ambas as expressões. Assim, os vértices candidatos a entrar no conjunto T devem satisfazer a expressão $d_r \leq frac(\sum_{i \in T} d_i/c) * c$, onde $frac(a)$ é representa a parte fracionária do valor real a. A última parcela da expressão corresponde à soma dos pesos das arestas incidentes ao vértice r que seriam removidas com o próprio vértice.

A busca pára na linha 5 porque as desigualdades que poderiam ser geradas a partir desse ponto provavelmente seriam dominadas pela desigualdade corrente. Na linha 6 o algoritmo fecha o laço principal e continua buscando novas desigualdades violadas.

Devido ao fato do algoritmo descrito anteriormente ser um algoritmo heurístico existe a possibilidade de que este possa não encontrar uma desigualdade de eliminação de subciclos violada pela solução da relaxação corrente do *Branch and Cut*, mesmo existindo uma.

Por essa razão, outro algoritmo heurístico é utilizado para encontrar desigualdades de eliminação de subciclos violadas. Nesse caso, as desigualdades separadas são da forma das desigualdades (3.14). Vale lembrar que, assim como as desigualdades do *VRP*, essas também são válidas apenas quando o conjunto de vértices contém apenas pontos de entrega.

Essas desigualdades correspondem ao segundo tipo de desigualdades de eliminação de subciclos separadas pelo algoritmo de *Branch and Cut* implementado no presente trabalho. A heurística de separação dessas desigualdades encontra-se descrita em [PG85].

As desigualdades representadas tanto por (3.33) quanto por (3.14) só são válidas quando o conjunto de vértices U é formado exclusivamente por pontos de entrega. No caso de conjuntos de vértices contendo depósitos, a desigualdade acima elimina subciclos que são permitidos pela formulação (aqueles subciclos formados pelas múltiplas passagens por depósitos).

Quando nenhuma desigualdade violada é encontrada pelos dois algoritmos anteriores, um terceiro tipo de desigualdades de eliminação de subciclos, representada por (3.34), é separada.

Essas desigualdades são válidas quando pelo menos um ponto de entrega encontra-se no conjunto U e outro no conjunto $V \setminus U$. Essas desigualdades impedem que sejam formados dois ciclos disjuntos, como solução para o PE, baseando-se no fato de que se existe um ponto de entrega em cada um dos conjuntos U e $V \setminus U$, deve existir pelo menos um arco que vai do primeiro conjunto para o segundo e vice-versa, independentemente do número de depósitos em cada conjunto.

Por outro lado, se um dos conjuntos U ou $V \setminus U$ não contém nenhum ponto de entrega, digamos U, pode existir uma solução viável para o PE que não utiliza os depósitos em U, e neste caso, não há arcos ligando esses dois conjuntos. Nesse caso, a desigualdade (3.34) deixa de ser válida.

A separação das desigualdades (3.34) pode ser feita utilizando-se qualquer algoritmo de cálculo de corte mínimo em redes. Dado o corte mínimo direcionado r de arestas do grafo G, particionando o conjunto de vértices V em dois conjuntos U e $U \setminus V$, se $\sum_{(i,j)\in r} x^*_{ij} < 1$, então a desigualdade (3.34) encontra-se violada pela solução corrente da relaxação.

No caso do presente trabalho, o algoritmo de separação dessas desigualdades utilizado foi o descrito em [NOI94] que tem complexidade $O(mn + n^2 log\ n)$, onde $n = |V|$ e $m = |A|$. No caso do PE, como é considerado o grafo completo, a complexidade do algoritmo é $O(n^3)$. Opcionalmente poderiam ter sido utilizados outros algoritmos, como por exemplo o algoritmo de cortes de capacidade mínima em grafos ([FF62] ou [PR90]) ou o algoritmo exato descrito em [PG85].

3.4 Desigualdades de *Flow Cover*

Outro conjunto de desigualdades válidas para o PE são as desigualdades de *Flow Cover* (desigualdades de cobertura de fluxo). Tais desigualdades são válidas para o poliedro genérico descrito por:

$$X = \{(x,y) \in I\!B^n \times I\!R^n_+ | \sum_{j\in N^+} y_j - \sum_{j\in N^-} y_j \leq d, y_j \leq m_j x_j, j \in N\} \tag{3.42}$$

O sistema (3.42) pode ser visto como uma rede de um único vértice com capacidades nas arestas (veja figura 3.6). O conjunto N contém os índices dos arcos que incidem no único vértice da rede. Os conjuntos N^+ e N^- são os índices dos arcos que entram e saem, respectivamente, do vértice, sendo que $N = N^+ \cup N^-$ e $n = |N|$. As variáveis y são fluxos nos arcos que obedecem à restrição de conservação de fluxo, com demanda externa d. Esses fluxos obedecem também às restrições de capacidade, onde m_j é a capacidade do arco j e x_j é a variável $0-1$ que indica se o arco j está sendo utilizado. Sem perda de generalidade, assume-se que os m_j estão ordenados em ordem decrescente, ou seja, $m_i \geq m_j$, $i \leq j$, e que o $m_1 = max\{m_j\}$, $\forall i \in N$.

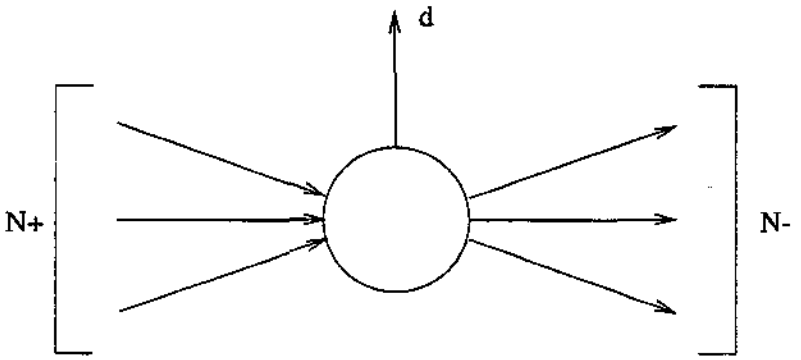


Figura 3.6: Modelo de fluxo num único nó

Várias famílias de desigualdades válidas para (3.42) são conhecidas. Em [NW88] são descritas duas dessas classes de desigualdades. Enquanto que em [GNS95] são apresentadas técnicas de *lifting* para essas desigualdades.

Em primeiro lugar, considerou-se o modelo de fluxo, representado na figura 3.6, apenas com arcos entrando no vértice, ou seja, $N^- = \emptyset$. Um conjunto $C^+ \subseteq N^+$ é chamado de *flow cover* (ou cobertura de fluxo), se $\sum_{j \in C^+} m_j > d$. A desigualdade

$$0 \leq d - \sum_{j \in C^+} y_j - \sum_{j \in C^{++}} (m_j - \lambda)(1 - x_j), \tag{3.43}$$

onde $\lambda = \sum_{j \in C^+} m_j - d$ e $C^{++} = \{j \in C^+ | m_j > \lambda\}$, é chamada **Flow Cover Inequality**, ou simplesmente *FCI* e é válida para (3.42) quando $N^- = \emptyset$.

Em seguida, o modelo de fluxo foi generalizado. O conjunto $C = C^+ \cup C^-$ é chamado de *flow cover* se $C^+ \subseteq N^+$, $C^- \subseteq N^-$ e $\sum_{j \in C^+} m_j - \sum_{j \in C^-} m_j = d + \lambda$ com $\lambda > 0$. A desigualdade

$$0 \leq d + \sum_{j \in C^-} m_j - \sum_{j \in C^+} y_j - \sum_{j \in C^{++}} (m_j - \lambda)(1 - x_j) + \sum_{j \in L^-} \lambda x_j + \sum_{j \in L^{--}} y_j, \tag{3.44}$$

onde $L^- = \{j \in N^- \backslash C^- | m_j > \lambda\}$ e $L^{--} = N^- \backslash (L^- \cup C^-)$, é chamada **Simple Generalized Flow Cover Inequality**, ou simplesmente *SGFCI* e é válida para (3.42). A desigualdade

$$\begin{aligned} 0 \leq\ & d + \sum_{j \in C^-} m_j - \sum_{j \in C^+ \cup L^+} y_j - \sum_{j \in C^{++}} (m_j - \lambda)(1 - x_j) \\ & - \sum_{j \in C^-} min\{\lambda, [m_j - (m_1 - \lambda)]^+\}(1 - x_j) + \sum_{j \in L^+} (max\{m_1, m_j\} - \lambda)x_j \\ & + \sum_{j \in L^-} max\{\lambda, m_j - (m_1 - \lambda)\}x_j + \sum_{j \in L^{--}} y_j, \end{aligned} \tag{3.45}$$

onde $L^+ \subseteq N^+ \setminus C^+$, $L^- \subseteq N^- \setminus C^-$, $L^{--} = N^- \setminus (C^- \cup L^-)$ e $[x]^+ = max\{x, 0\}$ é chamada **Extended Generalized Flow Cover Inequality**, ou *EGFCI* e também é válida para (3.42).

Observando a formulação do PE pode-se perceber que as desigualdades de fluxo ((3.37) e (3.38)), em conjunto com as restrições de capacidade nos arcos ((3.35) e (3.36)) formam um politopo bastante parecido com (3.42). Na verdade, é necessário fazer pequenas modificações nessas restrições para que seja possível provar que as desigualdades de *flow cover* são válidas para o PE.

Para se obter desigualdades da classe apresentada acima, válidas para o PE, é necessário reescrever as restrições (3.35), (3.36), (3.37) e (3.38) da formulação original, para obter restrições da forma do politopo (3.42). É necessário também determinar quais variáveis do problema compõem os conjuntos N^+ e N^-. As restrições resultantes são:

$$\sum_{\substack{k=1 \\ k\neq j}}^{n} y_{jk} - \sum_{\substack{i=1 \\ i\neq j}}^{n} y_{ij} \leq -d_j, \ \forall\, j \in P \tag{3.46}$$

com $N^+ = \{y_{jk} : k = 1, \ldots, n,\ k \neq j\}$ e $N^- = \{y_{ij} : i = 1, \ldots, n,\ i \neq j\}$, que é equivalente à equação (3.38);

$$\sum_{\substack{i=1 \\ i\neq j}}^{n} y_{ij} - \sum_{\substack{k=1 \\ k\neq j}}^{n} y_{jk} \leq d_j, \ \forall\, j \in D \tag{3.47}$$

com $N^+ = \{y_{ij} : i = 1, \ldots, n,\ i \neq j\}$ e $N^- = \{y_{jk} : k = 1, \ldots, n,\ k \neq j\}$.

$$\sum_{\substack{k=1 \\ k\neq j}}^{n} y_{jk} - \sum_{\substack{i=1 \\ i\neq j}}^{n} y_{ij} \leq -d_j, \ \forall\, j \in D \tag{3.48}$$

com $N^+ = \{y_{jk} : k = 1, \ldots, n,\ k \neq j\}$ e $N^- = \{y_{ij} : i = 1, \ldots, n,\ i \neq j\}$. As duas últimas desigualdades combinadas equivalem à equação (3.37).

As desigualdades (3.47) e (3.48) foram obtidas a partir da transformação da equação (3.37) como duas desigualdades.

Cada desigualdade da forma (3.47), (3.48) ou (3.46), combinadas com a desigualdade abaixo

$$y_{ij} \leq x_{ij}c, \ \forall\, (i,j) \in A, \ \forall\, i \in N, \tag{3.49}$$

define um novo politopo da forma (3.42). Considerando que para o caso particular tratado neste trabalho, a capacidade de todas as arestas $m_j = c$, $j \in N$, pode-se simplificar as desigualdades (3.44) e (3.45), obtendo-se respectivamente, as desigualdades

$$0 \leq d + c \cdot |C^-| - \sum_{j \in C^+} y_j - (c - \lambda) \cdot \sum_{j \in C^{++}} (1 - x_j) + \sum_{j \in L^-} \lambda x_j + \sum_{j \in L^{--}} y_j, \tag{3.50}$$

$$
\begin{aligned}
0 \leq\ & d + c \cdot |C^-| - \sum_{j \in C^+ \cup L^+} y_j - (c - \lambda) \cdot \sum_{j \in C^{++}} (1 - x_j) \\
& - \lambda \cdot \sum_{j \in C^-} (1 - x_j) + (c - \lambda) \cdot \sum_{j \in L^+} x_j \\
& + \sum_{j \in L^-} \lambda x_j + \sum_{j \in L^{--}} y_j,
\end{aligned} \tag{3.51}
$$

Pode-se então derivar *SGFCI's* e *EGFCI's* válidas para o *PE* da forma (3.50) e (3.51), respectivamente.

3.4.1 Lifting de *SGFCI*

Embora as *SGFCI*, derivadas para o *PE*, sejam válidas, pode-se gerar novas desigualdades mais fortes, através de uma operação de *lifting* em seus coeficientes.

A seguir, será descrita a técnica de *lifting* de *SGFCI* genéricas apresentada em [GNS95]. Em seguida serão feitas algumas simplificações sobre as desigualdades geradas a partir do *lifting* para aplicá-la ao caso do *PE*.

A desigualdade obtida pelo *lifting* de *SGFCI* é denominada *LSGFCI* (*Lifted Simple Generalized Flow Cover Inequality*) e é descrita como abaixo:

$$
\begin{aligned}
\sum_{j \in C^+} y_j + \sum_{j \in C^{++}} (m_j - \lambda)(1 - x_j) + \sum_{j \in N^+ \setminus C^+} \alpha_j y_j - \sum_{j \in N^+ \setminus C^+} \beta_j x_j \\
\leq d' - \sum_{j \in C^-} g(m_j)(1 - x_j) + \sum_{j \in L^-} \lambda x_j + \sum_{j \in L^{--}} y_j
\end{aligned} \tag{3.52}
$$

Os coeficientes α, β são definidos como:

$$
(\alpha_j, \beta_j) = \begin{cases} (0, 0) & M_i \leq m_j \leq M_{i+1} \\ (1, M_i - i\lambda) & M_i - \lambda \leq m_j \leq M_i \end{cases}
$$

e o coeficiente $g(z)$ é dado por:

$$
g(z) = \begin{cases}
i\lambda & M_i \leq z \leq M_{i+1} - \lambda,\ i = 0, \ldots, t-1 \\
z - M_i + i\lambda & M_i - \lambda \leq z \leq M_i,\ i = 1, \ldots, t-1 \\
z - M_i + i\lambda & M_i - \lambda \leq z \leq M_i - \lambda + m\ell + \rho_i,\ i = t, \ldots, r-1 \\
i\lambda & M_i - \lambda + m\ell + \rho_i \leq z \leq M_{i+1} - \lambda,\ i = t, \ldots, r-1 \\
z - M_r + r\lambda & M_r - \lambda \leq z \leq d''.
\end{cases}
$$

sendo que $d'' = d + \sum_{j \in C^-} m_j + \sum_{j \in N^- \setminus C^-} m_j$, $r = |C^{++} \cup L^-|$, $C^{++} \cup L^- = \{j_1, \ldots, j_r\}$ com $m_{j_i} \geq m_{j_{i+1}}$, $i = 1, \ldots, r-1$, $M_0 = 0$ e $M_i = \sum_{k=1}^{i} m_{j_k}$ para $i = 1, \ldots, r$. Além disso, $mp = min\{j \in C^{++} : m_j\}$, $m = \sum_{j \in C^+ \setminus C^{++}} m_j + \sum_{j \in L^{--}} m_j$, $m\ell = min(m, \lambda)$, t é o maior

índice em $C^{++} \cup L^-$ tal que $m_{j_t} = mp$ e finalmente, $\rho_i = max[0, m_{i+1} - (mp - \lambda) - m\ell]$ para $i = t, \ldots, r-1$.

No caso do *PE*, aplica-se novamente a simplificação $m_j = c$, $j \in N$ em (3.52), obtendo a seguinte desigualdade:

$$\sum_{j \in C^+} y_j + (c - \lambda) \cdot \sum_{j \in C^{++}} (1 - x_j) + \sum_{j \in N^+ \setminus C^+} \alpha_j y_j - \sum_{j \in N^+ \setminus C^+} \beta_j x_j \leq d' - g \cdot \sum_{j \in C^-} (1 - x_j) + \sum_{j \in L^-} \lambda x_j + \sum_{j \in L^{--}} y_j \tag{3.53}$$

onde os coeficientes α, β são definidos como

$$(\alpha_j, \beta_j) = \begin{cases} (0,0) & ic \leq c \leq (i+1)c \\ (1, i(c-\lambda)) & ic - \lambda \leq c \leq ic \end{cases}$$

e o coeficiente g é dado por

$$g = \begin{cases} i\lambda & ic \leq c \leq (i+1)c - \lambda,\ i = 0, \ldots, t-1 \\ c - ic + i\lambda & ic - \lambda \leq c \leq ic,\ i = 1, \ldots, t-1 \\ c - rc + r\lambda & rc - \lambda \leq c \leq d''. \end{cases}$$

tendo sido feitas as seguintes simplificações $d'' = d + c \cdot |N^-|$, $r = |C^{++} \cup L^-|$, $C^{++} \cup L^- = \{j_1, \ldots, j_r\}$ com $m_{j_i} = c$, $i = 1, \ldots, r$, $M_0 = 0$ e $M_i = ic$ para $i = 1, \ldots, r$, $mp = c$, $m = c|C^+ \setminus C^{++}| + c|L^{--}|$, $m\ell = min(m, \lambda)$, $t = r$ e finalmente, $\rho_i = \lambda - m\ell$ para $i = t, \ldots, r-1$.

Vale lembrar que a desigualdade (3.53) só é válida para o *PE* quando $C^{++} \neq \emptyset$, pois caso contrário, o valor de $mp = min_{j \in C^{++}} c$ não está definido.

3.4.2 Separação das Desigualdades de *Flow Cover*

Uma vez definidas as desigualdades de *Flow Cover* que serão utilizadas no algoritmo de *B&C*, é necessário desenvolver uma rotina de separação dessas desigualdades. A rotina de separação implementada no presente trabalho foi obtida em [NW88] e é descrita a seguir.

Uma *SGFCI* é definida quando se estabelecem os conjuntos C^+ e C^-. Em [NW88] é apresentada a seguinte desigualdade para o politopo (3.42),

$$\sum_{j \in C^+} \{y_j + (m_j - \lambda)(1 - x_j)\} \leq d + \sum_{j \in C^-} m_j + \sum_{j \in N^- \setminus C^-} \lambda x_j. \tag{3.54}$$

Esta desigualdade é obtida a partir de (3.44), fazendo as seguintes simplificações: $L^- = N^- \setminus C^-$, e conseqüentemente, $L^{--} = \emptyset$ e $C^{++} = C^+$. Estas modificações, tornam a desigualdade (3.54) mais fraca que a desigualdade original.

Conside que x^* e y^* constituem a solução fracionária da relaxação linear. Como $y_j \leq m_j x_j$, $\forall j \in N$, os valores de y_j^* podem ser substituídos pelos valores $m_j x_j^*$. Substituindo $d = \sum_{j \in C^+} m_j - \sum_{j \in C^+} m_j - \lambda$ em (3.54) pode-se calcular o limitante superior para a sua violação da desigualdade (3.54) através da expressão:

$$\lambda \left\{ - \sum_{i \in C^+} (1 - x_j^*) + \sum_{j \in C^-} x_j^* - \left(\sum_{j \in N^-} x_j^* - 1 \right) \right\}. \tag{3.55}$$

Para encontrar o valor máximo desse limitante superior, deve-se resolver o problema da mochila

$$\begin{aligned} \xi = max \quad & \left\{ \sum_{j \in N^+} (x_j^* - 1)\alpha_j + \sum_{j \in N^-} x_j^* \beta_j \right\} \\ & \sum_{j \in N^+} m_j \alpha_j - \sum_{j \in N^-} m_j \beta_j \geq d \\ & \alpha \in I\!B^{|N^+|}, \beta \in I\!B^{|N^-|}. \end{aligned} \tag{3.56}$$

Uma vez obtida a solução (α, β) do problema da mochila, são definidos quais os elementos de cada um dos conjuntos C^+, C^{++}, L^+, L^- e L^{--} da seguinte forma. O conjunto C^+ recebe os índices $i \in N$ tais que $\alpha_i = 1$, e C^-, os índices $i \in N$ tais que $\beta_i = 1$. O conjunto $C^+ +$ recebe os índices $j \in C^+$ tais que $m_j > \lambda$, $L^+ = N^+ \setminus C^+$, $L^- = N^- \setminus C^-$ e $L^{--} = \emptyset$.

Se alguma desigualdade (3.54) está violada, então o limitante superior para o valor da violação $\lambda[\xi - (\sum_{j \in N^-} x_j^* - 1)]$ deve ser positivo.

Note que mesmo que $\xi < \sum_{j \in N^-} x_j^* - 1$, a desigualdade (3.44) pode estar violada devido aos argumentos terem sido baseados em aproximações do valor da violação de (3.44).

Uma vez que o problema da mochila (3.57) dá apenas uma aproximação para o valor da violação de (3.44), não há necessidade de resolvê-lo à otimalidade. Além disso, uma vez que cada par C^+, C^- distinto resulta numa *SGFCI* diferente, é interessante obter vários pares C^+, C^-, diferentes, todos "quase-ótimos", para que se possa obter mais desigualdades separadas, aumentando as chances de se obter boas desigualdades violadas.

