Pontifícia Universidade Católica
do Rio de Janeiro

Carlos Eduardo Costa Vieira

Heurísticas para o Problema das p-Medianas Conectadas

Tese de Doutorado

Tese apresentada ao Programa de Pós–graduação em Informática do Departamento de Informática da PUC–Rio como requisito parcial para obtenção do título de Doutor em Informática

Orientador: Prof. Celso da Cruz Carneiro Ribeiro

Rio de Janeiro
Setembro de 2006

Pontifícia Universidade Católica
do Rio de Janeiro

Carlos Eduardo Costa Vieira

Heurísticas para o Problema das p-Medianas Conectadas

Tese apresentada ao Programa de Pós–graduação em Informática do Departamento de Informática do Centro Técnico Científico da PUC–Rio como requisito parcial para obtenção do título de Doutor em Informática. Aprovada pela Comissão Examinadora abaixo assinada.

Prof. Celso da Cruz Carneiro Ribeiro
Orientador
Departamento de Informática – PUC–Rio

Prof. Sérgio Lifschitz
Departamento de Informática – PUC-Rio

Prof. Simone de Lima Martins
Departamento de Ciência da Computação – UFF

Prof. Maurício Cardoso de Souza
Departamento de Engenharia de Produção – UFMG

Prof. Paulo Oswaldo Boaventura Netto
Programa de Engenharia de Produção – COPPE/UFRJ

Prof. Luiz Satoru Ochi
Departamento de Ciência da Computação – UFF

Prof. José Eugênio Leal
Coordenador Setorial do Centro Técnico Científico – PUC–Rio

Rio de Janeiro, 11 de Setembro de 2006

Carlos Eduardo Costa Vieira

Completou o segundo grau em Tecnologia em Informática Industrial na Escola Técnica Federal de Ouro Preto - Minas Gerais. Graduou–se em Ciência da Computação na Universidade Federal de Ouro Preto - Minas Gerais. Obteve o título de Mestre em Sistemas e Computação no Instituto Militar de Engenharia - Rio de Janeiro. Durante o mestrado foi bolsista da CAPES e desenvolveu o trabalho em heurísticas aplicadas ao problema de alocação de canal em sistemas de telecomunicações móveis.

Ficha Catalográfica

Vieira, Carlos Eduardo Costa

Heurísticas para o Problema das p-Medianas Conectadas / Carlos Eduardo Costa Vieira; orientador: Celso da Cruz Carneiro Ribeiro. – 2006.

v., 191 f: il. ; 30 cm

1. Tese (Doutorado em Informática) - Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2006.

Inclui bibliografia.

1. Informática – Teses. 2. Otimização Combinatória. 3. Projeto de Redes. 4. Localização de Facilidades. 5. Programação Linear Inteira. 6. Heurísticas. 7. Metaheurísticas. I. Ribeiro, Celso da Cruz Carneiro. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Dedico este trabalho à minha esposa Daniele.

Agradecimentos

Aos meus pais, Michele e Daniele pela força e apoio. À Leny e Luiz, Ana Paula, Cristiano, Jorge, Heidy e Rayane por serem a minha segunda família. Obrigado pelos computadores, apoio financeiro e acima de tudo, pela amizade e carinho.

Ao Departamento de Informática da Pontifícia Universidade Católica do Rio de Janeiro pela oportunidade. Um agradecimento especial aos prestativos funcionários do departamento: Rosane Teles e Cosme Leal (biblioteca), Deborah Gonçalves e Emanuelle Oliveira (secretaria), José Carlos, Anderson Oliveira e Luciana Almeida (suporte).

Aos colegas e principalmente grandes amigos que encontrei na PUC-Rio: Aletéia Favacho, Alexandre Duarte, Andréa Cynthia, Sebastián Urrutia e Thiago Noronha. Um agradecimento especial ao amigo Alexandre Duarte por disponibilizar, sempre que possível, um tempo para a discussão e troca de idéias.

Ao pesquisador Maurício Resende pelo tema de pesquisa. Ao Prof. Celso Ribeiro pela oportunidade, incentivo, troca de conhecimentos e advertências, quando necessário.

À CAPES, por financiar a pesquisa no Brasil e a todos aqueles que, direta ou indiretamente, contribuíram na elaboração deste trabalho.

Resumo

Vieira, C. E. C.; Ribeiro, C. C.. **Heurísticas para o Problema das p-Medianas Conectadas**. Rio de Janeiro, 2006. 191p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Esta tese define os problemas das p-medianas conectadas e o de localização de facilidades não-capacitadas conectadas. Possíveis aplicações incluem problemas de planejamento regional e o projeto de redes de telecomunicações ou de transporte. Para o primeiro problema, duas formulações de programação linear inteira são apresentadas e comparadas. Um destes modelos é adaptado para o segundo problema. Para o problema das p-medianas conectadas, algoritmos aproximados são desenvolvidos. Uma estratégia de busca local híbrida é proposta. Para acelerar as iterações do algoritmo de busca local, idéias como circularidade, melhoria iterativa e o descarte de vizinhos são incorporadas. Heurísticas GRASP e VNS são desenvolvidas incluindo a utilização de um filtro com o objetivo de diminuir os tempos de processamento e do procedimento de reconexão por caminhos com o objetivo de melhorar a qualidade das soluções encontradas. Diversos testes são realizados comparando-se esses algoritmos. Os resultados mostraram a necessidade de se executar um passo adicional de pós-otimização às heurísticas GRASP e VNS propostas.

Palavras–chave

Otimização Combinatória. Projeto de Redes. Localização de Facilidades. Programação Linear Inteira. Heurísticas. Metaheurísticas.

Abstract

Vieira, C. E. C.; Ribeiro, C. C.. **Heuristics for the Connected p-Median Problem**. Rio de Janeiro, 2006. 191p. PhD Thesis – Department of Informática, Pontifícia Universidade Católica do Rio de Janeiro.

In this work, the connected p-median and the connected facility location problems are defined. Applications arise in regional planning, design of telecommunications and transportation networks. For the first problem, two integer linear programming formulations are proposed. Adaptations are made in one of these formulations and are used to model the second problem. Approximation algorithms to solve the connected p-median problem are developed. A hybrid local search strategy is proposed. In order to speed up the local search iterations, ideas as circularity, first-improving strategy and discard neighbors are incorporated. A GRASP algorithm and a VNS heuristic are also proposed. A filter is used to reduce the computational time required and a path-relinking is applied to improve the results found. Computational experiments to compare the algorithms are reported. To improve these results, it is applied a post-optimization step to the GRASP and VNS heuristics.

Keywords

Combinatorial Optimization. Network Design. Facility Location. Integer Linear Programming. Heuristics. Metaheuristics.

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

Os problemas de projeto de redes e de localização de facilidades são normalmente estudados separadamente.

O objetivo do problema de projeto de redes, usufruídas por um conjunto de usuários ou clientes, é a identificação de topologias, dimensões e rotas de forma a reduzir custos e melhorar o desempenho e a qualidade dos serviços oferecidos aos usuários da rede. Possíveis áreas de aplicações incluem redes de comunicação ou de transportes. A maioria dos problemas de projeto de redes são classificados como NP-árduos [34]. Entre eles, pode-se citar problemas de otimização combinatória importantes tais como o problema de Steiner [61, 63] e o problema de Steiner com prêmios [10]. Para uma revisão detalhada dos problemas de projeto de redes, modelos e algoritmos, ver, por exemplo, o artigo de Magnanti e Wong [34].

O problema de localização de facilidades trata de decisões sobre onde localizar facilidades considerando usuários ou clientes que devem ser servidos de forma a otimizar algum critério. O termo facilidades é utilizado em um sentido geral e pode incluir fábricas, depósitos, escolas, hospitais, usinas nucleares, centros de comutação eletrônica, paradas de ônibus, estações de metrô. Os critérios a serem otimizados dependem do problema e podem estar relacionados ao tempo ou custo de viagem do cliente, à satisfação da demanda, à localização e potencial resposta dos competidores e ao número de clientes dentro de uma distância de cobertura pré-estabelecida, entre outros. Dentre os diversos problemas de localização, destacam-se o problema de recobrimento de conjuntos [59], os problemas dos p-centros e das p-medianas [24, 25] e o problema de localização de facilidades não-capacitadas [31]. Todos os problemas citados acima são NP-árduos [14]. Vale salientar que os clientes (usuários) utilizam a rede cujo projeto já está pré-estabelecido. Assim, a definição das localizações ótimas está restrita à estrutura da rede projetada a ser utilizada pelos clientes [8]. Revisões dos modelos de localização de facilidades podem ser encontradas em [13, 14, 41].

Muitas vezes, a topologia subjacente à rede tem um profundo impacto sobre a localização ótima das facilidades [36, 42, 54]. Em muitos casos, quando a topologia da rede não é bem projetada, os clientes podem obter um serviço extremamente pobre, mesmo estando as facilidades localizadas de maneira

ótima [4]. Em alguns problemas práticos, a localização de facilidades e o projeto de rede estão intimamente relacionados. Para exemplificar algumas aplicações, um modelo unindo os dois problemas poderia ser utilizado em [8, 36, 37, 54]:

- planejamento regional, onde o governo pode considerar simultaneamente a construção de um novo sistema de rodovias bem como a localização das facilidades públicas tais como correios, escolas e hospitais;
- projeto de redes de telecomunicações, onde existe um compromisso entre a adição de centros ou concentradores (facilidades) e cabos (arcos) para aumentar a capacidade da rede e;
- projeto de redes de transmissão de energia, onde as facilidades são as estações geradoras e subestações, e os arcos são as linhas de transmissão e distribuição.

A otimização simultânea do projeto da rede e da localização ótima das facilidades tem atraído o interesse da comunidade de pesquisa operacional [56]. Existe uma necessidade crescente de investigação de modelos onde as redes são projetadas levando-se em consideração as futuras localizações de facilidades.

Esta tese propõe dois problemas que possuem tanto características de projeto de rede quanto de localização de facilidades.

O primeiro será denominado de *problema das p-medianas conectadas* e, em linhas gerais, pode ser definido como se segue: são dados um grafo, um conjunto de potenciais localizações das facilidades, um conjunto de usuários ou clientes, os custos de servir cada usuário a partir de cada facilidade, o custo de instalação de cada aresta e uma constante p representando o número de facilidades a serem abertas. Deseja-se instalar p facilidades nos vértices do grafo, associando-se os usuários à facilidade aberta mais próxima e unindo-se as p facilidades abertas através de uma árvore. O objetivo do problema é minimizar o custo total: de interconexão das facilidades e o de atendimento (ou de serviço) dos usuários.

Generalizando-se este primeiro problema tem-se o *problema de localização de facilidades não capacitadas conectadas*, com a diferença de que existe um custo fixo associado à abertura de uma facilidade em um vértice do grafo e não existe um número fixo de facilidades a serem abertas, isto é, a quantidade de facilidades a serem abertas não é previamente determinada. Deseja-se instalar um conjunto de facilidades nos vértices do grafo, associando-se os usuários à facilidade aberta mais próxima e unindo-se as facilidades abertas por uma árvore. O objetivo do problema é minimizar o custo total: de interconexão das facilidades, seus custos de instalação e o custo de atendimento dos usuários. Ambos os problemas assumem que cada vértice é simultaneamente um usuário e uma potencial localização de facilidades.

Essencialmente, esses problemas modelam situações nas quais as facilidades abertas necessitam se comunicar umas com as outras. Uma possível aplicação seria, por exemplo, em um projeto de rede de telecomunicações [3, 56]. Um modelo comum de uma rede de telecomunicações consiste de um *backbone* central e de um conjunto de clientes que desejam se comunicar. O *backbone* consiste de um conjunto de vértices interconectados com uma alta largura de banda, sendo que cada nó possui uma capacidade de rotear dados/informações (operação de comutação) e um alto custo associado a essa operação. O projeto da rede consiste em selecionar um subconjunto de vértices a serem conectados através do *backbone*, interligando-os uns com os outros com o propósito de rotear dados/informações dos/para os clientes da rede. Procura-se minimizar o custo associado à operação de comutação nos vértices do *backbone*, à interconexão dos mesmos e à associação dos clientes a um determinado vértice do *backbone*.

A próxima seção mostrará o estado da arte dos modelos que tratam do problema de projeto de rede e de localização de facilidades simultaneamente. A última seção deste capítulo apresentará o objetivo da tese, suas contribuições e a descrição de sua organização.

1.1 Estado da Arte

Alguns artigos na literatura examinaram o relacionamento entre a localização de facilidades e a topologia das redes. Bandelt e Labbé [4] mostraram que quando a rede não é projetada apropriadamente, os clientes podem obter um serviço extremamente pobre, mesmo quando as facilidades estejam localizadas de uma maneira ótima. Berman et al. [7] trataram do seguinte problema: supondo que uma ou mais facilidades já estejam localizadas em uma rede, como modificá-la eficientemente com o objetivo de melhorar a localização da(s) facilidade(s) existentes? As mudanças ocorrem na topologia da rede e não através da escolha de novas localizações para as facilidades. Peeters e Thomas [42] investigaram o impacto de diferentes topologias de rede em soluções ótimas para o problema das p-medianas, encontrando resultados significativos.

Diversos trabalhos recentes incorporaram a localização de facilidades no processo de tomada de decisão envolvido ao se projetar uma rede.

Rolla [54] integrou os aspectos de localização discreta e de planejamento topológico, propondo o *problema de planejamento de redes com localização de facilidades capacitadas*. Esse problema é definido sobre um grafo, no qual o conjunto de vértices é dividido em nós de oferta, nós de demanda ou consumidores e nós intermediários ou de transbordo. Os custos envolvidos no problema são: custo fixo de instalação da facilidade em um nó de oferta, custo fixo de instalação dos arcos e custo variável no fluxo que passa pelos arcos. O objetivo do

problema é determinar a combinação de nós de oferta e arcos que, a um custo mínimo, conduzam os fluxos necessários ao atendimento das demandas dos nós consumidores, passando, se necessário, pelos nós intermediários. O termo "capacitadas" utilizado na denominação do problema se refere à capacidade dos nós de oferta candidatos à localização das facilidades. A autora apresenta uma formulação para o problema e resultados computacionais para diversas instâncias através de abordagens exata e heurística (busca tabu e relaxação lagrangeana com método de subgradientes).

Bhadury et al. [8] provaram pela primeira vez que modelos integrados de projeto de redes e de localização de facilidades são computacionalmente difíceis. Para isso, estudaram o seguinte problema: assumindo-se que uma facilidade esteja localizada em um determinado vértice de um grafo, deseja-se projetar uma rede de transporte usufruída pelos clientes aproveitando-se o serviço oferecido por essa única facilidade. Os autores mostraram que esse modelo é equivalente ao problema da árvore geradora mínima restrito pela mediana: dados um grafo e um vértice específico, deseja-se encontrar a árvore geradora mínima sendo o vértice específico a única mediana da árvore. Bhadury et al. provaram que a versão de decisão desse problema é NP-completo, mostrando a dificuldade de resolução de modelos integrados, mesmo no caso mais simples, onde somente uma facilidade deve ser localizada na rede.

Melkote e Daskin [36] observaram que, em várias aplicações, uma mudança na topologia da rede é freqüentemente mais efetiva do que adicionar facilidades com o objetivo de melhorar os níveis de serviço. Assim, investigaram o seguinte modelo integrado denominado de *problema de projeto de rede/localização de facilidades não-capacitadas* [36]: são dados um conjunto de vértices representando pontos de demanda e possíveis localizações de facilidades e um conjunto de arcos não capacitados. Cada vértice está associado a um custo fixo para construção de uma facilidade não-capacitada e cada arco possui um custo fixo de construção e um custo variável de transporte por unidade. O problema consiste em projetar a rede de transporte e determinar o conjunto de localizações de facilidades que minimiza o custo total do sistema: custo de construção dos arcos, custo de construção das facilidades e os custos de transporte nos arcos. O termo "não-capacitadas" se refere às facilidades que podem servir uma quantidade ilimitada de demanda. Melkote e Daskin [37] trabalharam em um problema semelhante ao anterior, porém as facilidades possuem limites na quantidade de demanda que podem servir, isto é, as facilidades são consideradas capacitadas. Em ambos os artigos são apresentadas formulações de programação inteira mista para os problemas e resultados computacionais para diversas instâncias geradas de forma aleatória utilizando-se um resolvedor de programação inteira.

Ravi e Sinha [45] investigaram três problemas que integram carac-

terísticas de projeto de rede e localização de facilidades. O primeiro problema foi denominado *problema de localização de facilidades com capacidade nos cabos*: são dados um grafo com custos nas arestas, um conjunto de potenciais facilidades com custo fixo para a sua abertura, um conjunto de clientes com demanda unitária e a capacidade dos cabos. Deseja-se abrir facilidades e construir uma rede de cabos tal que cada cliente seja servido por alguma facilidade aberta e que todas as capacidades dos cabos sejam obedecidas. O objetivo é minimizar os custos de abertura das facilidades e os custos de instalação dos cabos. O segundo problema denominado *problema das p-medianas com capacidade nos cabos* é uma variante do primeiro, onde não existe o custo para a abertura da facilidade. Contudo, no máximo p facilidades podem ser abertas. O terceiro problema, denominado *problema de localização de facilidades com k-cabos*, também é uma extensão do primeiro onde existem k tipos de cabos com custos fixos e capacidades diferentes. Os autores apresentaram algoritmos de aproximação para os três problemas estudados.