O algoritmo de separação de *SGFCI* e *EGFCI*, baseado na técnica de separação descrita acima, específico para o caso do *PE* é apresentado, na figura 3.7

Na primeira iteração do algoritmo, a mochila é resolvida utilizando-se o algoritmo guloso, descrito, por exemplo, em [NW88]. Nas iterações subseqüentes, são obtidas novas soluções através de perturbações na solução corrente. Estas perturbações consistem em adicionar ou remover um elemento da mochila corrente, sendo que as soluções repetidas da mochila são descartadas, utilizando-se para isso uma tabela de *hashing* para armazenar as soluções já obtidas.

procedimento *Separacao_Flow_Cover*()

1. $\mathcal{L} \leftarrow 0$
2. Resolva o problema da mochila (3.57), e, a partir da solução α_j e β_j, obtenha o par "quase-ótimo" C^+, C^-. O elemento $j \in C^+$ se $\alpha_j = 1$ e $j \in C^-$ se $\beta_j = 1$, para todo $j \in N$.
3. Crie a *SGFCI* (π, π_0) a partir do par C^+, C^-.
4. Aplique o *lifting* em (π, π_0) obtendo uma *LSGFCI* (π', π_0').
5. Se o *lifting* teve sucesso, significa que a desigualdade (π', π_0') é válida para o *PE*. Então faça $(a, a_0) \leftarrow (\pi', \pi_0')$. Senão, faça $(a, a_0) \leftarrow (\pi, \pi_0)$.
6. Verifique se a desigualdade (a, a_0) está violada pela solução x^t da relaxação do *PE*. Caso afirmativo adicione à lista $\mathcal{L}$ de desigualdades separadas. Caso contrário, descarte (a, a_0).
7. Crie a *EGFCI* (b, b_0) a partir do mesmo par C^+, C^-.
8. Verifique se a desigualdade (b, b_0) está violada pela solução x^t da relaxação do *PE*. Caso afirmativo adicione à lista $\mathcal{L}$ de desigualdades separadas. Caso contrário, descarte (b, b_0).
9. Se $|\mathcal{L}|$ é menor que o número máximo de desigualdades que podem ser separadas ou o número de iterações do algoritmo ainda não atingiu o máximo permitido, então perturbe a solução da mochila e repita o passo 2. Caso contrário, continue no passo 10.
10. Adicione as desigualdades contidas em $\mathcal{L}$ à formulação corrente do *Branch and Bound*.

fim_procedimento

Figura 3.7: Algoritmo de separação de *flow covers*.

Como pode-se observar no algoritmo de separação, as desigualdades *SGFCI* obtidas no procedimento são sempre submetidas ao procedimento de *lifting*, descrito na seção 3.4.1, na tentativa de se obter uma desigualdade que domine a primeira. Esse processo nem sempre é bem sucedido, devido às restrições quanto ao valor das funções que determinam os coeficientes do *lifting*.

No caso das *EGFCI*, ao invés de se estudar a violação da desigualdade pela solução corrente da relaxação, utiliza-se os mesmos conjuntos C^+ e C^- usados na construção da *SGFCI*.

3.5 Algoritmo de *Branch and Cut* para o *PE*

O algoritmo de *Branch and Cut* implementado para o *PE* segue o modelo genérico apresentado na seção 3.1.6, com algumas modificações no algoritmo de planos de corte.

As desigualdades separadas pelo algoritmo são aquelas descritas nas seções 3.3 e 3.4.

As desigualdades de eliminação de subciclos são separadas em todos os vértices do *Branch and Bound* pois têm algoritmos de separação eficientes (como será visto no capítulo 5, não tomam mais do que 5% do tempo de execução) e geram limitantes de boa qualidade. Além disso, não é possível gerar muitas desigualdades dessa classe a cada iteração, o que evita que o número de restrições da formulação corrente cresça muito.

Já a separação das desigualdades de *flow cover* é computacionalmente mais cara. Como visto na seção 3.4.2, é necessário resolver um problema da mochila para encontrar cada desigualdade violada. Além disso, o número de desigualdades que podem ser separadas é muito maior que o de desigualdades de eliminação de subciclos. Esse segundo fator implica em dois problemas. Primeiro que o número de problemas da mochila a serem resolvidos é proporcional ao número de desigualdades geradas. Mesmo sendo resolvido através do algoritmo guloso no caso do primeiro problema, o procedimento demanda um considerável esforço computacional (chegando em alguns casos a tomar mais de 35% do tempo total de execução).

Por essa razão foi criado um algoritmo de separação que funciona da seguinte maneira. Considera-se o nível i do vértice corrente do *Branch and Bound* e dois parâmetros ℓ e s. Só ocorre a separação de *flow covers* se $i \leq \ell$ ou $i \ mod \ s = 0$, onde $i \ mod \ s$ representa o resto da divisão inteira de i por s. Em outras palavras, separa-se *flow covers* se o nível i do vértice corrente da árvore de enumeração do *Branch and Bound* for inferior a ℓ, ou um múltiplo de s. Assim, através do parâmetro ℓ, os *flow covers* são separados nos primeiros vértices do *Branch and Bound*, onde a qualidade dos seus limitantes tende a ser melhor, economizando tempo de separação nos vértices mais profundos do *Branch and Bound*. O parâmetro s serve para permitir a separação de *flow covers* mesmo em níveis mais profundos, permitindo que bons *flow covers* que porventura tenham sido retirados da formulação em alguma iteração do algoritmo possam ser novamente separados.

A figura 3.8 a seguir mostra o algoritmo de planos de corte modificado para o caso do *PE*.

Detalhes da implementação do algoritmo de *Branch and Cut* para o *PE*, tais como a estratégia de *branching*, estratégia de armazenamento de desigualdades geradas e a determinação dos parâmetros do algoritmo serão apresentados no capítulo 5.

procedimento Planos_de_Corte_PE()

1. Resolva a relaxação da formulação do *PE* dada na seção 3.2.4 usando um algoritmo de programação linear, sendo x^t a solução ótima do *PL*, e z^t o seu valor ótimo. Se $x^t \in \mathbb{Z}_+^n$ e é valida para o *PE* então pare. O valor de z^t é o limitante inferior da sub-árvore de enumeração do *Branch-and-Bound*. Caso contrário continue.
2. Separe desigualdades de eliminação de subciclos da seção 3.3.
3. Se o nível i do vértice corrente da árvore de enumeração do *Branch and Bound* é tal que $i \leq \ell$ ou $i \ mod \ s = 0$, separe *SGFCI*, *LSGFCI* e *EGFCI*.
4. Se encontrar alguma desigualdade violada vá para 1, senão pare.

Figura 3.8: Algoritmo da fase de planos de corte do algoritmo de *Branch and Cut* do *PE*.

Capítulo 4

Sistema de Apoio À Decisão

Neste capítulo será apresentada a proposta de um Sistema Espacial de Apoio à Decisão para a Logística de Distribuição de Revistas a Assinantes. Tal sistema é baseado num *Sistema de Informação Geográfica* comercial onde são integrados os algoritmos combinatórios descritos no capítulo 2. Estes últimos desempenham papel fundamental no apoio ao processo decisório.

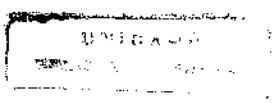
O restante deste capítulo está organizado da seguinte forma. Na seção 4.1, é feita uma introdução aos *Sistemas de Apoio à Decisão Espaciais*, definindo a terminologia e os principais conceitos sobre o tema utilizados no restante do capítulo. Esta seção é basicamente uma compilação das idéias apresentadas em [SMRY99]. Em seguida, na seção 4.2 é apresentada a proposta para o *Sistema Espacial de Apoio à Decisão para a Logística de Distribuição* propriamente dito. Na seção 4.3 é feita uma descrição completa do protótipo desenvolvido no presente trabalho, que consiste numa implementação parcial do sistema de apoio à decisão proposto na seção 4.2.

4.1 Introdução aos Sistemas de Apoio à Decisão

Um *Sistema de Apoio à Decisão* (*SAD*) é um sistema computacional projetado para aumentar a eficácia e a eficiência do processo decisório, tanto em termos operacionais quanto gerenciais nas empresas, provendo mecanismos para facilitar a interação do usuário com dados e modelos de análise.

Um Sistema Espacial de Apoio à Decisão (*SEAD*) é um *SAD* em que a dimensão espacial dos dados é fundamental para a análise das decisões. Em geral, as propriedades espaciais dos dados a serem analisados referem-se à sua localização na superfície da Terra. Estes são denominados *dados geo-referenciados*, e referem-se a dados sobre fenômenos geográficos associados à sua localização espacialmente referenciada à Terra.

Os *SEAD* baseiam-se fortemente em mapas, que formam a estrutura básica do apoio à decisão espacial. O processo de decisão espacial é feito através de dois estágios: (1) geração de soluções usando os mapas (usando o *SIG*) e (2) análise crítica das soluções geradas. Estes dois estágios podem ser iterativos, isto é, a análise crítica pode levar à geração de novas soluções, e assim por



diante, até que os tomadores de decisão estejam satisfeitos com as soluções obtidas [SMRY99].

Por essa razão, a principal ferramenta utilizada num *SEAD* é o *Sistema de Informação Geográfica*, ou *SIG*. Estes sistemas provêm mecanismos para armazenar, analisar, manipular e visualizar dados geo-referenciados. Os *SIG* são utilizados para auxíliar os tomadores de decisão na identificação de regiões que satisfazem um ou mais critérios, explorando as relações espaciais e temporais entre dados geo-referenciados, fornecendo dados para análises e simulações. Embora seja notória a importância de *SIG* no apoio à decisão espacial, a capacidade de tais sistemas para a tarefa é limitada.

Os *SIG* podem ser considerados como elementos de *software* que fornecem uma grande variedade de funções de análise sobre dados geo-referenciados, e oferecem ferramentas avançadas de visualização (cartográfica). Entretanto, estes não provêm meios de ajudar os usuários a selecionar as funções apropriadas para a análise e também não os guiam no processo de interpretação dos resultados.

Existem muitos esforços para obter ferramentas mais poderosas para o apoio à decisão espacial. Em linhas gerais, como ressaltado em [SMRY99], várias propostas de desenvolvimento de *SEAD* encontradas na literatura podem ser classificadas em três categorias, como se segue.

1. Estudos de caso;
2. Implementação de modelos de análise;
3. Projeto e arquitetura de sistemas.

Os estudos de caso (1) têm como objetivo mostrar como o uso de funções de *SIG* e visualização cartográfica melhoram o processo decisório em problemas específicos.

Os estudos de implementação de modelos (2), categoria em que se insere o presente trabalho, são voltados para a construção de ferramentas específicas que auxiliam os estudos envolvendo análises espaciais num *SIG*. Esta abordagem consiste na implementação de um conjunto de modelos num *SIG* para um dado domínio de aplicação. O *SIG* é responsável por fornecer os componentes de bancos de dados (espacial) e da interface com o usuário do *SEAD*, e o módulo adicionado provê o modelo de análise e simulação.

Exemplos de trabalhos nessa categoria são [DBB94] (localização de facilidades em ambiente urbano), [Nie96] (planejamento de transportes) e [VMP98] (estabelecimento de plantas industriais). Assim como o presente trabalho, esses estudos são também exemplos de trabalhos que integram *SIG* e *Pesquisa Operacional* no apoio à decisão.

A terceira e última categoria de trabalhos (3) trata do problema mais genérico de definir como devem ser construídos os sistemas de apoio à decisão. Alguns exemplos de estudos nessa categoria são citados em [SMRY99], trabalho que se enquadra neste último grupo.

4.2 Proposta para o Sistema Espacial de Apoio à Decisão

Nesta seção, propõe-se um *Sistema Espacial de Apoio à Decisão para a Logística de Distribuição* para o caso específico da distribuição de exemplares de revistas a assinantes. O objetivo deste sistema é auxiliar o especialista na logística, em nível operacional, a alocar os recursos disponíveis para a entrega de assinaturas de revistas da forma mais racional possível.

A proposta do sistema é especificada para que se possa estabelecer um contexto para a utilização dos algoritmos combinatórios apresentados nos capítulos anteriores no apoio à decisão na logística de distribuição. A especificação compreende apenas as tarefas que irão compor a logística de distribuição e como o sistema deverá dar suporte a cada tarefa.

4.2.1 Descrição das etapas da logística suportadas pelo *SEAD* proposto

Nesta seção será apresentado o *SEAD* para a logística de distribuição, que é descrito através de *workflows*. Nestes *workflows*, será utilizada a seguinte convenção: os retângulos representam tarefas a serem realizadas e as setas representam dependências entre tarefas. No canto superior esquerdo de cada tarefa é mostrado um número que identifica univocamente cada tarefa.

Num nível de abstração mais alto, pode-se dividir todo o processo de logística em duas etapas principais: planejamento e execução, conforme o *workflow* de nível 1, mostrado na figura 4.1.

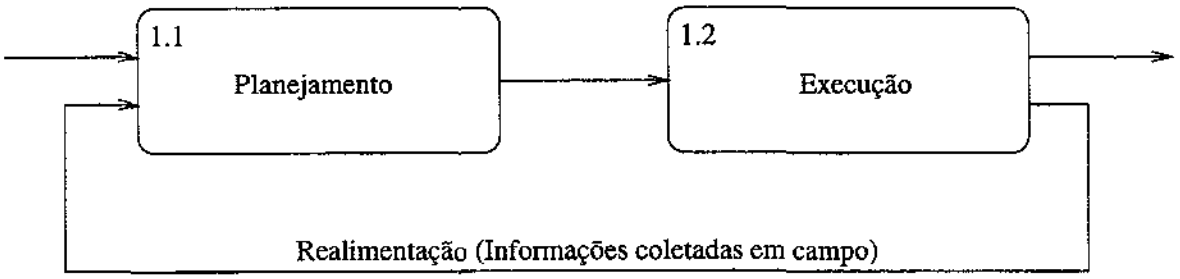


Figura 4.1: *Workflow* de nível 1 da logística de distribuição de assinaturas de revistas a assinantes

Durante a etapa de planejamento (tarefa 1.1 da figura 4.1), é estabelecida a maneira com que a distribuição dos exemplares será realizada, definindo-se, para cada zona de distribuição [1], os distritos que serão alocados a cada entregador e suas rotas de distribuição.

Na etapa de execução (tarefa 1.2 da figura 4.1), os resultados obtidos durante o planejamento são implementados em campo.

A figura 4.2 refina o processo de planejamento de distribuição (tarefa 1.1 mostrada na figura 4.1). O primeiro passo do planejamento é selecionar uma zona de distribuição dentre aquelas que constituem a região abrangida pela distribuição (tarefa 1.1.1). O texto logo abaixo dos retângulos das tarefas correspondem ao *status* da logística de distribuição no momento em que a tarefa está sendo executada. Estes valores servem para orientar o usuário ao longo do processo

[1] Relembrando a definição apresentada no capítulo 1, uma zona de distribuição é a unidade básica sobre a qual se executa o distritamento. Geralmente uma zona de distribuição corresponde à região pertencente a um bairro ou parte dele.

de logística, indicando o seu andamento. Mais detalhes sobre o uso destes valores são apresentados mais adiante na seção 4.2.2.

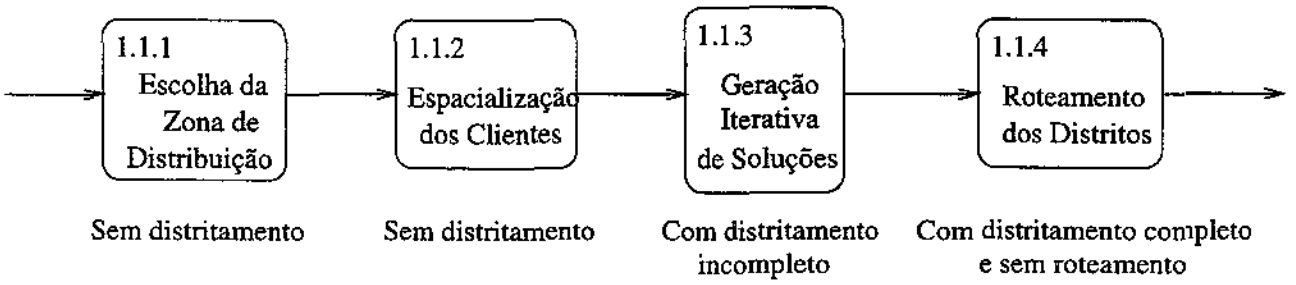


Figura 4.2: *Workflow* de nível 2 da fase de planejamento da logística de distribuição

Passa-se então à fase de espacialização dos clientes (tarefa 1.1.2) da zona de distribuição escolhida na tarefa anterior. A espacialização dos clientes consiste em determinar a localização de suas residências no mapa, através de seus endereços. Esta etapa é fundamental para a logística, pois a partir da espacialização é que serão calculados os tempos de entrega para cada trecho da zona de distribuição (dados necessários à resolução do *PD*) e também para a determinação das rotas dentro dos distritos (cada endereço espacializado no mapa corresponde a um ponto de entrega no *PE*).

A tarefa seguinte (1.1.3) consiste na geração de soluções para o problema do distritamento, na região considerada, seguida da análise crítica destas soluções. Nesta fase, são utilizados os modelos de análise implementados no *SIG*. Estes modelos têm como principal ferramenta é o algoritmo heurístico do *PD*, descrito na seção 2.2.

A tarefa 1.1.3 pode ser repetida diversas vezes. Esta iteração corresponde aos dois estágios do processo de decisão, conforme apresentado na seção 4.1: geração de mapas no *SIG*; análise crítica dos mapas gerados.

No contexto do projeto de sistema de apoio à decisão, a escolha de soluções onde se consideram múltiplos objetivos deve ficar a cargo do especialista. O sistema deve se encarregar apenas de fornecer um conjunto de soluções apropriado e também, ferramentas que facilitem a escolha do especialista.

Embora as funções objetivo i) minimização do número de distritos e ii) balanceamento de cargas de trabalho, consideradas no algoritmo apresentado sejam tratadas hierarquicamente, sendo a primeira prioritária sobre a segunda, nada impede que sejam feitas modificações para permitir a flexibilização da tomada de decisão.

Por exemplo, pode-se alterar o algoritmo de forma simples, para que este retorne ao usuário as soluções melhor balanceadas, com número de distritos entre p e $p+q$, onde p é o menor número de distritos encontrado na solução P, dentre todas as soluções geradas, e q é uma constante arbitrária qualquer. Nesse caso, o especialista poderia escolher uma solução P' como "ótima", com número de distritos $p' > p$, baseado em sua avaliação subjetiva de que o balanceamento de P' é significativamente melhor do que o de P.

Além disso, o sistema de apoio à decisão deve permitir que outras restrições não tratadas no modelo possam ser consideradas pelo especialista no momento da escolha da solução a ser implantada.

Considere o seguinte exemplo de um processo de decisão espacial, suportado pelo sistema proposto. Uma determinada empresa pode qual a melhor alocação de veículos para a distribuição de revistas considerando entre outros fatores, a disponibilidade de utilizar diferentes tipos de veículos, afetando cada um de maneira diferente o número de distritos obtidos para a mesma zona de distribuição. A entrega a pé, por exemplo, pode vir a gerar mais distritos na solução, pois a velocidade do entregador é menor (quando comparada com o uso de outros veículos) fazendo com que o entregador cubra uma área menor. Já a entrega de motocicleta geraria um número menor de distritos, tendo entretanto um custo superior ao da entrega a pé. Caberia ao usuário decidir qual das soluções é melhor, ou seja, decidir se o ganho no número de distritos obtido com a distribuição com motocicletas compensa o aumento de custo, com relação à entrega a pé.

O sistema também deve permitir que o usuário analise as soluções geradas, reavaliando os parâmetros utilizados, ajustando-os e gerando novas soluções. Adicionalmente, o usuário pode utilizar o sistema para gerar novas soluções, fornecendo como ponto de partida soluções disponíveis anteriormente.

A tarefa 1.1.3 é concluída quando o especialista considera satisfatória a solução encontrada, após o processo iterativo de decisão espacial. Nesse momento, a zona de distribuição passa a ter um conjunto de distritos válidos para a distribuição.

A tarefa seguinte, 1.1.4 consiste em realizar o roteamento para todos os distritos obtidos na tarefa anterior. No *workflow* este processo é simplificado e apresentado como uma única tarefa. Entretanto, este envolve um conjunto complexo de decisões. Este processo pode ser descrito em mais detalhes, como mostrado na figura 4.3.

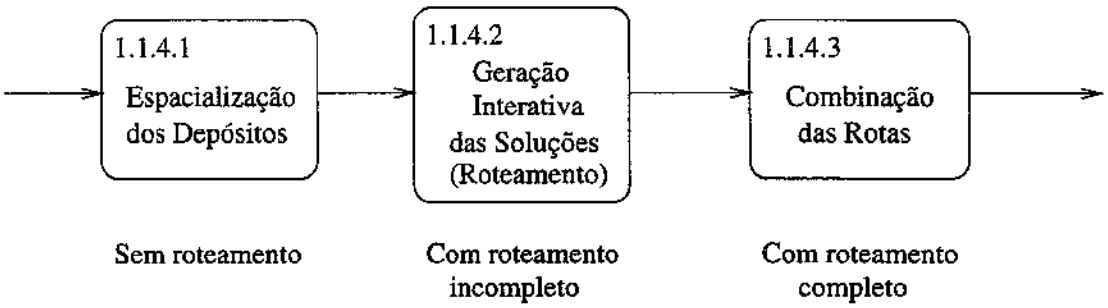


Figura 4.3: *Workflow* de nível 3 da fase de roteamento dos distritos

O primeiro passo na etapa de roteamento é a espacialização dos depósitos, correspondente à tarefa 1.1.4.1. Nesse processo, os depósitos que irão suprir o distrito em questão são dispostos sobre o mapa. Opcionalmente a espacialização dos depósitos pode ser realizada ao mesmo tempo em que se espacializa os clientes em toda região de distribuição, caso a localização dos depósitos seja conhecida *a priori*. Entretanto, em muitas empresas, o processo de contratação dos

depósitos é feita após a determinação de cada distrito. Nestes casos, só é possível espacializar os depósitos após completada a fase do distritamento. Relembrando a discussão sobre os depósitos no *PE*, apresentada no capítulo 1, estes são, em geral, pequenos espaços em portarias de edifícios residenciais ou em postos de gasolina. A sua contratação muitas vezes é feita através de acordos informais e sua localização é muito dependente da configuração do distrito onde se encontram.