Swamy e Kumar [56] investigaram o *problema de localização de facilidades conectadas* (PLFC), definido da seguinte forma: seja um grafo com custos associados às arestas, um conjunto de possíveis localizações de facilidades e seus custos de instalação e um conjunto de clientes com demandas a serem servidas. Deseja-se instalar um subconjunto de facilidades, associando-se cada cliente às mesmas, com custo proporcional à demanda e à distância mínima entre cliente e facilidade. Adicionalmente, deseja-se conectar as facilidades abertas por uma árvore. O objetivo do problema é minimizar o custo total da solução, incluindo-se os custos de instalação das facilidades, os custos de associação das demandas às mesmas e os custos de conectar as facilidades abertas. Os autores trabalharam também com variantes do PLFC como o *problema das k-medianas conectadas*, que possui a restrição adicional de que no máximo k facilidades podem ser abertas, e o problema *alugue-ou-compre*, que não possui custos para instalar as facilidades e todos os vértices da rede são localizações candidatas para a abertura das facilidades. Os autores apresentaram formulações de programação linear inteira baseadas em cortes e algoritmos de aproximação para esses problemas e outras variantes. O PLFC é semelhante ao *problema de localização de facilidades não capacitadas conectadas*, estudado nessa tese. No PLFC, os clientes possuem demandas a serem servidas e os conjuntos de potenciais localizações de facilidades e de usuários podem estar contidos ou não no conjunto de vértices do grafo.

Gupta et al. [23] trabalharam também com o PLFC e com uma redução de um dos problemas estudados ao PLFC em [21]. Gupta et al. [22] investigaram o problema alugue-ou-compre e o reduziram ao PLFC. Khuller e Zhu estudaram um caso especial do PLFC em [30]. Guha e Khuller [20] trabalharam com o *problema do conjunto dominante conectado*. O *problema do conjunto domi-*

nante é definido por: dado um grafo, deseja-se encontrar o menor subconjunto de vértices tal que cada nó não pertencente ao subconjunto seja adjacente a pelo menos um vértice do mesmo. Adicionando-se a restrição de que o subgrafo induzido pelos vértices do subconjunto seja conexo, tem-se o *problema do conjunto dominante conectado*. A abordagem utilizada na resolução dos problemas anteriores [20, 21, 22, 23, 30] foi a técnica de algoritmos de aproximação.

Labbé et al. [32] investigaram o *problema do ciclo mediano*. São dados um grafo completo, com arestas com pesos representando custos de roteamento e arcos com pesos representando custos de associação (laços são permitidos). Assume-se o número de vértices do grafo superior a seis e fixa-se um vértice qualquer (depósito). Uma solução para o problema é um ciclo passando por um subconjunto de vértices do grafo incluindo-se o depósito e pelo menos dois outros vértices. O custo de roteamento de uma solução é a soma dos custos de roteamento das arestas pertencentes ao ciclo. Vértices presentes no ciclo estão associados a si próprios e vértices não pertencentes ao ciclo estão associados aos vértices mais próximos presentes no ciclo. Assim, o custo de associação de uma solução é a soma dos custos de associação de cada vértice que não está no ciclo a vértices pertencentes ao ciclo. Os autores apresentaram formulações de programação inteira para duas versões do problema e resultados computacionais para diversas instâncias através de um algoritmo exato do tipo *branch and cut*. Na primeira versão, minimiza-se a soma dos custos de roteamento e associação e na segunda versão, minimiza-se o custo de roteamento, sujeito a um limite superior no custo de associação.

Duas linhas de pesquisa também relacionadas a modelos integrados são [36]: problemas de localização-roteamento, onde considera-se a localização simultânea das facilidades e a determinação de rotas ou caminhos de coleta/entrega de algum serviço ou mercadoria e problemas de localização de concentradores, onde deseja-se localizar os concentradores em uma determinada rede, conectá-los e alocar os pontos de demanda aos mesmos. Para uma revisão da literatura de localização-roteamento e de localização de concentradores, ver os artigos de Min et al. [39] e de Campbell [9], respectivamente.

1.2 Objetivos da Tese

A integração de modelos de projeto de redes com os de localização de facilidades é um tema recente de pesquisa, como pôde ser observado na revisão bibliográfica. As técnicas adotadas para a resolução destes problemas utilizadas na literatura são principalmente as abordagens exata [32, 36, 37, 54] e por algoritmos de aproximação [20, 21, 22, 23, 30, 45, 56]. Somente uma única abordagem heurística foi utilizada [54]. Este será o enfoque dado nessa tese.

O problema das p-medianas conectadas e o problema de localização de

facilidades não-capacitadas conectadas unem problemas NP-árduos clássicos de otimização combinatória bem estudados na literatura. Assim, os mesmos são também NP-árduos, sendo difíceis de serem resolvidos de forma exata. Além disso, muitas vezes, os métodos exatos nem sempre conseguem obter soluções ótimas em tempo hábil para problemas reais ou de grande porte. Por isso, uma abordagem heurística torna-se necessária. O objetivo desta tese é estudar o problema das p-medianas conectadas utilizando-se da abordagem heurística como ferramenta para sua solução. O termo heurísticas se refere, de uma forma geral, a algoritmos construtivos, de busca local e metaheurísticas.

O Capítulo 2 apresentará a definição do problema das p-medianas conectadas e do problema de localização de facilidades não-capacitadas conectadas. Duas formulações de programação linear inteira serão apresentadas para o primeiro problema e uma delas será estendida para o segundo. Também, comparações entre as duas formulações propostas para o primeiro problema serão realizadas em algumas instâncias, utilizando-se um resolvedor de programação linear inteira.

O Capítulo 3 descreverá o algoritmo de busca local para o problema das p-medianas conectadas, baseando-se em idéias tais como melhoria iterativa, circularidade, concatenação de buscas e descarte de vizinhos com o objetivo de acelerar os passos da busca local.

Os Capítulos 4 e 5 apresentarão, respectivamente, heurísticas GRASP *(Greedy Randomized Adaptive Search Procedures)* e VNS *(Variable Neighborhood Search)* para o problema das p-medianas conectadas. Ambas utilizarão uma estratégia de filtro com o objetivo de diminuir os tempos de processamento e o procedimento de reconexão por caminhos com o objetivo de melhorar a qualidade das soluções.

O Capítulo 6 mostrará as comparações entre as heurísticas GRASP e VNS para alguns problemas testes, em termos de qualidade das soluções encontradas e dos tempos de processamento obtidos.

Por último, o Capítulo 7 mostrará as conclusões da tese e possíveis extensões como trabalhos futuros.

2
Formulações Matemáticas para os Problemas Híbridos de Projeto de Redes e de Localização de Facilidades

Este capítulo propõe formulações matemáticas de programação linear inteira para os problemas híbridos de projeto de redes e de localização de facilidades, mostrando dois modelos para o problema das p-medianas conectadas e um modelo para o problema de localização de facilidades não-capacitadas conectadas.

Primeiramente, serão apresentadas as definições e formulações matemáticas de três problemas clássicos. A Seção 2.1 define o problema de Steiner em grafos e apresenta duas formulações para o mesmo: um modelo baseado no problema de fluxo em redes com várias comodidades [43, 64] e um modelo baseado no problema da árvore geradora mínima restrita por grau [6, 43]. As Seções 2.2 e 2.3 apresentam, respectivamente, as definições do problema das p-medianas e do problema de localização de facilidades não-capacitadas, assim como suas respectivas formulações clássicas.

Em seguida, serão mostradas as definições e formulações propostas para os problemas híbridos. A Seção 2.4 define o problema das p-medianas conectadas e apresenta dois modelos: o primeiro une a formulação por fluxos do problema de Steiner em grafos com a formulação de p-medianas e o segundo une a formulação por árvore do problema de Steiner em grafos com a formulação de p-medianas. A Seção 2.5 define o problema de localização de facilidades não-capacitadas conectadas e apresenta um modelo que une a formulação por árvore do problema de Steiner em grafos com a formulação clássica do problema de localização de facilidades não-capacitadas.

Na Seção 2.6, comparações entre as duas formulações propostas para o problema das p-medianas conectadas serão realizadas em instâncias testes, utilizando-se um resolvedor de programação linear inteira com o intuito de apontar a melhor formulação para o problema nas instâncias testadas. Por último, na Seção 2.7, conclusões e considerações finais serão apresentadas sobre as formulações propostas.

2.1 Problema de Steiner em Grafos

O problema de Steiner em grafos pode ser definido como se segue [61]: dados um grafo não-orientado $G = (V, E)$, com o conjunto V de índices dos vértices do grafo, arestas E, custos positivos $c_{ij} > 0$ associados às arestas e o conjunto $T \subseteq V$ de índices dos vértices terminais, encontrar um subgrafo conexo de G de custo mínimo que contenha todos os vértices de T. Como os custos das arestas são positivos, a solução ótima será sempre uma árvore, denominada árvore de Steiner mínima.

Esse problema é NP-árduo [28], sendo bastante estudado na literatura [61, 63] e possui diversas formulações de programação inteira (mista) [43]. Os modelos descritos a seguir são baseados no problema de fluxos em redes com várias comodidades e no problema da árvore geradora mínima restrita por grau.

2.1.1 Formulação Baseada em Fluxos

O problema de fluxos em uma rede consiste no envio de uma comodidade (mercadoria, produto) de um nó origem para um nó destino, otimizando-se algum critério e obedecendo-se restrições de capacidade. Contudo, existem aplicações que necessitam que várias comodidades, muitas vezes com restrições de fluxo diferentes, compartilhem a mesma rede. Os problemas em que as comodidades compartilham os mesmos recursos da rede são denominados de problemas de fluxos em redes com várias comodidades ou multi-comodidades. Assim, o fluxo correspondente a cada par origem-destino é visualizado como uma diferente comodidade.

Seja um grafo orientado $G = (V, A)$ formado a partir do grafo $G = (V, E)$, substituindo-se cada aresta $(i, j) \in E$ por dois arcos $[i, j], [j, i] \in A$, tal que $c_{ij} = c_{ji}$. Suponha, sem perda de generalidade, que o vértice indexado por 1 é um vértice terminal. Seja também $T_1 = T \setminus \{1\}$ o conjunto dos demais vértices terminais. No problema de Steiner em um grafo orientado, deseja-se encontrar uma sub-árvore de custo mínimo que contenha um caminho direcionado entre o vértice terminal 1 e cada vértice terminal $k \in T_1$. O custo total da árvore é a soma dos custos dos arcos que dela fazem parte.

Nesse modelo, cada caminho a partir do vértice terminal 1 corresponde a um fluxo de uma comodidade entre os vértices terminais 1 e k, resultando assim, em $|T| - 1$ comodidades diferentes. Desta maneira, representa-se a conectividade do problema de Steiner em grafos via um problema de fluxos em uma rede com várias comodidades [64]. Seja w_{ij}^k a quantidade de fluxo da comodidade k passando pelo arco $[i, j] \in A$ tendo como origem o vértice terminal 1 e como destino qualquer vértice terminal $k \in T_1$.

O problema de Steiner em um grafo orientado modelado por programação inteira 0-1 utiliza a seguinte variável de decisão [64]:

$$z_{ij} = \begin{cases} 1, & \text{se o arco } [i,j] \in A \text{ é incluído na solução,} \\ 0, & \text{caso contrário,} \end{cases}$$

e pode ser expresso como segue:

$$Min \sum_{[i,j]\in A} c_{ij} z_{ij} \tag{2-1}$$

sujeito a:

$$\sum_{[i,j]\in A} w_{ij}^k - \sum_{[j,i]\in A} w_{ji}^k = \left\{ \begin{array}{ll} 1, & \text{se } i = 1 \\ -1, & \text{se } i = k \\ 0, & \text{se } i \neq 1, k \end{array} \right\} \forall\ i \in V, \forall\ k \in T_1 \tag{2-2}$$

$$w_{ij}^k \leq z_{ij}, \forall\ [i,j] \in A, \forall\ k \in T_1 \tag{2-3}$$

$$w_{ij}^k \geq 0, \forall\ [i,j] \in A, \forall\ k \in T_1 \tag{2-4}$$

$$z_{ij} \in \{0,1\}, \forall\ [i,j] \in A \tag{2-5}$$

A função objetivo (2-1) minimiza a soma dos custos nos arcos que compõem a árvore. As restrições (2-2) são as equações de conservação de fluxo, indicando que uma unidade de fluxo originária do vértice terminal 1 deve ser enviada até o vértice terminal k, passando, se necessário, por outros vértices. Os primeiro e segundo termos do lado esquerdo da equação representam, respectivamente, a soma dos fluxos que saem e entram no vértice $i \in V$. A diferença entre a soma dos fluxos que saem e entram no vértice 1 (nó fonte) deve ser igual a 1. A diferença entre a soma dos fluxos que saem e entram no vértice k (nó consumidor) deve ser igual a -1. Para qualquer outro vértice, a diferença entre a soma dos fluxos que saem e entram no vértice (nó de transbordo) deve ser igual a 0. As restrições (2-3) indicam que o fluxo direcionado entre os vértices terminais 1 e k passando por um arco $[i,j]$ é permitido somente se este arco for incluído na solução. As restrições (2-4) impõem que o fluxo direcionado entre os vértices terminais 1 e k passando por um arco $[i,j]$ seja não-negativo. As restrições (2-5) estabelecem a variável de decisão z_{ij} como binária.

2.1.2
Formulação Baseada em Árvore Geradora Mínima Restrita por Grau

Segundo a formulação baseada em árvore, o problema de Steiner em grafos equivale a encontrar uma árvore geradora mínima restrita por grau

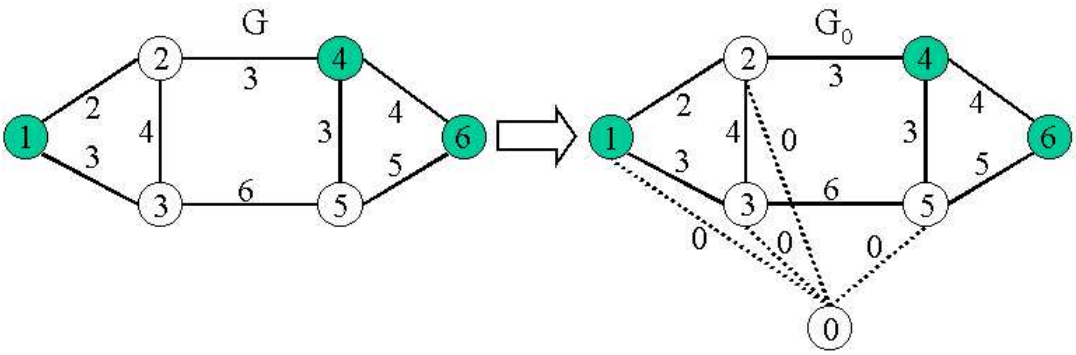

Figura 2.1: Grafo original $G = (V, E)$ e grafo modificado $G_0 = (V_0, E_0)$.

em um grafo modificado $G_0 = (V_0, E_0)$, produzido adicionando-se ao grafo original $G = (V, E)$ um vértice artificial indexado por 0 e conectando-o através de arestas de custo zero a todos os vértices no conjunto $V \setminus T$ e a um vértice terminal qualquer indexado por 1 [6, 43]. A Figura 2.1 mostra um exemplo de um grafo original G e de um grafo modificado G_0 com custos positivos associados às arestas, onde $T = \{1, 4, 6\}$ são os vértices terminais e $V \setminus T = \{2, 3, 5\}$ são os demais vértices do grafo.

A restrição por grau impõe que na árvore geradora mínima, cada vértice do conjunto $V \setminus T$ adjacente ao vértice artificial 0 deve possuir grau um. Assim, a solução ótima para o problema da árvore geradora mínima restrita por grau da Figura 2.1 é apresentada na Figura 2.2. A sub-árvore conectada ao vértice terminal 1 será a árvore de Steiner, isto é, a solução ótima para o problema de Steiner no grafo original G, representada na figura pelos vértices e arestas dentro do retângulo pontilhado. Os vértices que não estão na árvore estão diretamente conectados ao vértice artificial 0 por arestas de custo zero [6].

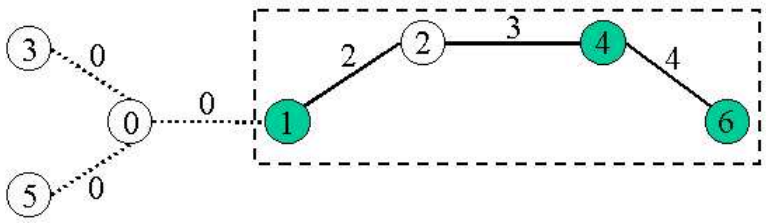

Figura 2.2: Solução ótima para o problema da árvore geradora mínima restrita por grau no grafo G_0.

Define-se δ_i como o conjunto de arestas do grafo original G que possuem uma extremidade no vértice i, isto é, $\delta_i = [(i, j) \mid (i, j) \in E] \cup [(j, i) \mid (j, i) \in E]$ [43]. Como exemplo, seja o grafo original $G = (V, E)$ mostrado na Figura 2.1. Para o vértice 2, tem-se $\delta_2 = \{(1, 2), (2, 3), (2, 4)\}$.