Em seguida, na tarefa 1.1.4.2, são geradas rotas, utilizando o algoritmo heurístico do *PE* (descrito na seção 2.3), implementado como parte do modelo de análise no *SEAD*. Esse processo é seguido da análise crítica das soluções geradas. Esta tarefa corresponde ao processo iterativo de decisão espacial, semelhante à tarefa 1.1.3 na fase de distritamento.

Durante a análise crítica das soluções o usuário pode modificar alguns parâmetros de otimização, como por exemplo, impedir a utilização de certos trechos nas rotas, impedir a que a rota resultante utilize arestas ligando diretamente dois pontos de entrega predeterminados. A cada modificação, há a possibilidade de executar novamente a tarefa precedente, gerando assim várias soluções alternativas.

A repetição dessa tarefa é muito importante para o processo decisório, pois permite que o usuário ajuste o modelo, acrescentando restrições que não fazem parte do modelo originalmente implementado pelo algoritmo.

Note que, o *workflow* da figura 4.3, até a tarefa 1.1.4.2, é executado uma vez para cada um dos distritos da solução obtida ao final da análise crítica. Cada uma destas execuções leva à conclusão do roteamento de um dos diversos distritos obtidos na fase de distritamento.

Finalmente, na tarefa 1.1.4.3, os resultados de todos os roteamentos são combinados e organizados, para que possam permitir a execução da solução obtida na fase de planejamento.

Uma vez concluída a fase de planejamento, pode-se detalhar melhor a fase de execução (tarefa 1.2 no *workflow* mostrado na figura 4.1). A figura 4.4 mostra mais detalhes desta tarefa.

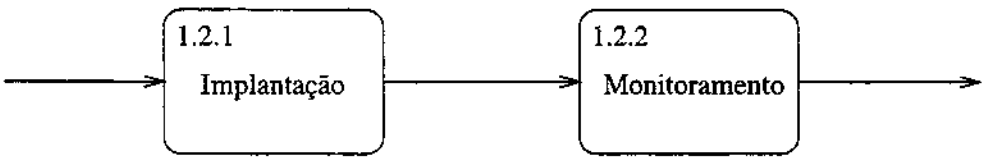


Figura 4.4: *Workflow* de nível 2 da fase de execução da logística de distribuição

No processo de implantação (tarefa 1.2.1, figura 4.4), os distritos são atribuídos aos entregadores disponíveis. Estes recebem mapas da região que irão atender com a rota de entrega (indicadas com setas de direção) e os pontos de entrega traçados, assim como a sua descrição textual com indicações tais como, onde iniciar a entrega, seguir entregando pela rua A, até o número x, entrando à esquerda na esquina com a rua B, entregando até o número y, até a esquina com a rua C, e assim por diante.

No processo de monitoramento (tarefa 1.2.2, figura 4.4), o resultado da implantação da logística é observado na prática: os entregadores são encarregados de reportar os problemas e dificuldades encontrados em campo. Estes problemas podem ter naturezas distintas e requerem tratamentos especiais em cada caso. Em linhas gerais, pode-se classificar estes problemas em

três categorias, com os respectivos impactos sobre a solução implantada:

- Erros na base de dados espacial. Pode haver erros na representação da rede de ruas do distrito, ou erros na espacialização dos clientes. O primeiro caso pode acarretar problemas mais graves, como por exemplo, a inviabilização do roteiro de entrega, devido a uma ligação inexistente entre ruas, ou a ausência no banco de dados de indicação de conversão proibida. Esses erros podem ainda inviabilizar um distritamento inteiro, caso a correção do erro de digitalização resulte em desconectar um dos distritos da solução. O segundo caso, a espacialização incorreta de um cliente pode inviabilizar a rota do distrito, obrigando a geração de uma nova rota. A correção do local onde reside o cliente pode também, com menor freqüência, causar uma modificação no balanceamento das cargas dos distritos, quando a nova localização estiver num distrito diferente do distrito onde o cliente encontrava-se originalmente. Essa modificação entretanto deve ser pouco relevante, visto que a contribuição de um único cliente no tempo de distribuição de um trecho é pouco significativa. Este tipo de erro começa a provocar maiores impactos quando o número de clientes espacializados em locais errados passa a ser grande, ou quando trata-se do endereço de um edifício com muitos assinantes.

- Imprevistos. Em geral, ocorrências não previstas de situações ligadas à geografia. Exemplos são: ruas interditadas devido a obras, modificação de mãos de direção e sinalização de trânsito em geral, entre outros. Estes devem receber tratamento semelhante ao do item anterior.

- Questões não tratadas pelo modelo decisório. Fatores não previstos pelo modelo de análise do sistema de apoio à decisão, que também não foram levados consideração pelo especialista na logística durante o planejamento, mas que influem significativamente na distribuição. Exemplos desta situação são os fatores sociais, intimamente ligados com o comportamento de cada entregador. Caso o sistema considere tais fatores, cria-se a possibilidade de se aproveitar a experiência do entregador para aumentar a qualidade das soluções implementadas.

Note que, para suportar as atualizações decorrentes das ocorrências detectadas durante a fase de monitoramento, é necessário prover o sistema de ferramentas de apoio à decisão para avaliação do impacto de alterações nos bancos de dados sobre as soluções correntemente implantadas.

Até aqui, foram especificados os requisitos do *SEAD* para que este dê apoio ao usuário na criação de soluções, em regiões em que não se dispõe de soluções pré-existentes (ou seja, soluções para áreas não atendidas). O sistema deve auxiliar o usuário a decidir se determinadas alterações no banco de dados tornam necessárias modificações nas soluções correntemente implantadas. Em caso afirmativo, estas ferramentas devem permitir ao especialista avaliar qual o impacto na solução corrente, se é possível fazer apenas modificações locais em alguns distritos para absorver as modificações ou se é necessário refazer todo o processo de logística.

4.2.2 Organização do trabalho do especialista durante a logística

Durante o processo de roteamento, o sistema deve oferecer um gerenciador de tarefas que permite ao especialista manter um registro do *status* de cada zona de distribuição, como uma lista de tarefas.

Esse controle de tarefas deve guiar o especialista desde o início do processo até a sua conclusão, e permitindo que possa verificar quais tarefas já foram concluídas e quais etapas ainda encontram-se em andamento ou devem ser iniciadas, dando uma noção de quanto esforço ainda deve ser empregado para a conclusão do trabalho. O mecanismo que guia o usuário desde o início do processo até a sua conclusão é descrito a seguir.

Uma vez iniciado o processo da logística de distribuição, o usuário tem acesso a uma lista com todas as zonas de distribuição que formam a região atendida pela empresa. Nesta lista, o usuário pode verificar o *status* corrente da logística, para cada zona de distribuição, de acordo com a tarefa em que a logística foi interrompida. Os valores válidos para o *status* são apresentados nos *workflows* mostrados anteriormente.

A indicação de *status* permite ao sistema guiar o trabalho do especialista, mostrando quais zonas de distribuição ainda não estão concluídas e nestas, quais distritos faltam ser roteados, utilizando os valores de *status* descritos acima.

Uma vez que todas as zonas de distribuição da região atendida pela empresa estejam com o *status* de *distritamento e roteamento completos*, todo o planejamento de distribuição da logística está concluído. O processo passa então para a fase de implantação e posterior monitoramento.

Além disso, devem ser providos recursos para a manipulação de metadados, visando documentar o trabalho dos especialistas, fundamentando as decisões tomadas, indicando possíveis pontos problemáticos e permitindo-se compartilhar a experiência adquirida por outros grupos de usuários.

4.2.3 Considerações finais sobre o sistema proposto

O sistema proposto é um sistema de apoio à decisão espacial, pois se enquadra na definição apresentada na seção 4.1.

Em primeiro lugar, o sistema provê ferramentas para facilitar a interação do usuário com os dados e com os modelos de análise. Com relação aos dados, o sistema fornece ao usuário meios de visualização de mapas utilizando o *SIG*. No que se refere aos modelos de análise, o sistema provê ferramentas automáticas para o distritamento e o roteamento, capazes de gerar várias soluções alternativas ao especialista, provendo também mecanismos para auxiliar a análise dos resultados.

São muitas as vantagens da utilização de um sistema como o proposto em empresas que trabalham com a distribuição de assinaturas de jornais e revistas. Em primeiro lugar, o esforço demandado atualmente força os especialistas a gerarem apenas uma única solução para o problema da logística de distribuição, por falta de ferramentas disponíveis. Devido ao custo extremamente elevado e do esforço humano envolvido na sua obtenção, tal solução é utilizada sem muitas modificações por vários anos seguidos até que a expansão do número de clientes das

áreas atendidas force a criação de uma nova solução. Em geral, isto só ocorre quando a solução correntemente implantada já apresenta sérias dificuldades na distribuição.

Como exemplo, no caso da *ECT*, o processo de distritamento é realizado em intervalos de cerca de 3 anos e quando é executado leva cerca de 6 meses para ser totalmente concluído. Esse tempo é mais que suficiente para que o crescimento populacional e a expansão dos grandes centros tornem inviáveis soluções que inicialmente satisfaziam todas as restrições dos modelos de análise.

Em muitos casos, para minimizar esse efeito, são feitas modificações *ad hoc*, nas soluções que não são refletidas nos documentos gerados durante o planejamento, causando um problema de consistência de dados.

4.3 Sistema Implementado

Foi desenvolvido um protótipo do *SEAD* apresentado na seção anterior. Visto que a implementação de um sistema com todas as funcionalidades descritas naquela seção demandariam muitos recursos (tempo, recursos humanos e financeiros), o protótipo implementado atende apenas a um subconjunto mínimo de funcionalidades do sistema proposto que permitem que se atinjam os propósitos do presente trabalho.

A implementação do protótipo visou em primeiro lugar: (1) testar a integração dos algoritmos implementados para o *PD* e *PE* com o *SIG*; (2) permitir a visualização das soluções encontradas tanto para o *PD* como para o *PE* sobre mapas.

O *workflow* mostrado na figura 4.5 caracteriza as funcionalidades do *SEAD* implementadas no protótipo simplificado.

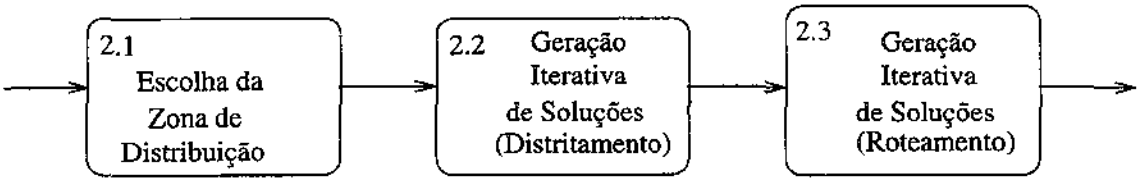


Figura 4.5: *Workflow* representando as fases da logística de distribuição suportadas pelo protótipo implementado

As simplificações mais importantes feitas sobre o *SEAD* proposto na seção anterior são:

- O sistema não permite que o usuário faça modificações sobre as soluções geradas tanto para o distritamento quanto para o roteamento. Por isso, caso o usuário não esteja satisfeito com as soluções geradas automaticamente pelo sistema (na etapa de análise de resultados), este é obrigado a retornar à etapa de geração de soluções e gerar novas alternativas;
- Não foram implementados os mecanismos que guiam o usuário durante o processo, através da lista de tarefas. No sistema implementado, este controle fica totalmente a cargo do usuário.

Dentre as funcionalidades oferecidas pelo sistema, pode-se citar:

- A escolha da zona de distribuição que será utilizada;
- Visualização dos resultados do distritamento de zonas de distribuição, tanto em formato de tabelas quanto a sua disposição espacial (com os distritos exibidos em cores diferentes);
- Visualização dos resultados do roteamento de distritos, também em formato de tabelas e sua disposição espacial (exibindo o ponto inicial e o ciclo percorrido pelo entregador, com setas indicando a direção).

Vale lembrar que há ainda problemas na implementação, sem entretanto comprometer os objetivos almejados. Dentre as principais deficiências, encontram-se aquelas relativas à visualização dos dados, como por exemplo, o uso de cores na exibição da solução do distritamento. Seria necessário utilizar um conjunto de cores contrastantes (facilmente diferenciadas pelo usuário), e algum algoritmo de coloração de grafos que evitasse que cores parecidas fossem utilizadas em distritos adjacentes.

Todas essas questões devem ser consideradas quando da implementação de um *SEAD* comercial.

4.3.1 Arquitetura do sistema implementado

O diagrama esquemático da arquitetura do sistema implementado é mostrado na figura 4.6.

No diagrama da figura 4.6, as setas demonstram a interação entre os módulos do sistema com o usuário e com os bancos de dados.

A *biblioteca de modelos* é o componente de *software* que contém a implementação dos algoritmos heurísticos para ambos os problemas tratados nessa dissertação (*PD* e *PE*). Numa implementação completa do *SEAD* proposto na seção 4.2, a biblioteca de modelos de análise conteria todas as ferramentas e mecanismos responsáveis por auxiliar o usuário na logística, que não fossem providos pelo *SIG*.

A interação com o usuário se dá através da interface do *SIG ArcView* com pequenas modificações para adicionar os botões e menus que dão acesso às bibliotecas de modelos de análise.

A interação entre o *SIG*, bancos de dados e as bibliotecas de modelos de análise se dá da seguinte forma. Quando o usuário aciona um botão para a realizar o distritamento sobre uma zona de distribuição, ou o roteamento sobre um distrito, o *SIG* obtém dos bancos de dados todas as informações necessárias para a criação das instâncias dos problemas, converte esses dados para o formato utilizado pelos modelos e faz a chamada aos modelos, passando como parâmetros os arquivos que contém os dados formatados.

Os modelos de análise, processam os dados fornecidos pelo *SIG*, geram as instâncias do *PD* ou *PE* correspondentes e aplicam os algoritmos heurísticos sobre as mesmas. As soluções encontradas são devolvidas ao *SIG*, em formato pré-estabelecido, que se encarrega de tratá-los e apresentá-los ao usuário.

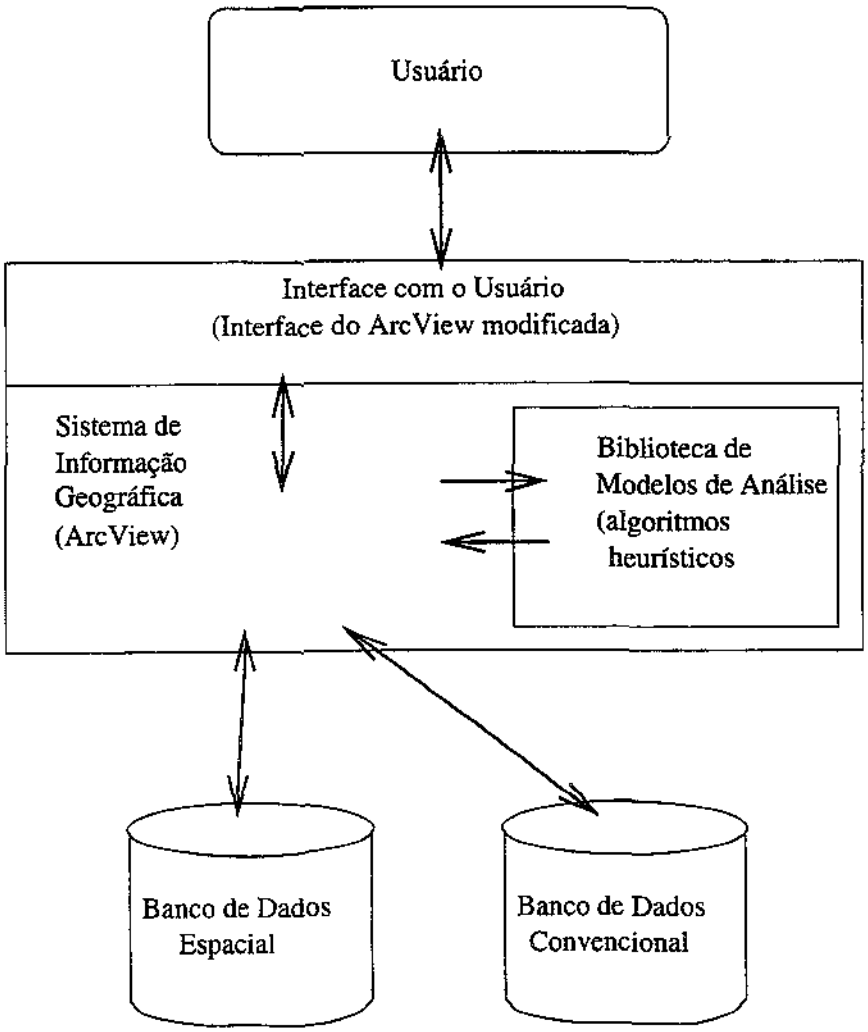


Figura 4.6: Diagrama esquemático da arquitetura do *SEAD* implementado.

A troca de dados entre *SIG* e os modelos de análise é feita através de arquivos texto, em ambos os sentidos, enquanto que a passagem de sinais de controle é feita através de chamadas de sistema.

Note que não há interação direta entre a biblioteca de modelos com os bancos de dados nem com o usuário. Toda a interação é mediada pelo *SIG*.

4.3.2 Cópias de tela do *SEAD* implementado

A seguir são apresentadas algumas cópias de tela do *SEAD* implementado.

A figura 4.7 apresenta uma zona de distribuição fictícia extraída do mapa da cidade de Goiânia, com os clientes já espacializados.

A figura 4.8 mostra um distritamento válido para a mesma zona de distribuição da figura anterior. Os distritos são apresentado em tons de cinza diferentes.

A figura 4.9 apresenta, em detalhe, um dos distritos apresentados na figura 4.8. Os círculos representam a localização dos pontos de entrega, enquanto que os quadrados denotam os depósitos. Nesta figura, pode-se observar uma rota de entrega gerada para o distrito.

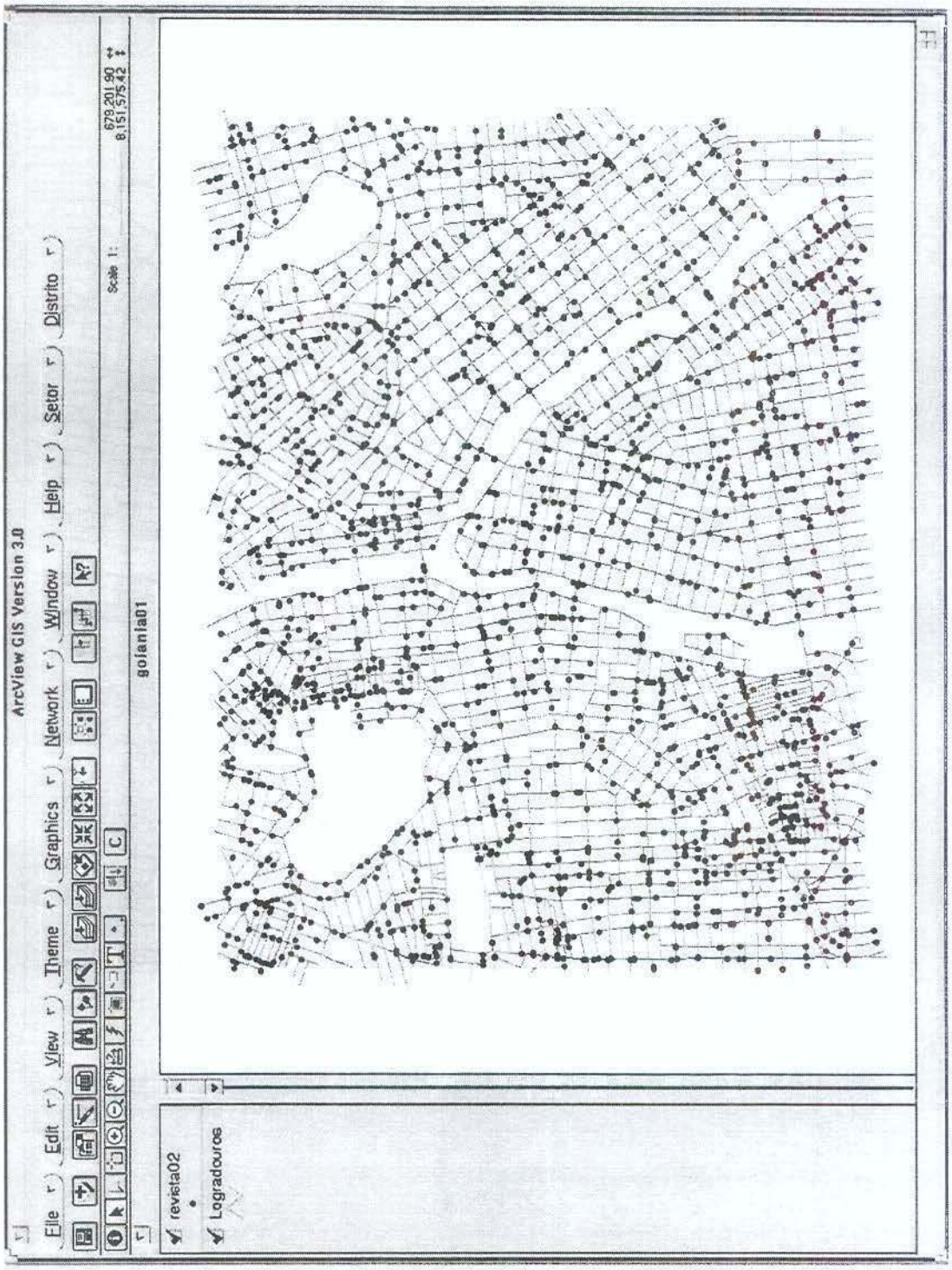


Figura 4.7: Cópia de tela do protótipo implementado, mostrando uma zona de distribuição fictícia na cidade de Goiânia, Goiás.