O problema de Steiner em um grafo não-orientado modelado por programação inteira 0-1 utiliza a seguinte variável de decisão [6]:

$$z_{ij} = \begin{cases} 1, & \text{se a aresta } (i,j) \in E_0 \text{ é incluída na solução,} \\ 0, & \text{caso contrário,} \end{cases}$$

e pode ser expresso como segue:

$$Min \sum_{(i,j)\in E_0} c_{ij} z_{ij} \tag{2-6}$$

sujeito a:

$$\sum_{(i,j)\in E_0} z_{ij} = |V_0| - 1 \tag{2-7}$$

$$\sum_{(i,j)\in E_0; i,j\in W} z_{ij} \leq |W| - 1, \forall\ W \neq \emptyset, W \subset V_0 \tag{2-8}$$

$$z_{0i} + z_{ij} \leq 1, \forall\ i \in V \setminus T, \forall\ (i,j) \in \delta_i \tag{2-9}$$

$$z_{ij} \in \{0,1\}, \forall\ (i,j) \in E_0 \tag{2-10}$$

A função objetivo (2-6) minimiza a soma dos custos das arestas que compõem a árvore. As restrições (2-7) e (2-8) asseguram que as arestas escolhidas formam uma árvore geradora no grafo modificado G_0. As restrições (2-7) relacionam-se à quantidade de arestas utilizadas e as restrições (2-8) são as de eliminação de sub-rotas, onde W representa um subconjunto qualquer de vértices do conjunto V_0. As restrições (2-9) asseguram que se a aresta $(0,i)$ for usada, com $i \in V \setminus T$, nenhuma outra aresta do grafo original G que possui uma extremidade no vértice i pode ser utilizada. Assim, garante-se que, na solução ótima do grafo modificado, o vértice i esteja conectado somente ao vértice artificial 0, não pertencendo, assim, à árvore de Steiner conectada ao vértice terminal 1. As restrições (2-10) estabelecem a variável de decisão z_{ij} como binária.

2.2 Problema das p-Medianas

O problema das p-medianas pode ser definido como se segue [24, 25]: dados o conjunto F de índices das m potenciais localizações de facilidades (medianas), o conjunto U de índices dos n usuários (clientes), os custos d_{ij} de servir cada usuário i a partir de cada potencial localização de facilidade $j, \forall\ i \in U, \forall\ j \in F$ e uma constante $p \leq m$, determine quais p facilidades instalar com o objetivo de minimizar a soma dos custos de servir cada usuário com a facilidade aberta mais próxima. O problema das p-medianas é NP-árduo [29] e é um dos mais conhecidos problemas de localização.

O problema das p-medianas modelado por programação inteira 0-1 utiliza as seguintes variáveis de decisão [50]:

$$x_{ij} = \begin{cases} 1, & \text{se o usuário } i \in U \text{ está associado à facilidade } j \in F, \\ 0, & \text{caso contrário,} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{se há uma facilidade localizada em } j \in F, \\ 0, & \text{caso contrário,} \end{cases}$$

e pode ser expresso como segue:

$$Min \sum_{i \in U} \sum_{j \in F} d_{ij} x_{ij} \tag{2-11}$$

sujeito a:

$$\sum_{j \in F} x_{ij} = 1, \forall\ i \in U \tag{2-12}$$

$$x_{ij} \leq y_j, \forall\ i \in U, \forall\ j \in F \tag{2-13}$$

$$\sum_{j \in F} y_j = p \tag{2-14}$$

$$x_{ij}, y_j \in \{0, 1\}, \forall\ i \in U, \forall\ j \in F \tag{2-15}$$

A função objetivo (2-11) minimiza a soma dos custos de servir cada usuário com a facilidade aberta mais próxima. As restrições (2-12) expressam que cada usuário será designado para exatamente uma única mediana. As restrições (2-13) previnem que um usuário seja designado para uma potencial localização de facilidade que não esteja selecionada como mediana. A restrição (2-14) força que o número total de facilidades abertas seja igual a p. As restrições (2-15) estabelecem as variáveis de decisão x_{ij} e y_j como binárias.

2.3 Problema de Localização de Facilidades Não-Capacitadas

O problema de localização de facilidades não-capacitadas é outro conhecido problema de localização, que difere do problema das p-medianas em dois aspectos: existe um custo associado com a instalação de uma facilidade em uma potencial localização e não existe um limite superior no número de facilidades que podem ser abertas.

O problema pode, assim, ser definido [31]: dados o conjunto F de índices das m potenciais localizações de facilidades, o conjunto U de índices dos n usuários, os custos d_{ij} de servir cada usuário i a partir de cada potencial localização de facilidade $j, \forall\ i \in U, \forall\ j \in F$ e os custos de instalação c_j de uma facilidade em uma potencial localização $j, \forall\ j \in F$, determine o subconjunto de facilidades a serem abertas, minimizando a soma dos custos de servir cada usuário com a facilidade aberta mais próxima com a soma

dos custos de instalação das facilidades. O termo "não-capacitadas" se refere às facilidades, que podem servir uma quantidade ilimitada de usuários. Esse problema também é NP-árduo [12].

O problema de localização de facilidades não-capacitadas modelado por programação inteira 0-1 utiliza as seguintes variáveis de decisão [31]:

$$x_{ij} = \begin{cases} 1, & \text{se o usuário } i \in U \text{ está associado à facilidade } j \in F, \\ 0, & \text{caso contrário,} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{se há uma facilidade localizada em } j \in F, \\ 0, & \text{caso contrário,} \end{cases}$$

e pode ser expresso como segue:

$$Min \sum_{j \in F} c_j y_j + \sum_{i \in U} \sum_{j \in F} d_{ij} x_{ij} \tag{2-16}$$

sujeito a:

$$\sum_{j \in F} x_{ij} = 1, \forall\ i \in U \tag{2-17}$$

$$x_{ij} \leq y_j, \forall\ i \in U, \forall\ j \in F \tag{2-18}$$

$$x_{ij}, y_j \in \{0, 1\}, \forall\ i \in U, \forall\ j \in F \tag{2-19}$$

A função objetivo (2-16) minimiza a soma dos custos de instalação das facilidades com a soma dos custos de servir cada usuário com a facilidade aberta mais próxima. As restrições (2-17) expressam que cada usuário será designado para exatamente uma única mediana. As restrições (2-18) previnem que um usuário seja designado para uma potencial localização de facilidade que não esteja selecionada como mediana. As restrições (2-19) estabelecem as variáveis de decisão x_{ij} e y_j como binárias.

2.4 Problema Híbrido das p-Medianas Conectadas

O problema das p-medianas conectadas pode ser definido como se segue. Seja um grafo não-orientado $G = (V, E)$, com o conjunto $V = \{1, 2, \ldots, n\}$ de índices dos vértices, o conjunto de arestas E e custos de instalação positivos c_{ij} de uma aresta $(i, j) \in E$. Considera-se também o mesmo conjunto V de índices das n potenciais localizações de facilidades (não-capacitadas), o mesmo conjunto V de índices dos n usuários ou clientes (com demanda unitária), os valores d_{ij} representando o custo de servir cada usuário i a partir de cada potencial localização de facilidade $j, \forall\ i, j \in V$ e uma constante $p \leq n$. Assim, cada vértice é simultaneamente um usuário e uma potencial

localização de facilidade. Deseja-se instalar p facilidades nos vértices do grafo, associando-se cada usuário com a facilidade aberta mais próxima, e conectar as p facilidades abertas por uma árvore. O objetivo do problema é minimizar o custo total de conectar as facilidades abertas e de servir os usuários. A árvore conecta todas as facilidades abertas, mas também inclui outros vértices que não foram selecionados como medianas (vértices de Steiner). Esses vértices são considerados usuários e, portanto, devem ser atendidos por uma das p facilidades abertas.

Retirando-se as restrições que conectam as p medianas instaladas através de uma árvore de Steiner, tem-se simplesmente o clássico problema das p-medianas, sendo, esse último então um caso especial do problema híbrido na qual as restrições de conectividade foram eliminadas. Como o problema das p-medianas é NP-árduo [29], também o é o problema das p-medianas conectadas, exceto para os casos em que $p = 1$ ou $p = n$. Quando $p = 1$, o problema híbrido torna-se o de encontrar a única mediana do grafo, pois o vértice escolhido como mediana não se conectará a nenhum outro e servirá aos clientes em todos os nós do grafo. Quando $p = n$, tem-se o problema da árvore geradora mínima, pois todos os usuários serão servidos por facilidades instaladas nos próprios vértices e uma árvore de custo mínimo conectará todos os nós do grafo.

Serão propostas duas formulações para o problema das p-medianas conectadas. A primeira une a formulação de Steiner baseada em fluxos com a formulação clássica das p-medianas. A segunda une a formulação de Steiner baseada em árvore geradora mínima restrita por grau com a formulação das p-medianas.

2.4.1 Formulação Baseada em Fluxos

Na formulação baseada em fluxos para o problema de Steiner em grafos, sabe-se, antecipadamente, qual vértice será o nó fonte (nó terminal 1) e quais vértices serão os nós consumidores (demais vértices terminais), pois o conjunto T é uma entrada para o problema. Assim, $|T| - 1$ fluxos unitários de comodidades diferentes originários do vértice terminal 1 serão consumidos pelos demais vértices terminais, passando, se necessário, pelos vértices de Steiner pertencentes ao conjunto $V \backslash T$. As restrições do modelo garantem encontrar um subgrafo conexo, mais precisamente uma árvore, e a função objetivo do problema garante que a árvore seja de custo mínimo.

A idéia da formulação baseada em fluxos para o problema das p-medianas conectadas é similar à descrita no parágrafo anterior. Porém, não se sabe antecipadamente onde localizar as p facilidades nos vértices do grafo. Associa-se, então, uma comodidade k a cada mediana a ser instalada e cria-se um vértice artificial 0 que funcionará como um nó fonte enviando fluxos unitários

de p comodidades diferentes para as medianas a serem instaladas, passando, se necessário, por vértices intermediários. O modelo garante um caminho mínimo direcionado entre o vértice artificial 0 e cada uma das p medianas instaladas, com os usuários servidos pela facilidade aberta mais próxima.

Seja um grafo orientado $G_0^{'} = (V_0^{'}, A_0^{'})$ formado a partir do grafo $G = (V, E)$, substituindo-se cada aresta $(i, j) \in E$ por dois arcos $[i, j], [j, i] \in A_0^{'}$, tal que $c_{ij} = c_{ji}$. Adiciona-se também ao grafo $G_0^{'}$ um vértice artificial indexado por 0 e arcos $[0, j], \forall\ j \in V$ com custos de instalação zero entre o vértice 0 e cada um dos vértices em V. Sejam o conjunto $K = \{1, 2, \ldots, p\}$ de índices da quantidade de medianas a serem instaladas e w_{ij}^k a quantidade de fluxo da comodidade $k \in K$ passando através do arco $[i, j] \in A_0^{'}$.

O problema das p-medianas conectadas pode ser modelado com as seguintes variáveis de decisão

$$x_{ij} = \begin{cases} 1, & \text{se o usuário } i \in V \text{ está associado à facilidade } j \in V, \\ 0, & \text{caso contrário,} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{se há uma facilidade localizada em } j \in V, \\ 0, & \text{caso contrário,} \end{cases}$$

$$f_j^k = \begin{cases} 1, & \text{se há uma facilidade localizada em } j \in V \text{ e associada à} \\ & \text{comodidade } k \in K, \\ 0, & \text{caso contrário,} \end{cases}$$

$$z_{ij} = \begin{cases} 1, & \text{se o arco } [i, j] \in A_0^{'} \text{ é incluído na solução,} \\ 0, & \text{caso contrário,} \end{cases}$$

e pode ser expresso como segue:

$$Min \sum_{i \in V} \sum_{j \in V} d_{ij} x_{ij} + \sum_{[i,j] \in A_0^{'}} c_{ij} z_{ij} \tag{2-20}$$

sujeito a:

$$\sum_{j \in V} x_{ij} = 1, \forall\ i \in V \tag{2-21}$$

$$x_{ij} \leq y_j, \forall\ i, j \in V \tag{2-22}$$

$$\sum_{j \in V} f_j^k = 1, \forall\ k \in K \tag{2-23}$$

$$y_j = \sum_{k \in K} f_j^k, \forall\ j \in V \tag{2-24}$$

$$\sum_{[i,j]\in A_0^{'}} w_{ij}^k - \sum_{[j,i]\in A_0^{'}} w_{ji}^k = \left\{ \begin{array}{ll} 1, & \text{se } j = 0 \\ -f_j^k, & \text{se } j \neq 0 \end{array} \right\} \forall\ j \in V_0^{'}, \forall\ k \in K \qquad (2\text{-}25)$$

$$\sum_{[0,j]\in A_0^{'}} z_{0j} = 1 \qquad (2\text{-}26)$$

$$w_{ij}^k \leq z_{ij}, \forall\ [i,j] \in A_0^{'}, \forall\ k \in K \qquad (2\text{-}27)$$

$$w_{ij}^k \geq 0, \forall\ [i,j] \in A_0^{'}, \forall\ k \in K \qquad (2\text{-}28)$$

$$x_{ij}, y_j, f_j^k \in \{0,1\}, \forall\ i,j \in V, \forall\ k \in K \qquad (2\text{-}29)$$

$$z_{ij} \in \{0,1\}, \forall\ [i,j] \in A_0^{'} \qquad (2\text{-}30)$$

A função objetivo (2-20) minimiza o custo de servir cada usuário com a facilidade aberta mais próxima e o custo de conectar as facilidades abertas. As restrições (2-21) expressam que cada usuário será designado para exatamente uma única mediana. As restrições (2-22) previnem que um usuário seja designado para uma potencial localização de facilidade que não esteja selecionada como mediana. As restrições (2-23) mostram que cada comodidade k deve ser associada a uma única facilidade localizada em j, indicando que não se pode associar facilidades diferentes à mesma comodidade. As restrições (2-24) asseguram que, se uma facilidade está localizada em j ($y_j = 1$), então exatamente uma comodidade k deve ser associada a j. As restrições (2-25) são as equações de conservação de fluxo para cada comodidade k. Os primeiro e segundo termos do lado esquerdo da equação representam, respectivamente, a soma dos fluxos que entram e saem no vértice $j \in V_0^{'}$. A diferença entre a soma dos fluxos que entram e saem no vértice artificial 0 deve ser igual a 1. Para outro vértice qualquer, a diferença entre a soma dos fluxos que entram e saem no vértice deve ser igual a $-f_j^k$. O modelo determinará, para cada comodidade k, o nó consumidor (caso em que $f_j^k = 1$) e os nós intermediários (casos em que $f_j^k = 0$). A restrição (2-26) assegura que somente um arco $[0,j] \in A_0^{'}$ pode ser utilizado. Assim, o vértice j do grafo original associado ao arco é responsável por distribuir o fluxo das diferentes comodidades para as medianas instaladas. Na solução ótima, o vértice j será sempre uma facilidade aberta devido à minimização ocorrida na função objetivo. As restrições (2-27) indicam que o fluxo passando por um arco $[i,j]$ associado a comodidade k é permitido somente se este arco for incluído na solução. As restrições (2-28) impõem que o fluxo passando por um arco $[i,j]$ associado a comodidade k seja não-negativo. As restrições (2-29) e (2-30) estabelecem as variáveis de decisão x_{ij}, y_j, f_j^k e z_{ij} como binárias.

2.4.2
Formulação Baseada em Árvore Geradora Mínima Restrita por Grau

Para se modelar o problema de Steiner em grafos através da formulação baseada em árvore, cria-se um grafo modificado adicionando-se ao grafo original um vértice artificial 0 e conectando-o através de arestas com custos zero a um vértice terminal qualquer (terminal 1) e aos vértices no conjunto $V\backslash T$. Resolvendo-se o problema da árvore geradora mínima com a restrição de que os vértices do conjunto $V\backslash T$ adjacentes ao vértice artifical 0 deve possuir grau um, tem-se a solução para o problema de Steiner no grafo original (árvore conectada ao vértice terminal 1). Nessa formulação, sabe-se, antecipadamente, onde conectar o vértice artifical 0 aos vértices do grafo original, pois o conjunto de nós terminais T é uma entrada para o problema.

A idéia da formulação baseada em árvore geradora mínima restrita por grau para o problema das p-medianas conectadas é similar à descrita no parágrafo anterior. Contudo, não se sabe antecipadamente onde localizar as p facilidades nos vértices do grafo. Criou-se, então, dois vértices artificiais 0 e $n+1$.

O vértice 0 ficará responsável por se conectar a uma das medianas a ser instalada. Em uma analogia com o problema de Steiner em grafos, essa ligação corresponde a conexão entre os vértices 0 (artificial) e 1 (terminal).

O vértice artificial $n+1$ ficará responsável pela restrição por grau. No problema de Steiner em grafos, os vértices terminais, com exceção do vértice 1, não estão ligados ao vértice artificial 0. Da mesma maneira, para o problema das p-medianas conectadas, ao se instalar uma mediana em um determinado vértice, este não ficará ligado ao vértice $n+1$. No problema de Steiner em grafos, cada vértice no conjunto $V\backslash T$ está ligado ao vértice 0 e permanecerá conectado ao mesmo com grau um caso não esteja na árvore de Steiner ótima. Similarmente, no problema das $p-$medianas conectadas, os vértices que não possuem facilidades instaladas estarão ligados ao vértice $n+1$ se os mesmos não estiverem na árvore de Steiner encontrada pela solução ótima. A árvore de Steiner formada pelas p medianas instaladas estará ligada ao vértice artificial 0, com os usuários sendo servidos pela facilidade aberta mais próxima.