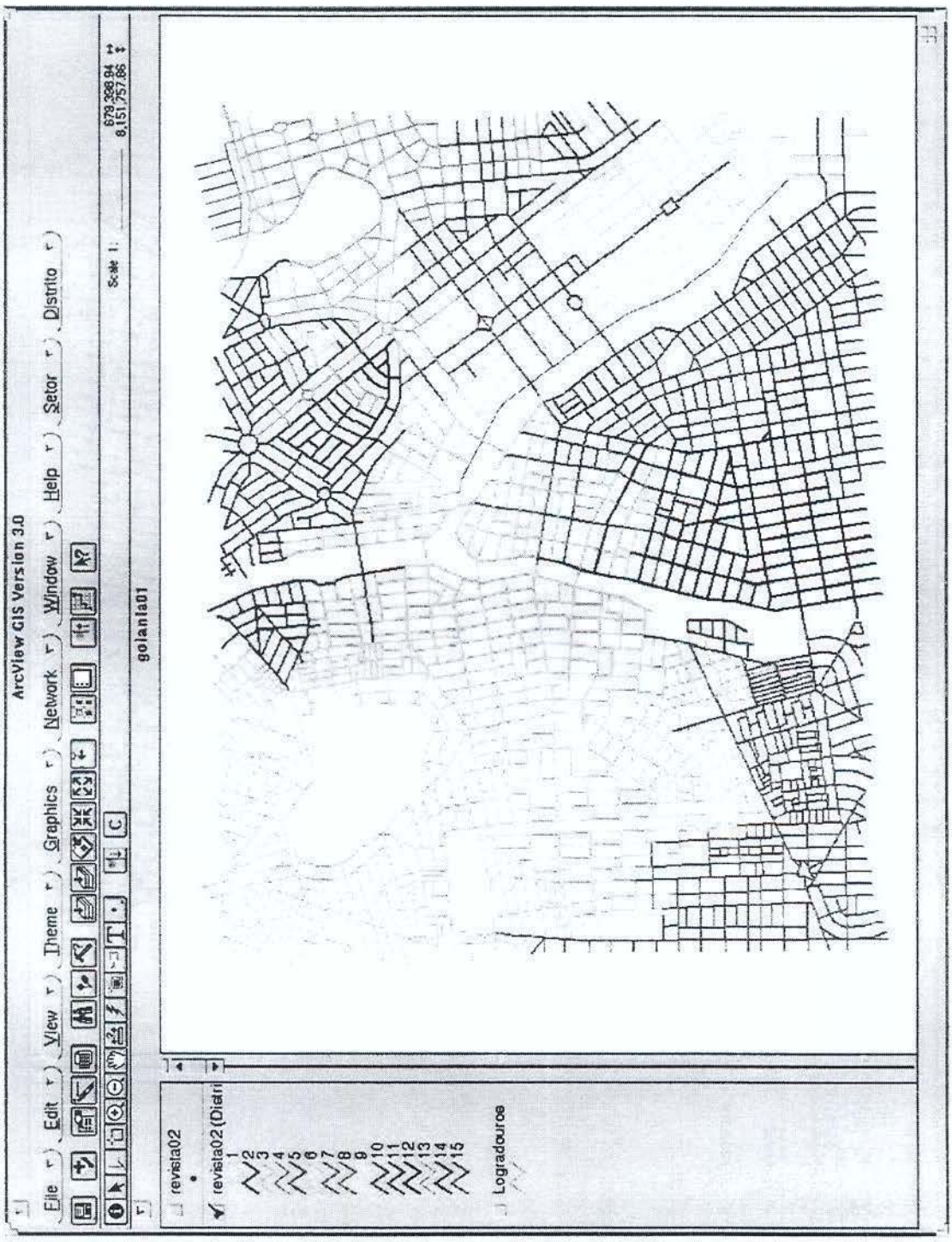


Figura 4.8: Cópia de tela do protótipo implementado, mostrando um distritamento gerado para uma zona de distribuição fictícia na cidade de Goiânia.

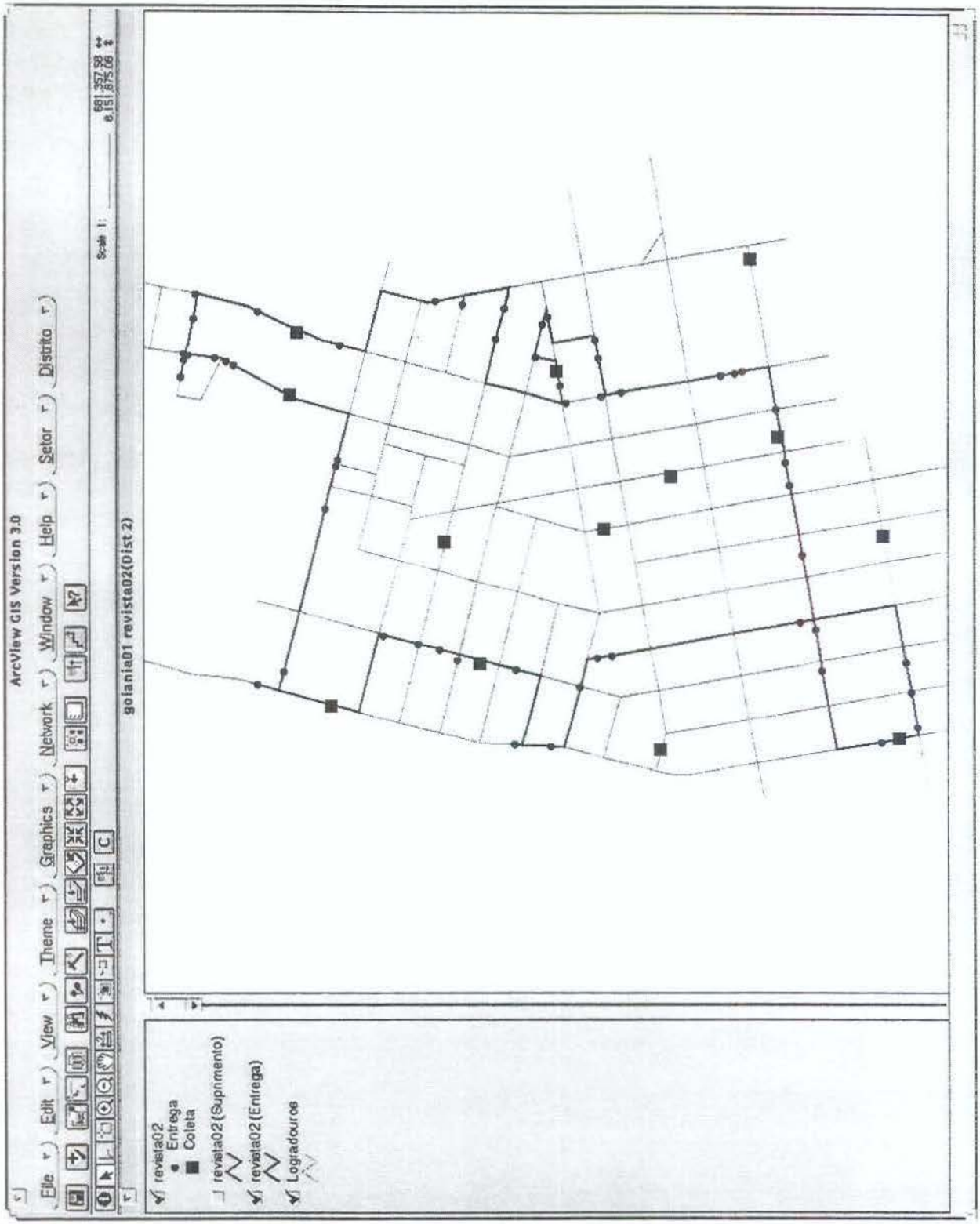


Figura 4.9: Cópia de tela do protótipo implementado, mostrando um distrito em detalhe e sua rota de distribuição, onde encontram-se os pontos de entrega e os depósitos.

Capítulo 5

Testes Computacionais

Neste capítulo serão apresentados os testes computacionais dos algoritmos desenvolvidos no presente trabalho. O seu conteúdo encontra-se dividido em duas partes: a primeira relacionada aos testes do *PD* (seção 5.1) e a segunda descrevendo os testes dos algoritmos do *PE* (seção 5.2). Na primeira parte do capítulo, relacionada aos testes do *PD*, serão apresentados: o procedimento de geração das instâncias do problema, o processo de ajuste dos parâmetros do *GRASP*, os testes computacionais envolvendo todas instâncias e a análise dos resultados obtidos. Na segunda parte do capítulo, serão descritos os procedimentos empregados na geração das instâncias, utilizadas nos testes do *PE*. Serão apresentados, em seguida, os testes computacionais de ambos os algoritmos (exato e heurístico) desenvolvidos para este problema, acompanhados de uma análise dos resultados.

5.1 Testes Computacionais para o *PD*

O algoritmo descrito na seção 2.2 para o *PD* foi implementado em linguagem *C++*, usando o compilador *gcc* versão *2.7.2* sob o sistema operacional *Unix Solaris 2.5*. Os testes desse algoritmo foram executados numa máquina *Sun SparcServer 1000* com 8 processadores com 40MHz de freqüência de *clock* compartilhando 308MB de memória *RAM*. Apesar de utilizar uma máquina multiprocessada, o algoritmo implementado não utilizou a arquitetura paralela, sendo executado seqüencialmente.

O principal objetivo dos testes computacionais do *PD* é verificar se o algoritmo heurístico desenvolvido obtém bons resultados, ou seja, se este obtém soluções de boa qualidade para o problema. Para isso foram criadas instâncias aleatórias para o problema cujas características mais se assemelham àquelas das instâncias reais encontradas nas empresas de distribuição de assinaturas de revistas.

Além disso, os testes computacionais do *PD* visaram verificar o comportamento do algoritmo quando aplicado a instâncias com características distintas com relação às instâncias típicas do problema. Com isso, pretende-se determinar sob quais circunstâncias o algoritmo continua a apresentar bons resultados.

Observando a definição do *PD*, apresentada no capítulo 1, uma instância para o problema é caracterizada pelos seguintes componentes: (1) grafo gerado a partir do mapa da região onde será feito o distritamento (caracterizado na seção 1.1); (2) estatísticas sobre o tempo de entrega (em minutos) gasto durante a distribuição de revistas em cada trecho de logradouro presente no mapa da região; (3) carga máxima de trabalho dos entregadores, em minutos.

Uma instância real típica do *PD*, conforme dados levantados em entrevistas com especialistas da *ECT - Empresa Brasileira de Correios e Telégrafos* e pela inspeção de instâncias reais do problema, apresenta as seguintes características: (1) número de trechos na região: entre 500 e 700; (2) número de segmentos de logradouro no mapa: desconhecido; (3) tempos médios de entrega por trechos: 6 ± 2 minutos, obedecendo a uma distribuição semelhante à distribuição normal; (4) carga máxima de trabalho: cerca de 400 minutos.

Baseados no caso típico descrito acima, foram construídos os mapas que serviram de base para as instâncias do *PD*. O procedimento adotado na criação desses mapas é descrito a seguir.

Em primeiro lugar, foram utilizados mapas digitais da cidade de *Goiânia*, no Estado de Goiás, como base de dados espaciais para as instâncias de testes. O mapa utilizado representa a rede de logradouros e segmentos de logradouros da cidade Goiânia e demais cidades vizinhas totalizando cerca de 30 mil segmentos de logradouros e aproximadamente 6 mil logradouros.

Esses mapas estão particionados em dois componentes de dados: dados espaciais e convencionais. Os dados espaciais representam os segmentos de logradouros que se encontram digitalizados no formato de *centerlines*, armazenadas em arquivos "SEQ" do sistema *MaxiCad*. O segundo compontente de dados é um banco de dados convencional no formato *dBase* ("DBF"), que armazena estatísticas como nome do logradouro, dados para o posicionamento do rótulo contendo o nome do logradouro, entre outros.

5.1.1 Ajuste dos parâmetros do *GRASP* para o PD

O primeiro teste do *PD* visou determinar qual conjunto de valores dos parâmetros do *GRASP* (número de iterações e o tamanho do conjunto reduzido de candidatos, ou *RCL*) implicaram nos melhores resultados.

As instâncias para esse teste foram criadas da seguinte forma. Primeiro foram tomados os 17 mapas extraídos do mapa completo de Goiânia. Para os trechos de cada um dos 17 mapas foram atribuídos tempos de distribuição de revistas aleatórios, segundo a distribuição normal, com 6 ± 2 minutos por trecho. Em cada instância foi adotada a carga de trabalho de 400 minutos.

A tabela 5.1 mostra o resumo das características das instâncias geradas.

Instância	Tempo total	Segmentos	Trechos	Grau máximo
pe_211.txt	2054.51	1606	350	39
pe_213.txt	1272.81	929	215	20
pe_221.txt	940.46	839	160	25
pe_222.txt	5080.27	3916	847	51
pe_223.txt	10754.93	8960	1796	65
pe_224.txt	5169.49	4000	860	34
pe_225.txt	2575.62	2221	434	36
pe_231.txt	1063.48	747	180	28
pe_232.txt	1339.23	1944	226	26
pe_233.txt	1874.41	3899	316	42
pe_234.txt	3096.81	3920	519	41
pe_300.txt	2298.79	1501	390	30
pe_302.txt	1030.33	672	173	30
pe_700.txt	728.73	795	123	25
pe_910.txt	736.52	480	124	57
pe_920.txt	751.30	466	126	29
pe_930.txt	684.01	518	115	22

Tabela 5.1: Resumo das características das instâncias de teste do *PD*.

A coluna **instância** contém os nomes das instâncias de teste geradas. A coluna **tempo total** mostra a soma dos tempos médios de entrega por trecho, expressa em minutos, para cada instância. As colunas **segmentos** e **trechos**, contêm o número de segmentos de logradouro e de trechos, respectivamente. Finalmente, a coluna **grau máximo** mostra o número máximo de trechos adjacentes dentre os trechos da instância, ou seja mostra o maior grau dentre os vértices do grafo correspondente).

O algoritmo do *PD* foi executado sobre 6 dessas instâncias várias vezes, sendo que a cada execução uma combinação diferente dos parâmetros do *GRASP* foi utilizada. Para o número de iterações foram utilizados os valores: 1000, 5000, 10000. Para o *RCL* foram utilizados os valores: 2, 3, 5, 10, 20, 30 e 50, resultando em 21 execuções para cada instância de teste, totalizando 126 testes. Os resultados obtidos são resumidos no gráfico da figura 5.1.

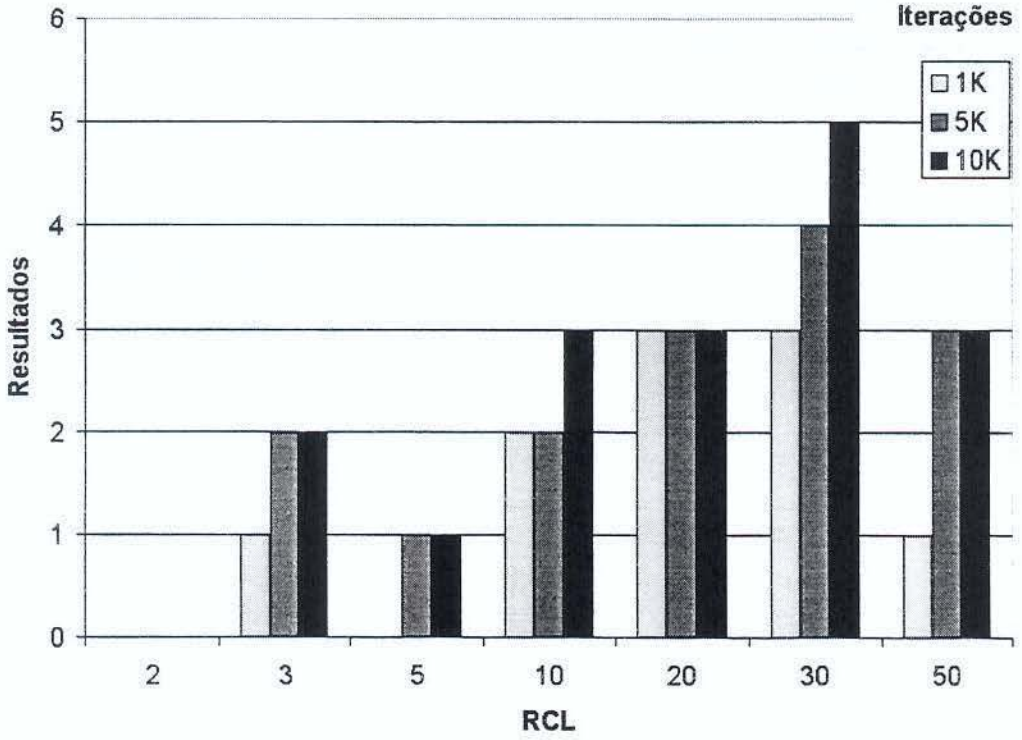


Figura 5.1: Resultados do teste de ajuste dos parâmetros *GRASP* do *PD*.

A figura 5.1 mostra, para cada combinação de parâmetros *GRASP*, o número de execuções do algoritmos em que se obteve o melhor resultado conhecido para aquela instância. O eixo das ordenadas expressa o número de vezes que a combinação de parâmetros gerou o melhor resultado encontrado para uma dada instância. Note que a soma dos valores dos resultados é superior a 6 pois há empates. O critério utilizado para a classificação dos resultados é apresentada a seguir.

Em primeiro lugar, são consideradas aquelas soluções cujo número de distritos obtido é mínimo, quando comparado com os outros resultados para a mesma instância. O desempate

entre duas soluções com o número mínimo de distritos é feito escolhendo-se aquela com menor desvio padrão da carga de trabalho dos distritos. Nesse caso, aceita-se uma tolerância de 25 % no valor do desvio padrão.

A partir da figura 5.1, pode-se chegar às seguintes conclusões: (1) Independentemente do número de iterações, os resultados favoráveis crescem com o aumento do valor do *RCL* até atingir o ápice com *RCL* igual a 30. A partir daí os resultados voltam a se deteriorar; (2) O melhor resultado foi obtido com a execução de 10000 iterações.

Provavelmente, o aumento do número de iteração para um valor acima de 10000 poderia trazer melhores resultados para *RCL* maior que 30. Entretanto, um aumento no número de iterações aumentaria muito o tempo computacional demandado em cada teste. Por essa razão, o número de iterações foi limitado a 10000. A figura 5.2 mostra as somas dos tempos de *CPU* de cada combinação utilizada.

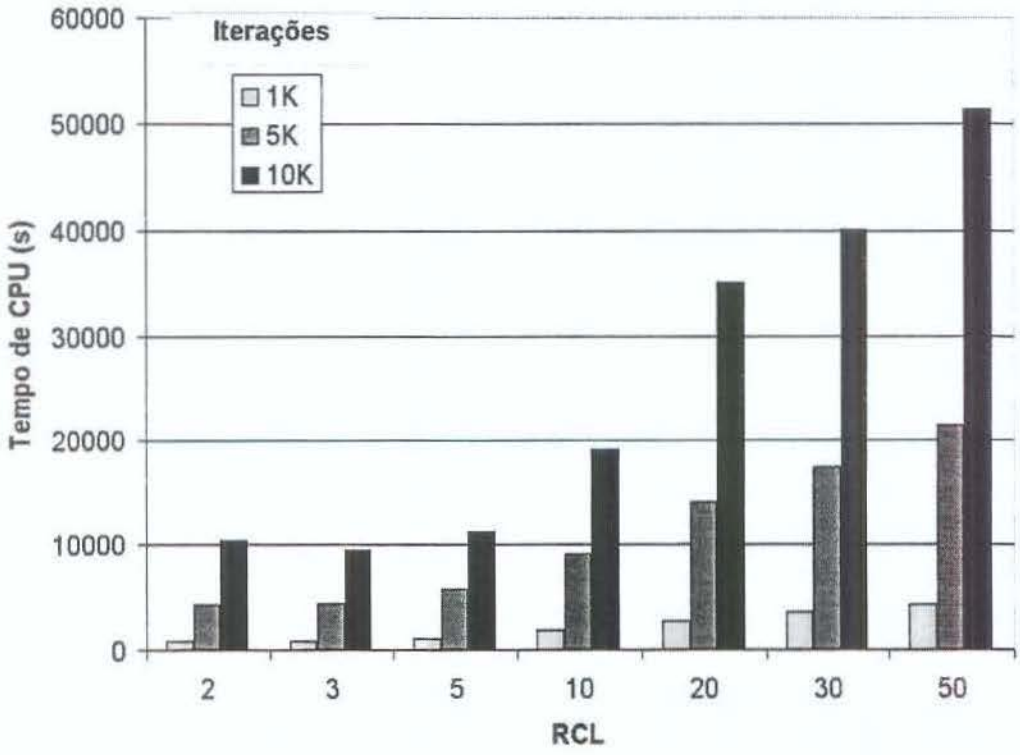


Figura 5.2: Tempos de execução dos testes de ajuste dos parâmetros *GRASP* do *PD*.

Dessa forma, os parâmetros escolhidos para os testes subseqüentes foram: i) número de iterações do *GRASP* igual a 10000; ii) tamanho do *RCL* igual a 30.

O segundo teste computacional projetado para o *PD* teve como objetivo verificar o comportamento do algoritmo em condições ligeiramente diferentes daquelas apresentadas pelas instâncias típicas do problema.

Foram fixados os parâmetros *GRASP* como apresentado anteriormente e foram utilizadas

diferentes combinações: i) da distribuição de tempos por trecho e ii) da carga de trabalho dos entregadores.

Nesse teste foram utilizados os mesmos 17 mapas utilizados no teste anterior, sendo que em cada um, foram utilizadas diferentes distribuições de tempos por trecho e diferentes cargas de trabalho. Os valores utilizados para a distribuição dos tempos por trecho foram: 3 ± 1 min, 6 ± 2 min e 10 ± 4 min. Com relação às cargas de trabalho, foram utilizados 3 valores distintos: 300 min, 400 min e 500 min, resultando num total de 153 instâncias distintas.

Pode-se avaliar a qualidade das soluções do *PD*, do ponto de vista da minimização de distritos, comparando-se os resultados obtidos nos testes com o limitante inferior para o número de distritos da respectiva instância. Tal limitante é dado pela seguinte expressão:

$$\underline{z} = \left\lceil \frac{\sum_{i \in V} t_i}{W} \right\rceil, \tag{5.1}$$

onde V é o conjunto de vértices do grafo G que representam os trechos da região considerada, t_i é o tempo de distribuição para o trecho representado pelo vértice $i \in V$ e W é a carga de trabalho máxima dos entregadores.

A figura 5.3 mostra os resultados do teste geral do algoritmo do *PD*, com relação ao número de distritos. No eixo das ordenadas do gráfico são mostrados o número de resultados em que foi obtida a solução ótima, ou seja, o número de distritos na solução obtida foi igual ao valor do limitante inferior. Dentre os resultados que não atingiram o ótimo, o pior resultado foi um número de distritos 3 unidades maior que o limite inferior, o que é bastante satisfatório, visto que a solução ótima pode ter valor superior ao dado pelo limitante inferior, dependendo da instância do problema. Além disso, no caso da *ECT - Empresa Brasileira de Correios e Telégrafos*, o valor das soluções obtidas, em geral, encontra-se 2 ou 3 unidades acima do limitante inferior.

Os resultados mostram que o algoritmo obtém boas soluções com relação ao número de distritos. No caso real típico, com carga máxima de 400 minutos e com distribuição de tempos de 6 ± 2 minutos, foram obtidas 16 soluções ótimas do total de 17 instâncias.

Com relação ao balanceamento de cargas de trabalho dos distritos, embora não se conheça um bom limitante inferior para o seu desvio padrão (além do limite inferior trivial igual a zero), é possível determinar a qualidade das soluções, comparando-se o desvio padrão das cargas de trabalho com valores de tolerância pré-estabelecidos. No presente trabalho, foram considerados resultados satisfatórios aqueles com desvio padrão abaixo de 1% da carga máxima de trabalho estabelecida.

A figura 5.4 mostra um gráfico com os resultados do teste geral com relação ao desvio padrão das cargas de trabalho dos distritos. O eixo das ordenadas deste gráfico representa o número de soluções em que o desvio padrão foi inferior à tolerância estabelecida de 1% da carga máxima de trabalho.

Percebe-se que no caso típico, em 16 das 17 instâncias testadas o valor do desvio padrão ficou abaixo da tolerância estabelecida, o que demonstra que o algoritmo tem bom desempenho, em termos de qualidade das soluções, também para a distribuição de tempos.

Considerando as outras combinações de cargas de trabalho e distribuição de tempos por

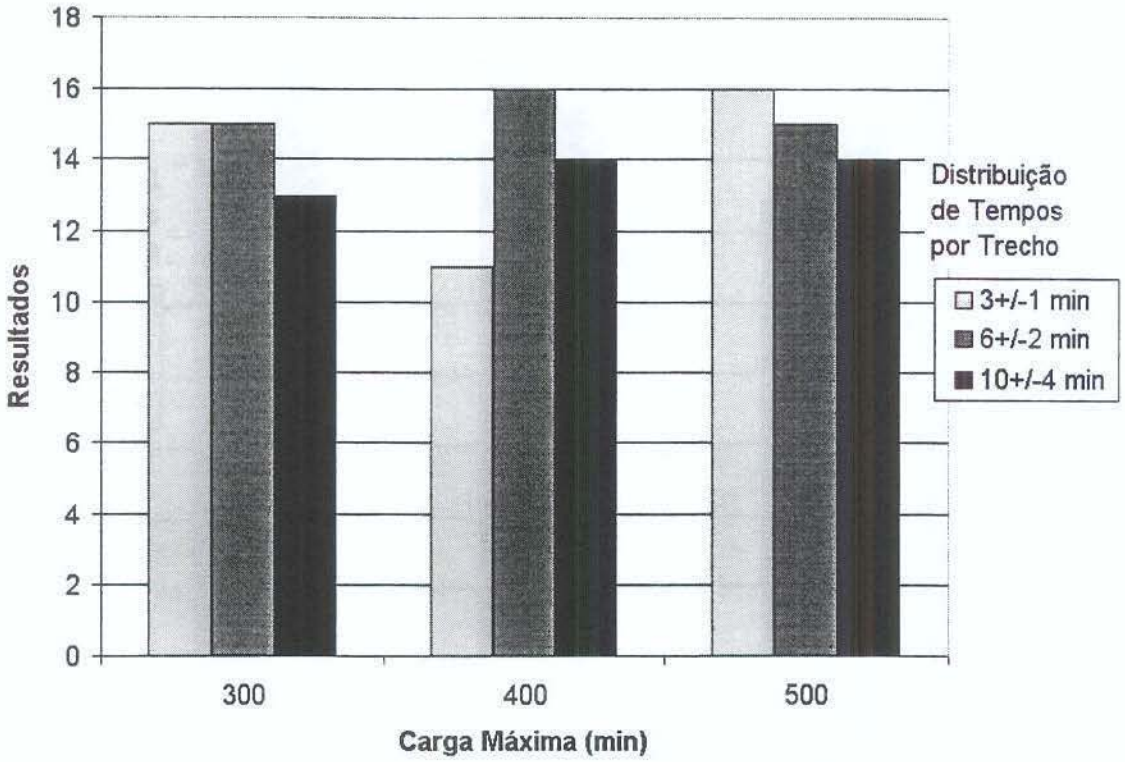


Figura 5.3: Resultados do teste geral do *PD* com relação ao número de distritos.

trecho, pode-se verificar que o algoritmo tem um desempenho um pouco superior, com relação à minimização do número de distritos, quando as cargas de trabalho são maiores (500 minutos) e o tempo médio de distribuição por trecho são menores ($3 \pm 1min$), o que já era esperado. Isto pode ser explicado pelo fato de que nessas condições, o número de trechos que cabem em cada distrito é maior, aumentando portanto o número de soluções viáveis do problema. Nesse caso, o algoritmo randomizado tem maiores chances de obter boas soluções. Quanto ao desvio padrão das cargas de trabalho, não se pode afirmar com certeza qual o comportamento do algoritmo com relação à modificação das características das instâncias. Mas, de um modo geral, o algoritmo não mostrou nenhuma deterioração sensível da qualidade para faixas razoáveis de tempo de distribuição e de carga de trabalho.

Testes de randomização

Foi realizado mais um teste com algoritmo do *PD*, para verificar se a randomização do algoritmo não interfere nas qualidade dos resultados, ou seja, pretende-se provar que o algoritmo é robusto quanto à randomização.

Foram escolhidas aleatoriamente três instâncias do *PD*, sobre as quais executou-se o algoritmo 10 vezes seguidas, cada qual utilizando uma semente do gerador pseudo-aleatório diferente.

Nas 10 repetições do algoritmo foram obtidos os mesmos números de distritos nas soluções

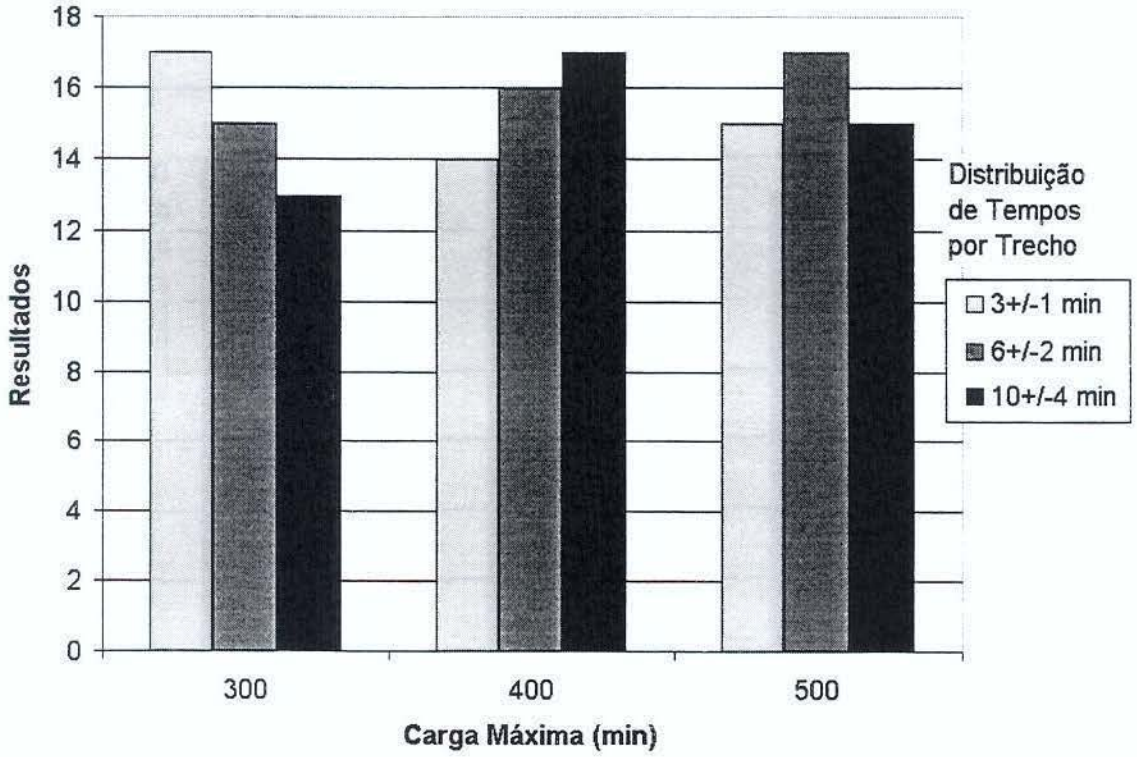


Figura 5.4: Resultados do teste geral do *PD* com relação ao desvio padrão da carga de trabalho dos distritos.

obtidas, para todas as 3 instâncias testadas. Ou seja, obteve-se desvio padrão nos resultados igual a zero. Quanto ao balanceamento das cargas de trabalho, não foram encontradas diferenças significativas (maiores que 0,5 minuto), entre os resultados dos testes, o que mostra que o algoritmo é muito robusto quanto à randomização *GRASP*.

5.2 Testes Computacionais para o *PE*

Como foi dito anteriormente, foram desenvolvidos dois algoritmos para resolver o *PE*, um algoritmo heurístico e outro exato.

O algoritmo heurístico para o *PE* foi desenvolvido e testado no mesmo ambiente usado nos testes do *PD*.

O algoritmo exato utilizou a mesma plataforma de desenvolvimento dos outros dois algoritmos. Entretanto, foi utilizado o *Framework*[1] *ABACUS - A Branch-And-CUt System* [Thi] para construir este algoritmo. O *ABACUS* é um framework orientado a objetos bastante flexível utilizado para o desenvolvimento de algoritmos de *Branch-and-Cut*. Ele permite desde a im-

[1]Um *Framework* é um ambiente genérico de desenvolvimento de programas, algoritmos e sistemas para um domínio de aplicação específico [GHJV94]

plementação algoritmos bastante simples, como algoritmos de *Branch-and-Bound* usando as estratégias padrão de branching e de busca, até a implementação personalizada de esquemas mais complexos envolvendo a separação de desigualdades válidas e estratégias de *branching* não convencionais.

A versão do *ABACUS* utilizada neste trabalho pode fazer chamadas a dois resolvedores de Programação Linear para computar as relaxações lineares, o *CPlex* e o *SoPlex*. No caso do algoritmo exato implementado neste trabalho, foi utilizado o *CPlex* versão 3.0.

Os testes do algoritmo exato foram realizados numa máquina *Sun Sparc Ultra-2* monoprocessada, com freqüência de *clock* de 200 MHz e 373MB de memória *RAM*. A execução do algoritmo exato foi limitada a 2 horas de tempo de *CPU* ou 4 horas de tempo total.

Os testes dos algoritmos do *PE* foram realizados em máquinas diferentes devido ao grande número de usuários e projetos alocados à *Sparc Ultra-2*. Como a comparação em termos de tempo não era tão importante, decidiu-se por utilizar esta máquina apenas para rodar os testes do algoritmo exato, que demanda maior esforço computacional.

Apenas a título de comparação entre os tempos de execução dos algoritmos desenvolvidos para o *PE*, pode-se considerar a razão entre a velocidade de processamento das máquinas igual à razão entre a freqüência de *clock* de seus processadores, ou seja, a máquina utilizada nos testes do algoritmo exato (*Sparc Ultra-2*) é cerca de cinco vezes mais rápida que a máquina utilizada para testar as heurísticas (*SparcServer 1000*).

Antes de apresentar os testes propriamente ditos, faz-se necessária uma explanação a respeito das instâncias do *PE* geradas.

5.2.1 Instâncias de teste para o PE

Uma instância genérica do *PE* é caracterizada pelos seguintes componentes.

- Grafo direcionado, completo e sem laços $G = (V, E)$. Os vértices $v \in V$ representam os pontos de coleta e os depósitos. O conjunto V é particionado em dois conjuntos: Um conjunto P representando os depósitos e outro conjunto D representando os pontos de entrega. Os arcos $a = (u, v) \in A$ representam passagem do entregador do vértice u ao v;
- Pesos w_a nos arcos $a = (u, v) \in A$ correspondentes ao custo que o entregador incorre ao visitar o vértice v, vindo do vértice u (denotado pela distância ou tempo gasto para ir do primeiro vértice ao segundo);
- Demandas $d_i > 0$, $i \in D$ dos pontos de entrega e estoques $d_i < 0$, $i \in P$ dos depósitos;
- Capacidade máxima de carga c do entregador.

No caso do *PE* também não foi possível utilizar instâncias reais para testes. Entretanto, foram geradas 4 tipos diferentes de instâncias, todas baseadas nas características observadas em instâncias reais típicas do problema, descritas a seguir.

Conforme dados das empresas de distribuição de revistas, em geral, a área coberta por um entregador tem entre 50 e 300 pontos de entregas, com 400 a 2000 exemplares a serem

distribuídos. Em geral, os entregadores utilizam cerca de 6 depósitos (nas regiões com maior número de assinantes) durante a distribuição (sem contar os depósitos não utilizados).

Embora a maior parte dessas empresas realize a distribuição com motocicletas, considerou-se no presente trabalho que a distribuição é feita a pé. Para tratar o caso motorizado, são necessárias várias informações adicionais a respeito das ruas da região de distribuição, como mãos de direção e tabelas de conversões proibidas, que não encontravam-se disponíveis. Mapas com tal nível de informação são ainda muito escassos e caros no Brasil.

Para a entrega a pé, estima-se que um entregador deva carregar consigo no máximo 8 kg de carga, para que não venha a ter problemas de saúde. Entretanto, é prática corrente nessas empresas, utilizarem cargas superiores, de 10 a 12 kg. Considerando que o peso médio de cada exemplar é de aproximadamente 200 g, calcula-se que a capacidade máxima do entregador seja de cerca de 50 exemplares.

Baseadas nessas informações, foram geradas 185 instâncias do problema, a partir de 4 fontes diferentes.

Devido a limitações, em termos de tempo de processamento, na resolução do *PE*, foram geradas instâncias de no mínimo 10 e no máximo 55 vértices (apesar das instâncias reais típicas apresentarem até 300 vértices). Embora o algoritmo heurístico possa tratar instâncias da ordem de grandeza das instâncias reais típicas, não foram feitos testes com instâncias desse tipo, pois não seria possível obter seus limitantes inferiores a partir do algoritmo exato. Portanto, não haveria nenhum parâmetro de comparação para os resultados da heurística.

O primeiro conjunto de instâncias foi obtido a partir de modificações em instâncias do *TSP* obtidas na *TSPLIB* [Rei]. Foram utilizadas 80 instâncias do tipo *EUC_2D*. Nessas instâncias, os vértices encontram-se numa superfície bidimensional, em que são fornecidas as coordenadas cartesianas de sua localização. Os custos dos arcos são obtidos através da distância euclidiana entre os vértices nos extremos dos arcos. Nessa caso, a matriz de distâncias é simétrica, ou seja, $w_{ij} = w_{ji}\ \forall (i,j) \in A$.

Para transformar uma instância do *TSP*, como a descrita acima, em uma instância do *PE*, foi aplicado o seguinte procedimento.

1. Determina-se o número de vértices que a instância do *PE* irá conter. Remove-se então o excedente de vértices da instância do *TSP* (aleatoriamente) até que o número de vértices desejado é atingido (isso porque as instâncias do *TSP* utilizadas têm de 50 a aproximadamente 18000 vértices, e as instâncias do *PE* têm número muito inferior de vértices);

2. Determina-se o número de vértices que serão transformados em pontos de entrega. Cada um recebe uma demanda d_i. A cerca de 35% dos pontos de entrega, atribui-se $d_i = 1$, para simular a existência de casas onde existe apenas um exemplar a ser entregue durante a distribuição. Nos outros pontos de entrega, $d_i = 10 \pm 5$ exemplares com distribuição uniforme, simulando edifícios em que mais de um exemplar é entregue;

3. Os vértices restantes são transformados em depósitos (foi utilizado cerca de 15% do total de vértices como depósitos). Atribui-se a cada depósito i, um estoque $d_i < 0$. A soma

dos estoques sempre supera a soma das demandas em 50%, ou seja, $|\sum_{i \in P} d_i| = 1,5 \times |\sum_{j \in D} d_j|$;

4. Associa-se então a cada arco $a = (u, v)$ o peso w_{uv} igual à distância euclidiana entre a localização dos vértices u e v;

5. Atribui-se a capacidade do entregador $c = 50$.

Utilizando-se o procedimento descrito acima, foram criadas 80 instâncias do *PE* a partir das instâncias do *TSP* da *TSPLIB*, sendo que as 8 primeiras instâncias têm 10 vértices cada, as 8 subseqüentes têm 15 vértices cada, e assim sucessivamente, até a criação de 8 instâncias com 55 vértices.

Um segundo conjunto de instâncias de teste para o *PE* foi criado a partir das instâncias de *Solomon* [Roc] para o *VRP* com um único depósito. Foi utilizado um processo, análogo ao descrito acima, para transformar essas instâncias em instâncias do *PE*. Da mesma forma que nas instâncias da *TSPLIB*, as instâncias de *Solomon* possuem um conjunto de pontos de entrega (clientes) cuja localização é expressa em termos de coordenadas cartesianas. Nesse caso, tanto as demandas dos pontos de entrega, quanto a capacidade do entregador são pré-estabelecidas. É necessário apenas identificar quais clientes serão transformados em depósitos e qual será o estoque alocado a cada um deles.

O procedimento de conversão de instâncias de *Solomon* para o *VRP* para instâncias do *PE* é descrito a seguir.

1. Determina-se o número de vértices da instância removendo aleatoriamente os excedentes, como no caso anterior;

2. São identificados os vértices que serão tranformados em depósitos e são alocados os estoques da mesma maneira que no procedimento de transformação das instâncias da *TSPLIB*. Note que nesse caso, as demandas dos pontos de entrega e a capacidade do entregador estão definidas *a priori*, e são mantidas como na instância original do *VRP*. Não existe portanto a simulação de casas e edifícios como no caso anterior.

3. Associa-se então a cada arco $a = (u, v)$ o peso w_{uv} igual à distância euclidiana entre a localização dos vértices u e v.

Usando o procedimento descrito acima, foram criadas 50 instâncias do *PE*, sendo 5 com 10 vértices, 5 com 15 vértices, e assim por diante, até as últimas 5 instâncias com 55 vértices. Assim como nas instâncias da *TSPLIB*, o número de vértices entre cada conjunto de instâncias salta de 5 em 5 unidades.