Considere o grafo $G_0'' = (V_0'', E_0'')$ formado adicionando-se ao grafo $G = (V, E)$ dois vértices artificiais indexados por 0 e $n+1$ e arestas $(0, j)$ e $(n+1, j), \forall\ j \in V$, entre os vértices artificiais e cada um dos vértices de V com custos de instalação zero. Além disso, adiciona-se uma aresta $(0, n+1)$ entre os vértices artificiais também com custo de instalação zero. Define-se δ_j como o conjunto de arestas do grafo original $G = (V, E)$ que possuem uma extremidade no vértice j, isto é, $\delta_j = [(i, j) \mid (i, j) \in E] \cup [(j, i) \mid (j, i) \in E]$.

O problema das p-medianas conectadas pode ser modelado por programação inteira 0-1 com as seguintes variáveis de decisão:

$$x_{ij} = \begin{cases} 1, & \text{se o usuário } i \in V \text{ está associado à facilidade } j \in V, \\ 0, & \text{caso contrário,} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{se há uma facilidade localizada em } j \in V, \\ 0, & \text{caso contrário,} \end{cases}$$

$$z_{ij} = \begin{cases} 1, & \text{se a aresta } (i,j) \in E_0^{''} \text{ é incluída na solução,} \\ 0, & \text{caso contrário,} \end{cases}$$

e pode ser expresso como segue:

$$Min \sum_{i \in V} \sum_{j \in V} d_{ij} x_{ij} + \sum_{(i,j) \in E_0^{''}} c_{ij} z_{ij} \tag{2-31}$$

sujeito a:

$$\sum_{j \in V} x_{ij} = 1, \forall\ i \in V \tag{2-32}$$

$$x_{ij} \leq y_j, \forall\ i,j \in V \tag{2-33}$$

$$\sum_{j \in V} y_j = p \tag{2-34}$$

$$\sum_{(0,j) \in E_0^{''}} z_{0j} = 1 \tag{2-35}$$

$$z_{0j} \leq y_j, \forall\ j \in V \tag{2-36}$$

$$\sum_{(i,j) \in E_0^{''}} z_{ij} = |V_0^{''}| - 1 \tag{2-37}$$

$$\sum_{(i,j) \in E_0^{''}; i,j \in W} z_{ij} \leq |W| - 1, \forall\ W \neq \emptyset, W \subset V_0^{''} \tag{2-38}$$

$$z_{(n+1)j} + y_j \leq 1, \forall\ j \in V \tag{2-39}$$

$$z_{(n+1)j} + z_{ij} \leq 1, \forall\ j \in V, \forall\ (i,j) \in \delta_j \tag{2-40}$$

$$x_{ij}, y_j \in \{0,1\}, \forall\ i,j \in V \tag{2-41}$$

$$z_{ij} \in \{0,1\}, \forall\ (i,j) \in E_0^{''} \tag{2-42}$$

A função objetivo (2-31) minimiza o custo de servir cada usuário com a facilidade aberta mais próxima e o custo de conectar as facilidades abertas. As restrições (2-32) expressam que cada usuário será designado para exatamente uma única mediana. As restrições (2-33) previnem que um usuário seja designado para uma potencial localização de facilidade que não esteja selecionada como mediana. A restrição (2-34) garante que p medianas serão instaladas nos

vértices do grafo. A restrição (2-35) assegura que uma única aresta $(0, j)$ será utilizada. As restrições (2-36) garantem que não se pode instalar a aresta $(0, j)$ se a potencial localização j não esteja selecionada como mediana. A restrição (2-35) e as restrições (2-36) asseguram a conexão entre o vértice artificial 0 e uma das p medianas instaladas. As restrições (2-37) e (2-38) asseguram que as arestas escolhidas formam uma árvore geradora no grafo G_0''. As restrições (2-37) relacionam-se à quantidade de arestas utilizadas e as restrições (2-38) são as de eliminação de sub-rotas, onde W representa um subconjunto qualquer de vértices do conjunto V_0''. As restrições (2-39) asseguram que, se uma mediana for instalada no vértice j, isto é, se $y_j = 1$, então a aresta $(n + 1, j)$ não será utilizada. As restrições (2-40) asseguram que, caso não ocorra a instalação de uma mediana no vértice j, isto é, se $y_j = 0$ e a aresta $(n + 1, j)$ for utilizada, então nenhuma outra aresta do grafo original G que possui uma extremidade no vértice j pode ser utilizada. Assim, os vértices que não pertencem à árvore de Steiner estarão diretamente ligados ao vértice artificial $n + 1$. As restrições (2-41) e (2-42) estabelecem as variáveis de decisão x_{ij}, y_j e z_{ij} como binárias.

2.5 Problema Híbrido de Localização de Facilidades Não-Capacitadas Conectadas

O problema de localização de facilidades não-capacitadas conectadas difere do problema das p-medianas conectadas em dois aspectos: existe um custo associado na instalação de uma facilidade em uma potencial localização e não existe um limite superior no número de facilidades que podem ser abertas.

O problema de localização de facilidades não-capacitadas conectadas pode ser definido como se segue: seja um grafo não-orientado $G = (V, E)$ com o conjunto $V = \{1, 2, \ldots, n\}$ de índices dos vértices do grafo, arestas E e custos de instalação positivos c_{ij} de uma aresta $(i, j) \in E$. Considera-se também o mesmo conjunto V de índices das n potenciais localizações de facilidades (não-capacitadas), o mesmo conjunto V de índices dos n usuários (com demanda unitária), os valores d_{ij} representando o custo de servir cada usuário i a partir de cada potencial localização de facilidade $j, \forall\ i, j \in V$ e o custo de instalação c_j de uma facilidade no vértice $j, \forall\ j \in V$. Assim, cada vértice é simultaneamente um usuário e uma potencial localização de facilidade. Deseja-se instalar um conjunto de facilidades nos vértices do grafo, associando-se os usuários à facilidade aberta mais próxima, e conectar as facilidades abertas por uma árvore. O objetivo do problema é minimizar o custo total de conectar as facilidades abertas, o custo fixo de instalação das mesmas e o custo de servir os usuários. A árvore conecta todas as facilidades abertas, mas também inclui outros vértices que não foram selecionados como medianas (vértices de Steiner). Esses vértices são considerados usuários e, portanto, devem ser

atendidos por uma das facilidades abertas.

Retirando-se as restrições que conectam as medianas instaladas através de uma árvore de Steiner, tem-se simplesmente o clássico problema de localização de facilidades não-capacitadas, sendo, esse último então um caso especial do problema híbrido na qual as restrições de conectividade foram eliminadas. Como esse problema é NP-árduo [12], também o é o problema de localização de facilidades não-capacitadas conectadas.

Na próxima seção será apresentada uma formulação que une a formulação de Steiner baseada em árvore geradora mínima restrita por grau com a formulação clássica do problema de localização de facilidades não-capacitadas.

2.5.1
Formulação Baseada em Árvore Geradora Mínima Restrita por Grau

A formulação baseada em árvore para o problema das p-medianas conectadas pôde facilmente ser adaptada para o problema de localização de facilidades não-capacitadas conectadas, acrescentando-se na função objetivo do primeiro problema uma parcela referente à soma dos custos de instalação de uma facilidade em uma potencial localização j e retirando-se a restrição (2-34) que força a instalação de um número fixo de medianas.

Considere o grafo $G_0^* = (V_0^*, E_0^*)$ formado adicionando-se ao grafo $G = (V, E)$ dois vértices artificiais indexados por 0 e $n+1$ e arestas $(0, j)$, $(n+1, j), \forall\, j \in V$, entre os vértices artificiais e cada um dos vértices de V com custos de instalação zero. Além disso, adiciona-se uma aresta $(0, n+1)$ entre os vértices artificiais, também com custo de instalação zero.

O problema de localização de facilidades não-capacitadas conectadas modelado por programação inteira 0-1 utiliza as seguintes variáveis de decisão:

$$x_{ij} = \begin{cases} 1, & \text{se o usuário } i \in V \text{ está associado à facilidade } j \in V, \\ 0, & \text{caso contrário,} \end{cases}$$

$$y_j = \begin{cases} 1, & \text{se há uma facilidade localizada em } j \in V, \\ 0, & \text{caso contrário,} \end{cases}$$

$$z_{ij} = \begin{cases} 1, & \text{se a aresta } (i,j) \in E_0^* \text{ é incluída na solução,} \\ 0, & \text{caso contrário,} \end{cases}$$

e pode ser expresso como segue:

$$Min \sum_{j \in V} c_j y_j + \sum_{i \in V} \sum_{j \in V} d_{ij} x_{ij} \; + \sum_{(i,j) \in E_0^*} c_{ij} z_{ij} \tag{2-43}$$

sujeito a:

$$\sum_{j \in V} x_{ij} = 1, \forall\ i \in V \tag{2-44}$$

$$x_{ij} \leq y_j, \forall\ i, j \in V \tag{2-45}$$

$$\sum_{(0,j) \in E_0^*} z_{0j} = 1 \tag{2-46}$$

$$z_{0j} \leq y_j, \forall\ j \in V \tag{2-47}$$

$$\sum_{(i,j) \in E_0^*} z_{ij} = |V_0^*| - 1 \tag{2-48}$$

$$\sum_{(i,j) \in E_0^*; i,j \in W} z_{ij} \leq |W| - 1,\ \forall\ W \neq \emptyset, W \subset V_0^* \tag{2-49}$$

$$z_{(n+1)j} + y_j \leq 1, \forall\ j \in V \tag{2-50}$$

$$z_{(n+1)j} + z_{ij} \leq 1, \forall\ j \in V_0^*, \forall\ (i,j) \in \delta_j \tag{2-51}$$

$$x_{ij}, y_j \in \{0,1\}, \forall\ i, j \in V \tag{2-52}$$

$$z_{ij} \in \{0,1\}, \forall\ (i,j) \in E_0^* \tag{2-53}$$

A função objetivo (2-43) minimiza o custo de instalação das facilidades, o custo de servir cada usuário com a facilidade aberta mais próxima e o de conectar as facilidades abertas. As restrições (2-44) expressam que cada usuário será designado para exatamente uma única mediana. As restrições (2-45) previnem que um usuário seja designado para uma potencial localização de facilidade que não esteja selecionada como mediana. A restrição (2-46) assegura que uma única aresta $(0, j)$ será utilizada. As restrições (2-47) garantem que não se pode instalar a aresta $(0, j)$ se a potencial localização j não estiver selecionada como mediana. A restrição (2-46) e as restrições (2-47) asseguram a conexão do vértice artificial 0 a uma das medianas instaladas. As restrições (2-48) e (2-49) asseguram que as arestas escolhidas formam uma árvore geradora no grafo G_0^*. As restrições (2-48) relacionam-se à quantidade de arestas utilizadas e as restrições (2-49) são as de eliminação de sub-rotas, onde W representa um subconjunto qualquer de vértices do conjunto V_0^*. As restrições (2-50) asseguram que, se uma mediana for instalada no vértice j, isto é, se $y_j = 1$, então a aresta $(n + 1, j)$ não será utilizada. As restrições (2-51) asseguram que, caso não ocorra a instalação de uma mediana no vértice j, isto é, se $y_j = 0$ e a aresta $(n + 1, j)$ for utilizada, então nenhuma outra aresta do grafo original G que possui uma extremidade no vértice j pode ser utilizada. Assim, os vértices que não pertencem à árvore de Steiner estarão diretamente ligados ao vértice artificial $n + 1$. As restrições (2-52) e (2-53) estabelecem as variáveis de decisão x_{ij}, y_j e z_{ij} como binárias.

O modelo garante que uma árvore de Steiner mínima conectará um

subconjunto de medianas instaladas, com os usuários sendo servidos pela facilidade aberta mais próxima.

2.6 Comparações entre os Modelos para o Problema das p-Medianas Conectadas

As comparações entre o modelo por fluxos e o modelo por árvore para o problema das p-medianas conectadas serão realizadas em grafos retirados da literatura de p-medianas [25], do problema de Steiner [60] e de modelos integrados de projeto de redes e de localização de facilidades [16, 36, 37].

O primeiro grupo de instâncias, denominado grupo de instâncias proporcionais, foi obtido como se segue. Dado um grafo $G = (V, E)$, os custos associados às arestas foram gerados aleatoriamente entre $[1, 20]$, utilizando-se o gerador de números aleatórios de Matsumoto e Nishimura [53]. O custo de servir cada usuário i a partir de cada facilidade j é dado pelo valor d_{ij} do caminho mais curto do vértice i ao vértice j no grafo G, calculado a partir dos custos associados às arestas. O custo de instalação c_{ij} de cada aresta é obtido multiplicando-se o custo da mesma pelo fator w, isto é, $c_{ij} = a_{ij} \times w$, onde a_{ij} é o custo associado à aresta (i, j) e w assume os valores 2, 5 e 10.

O segundo grupo de instâncias, denominado grupo de instâncias não proporcionais, foi obtido a partir do grupo de instâncias proporcionais da seguinte maneira. Dado um grafo $G = (V, E)$ com custos associados às arestas entre $[1, 20]$, o custo de servir cada usuário i a partir de cada facilidade j é dado pelo valor d_{ij} do caminho mais curto do vértice i ao vértice j no grafo G, como no primeiro grupo de instâncias. Porém, o custo de instalação c_{ij} de cada aresta é escolhido aleatoriamente no intervalo $[(w \times a_{ij}) - (w \times a_{ij})/2, (w \times a_{ij}) + (w \times a_{ij})/2]$, onde a_{ij} é o custo associado à aresta (i, j) e w assume os valores 2, 5 e 10.

Os modelos foram executados em uma máquina Pentium IV 3.2 GHz com 1 Gbyte de memória RAM sob o sistema operacional Windows XP utilizando-se a versão 8.0 do resolvedor de programação linear inteira ILOG CPLEX. Para a execução do modelo baseado em árvore, a restrição de eliminação de sub-rotas foi substituída pela restrição de Miller-Tucker-Zemlin utilizada no problema do caixeiro viajante [38].

As Tabelas 2.1 a 2.12 mostram os resultados obtidos pela formulação baseada em fluxos e pela formulação baseada em árvore para as instâncias proporcionais e não proporcionais. As colunas das tabelas indicam, respectivamente, o nome do problema (Instância), o número de vértices do grafo (n), o número de arestas do grafo ($|E|$), o número de facilidades a serem instaladas (p), o valor da solução ótima do problema (F_O), o tempo gasto em segundos para encontrar a solução ótima (Tempo (s)) e o número de nós explorados pelo

algoritmo de *branch and bound* (Nós).

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
P_1	9	20	3	36	0,30	128
P_2	9	20	4	36	1,33	973
P_3	10	14	3	18	0,30	119
P_4	10	14	4	17	1,00	438
P_5	10	18	3	49	0,59	325
P_6	10	18	4	48	1,59	1.163
P_7	15	30	3	93	1,55	647
P_8	15	30	4	85	12,50	5.096
P_9	15	30	5	83	296,17	111.600
P_10	17	40	3	97	3,27	880
P_11	17	40	4	94	52,25	18.532
P_12	17	40	5	91	61,19	16.796
P_13	21	38	3	76	2,30	322
P_14	21	38	4	70	129,83	36.830
P_15	21	38	5	68	576,72	103.452
P_16	21	38	6	66	12.396,83	1.809.532
P_17	21	45	3	138	3,84	708
P_18	21	45	4	126	521,22	141.452
P_19	21	45	5	118	15.383,72	3.084.294
P_20	21	45	6	114	119.958,31	20.410.121
P_21	21	52	3	124	3,09	633
P_22	21	52	4	121	24,98	4.117
P_23	21	52	5	118	1.709,16	273.834
P_24	21	52	6	116	7.333,94	1.056.254
P_25	25	40	3	117	3,44	363
P_26	25	40	4	109	96,75	6.428
P_27	25	40	5	103	3.813,91	406.974
P_28	25	40	6	100	2.014,67	220.022

Tabela 2.1: Resultados obtidos pela formulação por fluxos para as instâncias proporcionais ($w = 2$).

Para a formulação por fluxos, independente do valor de w e do grupo de instâncias (proporcionais e não proporcionais), observa-se que, mantendo-se o mesmo grafo e aumentando-se o número de facilidades a serem abertas, aumenta-se a dificuldade para encontrar a solução ótima em termos de tempo de processamento e número de nós explorados. Como exemplo, sejam as instâncias proporcionais P_19 e P_20 ($w = 2$). Ao instalar uma única facilidade a mais no mesmo grafo, a instância P_20 explora aproximadamente $6,6$ vezes mais nós do que a instância P_19, e gasta aproximadamente $7,7$ vezes mais tempo do que a mesma para encontrar a solução ótima. Já na formulação por árvore, em alguns grupos de instâncias (proporcionais e não proporcionais), ocorre o mesmo padrão de comportamento, porém, as diferenças em termos de tempos de processamento permanecem na ordem dos segundos, e não aumenta-se demasiadamente o número de nós explorados.