O terceiro conjunto de instâncias de testes foi obtido a partir de instâncias do *MDVRP* da *ORLIB* [Bea]. Estas instâncias são as que apresentam maiores semelhanças com instâncias do *PE*. Tais instâncias definem um conjunto de pontos de entrega com as respectivas demandas, um conjunto de depósitos e a capacidade de carga do entregador (veículo). Nesse caso, o processo de conversão de instâncias do *MDVRP* para o *PE* apenas remove o número excedente de vértices e

atribui estoques aos depósitos (pois no caso do *MDVRP* os depósitos tem capacidade ilimitada, ao contrário do *PE* que impõe restrições a essas capacidades).

O processo de conversão de instâncias do *MDVRP* para o *PE* é apresentado a seguir.

1. Determina-se o número de vértices que a instância deve ter, removendo-se os excedentes, como nos procedimentos anteriores.

2. São alocados os estoques aos depósitos seguindo a regra de que a soma dos estoques deve exceder a soma das demandas em 50%.

Seguindo o procedimento acima foram criadas 30 instâncias, sendo 3 instâncias com 10 vértices, 10 com 15 vértices, e assim sucessivamente, até as últimas 3 instâncias com 45 vértices cada.

O quarto e último conjunto de instâncias de testes foi gerado a partir dos mapas da cidade de Goiânia [dG96], mais especificamente, de distritos obtidos como solução do *PD*.

O procedimento de criação dessas instâncias é apresentado a seguir.

1. É escolhido o número de vértices da instância.

2. Os pontos de entrega (85% do total de vértices), são gerados sobre o mapa. Primeiro escolhe-se aleatoriamente um trecho do mapa. Em seguida, sorteia-se a posição desse trecho onde o ponto de entrega será posicionado. Da mesma forma que na geração de pontos de entrega nas instâncias da *TSPLIB*, 30% (do total de vértices) tem demanda unitária e o restante tem demanda $d_i = 10 \pm 5$ exemplares;

3. Os depósitos (restante dos vértices), são gerados da mesma forma que os pontos de entrega, e a estes são atribuídos estoques como nas instâncias da *TSPLIB*.

4. Associa-se então a cada arco $a = (u, v)$ o peso w_{uv} igual à distância do caminho mínimo sobre a rede de ruas entre os pontos no mapa representados pelos vértices u e v;

5. A capacidade de carga do entregador é dada por $c = 50$.

Seguindo o procedimento acima, foram criadas 25 instâncias, 4 têm 10 vértices cada, 2 têm 15 vértices, 2 têm 20 vértices, e assim por diante, até as últimas 2 instâncias com 55 vértices cada, totalizando 24 instâncias; Uma vez definidas as instâncias do problema, pode-se passar aos testes do algoritmo.

O formato dos nomes das instâncias, que será utilizado para identificá-las no decorrer do texto, é o seguinte: *pe000xx_yy.ext*, onde *xx* é o número de vértices da instância, *yy* é um número que distingüe as instâncias de mesmo tipo e número de vértices, *ext* é a extensão que identifica o tipo da instância, que pode assumir os seguintes valores: *tspm* para as instâncias do *TSP* modificadas, obtidas da *TSPLIB*; *solm* para as instâncias do *VRP* de *Solomon*; *mdvrpm* para as do *MDVRP*, obtidas da *ORLIB*; e *map* para as instâncias geradas a partir do mapa de Goiânia.

5.2.2 Testes do algoritmo heurístico do *PE*

Por se tratar de um problema original, não existem informações sobre instâncias de domínio público para o *PE*. Portanto, não é possível comparar os resultados do algoritmo heurístico, sem que se conheça as soluções ótimas ou limitantes inferiores para cada instância.

Por essa razão, serão apresentados nesta seção, apenas os testes de ajuste dos parâmetros *GRASP* e de randomização do algoritmo heurístico do *PE*. Os resultados gerais do algoritmo heurístico serão analisados junto com os resultados obtidos nos testes do algoritmo exato, apresentados na próxima seção (5.2.3).

Ajuste dos parâmetros *GRASP*

Assim como no caso do *PD*, inicialmente é necessário estabelecer os parâmetros *GRASP* para os quais o algoritmo do *PE* obtem melhores resultados.

Observando a descrição do algoritmo heurístico, apresentada na seção 2.3, existem 3 parâmetros a serem ajustados. São eles:

- Número de iterações do *GRASP*;
- Coeficiente α de ajuste da função exponencial que determina a posição de inserção de depósitos na fase construtiva do algoritmo (seção 2.3.2).
- Tamanho da lista restrita de candidatos (*RCL*) dentre os quais é feita a escolha de qual depósito será inserido na solução, também durante a fase construtiva do algoritmo (também na seção 2.3.2).

Foram realizados testes do algoritmo heurístico do *PE* sobre as 185 instâncias disponíveis, fixando-se os parâmetros *GRASP* em: 10000 iterações, $RCL = 5$ e $\alpha = 0.10$.

Para avaliar o comportamento do algoritmo heurístico com relação aos valores dos parâmetros *GRASP* foi realizado outro teste, descrito a seguir. Foi escolhido aleatoriamente um subconjunto de 10 instâncias dentre as instâncias da *ORLIB* e da *TSPLIB*. O algoritmo heurístico foi executado sobre cada uma das instâncias várias vezes, sendo que em cada execução foi utilizada uma combinação diferente dos três parâmetros *GRASP*. Para o número de iterações foram utilizados os valores: 1000, 5000 e 10000. O tamanho da *RCL* assumiu os valores 2, 5 e 10. Finalmente o coeficiente α recebeu os valores 0.1, 0.05 e 0.01, totalizando 27 combinações de parâmetros diferentes. O gráfico da figura 5.5 resume os resultados obtidos neste primeiro teste.

Na figura 5.5, as barras mostram o número de vezes em que a execução de cada combinação de parâmetros *GRASP* resultou na melhor solução conhecida para a instância correspontente. Observando a figura, percebe-se que os resultados são bastante homogêneos (ao contrário do teste equivalente do *PD* mostrado na figura 5.1). Todas as 27 combinações de parâmetros encontraram as melhores soluções para pelo menos 70 % das instâncias testadas.

Estes resultados mostram que a escolha dos parâmetros *GRASP* no primeiro teste foram adequadas, visto que a combinação de 10000 iterações, $RCL = 5$ e $\alpha = 0.10$ obteve sucesso nas 10 instâncias testadas. Entretanto, observando a figura 5.1, percebe-se que a execução com 5000

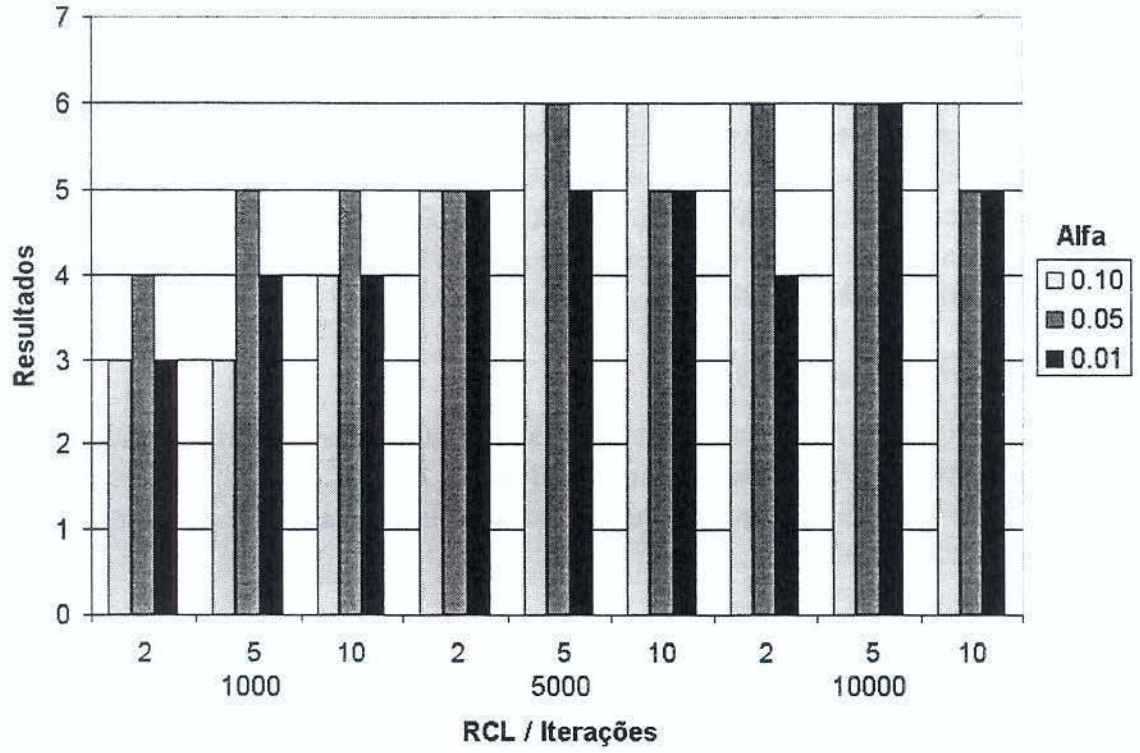


Figura 5.5: Resultados do teste de ajuste dos parâmetros *GRASP* do *PE*.

iterações teria sido mais apropriada, pois é equivalente à combinação escolhida em termos de qualidade das soluções, e tem tempo de execução cerca de 50 % inferior.

A escolha de $\alpha = 0.10$ foi adequada pois apresenta qualidade equivalente aos outros dois valores (0.05 e 0.01), mas tem tempo de execução cerca de 20 % inferior ao primeiro valor e de 50 % com relação ao segundo, conforme pode ser observado na figura 5.6.

Testes de randomização

Por também se tratar de um algoritmo randomizado, foi realizado mais um teste com algoritmo do *PE*, para verificar se o algoritmo é robusto quanto à randomização, da mesma maneira que foi feito com o *PD*.

Foram escolhidas aleatoriamente três instâncias do *PE*, daquelas criadas a partir das instâncias da *TSPLIB*, sobre as quais executou-se o algoritmo 10 vezes seguidas, cada qual utilizando uma semente do gerador pseudo-aleatório diferente.

Os resultados do teste de randomização para o *PE* também foram bastante satisfatórios, pois não foram encontradas diferenças maiores que $1,5$ % nos valores das soluções encontradas. Na maior parte dos resultados, as diferença ficou entre 0 e $0,5$ %. Portanto, pode-se concluir que o algoritmo heurístico implementado para o *PE* é robusto com relação à randomização.

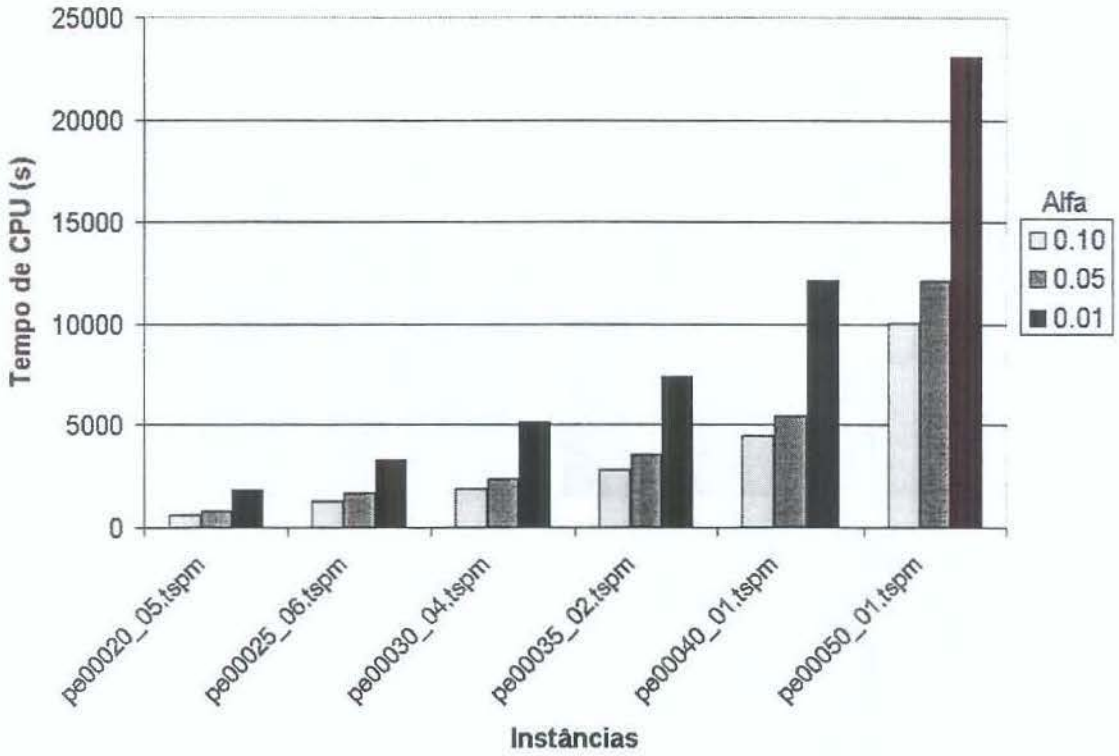


Figura 5.6: Tempos de *CPU* dos testes de ajuste dos parâmetros *GRASP* do *PE*.

5.2.3 Testes do algoritmo exato do *PE*

Assim como os dois algoritmos heurísticos, o algoritmo exato para o *PE* também tem parâmetros a serem definidos. Como apresentado na seção 3.5, o algoritmo exato do *PE* utiliza dois parâmetros que controlam a separação de desigualdades de *flow cover* ((3.53) e (3.51)). Os dois parâmetros são denotados por ℓ e s (inteiros e positivos) que têm as seguintes interpretações. As desigualdades de *flow cover* (3.53) e (3.51) são separadas nos ℓ primeiros níveis da árvore de enumeração e, a partir daí, apenas a cada s níveis da árvore.

Foram escolhidas 6 instâncias aleatórias, com 25 a 55 vértices, dentre as instâncias da *TSPLIB* modificadas, para o ajuste dos parâmetros. Para cada instância, foi executado o algoritmo exato com a seguinte combinação de parâmetros: ($\ell = \infty$, $s = 0$), ou seja, com separação em todos os vértices da árvore de *B&B*, ($\ell = -1$, $s = \infty$), ou seja, sem separação de *flow covers*, ($\ell = 4$, $s = 2$), ($\ell = 4$, $s = 4$), ($\ell = 4$, $s = 8$) e ($\ell = 8$, $s = 4$). Os resultados obtidos nesse teste encontram-se resumidos no gráfico da figura 5.7.

Observando os resultados na figura 5.7, percebe-se que a separação de *flow covers* melhora a qualidade das soluções, principalmente para as instâncias com 25 vértices ou mais. Quando separadas em todos os níveis, os resultados têm uma perda na qualidade, para as instâncias menores. Todas as outras combinações de valores têm resultados bastante equilibrados. Baseando-se nestes resultados, escolheu-se a combinação de parâmetro $\ell = 4$ e $s = 4$.

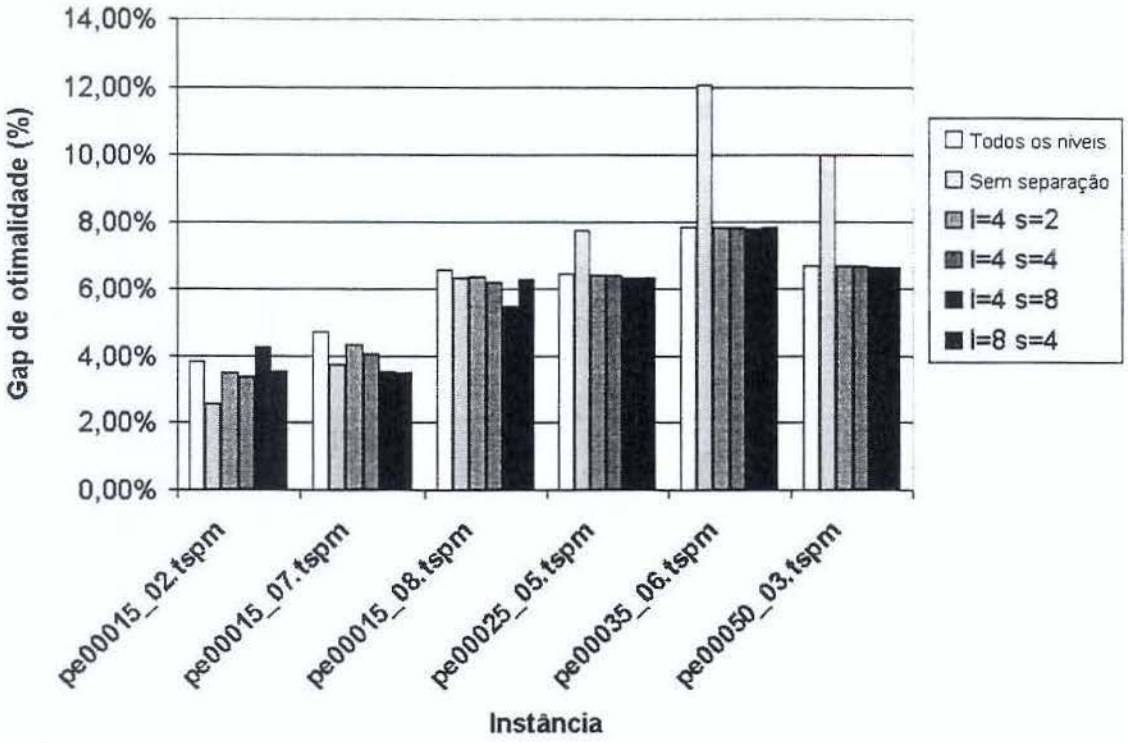


Figura 5.7: Resumo dos resultados do teste de ajuste de parâmetros do algoritmo exato do *PE*.

A estratégia de *branching* utilizada no algoritmo foi aquela conhecida por *best bound*, que consiste em explorar o vértice da árvore de *Branch & Bound* com melhores limitantes locais. Além disso, a execução do algoritmo foi limitada a 100 níveis na árvore de *Branch & Bound* (que entretanto não foi atingido em nenhum dos testes).

Foram então realizados novos testes, agora sobre todas as instâncias, para determinar o desempenho do algoritmo exato de forma geral. Os resultados obtidos encontram-se resumidos nas figuras 5.8, 5.9, 5.10, 5.11 e 5.12.

As quatro primeiras figuras (5.8, 5.9, 5.10 e 5.11) mostram o desempenho do algoritmo exato do *PE* quando aplicado a cada um dos tipos de instâncias geradas para o problema (instâncias originadas do mapa de Goiânia, da *ORLIB*, das instâncias de *Solomon* e da *TSPLIB*, respectivamente). A figura 5.12 mostra a combinação dos dados das outras quatro figuras.

Cada figura contém um gráfico que expressa o valor do *gap* de otimalidade dado pela diferença entre os melhores limitantes superior (dado pelo algoritmo heurístico) e inferior (dado pelo algoritmo exato). Para cada instância, o valor do *gap* é fornecido em termos percentuais. A linha superior dos gráficos representa os valores percentuais máximos, a linha intermediária representa o valor médio e a linha inferior, os valores mínimos para cada conjunto de instâncias com mesmo número de vértices.

Observando-se os resultados resumidos nas figuras anteriores, pode-se chegar às seguintes

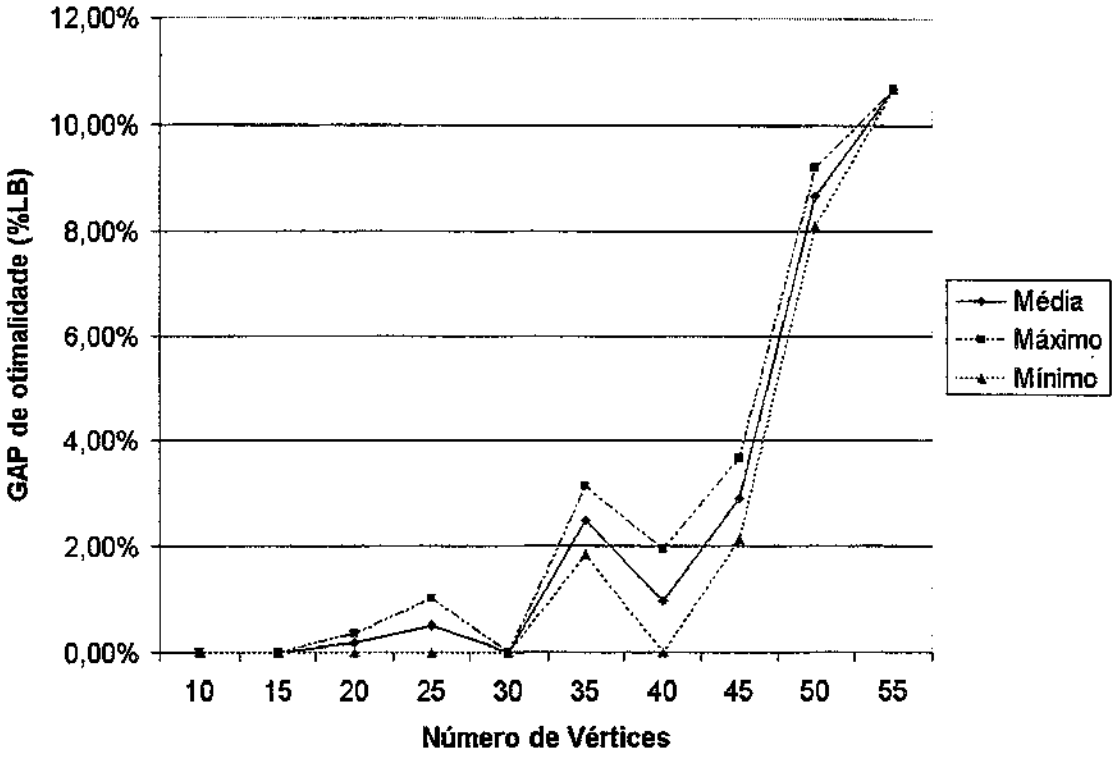


Figura 5.8: Resultados do algoritmo exato do *PE* sobre as instâncias geradas a partir dos mapas de Goiânia.

conclusões. Em primeiro lugar, observa-se que o algoritmo exato apresentou bom desempenho na resolução das instâncias de *Solomon*, da *ORLIB* e dos mapas de Goiânia, para instâncias de até 45 vértices, obtendo *gap* de otimalidade sempre inferior a 4% (exceção para uma instância de *Solomon* que teve *gap* de cerca de 4,5%). Grande parte das instâncias com menos de 30 vértices, dos tipos citados acima, foram resolvidas à otimalidade (no caso da *ORLIB*, todas as instâncias com até 40 vértices foram resolvidas à otimalidade, dentro do limite de 2 horas de tempo de *CPU*).