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
P_29	9	20	3	50	0,22	67
P_30	9	20	4	58	1,59	1.543
P_31	10	14	3	26	0,30	126
P_32	10	14	4	29	1,52	1.024
P_33	10	18	3	68	0,42	122
P_34	10	18	4	72	0,81	484
P_35	15	30	3	123	2,00	752
P_36	15	30	4	126	7,34	3.404
P_37	15	30	5	128	172,36	64.730
P_38	17	40	3	119	2,25	599
P_39	17	40	4	122	17,84	5.192
P_40	17	40	5	126	100,69	27.426
P_41	21	38	3	96	4,31	1.184
P_42	21	38	4	94	74,17	18.094
P_43	21	38	5	96	212,75	45.075
P_44	21	38	6	98	2.000,20	399.423
P_45	21	45	3	185	28,72	8.263
P_46	21	45	4	187	253,89	65.712
P_47	21	45	5	185	7.331,91	1.377.828
P_48	21	45	6	184	27.696,42	4.450.700
P_49	21	52	3	147	2,19	387
P_50	21	52	4	149	19,91	4.137
P_51	21	52	5	152	246,53	53.180
P_52	21	52	6	158	1.617,08	242.045
P_53	25	40	3	139	10,27	1.596
P_54	25	40	4	136	45,70	8.062
P_55	25	40	5	135	320,22	48.937
P_56	25	40	6	133	1.133,42	162.556

Tabela 2.2: Resultados obtidos pela formulação por fluxos para as instâncias proporcionais ($w = 5$).

Observa-se, em geral, que a formulação por fluxos e por árvore encontra a solução ótima das instâncias não proporcionais em um menor tempo de processamento (explorando um menor número de nós) quando comparado com o tempo gasto pelas mesmas formulações para encontrar a solução ótima das instâncias proporcionais.

Ambos os modelos encontram a solução ótima nas 168 instâncias testadas. Porém, verifica-se uma maior explosão do número de nós explorados na formulação por fluxos do que na formulação por árvore, caracterizando uma maior dificuldade de resolução dos problemas pelo modelo por fluxos em relação ao modelo por árvore. A formulação por árvore é, na maioria das vezes, muito mais rápida do que a formulação por fluxos para encontrar a solução ótima dos problemas. Todas as instâncias foram resolvidas em menos de um minuto pela formulação por árvore, enquanto que algumas instâncias são resolvidas em horas pela formulação por fluxos.

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
P_57	9	20	3	65	0,20	37
P_58	9	20	4	83	0,73	464
P_59	10	14	3	36	0,25	82
P_60	10	14	4	45	1,22	747
P_61	10	18	3	92	0,36	100
P_62	10	18	4	112	0,80	477
P_63	15	30	3	143	1,63	472
P_64	15	30	4	166	3,55	1.503
P_65	15	30	5	178	44,67	15.914
P_66	17	40	3	133	0,70	98
P_67	17	40	4	149	6,03	1.471
P_68	17	40	5	165	37,16	11.249
P_69	21	38	3	111	2,52	349
P_70	21	38	4	118	21,08	5.937
P_71	21	38	5	126	107,91	23.952
P_72	21	38	6	138	812,76	173.479
P_73	21	45	3	225	7,69	1.382
P_74	21	45	4	241	103,36	22.175
P_75	21	45	5	249	1.801,94	303.584
P_76	21	45	6	259	5.489,17	780.932
P_77	21	52	3	162	1,78	178
P_78	21	52	4	178	14,59	2.305
P_79	21	52	5	192	74,86	19.264
P_80	21	52	6	208	543,00	99.529
P_81	25	40	3	154	3,66	499
P_82	25	40	4	158	21,72	4.028
P_83	25	40	5	166	261,45	48.545
P_84	25	40	6	174	1.455,19	305.195

Tabela 2.3: Resultados obtidos pela formulação por fluxos para as instâncias proporcionais ($w = 10$).

2.7 Considerações Finais

Neste capítulo foram formulados dois problemas que possuem características de projeto de redes e de localização de facilidades simultaneamente e que procuram localizar facilidades em uma rede, conectando-as por uma árvore de Steiner e minimizando o custo de atendimento oferecidos pelas facilidades abertas aos usuários da rede. Foram apresentadas, também, formulações de programação linear inteira para os mesmos. Para o primeiro, duas modelagens foram propostas: uma baseada na junção da formulação por fluxos para Steiner com a formulação clássica de p-medianas e a outra unindo a formulação por árvore de Steiner restrita por grau com a formulação das p-medianas. Esse último modelo pôde ser facilmente adaptado para o segundo problema, bastando para isso incluir um custo fixo para a instalação de uma facilidade em um vértice da rede e eliminar a restrição de limite superior no número de fa-

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
P_1	9	20	3	36	0,17	18
P_2	9	20	4	36	0,08	10
P_3	10	14	3	18	0,06	13
P_4	10	14	4	17	0,08	0
P_5	10	18	3	49	0,11	20
P_6	10	18	4	48	0,13	20
P_7	15	30	3	93	0,63	435
P_8	15	30	4	85	0,73	810
P_9	15	30	5	83	1,88	2.933
P_10	17	40	3	97	0,38	133
P_11	17	40	4	94	0,45	207
P_12	17	40	5	91	0,59	287
P_13	21	38	3	76	1,05	263
P_14	21	38	4	70	1,36	576
P_15	21	38	5	68	1,31	842
P_16	21	38	6	66	1,91	1.806
P_17	21	45	3	138	24,72	22.264
P_18	21	45	4	126	33,83	34.504
P_19	21	45	5	118	46,56	50.829
P_20	21	45	6	114	53,64	60.848
P_21	21	52	3	124	0,55	42
P_22	21	52	4	121	0,63	93
P_23	21	52	5	118	0,95	254
P_24	21	52	6	116	1,47	766
P_25	25	40	3	117	6,33	2.905
P_26	25	40	4	109	7,72	4.067
P_27	25	40	5	103	11,53	6.966
P_28	25	40	6	100	12,97	9.551

Tabela 2.4: Resultados obtidos pela formulação por árvore para as instâncias proporcionais ($w = 2$).

cilidades a serem abertas, já que o próprio modelo determina o subconjunto ótimo de facilidades. Como a formulação por fluxos do primeiro problema é dependente do número de facilidades abertas p e o segundo problema não o é, a formulação por fluxos não pôde ser adaptada para o segundo problema.

Por último, os dois modelos propostos para o problema das p-medianas conectadas foram comparados utilizando-se um resolvedor de programação linear inteira. Os resultados mostraram que a formulação por árvore geradora restrita por grau foi superior à formulação por fluxos nas instâncias testadas (proporcionais e não proporcionais) e será o modelo utilizado para encontrar soluções ótimas ou limites inferiores na avaliação das heurísticas propostas.

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
P_29	9	20	3	50	0,09	11
P_30	9	20	4	58	0,09	24
P_31	10	14	3	26	0,09	11
P_32	10	14	4	29	0,08	5
P_33	10	18	3	68	0,09	12
P_34	10	18	4	72	0,08	11
P_35	15	30	3	123	0,50	189
P_36	15	30	4	126	0,45	264
P_37	15	30	5	128	0,51	454
P_38	17	40	3	119	0,28	17
P_39	17	40	4	122	0,28	50
P_40	17	40	5	126	0,27	25
P_41	21	38	3	96	0,81	120
P_42	21	38	4	94	0,64	85
P_43	21	38	5	96	0,69	167
P_44	21	38	6	98	0,97	551
P_45	21	45	3	185	6,02	3.425
P_46	21	45	4	187	10,97	8.739
P_47	21	45	5	185	24,92	23.744
P_48	21	45	6	184	20,91	20.126
P_49	21	52	3	147	0,41	18
P_50	21	52	4	149	0,47	42
P_51	21	52	5	152	0,42	30
P_52	21	52	6	158	0,50	32
P_53	25	40	3	139	3,22	1.172
P_54	25	40	4	136	3,75	1.556
P_55	25	40	5	135	5,69	2.998
P_56	25	40	6	133	5,41	3.271

Tabela 2.5: Resultados obtidos pela formulação por árvore para as instâncias proporcionais ($w = 5$).

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
P_57	9	20	3	65	0,06	3
P_58	9	20	4	83	0,05	5
P_59	10	14	3	36	0,09	12
P_60	10	14	4	45	0,09	5
P_61	10	18	3	92	0,08	9
P_62	10	18	4	112	0,08	5
P_63	15	30	3	143	0,30	26
P_64	15	30	4	166	0,28	51
P_65	15	30	5	178	0,17	17
P_66	17	40	3	133	0,13	2
P_67	17	40	4	149	0,20	16
P_68	17	40	5	165	0,17	13
P_69	21	38	3	111	0,61	43
P_70	21	38	4	118	0,48	50
P_71	21	38	5	126	0,56	39
P_72	21	38	6	138	0,53	95
P_73	21	45	3	225	2,91	1.546
P_74	21	45	4	241	3,42	1.822
P_75	21	45	5	249	3,74	2.754
P_76	21	45	6	259	5,94	5.325
P_77	21	52	3	162	0,31	6
P_78	21	52	4	178	0,25	7
P_79	21	52	5	192	0,23	9
P_80	21	52	6	208	0,24	5
P_81	25	40	3	154	0,69	33
P_82	25	40	4	158	0,70	111
P_83	25	40	5	166	0,97	297
P_84	25	40	6	174	1,17	469

Tabela 2.6: Resultados obtidos pela formulação por árvore para as instâncias proporcionais ($w = 10$).

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
NP_1	9	20	3	32	0,20	52
NP_2	9	20	4	30	0,75	548
NP_3	10	14	3	17	0,31	84
NP_4	10	14	4	16	0,91	463
NP_5	10	18	3	44	0,31	118
NP_6	10	18	4	42	1,11	692
NP_7	15	30	3	87	1,22	362
NP_8	15	30	4	78	13,97	6.516
NP_9	15	30	5	76	183,30	81.684
NP_10	17	40	3	92	2,03	304
NP_11	17	40	4	90	29,53	6.952
NP_12	17	40	5	87	150,23	37.420
NP_13	21	38	3	72	3,70	278
NP_14	21	38	4	66	30,25	7.189
NP_15	21	38	5	64	477,77	121.651
NP_16	21	38	6	62	1.957,30	429.284
NP_17	21	45	3	127	8,78	2.274
NP_18	21	45	4	115	50,92	11.096
NP_19	21	45	5	108	5.074,88	1.179.114
NP_20	21	45	6	104	100.080,89	17.833.865
NP_21	21	52	3	121	5,11	853
NP_22	21	52	4	116	76,72	14.107
NP_23	21	52	5	110	480,27	83.500
NP_24	21	52	6	106	11.023,27	1.746.696
NP_25	25	40	3	110	3,45	283
NP_26	25	40	4	102	13,11	1.851
NP_27	25	40	5	96	223,08	24.593
NP_28	25	40	6	91	681,56	66.593

Tabela 2.7: Resultados obtidos pela formulação por fluxos para as instâncias não proporcionais ($w = 2$).

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
NP_29	9	20	3	43	0,25	90
NP_30	9	20	4	46	0,38	196
NP_31	10	14	3	25	0,41	131
NP_32	10	14	4	29	1,69	1.179
NP_33	10	18	3	62	0,39	142
NP_34	10	18	4	64	0,63	313
NP_35	15	30	3	120	1,53	475
NP_36	15	30	4	119	19,09	8.643
NP_37	15	30	5	119	44,03	20.844
NP_38	17	40	3	119	2,25	458
NP_39	17	40	4	116	25,28	7.199
NP_40	17	40	5	118	24,50	6.575
NP_41	21	38	3	93	3,50	376
NP_42	21	38	4	91	48,61	10.755
NP_43	21	38	5	91	238,14	61.981
NP_44	21	38	6	90	704,73	141.179
NP_45	21	45	3	172	24,88	6.805
NP_46	21	45	4	168	169,02	34.607
NP_47	21	45	5	168	644,88	152.827
NP_48	21	45	6	167	3.274,08	568.291
NP_49	21	52	3	146	5,34	625
NP_50	21	52	4	146	47,70	9.727
NP_51	21	52	5	146	180,47	23.480
NP_52	21	52	6	148	1.346,05	215.413
NP_53	25	40	3	133	6,97	730
NP_54	25	40	4	124	35,20	5.894
NP_55	25	40	5	121	241,42	27.416
NP_56	25	40	6	119	770,66	71.758

Tabela 2.8: Resultados obtidos pela formulação por fluxos para as instâncias não proporcionais ($w = 5$).

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
NP_57	9	20	3	54	0,23	36
NP_58	9	20	4	62	0,31	123
NP_59	10	14	3	35	0,44	87
NP_60	10	14	4	48	1,30	1.109
NP_61	10	18	3	90	0,41	170
NP_62	10	18	4	100	0,75	360
NP_63	15	30	3	138	1,28	178
NP_64	15	30	4	160	4,22	1.904
NP_65	15	30	5	172	33,67	13.567
NP_66	17	40	3	134	1,39	215
NP_67	17	40	4	149	9,89	2.601
NP_68	17	40	5	158	19,44	5.928
NP_69	21	38	3	114	4,91	587
NP_70	21	38	4	115	29,55	6.350
NP_71	21	38	5	123	85,16	19.785
NP_72	21	38	6	139	976,14	245.514
NP_73	21	45	3	215	15,44	2.869
NP_74	21	45	4	230	121,08	23.298
NP_75	21	45	5	229	572,55	115.116
NP_76	21	45	6	242	5.478,03	950.650
NP_77	21	52	3	153	4,50	256
NP_78	21	52	4	166	10,41	1.368
NP_79	21	52	5	188	42,95	10.357
NP_80	21	52	6	203	468,50	97.320
NP_81	25	40	3	144	5,49	478
NP_82	25	40	4	147	19,03	3.810
NP_83	25	40	5	154	322,63	46.331
NP_84	25	40	6	155	1.467,81	251.148

Tabela 2.9: Resultados obtidos pela formulação por fluxos para as instâncias não proporcionais ($w = 10$).

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
NP_1	9	20	3	32	0,08	4
NP_2	9	20	4	30	0,03	9
NP_3	10	14	3	17	0,08	21
NP_4	10	14	4	16	0,06	10
NP_5	10	18	3	44	0,03	0
NP_6	10	18	4	42	0,03	10
NP_7	15	30	3	87	0,59	375
NP_8	15	30	4	78	0,69	635
NP_9	15	30	5	76	0,77	1.045
NP_10	17	40	3	92	0,25	31
NP_11	17	40	4	90	0,27	72
NP_12	17	40	5	87	0,23	50
NP_13	21	38	3	72	1,14	455
NP_14	21	38	4	66	1,30	689
NP_15	21	38	5	64	2,83	2.699
NP_16	21	38	6	62	2,80	3.203
NP_17	21	45	3	127	7,22	4.696
NP_18	21	45	4	115	11,52	8.617
NP_19	21	45	5	108	7,59	6.436
NP_20	21	45	6	104	8,23	8.398
NP_21	21	52	3	121	0,56	84
NP_22	21	52	4	116	0,94	274
NP_23	21	52	5	110	0,77	121
NP_24	21	52	6	106	0,69	140
NP_25	25	40	3	110	3,58	1.364
NP_26	25	40	4	102	4,78	2.130
NP_27	25	40	5	96	4,89	2.657
NP_28	25	40	6	91	5,08	2.232

Tabela 2.10: Resultados obtidos pela formulação por árvore para as instâncias não proporcionais ($w = 2$).

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
NP_29	9	20	3	43	0,05	5
NP_30	9	20	4	46	0,03	5
NP_31	10	14	3	25	0,09	14
NP_32	10	14	4	29	0,06	15
NP_33	10	18	3	62	0,05	17
NP_34	10	18	4	64	0,03	5
NP_35	15	30	3	120	0,45	173
NP_36	15	30	4	119	0,58	406
NP_37	15	30	5	119	0,28	151
NP_38	17	40	3	119	0,23	24
NP_39	17	40	4	116	0,16	10
NP_40	17	40	5	118	0,17	11
NP_41	21	38	3	93	0,94	236
NP_42	21	38	4	91	1,22	546
NP_43	21	38	5	91	1,05	510
NP_44	21	38	6	90	0,80	417
NP_45	21	45	3	172	5,86	4.060
NP_46	21	45	4	168	7,83	6.278
NP_47	21	45	5	168	10,70	10.649
NP_48	21	45	6	167	8,86	8.370
NP_49	21	52	3	146	0,48	36
NP_50	21	52	4	146	0,42	26
NP_51	21	52	5	146	0,39	18
NP_52	21	52	6	148	0,56	67
NP_53	25	40	3	133	3,55	1.786
NP_54	25	40	4	124	3,02	1.084
NP_55	25	40	5	121	2,20	932
NP_56	25	40	6	119	2,09	877

Tabela 2.11: Resultados obtidos pela formulação por árvore para as instâncias não proporcionais ($w = 5$).

Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
NP_57	9	20	3	54	0,03	5
NP_58	9	20	4	62	0,01	3
NP_59	10	14	3	35	0,05	7
NP_60	10	14	4	48	0,06	7
NP_61	10	18	3	90	0,05	6
NP_62	10	18	4	100	0,03	7
NP_63	15	30	3	138	0,23	57
NP_64	15	30	4	160	0,22	47
NP_65	15	30	5	172	0,14	15
NP_66	17	40	3	134	0,17	8
NP_67	17	40	4	149	0,14	11
NP_68	17	40	5	158	0,16	8
NP_69	21	38	3	114	0,80	183
NP_70	21	38	4	115	0,67	80
NP_71	21	38	5	123	0,66	182
NP_72	21	38	6	139	0,80	434
NP_73	21	45	3	215	1,89	549
NP_74	21	45	4	230	2,73	1.293
NP_75	21	45	5	229	2,53	1.600
NP_76	21	45	6	242	3,09	2.216
NP_77	21	52	3	153	0,25	3
NP_78	21	52	4	166	0,33	17
NP_79	21	52	5	188	0,38	31
NP_80	21	52	6	203	0,36	81
NP_81	25	40	3	144	0,83	44
NP_82	25	40	4	147	0,91	78
NP_83	25	40	5	154	0,88	214
NP_84	25	40	6	155	0,72	132

Tabela 2.12: Resultados obtidos pela formulação por árvore para as instâncias não proporcionais ($w = 10$).

3
Algoritmo de Busca Local

Um algoritmo de busca local define, para cada solução, uma vizinhança composta por um conjunto de soluções com características "muito próximas". Dada uma solução corrente, uma das formas de implementar um algoritmo de busca local é percorrer a vizinhança dessa solução em busca de outra com valor menor (para um problema de minimização). Se tal solução vizinha for encontrada, torna-se a nova solução corrente e o algoritmo continua. Caso contrário, a solução corrente é um ótimo local em relação à vizinhança adotada.

A abordagem utilizada no algoritmo de busca local decompõe o problema das p-medianas conectadas em dois sub-problemas: localizar p facilidades nos vértices do grafo, com os usuários servidos pela facilidade aberta mais próxima (SP-1) e conectar as p facilidades abertas por uma árvore de Steiner (SP-2). Para resolver SP-1 e SP-2, respectivamente, utiliza-se o algoritmo de busca local do problema das p-medianas, cuja implementação foi proposta por Resende e Werneck [47] e a heurística construtiva gulosa de Takahashi e Matsuyama (Prim) [57].

O algoritmo de busca local básico é inicializado através de uma heurística construtiva. A vizinhança é definida como o conjunto de soluções alcançadas pela troca de uma facilidade aberta (presente na solução) por outra fechada (que não faz parte da solução). Em cada iteração, o algoritmo procura trocar uma facilidade pertencente à solução por outra que não pertence à solução, avaliando essa troca em termos do custo de atendimento dos usuários. As facilidades abertas são conectadas por uma árvore de Steiner (custo de interconexão das facilidades). Caso ocorra uma diminuição no valor da função objetivo, efetua-se a troca e tem-se uma nova solução corrente. O algoritmo continua até que nenhum melhoramento aconteça, permanecendo assim em um ótimo local.

Porém, conectar as p facilidades abertas através de uma árvore de Steiner a cada nova troca é uma rotina extremamente custosa computacionalmente. Para acelerar o algoritmo, propõe-se uma estratégia híbrida que concatena duas variantes da busca local básica, incorporando idéias como a estratégia de busca por melhoria iterativa, circularidade e um teste simples com o objetivo de descartar vizinhos, evitando-se a execução da heurística de Steiner a cada nova troca.

Como no problema das p-medianas conectadas cada vértice é simultaneamente uma potencial localização de facilidade e um usuário, tem-se que $m = n$, onde m representa o número de possíveis localizações de facilidades e n representa o número de usuários. Quando necessário, a distinção será mantida.

Esse capítulo está organizado como se segue: a Seção 3.1 apresenta a heurística construtiva utilizada como solução inicial nos algoritmos de busca local. A Seção 3.2 descreve o algoritmo básico de busca local para o problema das p-medianas conectadas. A Seção 3.3 mostra as descrições das três versões da busca local básica e o estudo computacional comparando essas versões com o algoritmo básico. A Seção 3.4 apresenta a estratégia híbrida (busca local concatenada) e os resultados computacionais em relação às demais buscas locais. Por último, a Seção 3.5 apresenta as considerações finais apontando a melhor busca local para o problema das p-medianas conectadas em termos de qualidade das soluções encontradas e dos tempos de processamento obtidos nas instâncias testadas.

3.1 Solução Inicial dos Algoritmos de Busca Local

Heurísticas construtivas são algoritmos que criam soluções viáveis. Podem ser utilizadas de maneira isolada, contudo, são freqüentemente utilizadas em métodos mais elaborados como algoritmos de busca local, metaheurísticas e algoritmos exatos.

Para a geração das soluções iniciais dos algoritmos de busca local, o sub-problema SP-1 será resolvido utilizando-se a heurística Sample Greedy, uma variante da heurística construtiva clássica para o problema das p-medianas e o sub-problema SP-2 será resolvido utilizando-se a heurística Prim, uma extensão direta do algoritmo exato para o problema da árvore geradora mínima adaptada para o problema de Steiner em grafos. Essas heurísticas foram escolhidas devido aos bons resultados alcançados nos problemas das p-medianas e no problema de Steiner [48, 61]. Assim, pretende-se resolver cada um dos sub-problemas separadamente, combinando-se as suas soluções.

A Figura 3.1 descreve o algoritmo, onde $C(f)$ representa o acréscimo obtido no valor da função objetivo relativo ao custo de atendimento dos usuários ao adicionar a facilidade $f \in V$ à solução sendo construída. A função recebe como parâmetros a *semente* usada para inicializar o gerador de números aleatórios, o número de vértices do grafo e o número de facilidades a serem instaladas.

A heurística construtiva clássica para o problema das p-medianas constrói uma solução iterativamente, selecionando as facilidades seqüencialmente, até que p facilidades tenham sido escolhidas, minimizando o aumento na função objetivo em cada passo. A heurística Sample Greedy (linhas 1 a 16) seleciona

Função SampleGreedy_Prim (*semente*, n, p)
1. $S \leftarrow \emptyset$;
2. Inicialize o conjunto V das possíveis localizações de facilidades;
3. $q \leftarrow \lceil \log_2(n/p) \rceil$;
4. **Enquanto** $(|S| \neq p)$ **Faça**
5. $\quad k \leftarrow 0, Q \leftarrow \emptyset, V_{aux} = V$;
6. $\quad$ **Enquanto** $(k \neq q)$ **Faça**
7. $\quad\quad$ Selecione f aleatoriamente de V_{aux};
8. $\quad\quad$ Avalie o acréscimo $C(f)$;
9. $\quad\quad$ $Q \leftarrow Q \cup \{f\}$;
10. $\quad\quad$ $V_{aux} \leftarrow V_{aux} \setminus \{f\}$;
11. $\quad\quad$ $k \leftarrow k + 1$;
12. $\quad$ **Fim-Enquanto**
13. $\quad$ $f \leftarrow \text{argmin}\{C(f), f \in Q\}$;
14. $\quad$ $S \leftarrow S \cup \{f\}$;
15. $\quad$ $V \leftarrow V \setminus \{f\}$;
16. **Fim-Enquanto**
17. Inicie com uma sub-árvore $T_1(S)$ consistindo de um terminal $f \in S$ escolhido aleatoriamente;
18. $k \leftarrow 1, X_k \leftarrow X_k \cup \{f\}$;
19. **Enquanto** $(k < p)$ **Faça**
20. $\quad$ Determine um terminal $f \in S \setminus X_k$ mais próximo de $T_k(S)$; Construa uma árvore $T_{k+1}(S)$ adicionando o caminho mínimo entre f e $T_k(S)$;
21. $\quad$ $k \leftarrow k + 1, X_k \leftarrow X_k \cup \{f\}$;
22. **Fim-Enquanto**
23. $T(S) \leftarrow T_p(S)$;
24. **Retorne** S;

Fim

Figura 3.1: Algoritmo que gera as soluções iniciais das buscas locais.

aleatoriamente, em cada uma das p iterações, $q < n$ possíveis localizações de facilidades (linhas 5 a 12), escolhendo aquela que minimiza o aumento na função objetivo relativo ao custo de atendimento dos usuários (linha 13). A idéia é fazer q suficientemente pequeno para reduzir o tempo de execução em comparação ao algoritmo puramente guloso, mantendo um certo grau de aleatorização. Como em [48], utiliza-se $q = \lceil \log_2(n/p) \rceil$ (linha 3). A complexidade de pior caso da heurística Sample Greedy para o problema das p-medianas é $O(m + pqn)$ [48]. No problema das p-medianas conectadas tem-se $m = n$, então a complexidade de pior caso da heurística que resolve SP-1 (linhas 1 a 16) é $O(pqn)$.

A heurística Prim (linhas 17 a 23) é um dos algoritmos mais eficientes e mais utilizados na literatura para o cálculo de soluções aproximadas gulosas [52]. Inicia-se o algoritmo com uma sub-árvore $T_1(S)$ e um conjunto X contendo um terminal qualquer (linhas 17 e 18). Em cada uma das $p - 1$ iterações

(linhas 19 a 22), o algoritmo determina qual terminal $f \in S \backslash X$ possui distância mínima à sub-árvore corrente T_k, adicionando à mesma o terminal e os vértices e arestas no caminho entre ele e a sub-árvore (linha 20). Na linha 21 atualiza-se o número de iterações e o conjunto X_k. Ao final das iterações, o algoritmo conectará os p vértices terminais e possíveis não-terminais através de uma árvore de Steiner (linha 23). A complexidade de pior caso da heurística Prim depende da estrutura de dados utilizada. Utilizando-se um *heap* de Fibonacci (resp. *heap* binário), a complexidade é $O(p(|E|+n\log n))$ (resp. $O(p\,|E|\log n)$), onde $|E|$ corresponde ao número de arestas do grafo. Não utilizando um *heap*, a complexidade é $O(pn^2)$. Na tese foi utilizada a estrutura de dados *heap* binário. Essa implementação é bastante eficiente e, na prática, o fator p da complexidade de pior caso da heurística não se manifesta [61].

Por último, na linha 24, o algoritmo retorna uma solução definida pelo conjunto S formado pelas p facilidades abertas e pelo conjunto $T(S)$ de arestas obtidas quando se conecta estas facilidades por uma árvore de Steiner.

A complexidade de pior caso do algoritmo da Figura 3.1 é $O(p\,|E|\log n)$, dominada pela complexidade da heurística Prim que resolve o sub-problema SP2 (linhas 17 a 23).

3.2 Algoritmo Básico de Busca Local

O algoritmo básico de busca local para o problema das p-medianas conectadas utiliza o algoritmo de busca local clássico para o problema das p-medianas, juntamente com a heurística Prim.

O algoritmo de busca local para o problema das p-medianas, proposto por Teitz e Bart [58], é baseado na troca de facilidades. Dada uma solução composta por p facilidades, uma iteração do algoritmo determina, para cada facilidade que não está presente na solução, qual facilidade presente na solução (se houver) melhora ao máximo o valor da função objetivo se ambas forem trocadas. Se tal movimento existe, uma nova iteração será realizada a partir desta nova solução. Caso contrário, o algoritmo termina com um mínimo local sendo encontrado. A complexidade de pior caso do algoritmo é $O(pmn)$ por iteração.

Whitaker [62] descreveu uma implementação eficiente desse método, denominado de *Fast Interchange*, com complexidade $O(mn)$ por iteração. Resende e Werneck [47] propuseram uma implementação alternativa utilizando as mesmas operações básicas usadas por Whitaker em uma ordem diferente e com a mesma complexidade de pior caso. Apesar de requerer espaço de memória superior à heurística de Whitaker, acelerações significativas foram obtidas na prática em termos de tempo de processamento de até três ordens de magnitude nas instâncias testadas em relação à *Fast Interchange* [47].

O algoritmo de busca local proposto por Resende e Werneck foi modificado para se adaptar ao problema das p-medianas conectadas, onde as operações relacionadas à localização das p facilidades e à associação dos usuários às facilidades abertas mais próximas são realizadas pelo algoritmo de busca local para o problema das p-medianas [47], sendo que, a cada troca, as p facilidades abertas são conectadas via heurística de Prim.

3.2.1 Descrição do Algoritmo

A primeira e a segunda facilidade mais próxima do usuário $u \in V$ serão denotadas por ϕ_1^u e ϕ_2^u, respectivamente; d_1^u e d_2^u representam a distância do usuário u às facilidades ϕ_1^u e ϕ_2^u, respectivamente; a solução corrente é definida pelo conjunto S formado pelas p facilidades abertas e pelo conjunto $T(S)$ de arestas obtidas quando se conecta estas facilidades por uma árvore de Steiner através da heurística Prim; $F(S)$ é o custo da solução S e $F_T(S)$ é o custo da árvore de Steiner calculada acima. Representa-se por $i \notin S$ (resp. $r \in S$) uma facilidade candidata a inserção (resp. remoção) no (resp. do) conjunto de p facilidades abertas; uma solução vizinha $S^{'}$ da solução corrente S é obtida trocando-se uma facilidade candidata a remoção $r \in S$ por uma facilidade candidata a inserção $i \notin S$.

A descrição do algoritmo de busca local será baseada na descrição do algoritmo para o problema das p-medianas proposto em [47], realçando-se, quando necessário, as adaptações realizadas. Com o objetivo de acelerar passos posteriores do algoritmo, resultados parcias obtidos são armazenados em estruturas de dados auxiliares (*ganho*, *perda* e *extra*).

Dada uma solução S, para cada facilidade i candidata a inserção, define-se $ganho(i)$ como o total economizado quando i é adicionada a solução, não ocorrendo remoção de facilidade alguma. A economia obtida ao associar cada usuário a i cuja facilidade mais próxima estava mais distante do que i é dada por

$$ganho(i) = \sum_{u \in V} \max\{0, d_1^u - d_{ui}\}. \tag{3-1}$$

Similarmente, para cada facilidade r candidata a remoção, define-se $perda(r)$ como o aumento no valor da solução, resultante da remoção de r de S, não ocorrendo inserção de facilidade alguma. Este é o custo de transferir cada usuário associado a r à sua segunda facilidade mais próxima

$$perda(r) = \sum_{u:\phi_1^u = r} (d_2^u - d_1^u). \tag{3-2}$$

O cálculo de $perda(r)$ assume que o usuário u sempre será transferido da facilidade candidata a remoção r para a sua segunda facilidade mais próxima ϕ_2^u, sem levar em consideração a entrada da facilidade candidata a inserção i.

Porém, ocorrerá situações em que o usuário u estará mais próximo de i do que de ϕ_2^u. Assim, algumas parcelas no cálculo de $perda(r)$ precisam ser corrigidas. A estrutura de dados $extra(i, r)$ é responsável por essas correções. Para a sua definição, observa-se que, para cada usuário u, tem-se os seguintes casos:

1. $\phi_1^u \neq r$ - quando a facilidade mais próxima do usuário u não é r, pode-se eventualmente economizar, associando-se o usuário à facilidade i candidata a inserção. O valor economizado já está sendo embutido em $ganho(i)$.

2. $\phi_1^u = r$ - quando a facilidade mais próxima do usuário u é r, existem três possibilidades:

   (a) $d_1^u \leq d_2^u \leq d_{ui}$ - Como a distância do usuário u a i é maior do que a distância do usuário u a ϕ_2^u e, conseqüentemente a ϕ_1^u, o usuário deve ser transferido de r para ϕ_2^u, o que já está contabilizado no cálculo de $perda(r)$.

   (b) $d_1^u \leq d_{ui} < d_2^u$ - Como a distância do usuário u a i é menor do que a distância do usuário u a ϕ_2^u e maior ou igual a distância do usuário u a ϕ_1^u, o usuário deve ser transferido de r para i. Porém, o cálculo de $perda(r)$ assume que o usuário foi transferido de r para ϕ_2^u e deve ser corrigido pela parcela $d_2^u - d_{ui}$.

   (c) $d_{ui} < d_1^u \leq d_2^u$ - Como a distância do usuário u a i é menor do que a distância do usuário u a ϕ_1^u e, conseqüentemente a ϕ_2^u, o usuário u deve ser transferido de r para i, hipótese já contabilizada no cálculo de $ganho(i) = d_1^u - d_{ui}$. Porém, o cálculo de $perda(r)$ assume que o usuário foi transferido de r para ϕ_2^u e deve ser corrigido pela parcela $d_2^u - d_1^u$.

A função $extra(i, r)$ calcula as parcelas associadas aos itens (2b) e (2c) acima, fazendo as correções necessárias, e pode ser expressa como em [47]:

$$\begin{aligned} extra(i,r) = & \sum_{u:(\phi_1^u = r)\ e\ (d_1^u \leq d_{ui} < d_2^u)} (d_2^u - d_{ui}) + \\ & \sum_{u:(\phi_1^u = r)\ e\ (d_{ui} < d_1^u \leq d_2^u)} (d_2^u - d_{ur}) = \\ & \sum_{u:(\phi_1^u = r)\ e\ (d_{ui} < d_2^u)} (d_2^u - max\{d_{ui}, d_{ur}\}). \end{aligned} \tag{3-3}$$

A economia obtida no problema das p-medianas ao se trocar a facilidade i pela facilidade r pode ser dada pela seguinte equação [47]:

$$lucroP(i,r) = ganho(i) - perda(r) + extra(i,r). \tag{3-4}$$

Quando uma facilidade aberta r é substituída por uma nova facilidade i, os valores nas estruturas *ganho*, *perda*, *extra*, ϕ_1^u e ϕ_2^u ficam desatualizados. Uma maneira direta para a atualização dessas estruturas para a próxima iteração da busca local é recalcular ϕ_1^u e ϕ_2^u para cada usuário u, reinicializar todas as entradas das estruturas auxiliares e então atualizar *ganho*, *perda* e *extra*. Uma desvantagem dessa atualização é que nenhuma informação reunida em iterações anteriores é utilizada em iterações subseqüentes do algoritmo, podendo ocorrer repetição de cálculos.