Esse fato pode ser explicado, para as instâncias da *ORLIB* e de *Solomon*, pois estas correspondem a instâncias do *VRP*, em que os valores para a capacidade dos veículos e as demandas é diferente daquela observada nas instâncias reais típicas do *PE*. Nessas instâncias, a relação entre a capacidade do veículo e a demandas dos clientes é tipicamente maior que aquelas observadas na entrega de revistas, levando a instâncias com fluxo menos restrito. Dessa forma, a formulação do *PE* tende a se comportar como o *TSP*. Nesse caso, a qualidade dos limitantes inferiores obtidos a partir das relaxações lineares são melhores pois as desigualdades de eliminação de subciclos tornam-se mais efetivas.

Já para o caso das instâncias geradas a partir das instâncias do *TSP* obtidas na *TSPLIB*, o algoritmo exato teve um desempenho relativamente inferior. A maior instância resolvida à

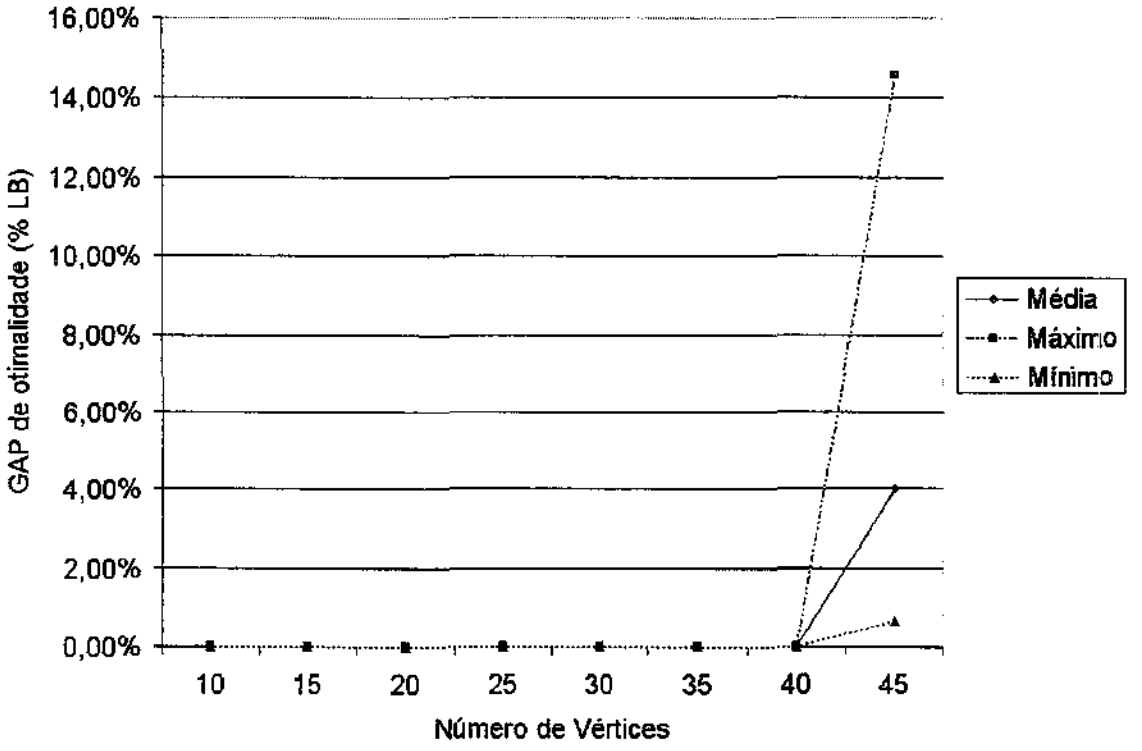


Figura 5.9: Resultados do algoritmo exato do *PE* sobre as instâncias modificadas do *MDVRP* da *ORLIB*.

otimalidade dentro de 2 horas de execução do algoritmo tem 35 vértices. Entretanto, já a partir de 25 vértices, o *gap* de otimalidade começa a se expandir.

No gráfico da figura 5.12, percebe-se a grande diferença entre os *gaps* mínimo e máximo, em que os valores mínimos sofrem a influência dos bons resultados sobre as instâncias dos mapas, da *ORLIB* e de *Solomon*, enquanto que os valores máximos são dilatados pelos resultados do algoritmo sobre as instâncias da *TSPLIB*.

Pode-se identificar claramente uma tendência ao aumento do *gap* de otimalidade à medida que aumenta o número de vértices da instância. Por essa razão, é óbvio que o algoritmo exato não é capaz de resolver grande parte das instâncias reais típicas do *PE* (com até 300 vértices). Entretanto, o algoritmo apresentou desempenho razoável para instâncias próximas às menores instâncias típicas (50 vértices).

Passa-se então à análise dos resultados obtidos pela heurística. Em primeiro lugar, observa-se que nenhum dos testes do algoritmo exato obteve um valor de limitante superior melhor que o previamente fornecido ao algoritmo. Ou seja, durante a execução dos testes do algoritmo exato, não foi encontrado nenhuma solução viável melhor que aquela fornecida pela heurística. Entretanto, percebe-se que o *gap* de otimalidade entre os resultados do algoritmo exato e do algoritmo heurístico tendem a crescer com o número de vértices da instância.

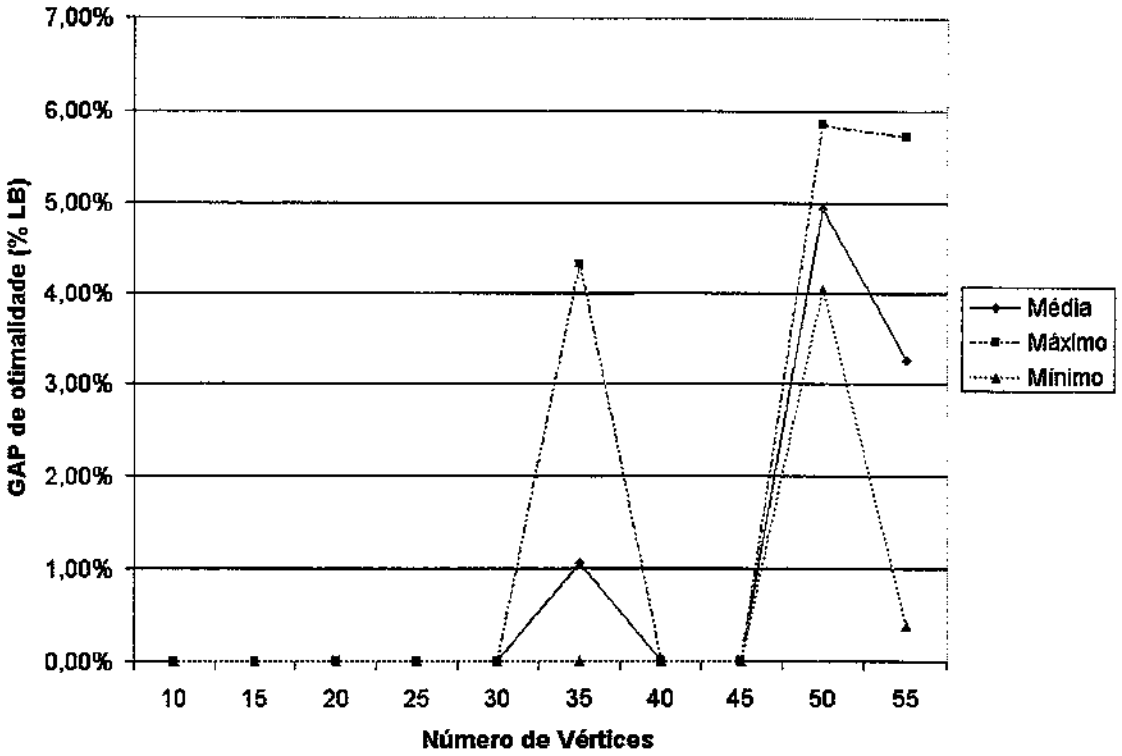


Figura 5.10: Resultados do algoritmo exato do *PE* sobre as instâncias modificadas do *VRP* de *Solomon*.

Existem três cenários possíveis para interpretação dos resultados. No primeiro cenário, mais otimista, o algoritmo exato não encontrou melhores limitantes superiores porque os limitantes primais obtidos pela heurística já estavam muito próximos dos valores para a solução ótima. Nesse caso, os grandes *gaps* de otimalidade resultantes seriam devidos a uma possível má qualidade dos limitantes inferiores, obtidos pelo algoritmo exato.

No segundo cenário, mais pessimista, os limitantes inferiores estariam bastante próximos dos valores das soluções ótimas. Os *gaps* de otimalidade seriam devidos à possível má qualidade das soluções obtidas pela heurística, combinadas com a dificuldade do algoritmo exato em encontrar boas soluções viáveis durante a execução.

O terceiro e último cenário é resultado da combinação das duas situações anteriores, em que os valores das soluções ótimas estariam situados numa posição intermediária com relação aos limitantes inferiores e às soluções da heurística. Nesse caso, haveria espaço para estudos visando melhorar tanto a heurística do *PE* quanto para obter outras desigualdades válidas fortes para melhoria dos limitantes inferiores. Este cenário parece ser o que melhor se adequa aos resultados encontrados. Mas para instâncias do porte daquelas que foram analisadas aqui, a primeira hipótese também é bastante razoável.

Além desses testes (teste de ajuste de parâmetros e o teste geral), foi realizado um outro teste computacional para verificar a qualidade das desigualdades de *flow cover* adicionadas na

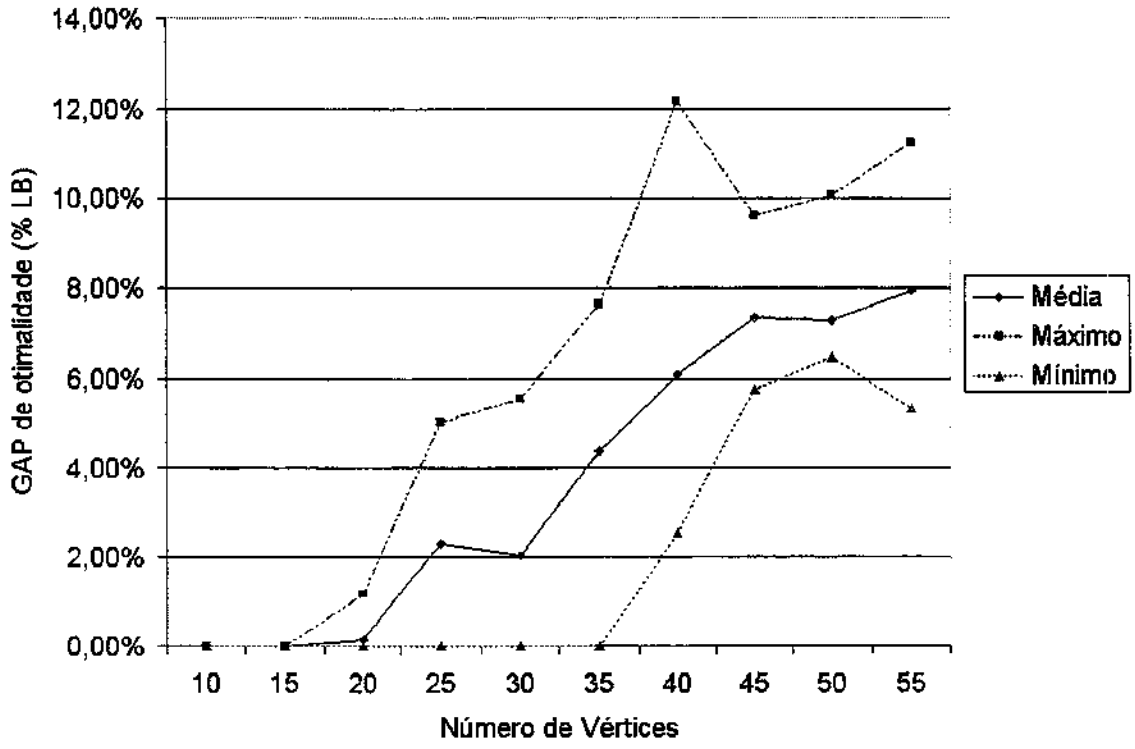


Figura 5.11: Resultados do algoritmo exato do *PE* sobre as instâncias modificadas do *TSP* obtidas da *TSPLIB*.

fase de *cutting planes* do algoritmo exato.

Para isso, foram comparados os limitantes inferiores obtidos pelo algoritmo implementado no presente trabalho com os limitantes obtidos por outro *framework* utilizado para implementar algoritmos de *Branch & Cut*, o *MINTO*, *Mixed INTeger Optimizer* [NSS94], em sua versão *2.3*.

O *MINTO* provê facilidades para a implementação de algoritmos de *B&C* em linguagem *C*, e, diferentemente do *framework* utilizado para implementar os algoritmos aqui descritos (o *ABACUS*), incorpora uma série de resultados teóricos recentes da área de *Programação Linear Interia Mista*. Como principais exemplos pode-se citar: a separação de desigualdades de cliques, *knapsack covers*, desigualdades obtidas por implicação lógica, além da desigualdade tratada no presente trabalho, as desigualdades de *flow cover*. Para uma descrição completa das funcionalidades desse sistema, refira-se a [SN96].

O objetivo desses testes é avaliar se a qualidade dos limitantes inferiores obtidos com a separação de *flow covers* pelo algoritmo exato é comparável à qualidade dos limitantes inferiores obtidos pelo *MINTO* separando as mesmas desigualdades. Por ser uma ferramenta distribuída comercialmente, considerou-se que o *MINTO* seria um bom parâmetro para a avaliação da qualidade dos *flow covers* gerados pela nossa separação.

Para realizar esses testes, foram necessárias algumas modificações no algoritmo exato.

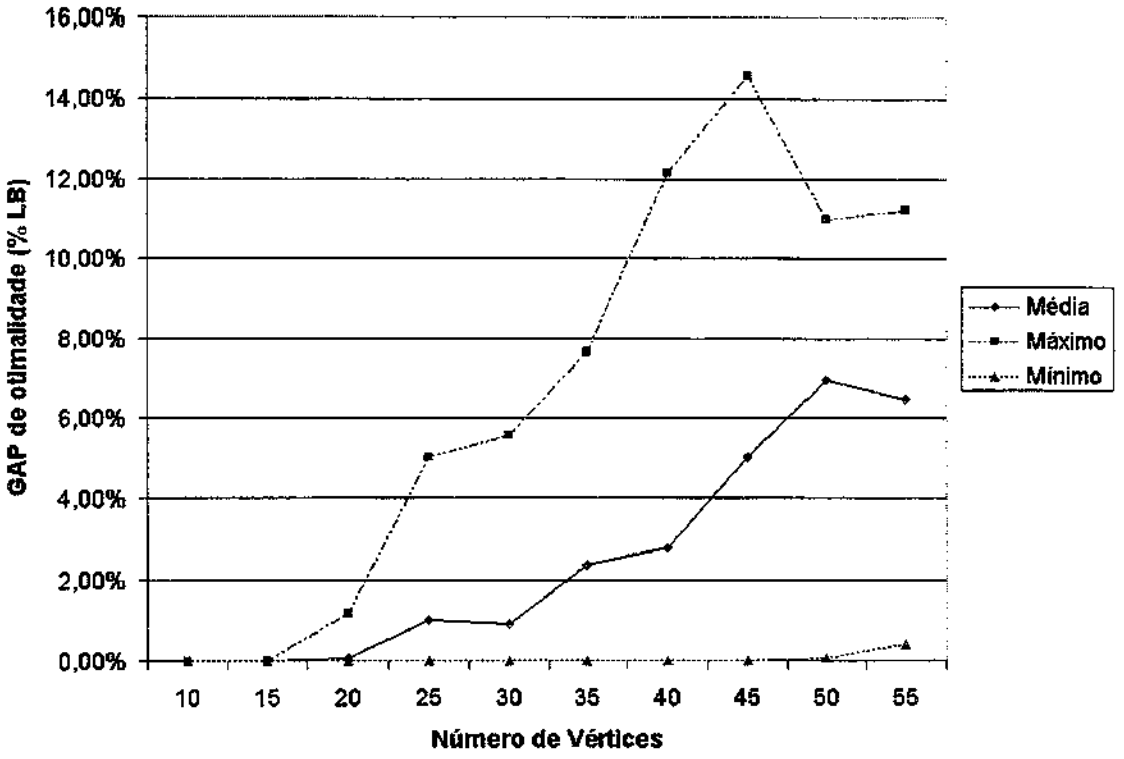


Figura 5.12: Resultados do algoritmo exato do *PE* sobre todas as instâncias.

Em primeiro lugar, não seria possível separar desigualdades de eliminação de subciclos no *MINTO*, sem que se empregasse um esforço de implementação considerável. Sendo assim, o *MINTO* seria incapaz de resolver o *PE* à otimalidade, visto que a ausência das desigualdades de eliminação de subciclos tornam a formulação do problema incorreta.

Dessa forma, o teste ficou limitado à avaliação dos limitantes inferiores calculados apenas no vértice raiz da árvore de *Branch & Bound*, executando-se assim, um algoritmo de *cutting planes* puro.

Foram então escolhidas, aleatoriamente, 13 instâncias, sobre as quais foi executado o algoritmo exato implementado no *ABACUS*, separando apenas desigualdades de eliminação de subciclos, no vértice raiz do *B&B*. Como saída, obteve-se um *PL* contendo a formulação inicial, mais as desigualdades adicionadas pelo *ABACUS*.

Esse *PL* foi fornecido como entrada para o *MINTO*, onde foram separadas desigualdades de *flow cover* até que o programa não encontrasse mais nenhuma desigualdade violada.

O mesmo *PL* fornecido ao *MINTO*, foi fornecido ao algoritmo implementado no *ABACUS*, onde foram separadas desigualdades de *flow cover*, até que mais nenhuma desigualdade violada fosse encontrada.

Finalmente, os valores dos limitantes inferiores e o número de cortes adicionados pelo *MINTO* e pelo algoritmo exato foram coletados. Estes são apresentados na tabela 5.2.

Nome da Instância	z_{LP}	$\#_{sub}$	$z_{subtour}$	$\#_{Ab}$	z_{ABACUS}	$\#_{Mnt}$	z_{MINTO}	Res.
pe00015_02.tspm	11628.8	13	14328.5	39	14451.8	44	14451.8	=
pe00030_05.tspm	3356.73	30	3673.41	92	3708.36	37	3673.41	A
pe00035_03.tspm	8586.65	52	9763.95	177	9857.77	82	9834.03	A
pe00040_07.tspm	13276.1	42	14678.0	140	14772.3	124	14805.4	
pe00050_01.tspm	17242.4	42	18750.2	211	19131.0	123	18930.9	A
pe00055_03.tspm	33222.5	56	36767.5	266	37784.4	61	36945.1	A
pe00020_03.solm	200.557	20	281.575	2	281.575	55	281.598	
pe00030_04.solm	258.801	33	310.399	78	310.399	63	310.399	=
pe00055_01.solm	404.743	45	480.238	15	481.010	60	481.690	
pe00035_02.mdvrpm	732.872	35	927.026	77	927.328	61	927.072	A
pe00050_01.mdvrpm	1066.68	35	1336.39	139	1337.05	61	1336.62	A
pe00015_02.map	5984.65	16	7778.35	97	7946.20	81	7946.20	=
pe00025_02.map	1338.43	43	2200.65	51	2200.65	47	2200.65	=

Tabela 5.2: Resultados da comparação entre a separação de *flow covers* do *MINTO* e do algoritmo proposto nesta dissertação.

As colunas da tabela 5.2 têm o seguinte significado. A primeira coluna contém o nome da instância utilizada no testes. A segunda coluna, z_{LP}, mostra o valor ótimo da primeira relaxação linear. Em seguida são mostrados o número de desigualdades de eliminação de subciclos encontradas pelo *ABACUS* (coluna $\#_{sub}$) e o valor ótimo do *LP* obtido com a adição desses cortes (coluna $z_{subtour}$). Essa última coluna mostra o limitante inferior para a formulação que foi fornecida ao *MINTO* e ao *ABACUS* para a separação de *flow covers*. Nas próximas duas colunas são mostrados os resultados obtidos pela separação de *flow covers* pelo *ABACUS*. Na coluna $\#_{Ab}$ é mostrado o número de *flow covers* adicionados e na coluna z_{ABACUS}, é mostrado o respectivo valor do limitante inferior obtido após a adição dos cortes. Nas colunas $\#_{Mnt}$ e z_{MINTO} são apresentados os mesmos resultados, agora obtidos pelo *MINTO*. A coluna $\#_{Mnt}$ mostra o número de cortes adicionados, e a coluna z_{MINTO} o limitante inferior correspondente.

Finalmente, na coluna **Res.**, são mostrados os resultados dos testes. Esta coluna pode assumir três valores: o valor *"A"* quando o limitante inferior do *ABACUS* é melhor que o do *MINTO*; o valor *"="* quando os limitantes são iguais; e valor nulo quando o limitante obtido pelo *MINTO* é melhor que o do *ABACUS*.

Observando a tabela 5.2, percebe-se que uma pequena vantagem do algoritmo implementado sobre o *ABACUS* com relação ao *MINTO*. Em 6 instâncias, o algoritmo exato implementado sobre o *ABACUS* obteve melhores limitantes inferiores, em 3 instâncias obteve piores limitantes, tendo encontrado os mesmos valores que o *MINTO* em outras 4 instâncias.

Percebe-se também que, em geral, nas instâncias obtidas a partir da *TSPLIB* (primeiras 6 linhas da tabela) e naquelas obtidas da *ORLIB* (linhas 10 e 11), o algoritmo proposto nesta dissertação supera o *MINTO* em quase todas as instâncias, obtendo valores consideravelmente maiores que este último. Entretanto, nas outras instâncias, o *MINTO* apresenta melhor desempenho. Nessas últimas instâncias, nenhum dos dois algoritmos obtém melhorias consideráveis nos limitantes (na maior parte dos casos, nenhuma melhoria é conseguida).

Deve-se ressaltar, que apesar do *MINTO* ter apresentado desempenho inferior, este implementa a separação de *flow covers* para quaisquer *PLIM* fornecidos como entrada, enquanto que no algoritmo proposto nesta dissertação, a separação é feita sobre um problema cuja estrutura das desigualdades que compõem a formulação é bem conhecida. Esse fato pode deixar o *MINTO* em ligeira desvantagem nesse teste.

Mesmo assim, pode-se concluir que a qualidade das desigualdades separadas pela implementação no *ABACUS* é bastante satisfatória.

Capítulo 6

Conclusões

Neste capítulo serão apresentadas as conclusões sobre os resultados dos estudos práticos e teóricos desenvolvidos na dissertação. Ao final do capítulo serão apresentadas propostas para futuras extensões do presente trabalho (seção 6.2).

6.1 Conclusões

O objetivo principal deste trabalho, conforme apresentado no capítulo 1 foi estudar e desenvolver uma solução computacional completa para o problema da *Logística de Distribuição de Revistas para Assinantes*. Para isso, foram estabelecidas três metas específicas: (1) Projetar e implementar algoritmos heurísticos capazes de resolver os problemas de otimização em grafos que compõem o processo de logística de distribuição de revistas (*PD* e *PE*); (2) Realizar um estudo teórico a respeito do *PE* baseado na *Otimização Combinatória* resultando no projeto e implementação de um algoritmo exato para o problema, baseado em *Programação Linear Inteira*; (3) Propor e implementar um *Sistema Espacial de Apoio à Decisão* para dar suporte ao processo decisório da logística de distribuição de revistas, baseado num *Sistema de Informação Geográfica* comercial, em que os algoritmos heurísticos obtidos como resultado de (1) desempenham papel fundamental no auxílio ao processo decisório.

Contemplando o objetivo (1), o capítulo 2 apresenta os algoritmos heurísticos projetados e implementados para resolver os problemas propostos no trabalho. Analisando os resultados obtidos por estes algoritmos rodando sobre instâncias de teste, apresentados no capítulo 5, pode-se constatar que ambos apresentam resultados bastante satisfatórios.

Em primeiro lugar, o algoritmo heurístico do *PD* obteve bons resultados tanto em relação à minimização do número de distritos, em que o algoritmo atingiu o valor ótimo para a maioria das instâncias de teste, quanto em relação ao balanceamento de cargas, onde foram obtidos desvios padrão da carga de trabalho dos distritos inferiores a 1 % da carga máxima de trabalho na maioria das instâncias testadas, ou seja, foram obtidos desvios entre as cargas de trabalhos dos entregadores inferiores a 5 minutos.

O algoritmo heurístico do *PE* também obteve resultados bastante satisfatórios. Através dos testes apresentados no capítulo 5, observa-se que em todas as instâncias em que foi provada a otimalidade, o algoritmo heurístico obteve a solução ótima. Naquelas em que não se conhece a solução ótima, o *gap* de otimalidade ficou abaixo de 10% do limitante inferior (exceto em 6 dentre as 184 instâncias de teste).

Quando o algoritmo heurístico do *PE* é executado sobre instâncias de mais de 40 vértices, percebe-se que ocorre um distanciamento entre o valor das soluções obtidas e os respectivos limitantes inferiores. Pode-se observar que há uma tendência ao crescimento desta distância, com o aumento do número de vértices das instâncias testadas.

Entretanto, este fato não demonstra necessariamente a perda de qualidade das soluções obtidas, uma vez que não há provas de que as soluções encontradas pelo algoritmo heurístico não são ótimas. Provavelmente, o distanciamento observado é devido à baixa qualidade dos limitantes inferiores obtidos pelo algoritmo exato. Dessa forma, conclui-se que o objetivo (1) descrito acima foi satisfatoriamente atingido.

O capítulo 3 contempla a meta estabelecida no objetivo (2). Neste capítulo foi realizado um estudo teórico a respeito do *PE*, em que se utilizou a *Programação Linear Inteira* para a criação de um modelo matemático que resultou no projeto e implementação de um algoritmo exato para o problema, baseado na técnica de *Branch and Cut*. Vale ressaltar que o problema em questão consiste num problema real de roteamento ainda não estudado, muito semelhante ao *MDVRP* não fixo, que tem relevância prática para um determinado conjunto de empresas.

Além disso, a formulação do *PE* apresenta um subconjunto de desigualdades ((3.37) e (3.38)) que modelam a restrição de capacidade sobre estoques em depósitos no *VRP*. Em geral, as versões do *VRP* tratadas na literatura consideram esse estoque ilimitado. A formulação apresentada neste trabalho permite modelar versões do *VRP* em que a quantidade de produto armazenada em estoques é limitada.

Finalmente, com relação ao objetivo (3), tem-se a proposta e a implementação de um *Sistema Espacial de Apoio à Decisão*, apresentado no capítulo 4, específico para o caso da logística de distribuição de revistas, integrando os algoritmos heurísticos apresentados no capítulo 2. A integração dos resultados de (1) e (3) resulta na descrição genérica e na implementação parcial de em um sistema computacional para a resolução de um problema operacional complexo encontrado nas empresas de distribuição de revistas.

Com os três resultados apresentados acima conclui-se que todos os objetivos propostos na presente dissertação de Mestrado foram satisfatoriamente atingidos.

6.2 Extensões

Como futuras extensões a este trabalho, propõe-se:

- A implementação da separação de desigualdades de *flow covers* para conjuntos de vértices, como proposto em [RW86], no algoritmo exato, na expectativa de melhorar os limitantes inferiores obtidos para o *PE*;
- Estudo de uma formulação para o *PE* que tenha melhor desempenho sobre instâncias cuja capacidade do entregador seja mais apertada, visto que a formulação aqui apresentada se comporta melhor no caso oposto, quando a capacidade é mais frouxa. Uma alternativa seria o estudo de formulações baseadas no particionamento de conjuntos (*set-partitioning*), pois podem ter melhor desempenho, uma vez que o número de rotas viáveis, considerado por essas formulações, é menor no caso de capacidade mais restrita;
- Estudo da estrutura poliédrica do *PE*, com análise da dimensão do poliedro e das faces definidas pelas desigualdades presentes na formulação, provando se estas são ou não facetas do poliedro do problema.
- A implementação completa do *SEAD* proposto na seção 4.2;
- Validação do *SEAD* proposto junto às empresas de distribuição de assinaturas de revistas;
- Extensões ao modelo do banco de dados espacial e das formulações para a utilização do sistema com veiculos motorizados;

Apêndice A

Glossário

PD: Problema do Distritamento, também referido no texto simplesmente como **distritamento**;

PE: Problema da Entrega de Revistas;

TSP: *Traveling Salesman Problem*, ou Problema do Caixeiro Viajante;

VRP: *Vehicle Routing Problem*, ou Problema do Roteamento de Veículos;

FCN: *Fixed-Charge Network Flow Problem*, ou Problema do Fluxo em Redes com Custos Fixos;

MDVRP: *Multi-depot Vehicle Routing Problem*, ou Problema do Roteamento de Veículos com Múltiplos Depósitos;

SVPDPTW: *Single Vehicle Pickup and Delivery Problem with Time-Windows*, ou Problema da Coleta e Entrega com um Único Veículo e Janelas de Tempo;

GRASP: *Greedy Randomized Adaptative Search Procedure*, ou Procedimento Guloso Randomizado e Adaptativo;

RCL: *Restricted Candidate List*, ou Lista Restrita de Candidatos;

PE fixo: Modificação no *PE* em que o número de vezes que cada depósito deve ser utilizado é fixo e definido *a priori*;

PL: Problema de Programação Linear;

PLI: Problema de Programação Linear Inteira;

PLM: Problema de Programação Linear Inteira Mista;

$\mathbb{R}$: Conjunto dos números reais;

$\mathbb{Z}$: Conjunto dos números inteiros;

$\mathbb{B}$: Conjunto binário contendo os elementos 0 e 1;

$\mathbb{R}^n_+$: Conjunto dos vetores de dimensão n, reais e não negativos;

FCI: *Flow Cover Inequality* ou Desigualdade de cobertura de fluxo;

SGFCI: *Simple Generalized Flow Cover Inequality* ou Desigualdade de cobertura de fluxo genérica simples;

EGFCI: *Extended Generalized Flow Cover Inequality* ou Desigualdade de cobertura de fluxo genérica extendida;

LSGFCI: *Lifted Simple Generalized Flow Cover Inequality* ou Desigualdade de cobertura de fluxo genérica simples com *lifting*;

Bibliografia

[ABCC91] D. Applegate, R. E. Bixby, V. Chvátal, e W. Cook. Personal communication, 1991.

[ABCC93] D. Applegate, R. E. Bixby, V. Chvátal, e W. Cook. Personal communication, 1993.

[AMS89] Y. Agarwal, K. Mathur, e H. M. Salkin. A set-partitioning-based algorithm for the vehicle routing problem. *Networks*, 19:731–750, 1989.

[BCCN96] E. Balas, S. Ceria, G. Cornuéjols, e G. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19:1–9, 1996.

[Bea] J. E. Beasley. Operations research library (orlib). http://mscmga.ms.ic.ac.uk/info.html.

[BGAB83] L. D. Bodin, B. L. Golden, A. Assad, e M. Ball. Routing and scheduling of vehicles and crews: the state of the art. *Computers and Operations Research*, 10:69–211, 1983.

[BJ90] Bazaraa e Jarvis. *Linear Programming and Network Flow.* Wiley, 1990.

[BLS93] L. J. J. Van Der Bruggen, J. K. Lenstra, e P. C. Schuur. Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science*, 27(3):298–311, agosto de 1993.

[BMMN95] M. O. Ball, T. L. Magnanti, C. L. Monma, e G.L. Nemhauser. *Handbooks in Operations Research and Management Science: Network Routing.* North-Holland, 1995.

[CCH+96] G. Câmara, M. A. Casanova, A. S. Hemerly, G. C. Magalhães, e C. M. B. Medeiros. *Anatomia de Sistemas de Informação Geográfica.* Instituto de Computação, UNICAMP, Campinas, Brasil, 1996.

[Chr85] N. Christofides. Vehicle routing. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, e D. B. Shmoys, editores, *The Traveling Salesman Problem, A Guided Tour of Combinatorial Optimization*, capítulo 12, pp. 431–448. Wiley, 1985.

[Cor90] T. H. Cormen. *Introduction to Algorithms.* MIT Press, 1990.

[CP80] H. Crowder e M. W. Padberg. Solving large-scale symmetric traveling salesman problems to optimality. *Management Science*, 26:495–509, 1980.

[DBB94] Y. Ding, A. Baveja, e R. Batta. Implementing Larson and Sadiq's Location Model in a Geographic Information System. *Computers Ops Research*, 21(4):447–454, 1994.

[DDS91] M. Desrochers, J. Desrosiers, e M. M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 1991.

[dG96] Prefeitura Municipal de Goi nia. Mapa urbano digital b sico de goi nia / cd-rom, 1996.

[DLS90] M. Desrochers, J. K. Lenstra, e M. W. P. Savelsbergh. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research*, 46:322–332, 1990.

[DSD84] J. Desrosiers, F. Soumis, e M. Desrochers. Routing with time-windows by column generation. *Networks*, 14:545–565, 1984.

[EWGC71] S. Eilon, C. D. T. Watson-Gandy, e N. Christofides. *Distribution Management: Mathematical Modelling and Practica Analysis*. Griffin, London, 1971.

[FF62] L. Ford e D. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[FJ78] M. L. Fisher e R. Jaikumar. A decomposition algorithm for large-scale vehicle routing. Technical Report 78-11-05, University of Pennsylvania, Department of Decision Sciences, novembro de 1978.

[FJ81] M. L. Fisher e R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11:109–124, 1981.

[FR76] B. Foster e D. Ryan. An integer programming approach for the vehicle scheduling problem. *Operations Research Quarterly*, 27:367–384, 1976.

[FR95] T. A. Feo e M. G. C. Rezende. Greedy randomized adaptative search procedures. *Journal of Global Optimization*, 1:1–27, 1995.

[GA86] B. L. Golden e A. A. Assad. Perspectives on vehicle routing: exciting new developments. *Operations Research*, 34(5):803–810, 1986.

[GHJV94] E. Gamma, R. Helm, R. Johnson, e J. Vlissides. *Design Patterns: Elements of Reusable Objects Oriented Software*. Addison Wesley, Massachusets, USA, 1994.

[GJ79] M. R. Garey e D. S. Johnson. *Computer and Intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, USA, 1979.

[Glo89] F. Glover. Tabu search - part 1. *ORSA Journal on Computing*, 1:190–206, 1989.

[Glo90] F. Glover. Tabu search - part 2. *ORSA Journal on Computing*, 2:4–32, 1990.

[GNS95] Z. Gu, G. L. Nemhauser, e M. W. P. Savelsbergh. Lifted flow cover inequalities for mixed 0-1 integer programs. Technical Report LEC-95-05, Georgia Institute of Technology, School of Industrial and Systems Engineering, Atlanta, GA, 30332-0205, maio de 1995.

[Gom58] R. E. Gomory. Outline for an algorithm for integer solutions to linear programs. *Bull. Amer. Math. Soc.*, 64:275–278, 1958.

[Gom60] R. E. Gomory. Solving linear programming problems in integers. *Proc. Sympos. Appl. Math.*, 10:211–215, 1960.

[Gom63] R. E. Gomory. An algorithm for integer solutions to linear programs. In R. L. Graves e P. Wolfe, editores, *Recent Advances in Mathematical Programming*, pp. 269–302. McGraw-Hill, New York, 1963.

[GP85] M. Grötschel e M. W. Padberg. Polyhedral theory. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, e D. B. Shmoys, editores, *The Traveling Salesman Problem, A Guided Tour of Combinatorial Optimization*, capítulo 8, pp. 251–306. Wiley, 1985.

[Grö80] M. Grötschel. On the symmetric traveling salesman problem: Solution of a 120-city problem. *Mathematical Programming Studies*, 12:61–77, 1980.

[Grö91] M. Grötschel. Solution of large-scale symmetric traveling salesman problems. *Mathematical Programming*, 51:141–202, 1991.

[JAM91] D. S. Johnson, C. R. Aragon, e L. A. McGeoch. Optimization by simulated annealing - an experimental evaluation (part ii). *Operations Research*, 39:378–406, 1991.

[JAMS89] D. S. Johnson, C. R. Aragon, L. A. McGeoch, e C. Schevon. Optimization by simulated annealing - an experimental evaluation (part i). *Operations Research*, 37:865–892, 1989.

[Lap92] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[LFE94] M. Laguna, T. A. Feo, e H. C. Elrod. A greedy randomized adaptative search procedure for the 2-partition problem. *Operations Research*, 42:677–687, 1994.

[LK73] S. Lin e B. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21:498–516, 1973.

[LLRS85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, e D. B. Shmoys, editores. *The Traveling Salesman Problem, A Guided Tour of Combinatorial Optimization*. Wiley, 1985.

[LN86] G. Laporte e Y. Nobert. An exact algorithm for the asymetrical capacitated vehicle routing problem. *Networks*, 16:33–46, 1986.

[LN87] G. Laporte e Y. Nobert. Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics*, 31:147–184, 1987.

[LND85] G. Laporte, Y. Nobert, e M. Desrochers. Optimal routing under capacity and distance restrictions. *Operations Research*, 33:1050–1073, 1985.

[Man89] U. Manber. *Introduction to Algorithms, A Creative Approach.* Addison Wesley, 1989.

[Mar99] C. A. J. Martinhom. *Relaxação Lagrangeana com Geração de Desigualdades Válidas Aplicada ao Problema do Roteamento de Veículos.* PhD thesis, Universidade Federal do Rio de Janeiro, 1999.

[MT90] S. Martello e P. Toth. Generalized assignment problem. In S. Martello e P. Toth, editores, *Knapsack Problems. Algorithms and Computer Implementations*, pp. 189–220. Wiley, 1990.

[Nie96] O. Nielsen. Using GIS in Denmark for Traffic Planning and Decision Support. *Journal of Advanced Transportation*, 29(3):335–354, 1996.

[NOI94] H. Nagamochi, T. Ono, e T. Ibaraki. Implementing an efficient minimum capacity cut algorithm. *Mathematical Programming*, 67:325–341, 1994.

[NSS94] G. L. Nemhauser, M. W. P. Savelsbergh, e G. C. Sigismondi. Minto, a mixed integer optimizer. *Operations Research Letters*, 15:47–58, 1994.

[NW88] G. L. Nemhauser e L. A. Wolsey. *Integer and Combinatorial Optimization.* Wiley, 1988.

[Or76] I. Or. *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistics of Blood Banking.* PhD thesis, Department of Industrial Engineering and Management Science, Northwestern University, 1976.

[Orl76] C. Orloff. Route-constrained fleet scheduling. *Transportation Science*, 10:149–168, 1976.

[Pap73] C. H. Papadimitriou. *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, New Jersey, 1973.

[PG85] M. W. Padberg e M. Grötschel. Polyhedral computations. In E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, e D. B. Shmoys, editores, *The Traveling Salesman Problem, A Guided Tour of Combinatorial Optimization*, capítulo 9, pp. 307–360. Wiley, 1985.

[PR87] M. Padberg e G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1–7, 1987.

[PR90] M. Padberg e G. Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, 47:19–36, 1990.

[PR91] M. Padberg e G. Rinaldi. A branch and cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, 33:1–201, 1991.

[Rei] G. Reinelt. Tsplib. http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html.

[Rei94] G. Reinelt. The traveling salesman problem, computational solutions for tsp applications. *Lecture Notes In Computer Science*, 840, 1994.

[Roc] Y. Rochat. Solomon data sets for the vrptw. http://masg1.epfl.ch/rose.mosaic/rochat_data/solomon.html.

[RSORS96] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, e G. D. Smith. *Modern Heuristic Search Methods*. Wiley, 1996.

[RW86] T. J. Van Roy e L. A. Wolsey. Valid inequalities for mixes 0-1 programs. *Discrete Applied Mathematics*, 14:199–213, 1986.

[RW87] T. J. Van Roy e L. A. Wolsey. Solving mixed integer programming problems using automatic reformulation. *Operations Research*, 35:45–57, 1987.

[RZ68] M. R. Rao e S. Zionts. Allocation of transportation units to alternative trips - a column generation scheme with out-of-kilter subproblems. *Operations Research*, 16:52–63, 1968.

[SMP96] C. C. Souza, C. B. Medeiros, e R. S. Pereira. Integrating heuristics and spatial databases: A case study. Technical Report IC-96-18, Unicamp, Instituto de Computação, novembro de 1996.

[SMRY99] L. Seffino, C. B. Medeiros, J. Rocha, e B. Yi. WOODSS - A Spatial Decision Support System based on Workflows. *Decision Support Systems*, 1999.

[SN96] M. W. P. Savelsbergh e G. L. Nemhauser. Functional description of minto, a mixed integer optimizer - version 2.3. Technical report, Georgia Institute of Technology - School of Industrial and Systems Engineering, Atlanta, GA 30332-0205, USA, novembro de 1996.

[Thi] S. Thienel. Abacus - a branch and cut system. http://www.informatik.uni-koeln.de/ls_juenger/projects/abacus.html.

[VMP98] I. Mendes V. Maniezzo e M. Paruccini. Decision Support for Siting Problems. *Decision Support Systems*, 23:273–284, 1998.

[Wol98] L. A. Wolsey. *Integer Programming*. John Willey, 1998.