Pode-se, porém, acelerar a busca local, pois as ações realizadas para atualizar as estruturas auxiliares dependem exclusivamente de u, ϕ_1^u e ϕ_2^u. Portanto, se ϕ_1^u e ϕ_2^u não mudam de uma iteração para outra, não existe a necessidade de atualizar *ganho*, *perda* e *extra* novamente para o usuário u.

Um usuário u é considerado afetado, se, após a troca entre i e r, uma das seguintes condições é satisfeita:

1. r é a primeira ou segunda facilidade mais próxima do usuário u;
2. a facilidade i está mais próxima do usuário u do que a sua segunda facilidade mais próxima ϕ_2^u.

As estruturas *ganho*, *perda* e *extra* só precisam ser atualizadas para os usuários afetados. Se o número de usuários afetados é pequeno, ganhos significativos em termos de tempo de processamento podem ser obtidos.

A Figura 3.2 mostra o algoritmo de busca local para o problema das p-medianas conectadas que recebe como parâmetros uma solução inicial S_0 gerada pela heurística construtiva apresentada na Seção 3.1 e a primeira e segunda facilidade mais próximas de cada usuário u, ϕ_1^u e ϕ_2^u respectivamente.

A linha 2 do algoritmo consiste em construir, para cada usuário u, uma lista com todas as facilidades ordenadas pela distância ao usuário. Possui como objetivo reduzir o tempo gasto na atualização das estruturas *ganho*, *perda* e *extra*, como será visto na explicação das funções *Atualiza_Estruturas* e *Desfaz_Atualiza_Estruturas*. Esse passo é executado uma única vez, mesmo quando a busca local for utilizada diversas vezes dentro de outro algoritmo, como, por exemplo, uma metaheurística [47]. Na linha 3, o conjunto de usuários afetados V_a é inicializado com todos os usuários. Na linha 4, as estruturas auxiliares *ganho*, *perda* e *extra* são inicializadas. O laço das linhas 6 a 27 é executado enquanto existir uma solução vizinha melhor do que a solução corrente.

O laço das linhas 7 a 9 é responsável pela atualização eficiente das estruturas auxiliares. Pelas Equações (3-1), (3-2) e (3-3), nota-se que, cada parcela calculada em *ganho*, *perda* e *extra* é uma soma percorrendo um subconjunto de usuários. Assim, a contribuição de cada cliente pode ser calculada independentemente. A função *Atualiza_Estruturas*, executada somente para os usuários

Algoritmo `Busca_Local_p-Medianas_Conectadas`(S_0, ϕ_1, ϕ_2)
1. $S \leftarrow S_0$;
2. Para cada $u \in V$, ordenar as facilidades de acordo com a distância ao usuário;
3. $V_a \leftarrow V$;
4. $ganho(i) \leftarrow 0, perda(r) \leftarrow 0, extra(i,r) \leftarrow 0, \forall\ i \notin S, \forall\ r \in S$;
5. $Melhoria \leftarrow Verdadeiro$;
6. **Enquanto** $(Melhoria = Verdadeiro)$ **Faça**
7. **Para Todo** $(u \in V_a)$ **Faça**
8. *Atualiza_Estruturas(ganho, perda, extra, u,* ϕ_1^u, ϕ_2^u*)*;
9. **Fim-Para-Todo**
10. $(i, r, lucro(i,r)) \leftarrow$ *Primeiro_Aprimorante(S, ganho, perda, extra)*;
11. **Se** $(lucro(i,r) \leq 0)$ **Então**
12. $Melhoria \leftarrow Falso$;
13. Vá para a linha 27;
14. **Fim-Se**
15. $V_a \leftarrow \emptyset$;
16. **Para Todo** $(u \in V)$ **Faça**
17. **Se** $(\phi_1^u = r$ **ou** $\phi_2^u = r$ **ou** $d_{ui} < d_2^u)$ **Então**
18. $V_a \leftarrow V_a \cup \{u\}$;
19. **Fim-Se**
20. **Fim-Para-Todo**
21. **Para Todo** $(u \in V_a)$ **Faça**
22. *Desfaz_Atualiza_Estruturas(ganho, perda, extra, u,* ϕ_1^u, ϕ_2^u*)*;
23. **Fim-Para-Todo**
24. $S \leftarrow S \backslash \{r\}$;
25. $S \leftarrow S \cup \{i\}$;
26. Atualizar a primeira e segunda facilidade mais próxima de cada usuário $u \in V_a$;
27. **Fim-Enquanto**
28. **Retorne** S;

Fim

Figura 3.2: Algoritmo de busca local para o problema das p-medianas conectadas.

afetados é responsável por essa atualização e seu pseudo-código será explicado posteriormente. A função *Primeiro_Aprimorante* (linha 10) utiliza a estratégia de melhoria iterativa para acelerar os passos da busca local, selecionando, quando houver, a primeira solução aprimorante encontrada na vizinhança da solução corrente. A função *Primeiro_Aprimorante* será explicada com mais detalhes posteriormente. Quando não existir uma solução vizinha melhor do que a solução corrente (caso em que $lucro(i, r) \leq 0$), o algoritmo é finalizado nas linhas 11 a 14 e o ótimo local é retornado na linha 28. Caso contrário, ocorre a atualização do conjunto de usuários afetados nas linhas 15 a 20, conforme descrito anteriormente. As informações presentes em *ganho*, *perda* e *extra* encontram-se desatualizadas em relação à troca ocorrida. Para que uma nova iteração seja realizada, a contribuição da iteração anterior de cada usuário u deve ser subtraída antes que uma nova adição seja feita pela função *Atualiza_Estruturas*. A função *Desfaz_Atualiza_Estruturas*, executada somente para os usuários afetados (laço das linhas 21 a 23), é responsável por essa operação e é similar à função *Atualiza_Estruturas*. As linhas 24 e 25, respectivamente, são responsáveis pela remoção e inserção das facilidades r e i no conjunto das p facilidades abertas na solução corrente. Na linha 26, ocorre a atualização da primeira e da segunda facilidade mais próxima de cada usuário afetado, levando-se em consideração a troca efetuada.

Funções Atualiza_Estruturas e Desfaz_Atualiza_Estruturas

A Figura 3.3 apresenta o pseudo-código da função *Atualiza_Estruturas* que recebe como parâmetros as estruturas auxiliares *ganho*, *perda* e *extra*, além do usuário e de suas respectivas facilidades mais próximas. A expressão $a \overset{+}{\leftarrow} b$ significa que o valor de a é incrementado com b unidades. A função *Desfaz_Atualiza_Estruturas* é similar a essa função, com $a \overset{+}{\leftarrow} b$ sendo substituída por $a \overset{-}{\leftarrow} b$.

Função *Atualiza_Estruturas(ganho, perda, extra, u, ϕ_1^u, ϕ_2^u)*
1. $r \leftarrow \phi_1^u$;
2. $perda(r) \overset{+}{\leftarrow} (d_2^u - d_1^u)$;
3. **Para Todo** $(i \notin S)$ **Faça**
4. **Se** $(d_{ui} < d_2^u)$ **Então**
5. $ganho(i) \overset{+}{\leftarrow} \max\{0, d_1^u - d_{ui}\}$;
6. $extra(i, r) \overset{+}{\leftarrow} d_2^u - \max\{d_{ui}, d_{ur}\}$;
7. **Fim-Se**
8. **Fim-Para-Todo**

Fim

Figura 3.3: Função que atualiza as estruturas *ganho*, *perda* e *extra*.

Na linha 1 obtém-se a facilidade mais próxima do usuário u. A con-

tribuição individual de cada usuário para a estrutura *perda* é calculada na linha 2. O laço das linhas 3 a 8 percorre as $(n - p)$ facilidades que não estão presentes na solução. A contribuição individual de cada usuário para as estrututras *ganho* e *extra* só ocorrem nas linhas 5 e 6, respectivamente, se a distância do usuário u à facilidade i não é maior do que d_2^u (teste da linha 4).

Com o objetivo de tentar reduzir o tempo gasto em cada chamada às funções *Atualiza_Estruturas* e *Desfaz_Atualiza_Estruturas*, utiliza-se a lista com todas as facilidades ordenadas pela distância ao usuário u obtida na linha 2 do algoritmo de busca local (Figura 3.2). Dessa maneira, ao invés de percorrer todas as facilidades que não estão presentes na solução na linha 3 (Figura 3.3), percorre-se o conjunto de facilidades $i \notin S$ ordenadas pela distância ao usuário afetado até que a distância de u a i seja maior ou igual a distância de u a ϕ_2^u ($d_{ui} \geq d_2^u$), que é potencialmente um subconjunto pequeno de facilidades em relação a $(n - p)$.

Função Primeiro_Aprimorante

Resende e Werneck [47] utilizaram a estratégia de busca de descida mais rápida no problema das p-medianas, calculando-se o lucro obtido em cada troca entre uma facilidade pertencente e outra não pertencente à solução corrente através da equação (3-4). Utiliza-se a estratégia de melhoria iterativa, sendo a função *Primeiro_Aprimorante* responsável por encontrar o primeiro vizinho que melhore o valor da função objetivo ao se trocar a facilidade r pela facilidade i, calculando-se o lucro obtido no problema das p-medianas (equação (3-4)) e o lucro obtido no problema de Steiner (diferença no custo entre a árvore de Steiner corrente e a árvore de Steiner obtida na troca de r por i).

A Figura 3.4 apresenta a função que recebe como parâmetros a solução corrente composta pelo conjunto S de p facilidades abertas e pela árvore de Steiner $T(S)$ correspondente, além das estruturas *ganho*, *perda* e *extra*.

A função é composta por dois laços aninhados. O externo (linhas 2 a 11) percorre as facilidades que não estão na solução corrente e o interno (linhas 3 a 10) percorre as facilidades pertencentes à solução corrente. Ambos são executados até que seja encontrado o primeiro vizinho aprimorante ou então até que nenhum melhoramento seja possível. Se uma solução vizinha melhor do que a solução corrente é encontrada ($lucro(i, r) > 0$ na linha 7), retorna-se, na linha 12, as facilidades candidatas a inserção e remoção i e r, respectivamente, e o lucro total obtido. Caso contrário, se $lucro(i, r) \leq 0$, retorna-se na linha 12, as facilidades i e r da última avaliação realizada e seu respectivo lucro.

Para acelerar a busca, a investigação da vizinhança inicia-se do vértice seguinte àquele que obteve o movimento aprimorante na iteração prévia, tanto no conjunto dos candidatos a inserção quanto no conjunto dos candidatos a remoção. A circularidade foi utilizada, por exemplo, para acelerar os passos

```
Função Primeiro_Aprimorante(S, ganho, perda, extra)
1.  Melhoria ← Falso;
2.  Circ Para Todo (i ∉ S) e (Melhoria = Falso) Faça
3.      Circ Para Todo (r ∈ S) e (Melhoria = Falso) Faça
4.          lucroP(i, r) ← ganho(i) − perda(r) + extra(i, r);
5.          lucroS(i, r) ← F_T(S) − F_T(S');
6.          lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
7.          Se (lucro(i, r) > 0) Então
8.              Melhoria ← Verdadeiro;
9.          Fim-Se
10.     Fim-Circ-Para-Todo
11. Fim-Circ-Para-Todo
12. Retorne i, r, lucro(i, r);
Fim
```

Figura 3.4: Função que determina o primeiro vizinho aprimorante na busca local básica.

do algoritmo de busca local em [10]. O lucro obtido ao se trocar r por i para o problema das p-medianas é calculado na linha 4. O lucro obtido para o problema de Steiner é calculado na linha 5 como a diferença entre o custo da árvore de Steiner corrente ($F_T(S)$) e o custo da árvore de Steiner obtida ao se trocar r por i executando-se Prim ($F_T(S')$). O lucro total obtido é calculado na linha 6 como a soma dos lucros obtidos com o problema das p-medianas e com o problema de Steiner.

Complexidade do Algoritmo de Busca Local

A linha 2 do algoritmo possui complexidade $O(n^2 \log n)$ desde que, no máximo n usuários terão suas n facilidades ordenadas com complexidade $O(n \log n)$. A inicialização do conjunto de usuários afetados (linha 3) possui complexidade $O(n)$. A inicialização das estruturas de dados auxiliares (linha 4) possui complexidade $O(pn)$, pois *extra* possui $p \times (n-p)$ entradas. As funções *Atualiza_Estruturas* e *Desfaz_Atualiza_Estruturas* (linhas 8 e 22, respectivamente) possuem complexidade $O(n)$, pois apresentam um laço percorrendo as $(n-p)$ facilidades que não estão presentes na solução. Levando-se em consideração que no máximo n chamadas são executadas para cada uma dessas funções, a complexidade das linhas 7 e 8 assim como das linhas 21 e 22 é $O(n^2)$. A complexidade da função *Primeiro_Aprimorante* é $O(p^2 |E| n \log n)$, pois, no pior caso, $p \times (n-p) = O(pn)$ trocas serão avaliadas, onde, em cada troca ocorre a execução da heurística Prim com complexidade $O(p |E| \log n)$, utilizando-se a estrutura de dados *heap* binário [61]. A atualização do conjunto de usuários afetados (linhas 15 a 20) possui complexidade $O(n)$. A remoção (resp. inserção) de uma facilidade do (resp. no) conjunto de p facilidades abertas (linhas 24 e 25, respectivamente) possui complexidade $O(1)$. Por último,

a linha 26 possui complexidade $O(pn)$, pois para atualizar a primeira e segunda facilidade mais próxima de cada usuário afetado u, deve-se percorrer o conjunto das p facilidades abertas pertencentes à solução corrente.

A complexidade de pior caso de cada iteração do algoritmo de busca local (laço das linhas 6 a 27) é dominada pela complexidade da função *Primeiro_Aprimorante* que é $O(p^2 \left|E\right| n \log n)$.

3.3 Otimizando o Algoritmo Básico de Busca Local

O algoritmo básico de busca local demanda um alto custo computacional, principalmente na função *Primeiro_Aprimorante*, pois, em cada troca, uma árvore de Steiner deve ser recalculada quando troca-se a facilidade candidata a remoção r pela facilidade candidata a inserção i. Pode-se, porém, utilizar um simples teste com o objetivo de filtrar possíveis bons vizinhos.

Seja a solução corrente definida pelo conjunto S das p facilidades abertas e pela árvore de Steiner $T(S)$ correspondente, com seu respectivo custo $F_T(S)$. O teste é iniciado pela retirada de um vértice $r \in S$. Em seguida, calcula-se o custo da árvore de Steiner sem o vértice r, isto é, executa-se a heurística Prim com $p-1$ facilidades. Seja $F_T(S\backslash\{r\})$ o custo dessa árvore. Define-se, agora, a possível melhoria de Steiner (pms) como a diferença entre o custo da árvore de Steiner corrente e o custo da árvore de Steiner sem o vértice r, isto é, $pms = F_T(S) - F_T(S\backslash\{r\})$. Seja i uma facilidade qualquer candidata a inserção. O valor *lucroP* (i, r) representa o lucro obtido ao se trocar r por i no problema das p-medianas. Pode-se então definir o seguinte teste com o objetivo de selecionar possíveis bons vizinhos: se $lucroP(i, r) + pms \leq 0$, descarta-se a facilidade candidata a inserção i; caso contrário, i é um vizinho que deve ser investigado.

Três diferentes versões da busca local básica foram propostas incorporando esse teste na função *Primeiro_Aprimorante* e serão descritas a seguir.

3.3.1 Busca Local com Teste

A Figura 3.5 mostra a modificação realizada na função *Primeiro_Aprimorante*, incorporando o teste descrito acima. Por isso, essa otimização será denominada de busca local com teste. Essa função recebe como parâmetros a solução corrente composta pelo conjunto S de p facilidades abertas e pela árvore de Steiner $T(S)$ correspondente, além das estruturas *ganho*, *perda* e *extra*.

A função possui dois laços aninhados. O externo (linhas 2 a 18) percorre as facilidades pertencentes a solução corrente e o interno (linhas 8 a 17) percorre as facilidades que não estão na solução corrente. Ambos são executa-

```
Função Primeiro_Aprimorante_Teste(S, ganho, perda, extra)
1.  Melhoria ← Falso;
2.  Circ Para Todo (r ∈ S) e (Melhoria = Falso) Faça
3.      pms ← F_T(S) − F_T(S\{r});
4.      Se (pms < 0) Então
5.          T(S\{r}) ← T(S);
6.          pms ← 0;
7.      Fim-Se
8.      Circ Para Todo (i ∉ S) e (Melhoria = Falso) Faça
9.          lucroP(i, r) ← ganho(i) − perda(r) + extra(i, r);
10.         Se (lucroP(i, r) + pms > 0) Então
11.             lucroS(i, r) ← F_T(S) − F_T(S');
12.             lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
13.             Se (lucro(i, r) > 0) Então
14.                 Melhoria ← Verdadeiro;
15.             Fim-Se
16.         Fim-Se
17.     Fim-Circ-Para-Todo
18. Fim-Circ-Para-Todo
19. Retorne i, r, lucro(i, r);
Fim
```

Figura 3.5: Função que determina o primeiro vizinho aprimorante na busca local com teste.

dos até que seja encontrado o primeiro vizinho aprimorante ou então até que nenhum melhoramento seja possível.

Na linha 3 calcula-se o valor da possível melhoria de Steiner como a diferença entre o custo da árvore de Steiner corrente e o custo da árvore de Steiner obtida pela execução de Prim sem o vértice r. A abordagem utilizada para a construção da árvore de Steiner é heurística e, por isso, pode acontecer que, ao se retirar um determinado vértice r da árvore corrente, o custo da árvore resultante seja pior do que o custo da árvore corrente. Assim, *pms* terá um valor negativo. Caso isso aconteça (linha 4), a árvore $T(S\backslash\{r\})$ será a árvore corrente $T(S)$, passando r de vértice terminal a vértice de Steiner na árvore e fazendo *pms* igual a 0 (linhas 5 e 6).

O lucro obtido no problema das p-medianas é calculado na linha 9 e o teste que otimiza a busca local é realizado na linha 10. Se a facilidade candidata a inserção passa no teste, executa-se Prim com p facilidades trocando-se r por i e verifica-se o lucro de Steiner ao realizar essa troca em relação ao custo da árvore corrente (linha 11). Na linha 12 calcula-se o lucro total obtido. Se i não passa no teste, avalia-se a próxima facilidade candidata a inserção. Se $lucro(i, r) > 0$ (linha 13), encontra-se o primeiro vizinho aprimorante na vizinhança da solução corrente. Retorna-se na linha 19 as facilidades candidatas a inserção e remoção i e r, respectivamente, e o lucro total obtido.

Caso contrário, se $lucro(i, r) \leq 0$, retorna-se na linha 19, as facilidades i e r da última avaliação realizada e seu respectivo lucro.

3.3.2 Busca Local pelas Bordas

Intuitivamente, facilidades próximas à árvore de Steiner podem ser melhores candidatas a inserção do que facilidades mais distantes. A modificação realizada na função *Primeiro_Aprimorante* incorpora o teste definido anteriormente e pode ser assim explicada, em linhas gerais: se i não passa no teste, avalia-se a próxima facilidade candidata; se i passa no teste, verifica-se se a mesma é adjacente à árvore de Steiner $T(S\backslash\{r\})$. Caso i seja adjacente a $T(S\backslash\{r\})$, executa-se Prim com p facilidades trocando-se r por i; caso contrário avalia-se a próxima facilidade candidata. Diz-que a facilidade i é adjacente à árvore de Steiner $T(S\backslash\{r\})$ se possui, na sua lista de adjacências, algum nó pertencente à essa árvore. Por priorizar facilidades adjacentes à árvore, a otimização realizada será denominada de busca local pelas bordas.

A Figura 3.6 mostra a modificação realizada na função *Primeiro_Aprimorante* que recebe como parâmetros a solução corrente composta pelo conjunto S de p facilidades abertas e pela árvore de Steiner $T(S)$ correspondente, além das estruturas *ganho*, *perda* e *extra*.

A função possui dois laços aninhados. O externo (linhas 2 a 35) percorre as facilidades pertencentes a solução corrente e o interno (linhas 8 a 34) percorre as facilidades que não estão na solução corrente. Ambos são executados até que seja encontrado o primeiro vizinho aprimorante ou então até que nenhum melhoramento seja possível.

Como na seção anterior, calcula-se *pms* na linha 3, o lucro das p-medianas na linha 9 e executa-se o teste que otimiza a busca local na linha 10. Se i não passa no teste, avalia-se a próxima facilidade candidata a inserção; caso contrário, verifica-se se i pertence ou não à árvore $T(S\backslash\{r\})$; se a facilidade candidata não pertence à árvore (linha 11), o próximo passo é verificar se a facilidade é adjacente à mesma. Para isso basta que a facilidade i possua, na sua lista de adjacências ($Adj(i)$), um nó pertencente a $T(S\backslash\{r\})$ (linhas 12 a 17). Em caso afirmativo, executa-se Prim com p facilidades e calcula-se o lucro de Steiner (linha 19) e o lucro total obtido (linha 20); caso contrário, avalia-se a próxima facilidade candidata i. Caso i pertença à árvore $T(S\backslash\{r\})$, isto é, i é um vértice de Steiner nessa árvore, não é necessário executar Prim com p facilidades e o custo da árvore será igual ao custo de $T(S\backslash\{r\})$; assim, o lucro para o problema de Steiner é igual ao valor *pms* (linha 27). Se $lucro(i, r) > 0$ (linha 21 ou 29), encontra-se o primeiro vizinho aprimorante na vizinhança da solução corrente, retornando-se, na linha 36, as facilidades candidatas a inserção e remoção i e r, respectivamente, e o lucro total obtido. Caso contrário,

```
Função Primeiro_Aprimorante_B(S, ganho, perda, extra)
1.  Melhoria ← Falso;
2.  Circ Para Todo (r ∈ S) e (Melhoria = Falso) Faça
3.      pms ← F_T(S) − F_T(S\{r});
4.      Se (pms < 0) Então
5.          T(S\{r}) ← T(S);
6.          pms ← 0;
7.      Fim-Se
8.      Circ Para Todo (i ∉ S) e (Melhoria = Falso) Faça
9.          lucroP(i, r) ← ganho(i) − perda(r) + extra(i, r);
10.         Se (lucroP(i, r) + pms > 0) Então
11.             Se (i ∉ T(S\{r})) Então
12.                 Adj_Arvore ← Falso;
13.                 Para Todo (f ∈ Adj(i)) Faça
14.                     Se (f ∈ T(S\{r})) Então
15.                         Adj_Arvore ← Verdadeiro;
16.                     Fim-Se
17.                 Fim-Para-Todo
18.                 Se (Adj_Arvore = Verdadeiro) Então
19.                     lucroS(i, r) ← F_T(S) − F_T(S′);
20.                     lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
21.                     Se (lucro(i, r) > 0) Então
22.                         Melhoria ← Verdadeiro;
23.                     Fim-Se
24.                 Fim-Se
25.             Fim-Se
26.             Senão
27.                 lucroS(i, r) ← pms;
28.                 lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
29.                 Se (lucro(i, r) > 0) Então
30.                     Melhoria ← Verdadeiro;
31.                 Fim-Se
32.             Fim-Senão
33.         Fim-Se
34.     Fim-Circ-Para-Todo
35. Fim-Circ-Para-Todo
36. Retorne i, r, lucro(i, r);
Fim
```

Figura 3.6: Função que determina o primeiro vizinho aprimorante na busca local pelas bordas.

se $lucro(i, r) \leq 0$, retorna-se na linha 36, as facilidades i e r da última avaliação realizada e seu respectivo lucro.

3.3.3
Busca Local pelas Bordas com Teste da Menor Aresta

Essa modificação também prioriza facilidades que são adjacentes à árvore de Steiner $T(S\backslash\{r\})$. Adicionalmente, para a execução de Prim com p facilidades trocando-se r por i, o seguinte teste deve ser satisfeito: $MA - pms - lucroP(i, r) < 0$, onde MA é a aresta de menor custo que liga a facilidade i a $T(S\backslash\{r\})$. O valor $MA - pms < 0$ indica que a facilidade candidata i pode melhorar o valor da função objetivo referente ao problema de Steiner. Já o valor $lucroP(i, r) > 0$ indica que uma possível troca entre r e i melhora o valor da função objetivo referente ao problema das p-medianas. Assim, o teste $MA - pms - lucroP(i, r) < 0$ indica que a facilidade i é um possível bom vizinho a ser investigado.

A Figura 3.7 mostra a modificação realizada na função *Primeiro_Aprimorante* que recebe como parâmetros a solução corrente composta pelo conjunto S de p facilidades abertas e pela árvore de Steiner $T(S)$ correspondente, além das estruturas *ganho*, *perda* e *extra*.

A função possui dois laços aninhados. O externo (linhas 2 a 35) percorre as facilidades pertencentes a solução corrente e o interno (linhas 8 a 34) percorre as facilidades que não estão na solução corrente. Ambos são executados até que seja encontrado o primeiro vizinho aprimorante ou então até que nenhum melhoramento seja possível.

Como nas duas seções anteriores, calcula-se *pms* na linha 3, o lucro das p-medianas na linha 9 e executa-se o teste que otimiza a busca local na linha 10. Se i não passa no teste, avalia-se a próxima facilidade candidata a inserção. Caso contrário, testa-se se i pertence ou não à árvore $T(S\backslash\{r\})$ (linha 11). Se a facilidade candidata i não pertence à árvore, verifica-se a aresta de menor custo que liga a facilidade à árvore. Para isso, percorre-se a lista de adjacências de i ($Adj(i)$), procurando algum nó pertencente a $T(S\backslash\{r\})$. Em caso afirmativo, armazena-se a aresta de menor custo (linhas 12 a 17). Se o teste $MA - pms - lucroP(i, r) < 0$ for verdadeiro (linha 18), executa-se Prim com p facilidades e calcula-se o lucro de Steiner (linha 19) e o lucro total obtido (linha 20). Caso contrário, avalia-se a próxima facilidade i. Se não existe aresta que liga i à árvore ($MA = \infty$), então i não é adjacente à árvore e passa-se para a próxima facilidade candidata. Caso i pertença à árvore $T(S\backslash\{r\})$, não é necessário executar Prim e o custo da árvore será igual ao custo de $T(S\backslash\{r\})$. Assim, o lucro para o problema de Steiner é igual ao valor *pms* (linha 27). Se $lucro(i, r) > 0$ (linha 21 ou 29), encontra-se o primeiro vizinho aprimorante na vizinhança da solução corrente, retornando-se, na linha 36, as facilidades

```
Função Primeiro_Aprimorante_BTMA(S, ganho, perda, extra)
1.  Melhoria ← Falso;
2.  Circ Para Todo (r ∈ S) e (Melhoria = Falso) Faça
3.      pms ← F_T(S) − F_T(S\{r});
4.      Se (pms < 0) Então
5.          T(S\{r}) ← T(S);
6.          pms ← 0;
7.      Fim-Se
8.      Circ Para Todo (i ∉ S) e (Melhoria = Falso) Faça
9.          lucroP(i, r) ← ganho(i) − perda(r) + extra(i, r);
10.         Se (lucroP(i, r) + pms > 0) Então
11.             Se (i ∉ T(S\{r})) Então
12.                 MA ← ∞;
13.                 Para Todo (f ∈ Adj(i)) Faça
14.                     Se (f ∈ T(S\{r})) e (c_fi < MA) Então
15.                         MA ← c_fi;
16.                     Fim-Se
17.                 Fim-Para-Todo
18.                 Se (MA − pms − lucroP(i, r) < 0) Então
19.                     lucroS(i, r) ← F_T(S) − F_T(S′);
20.                     lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
21.                     Se (lucro(i, r) > 0) Então
22.                         Melhoria ← Verdadeiro;
23.                     Fim-Se
24.                 Fim-Se
25.             Fim-Se
26.             Senão
27.                 lucroS(i, r) ← pms;
28.                 lucro(i, r) ← lucroP(i, r) + lucroS(i, r);
29.                 Se (lucro(i, r) > 0) Então
30.                     Melhoria ← Verdadeiro;
31.                 Fim-Se
32.             Fim-Senão
33.         Fim-Se
34.     Fim-Circ-Para-Todo
35. Fim-Circ-Para-Todo
36. Retorne i, r, lucro(i, r);
Fim
```

Figura 3.7: Função que determina o primeiro vizinho aprimorante na busca local pelas bordas com teste da menor aresta.

candidatas a inserção e remoção i e r, respectivamente, e o lucro total obtido. Caso contrário, se $lucro(i, r) \leq 0$, retorna-se na linha 36, as facilidades i e r da última avaliação realizada e seu respectivo lucro.

3.3.4 Resultados Computacionais

Essa seção apresenta os resultados computacionais, comparando-se a busca local básica com as três variantes em termos de qualidade das soluções e tempos de processamento. As buscas locais que utilizam as funções apresentadas nas Figuras 3.4, 3.5, 3.6 e 3.7 serão denominadas, respectivamente, de BL_Básica, BL_Teste, BL_B e BL_BTMA. Para cada par instância-algoritmo, foram realizadas 15 execuções com soluções iniciais diferentes geradas inicializando-se o gerador de números aleatórios com sementes diferentes.

Ambiente de Teste, Instâncias e Medidas Utilizadas

A heurística construtiva e os algoritmos de busca local foram implementados em C com o parâmetro de otimização -O3 e executados em uma máquina Pentium IV 3.2 GHz com 1 Gbyte de memória RAM sob o sistema operacional Linux RedHat 9.0. O gerador de números aleatórios utilizado foi o de Matsumoto e Nishimura [53].

Para a comparação dos algoritmos de busca local, três classes de instâncias utilizadas no problema das p-medianas foram adaptadas para o problema híbrido. Devido as características explicadas no próximo parágrafo, serão denominadas de instâncias proporcionais.

A primeira classe é uma modificação das instâncias da ORLIB [5], denominada aqui de ORM_P, contendo 40 grafos com 100 a 900 vértices e p variando entre 5 e 200. A segunda classe é uma modificação das instâncias GR [18], denominada aqui de GRM_P. Contém dois grafos completos, com 100 e 150 vértices e valores de p entre 5 e 50 (resp. entre 5 e 60) para o primeiro (resp. segundo) grafo. A terceira classe é uma modificação das instâncias SL [55], denominada aqui de SLM_P. Essa classe contém três instâncias baseadas em grafos da ORM_P: SLM_P700 utiliza o mesmo grafo que ORM_P34, mas com $p = 233$; SLM_P800 utiliza o mesmo grafo que ORM_P37, mas com $p = 267$; e SLM_P900 utiliza o mesmo grafo que ORM_P40, mas com $p = 300$. Seja então um grafo $G = (V, E)$ com custos associados às arestas. O custo de servir cada usuário i a partir de cada facilidade j é dado pelo valor d_{ij} do caminho mais curto do vértice i ao vértice j no grafo G, calculado a partir dos custos associados às arestas. O custo de instalação c_{ij} de cada aresta é obtido multiplicando-se o custo da mesma pelo fator w, isto é, $c_{ij} = a_{ij} \times w$, onde a_{ij} é o custo associado à aresta (i, j) e w assume os valores 2, 5 e 10.

As Tabelas 3.1 a 3.3 mostram, respectivamente, as 40, 20 e três instâncias com seus respectivos número de vértices (n), número de arestas ($|E|$) e quantidade de facilidades a serem abertas (p). Multiplicando-se pelos três valores assumidos por w, tem-se um total de 189 problemas.

Instância	n	$\|E\|$	p	Instância	n	$\|E\|$	p
ORM_P1	100	198	5	ORM_P21	500	4.909	5
ORM_P2		193	10	ORM_P22		4.896	10
ORM_P3		198	10	ORM_P23		4.903	50
ORM_P4		196	20	ORM_P24		4.914	100
ORM_P5		196	33	ORM_P25		4.894	167
ORM_P6	200	786	5	ORM_P26	600	7.068	5
ORM_P7		779	10	ORM_P27		7.072	10
ORM_P8		792	20	ORM_P28		7.054	60
ORM_P9		785	40	ORM_P29		7.042	120
ORM_P10		786	67	ORM_P30		7.042	200
ORM_P11	300	1.772	5	ORM_P31	700	9.601	5
ORM_P12		1.758	10	ORM_P32		9.584	10
ORM_P13		1.760	30	ORM_P33		9.616	70
ORM_P14		1.771	60	ORM_P34		9.585	140
ORM_P15		1.754	100	ORM_P35	800	12.548	5
ORM_P16	400	3.153	5	ORM_P36		12.560	10
ORM_P17		3.142	10	ORM_P37		12.564	80
ORM_P18		3.134	40	ORM_P38	900	15.898	5
ORM_P19		3.134	80	ORM_P39		15.896	10
ORM_P20		3.144	133	ORM_P40		15.879	90

Tabela 3.1: Classe de instâncias ORM_P para o problemas das p-medianas conectadas.

Instância	n	$\|E\|$	p	Instância	n	$\|E\|$	p
GRM_P1	100	4.950	5	GRM_P11	150	11.175	10
GRM_P2			10	GRM_P12			15
GRM_P3			15	GRM_P13			20
GRM_P4			20	GRM_P14			25
GRM_P5			25	GRM_P15			30
GRM_P6			30	GRM_P16			35
GRM_P7			35	GRM_P17			40
GRM_P8			40	GRM_P18			45
GRM_P9			50	GRM_P19			50
GRM_P10	150	11.175	5	GRM_P20			60

Tabela 3.2: Classe de instâncias GRM_P para o problemas das p-medianas conectadas.

Na avaliação da qualidade das soluções encontradas, foram utilizadas três medidas relativas [48, 52, 61]:

– *Desvio relativo percentual médio (drpm)* - Dada uma instância, define-se o *desvio relativo percentual* de um método como a diferença percentual

Instância	n	$\|E\|$	p
SLM_P700	700	9.585	233
SLM_P800	800	12.564	267
SLM_P900	900	15.879	300

Tabela 3.3: Classe de instâncias SLM_P para o problemas das p-medianas conectadas.

entre a média do valor da solução encontrada pelo método e a média do valor da solução encontrada por todos os métodos. O *desvio relativo percentual médio* é a média dos desvios percentuais obtida pelo método para todas as instâncias do grupo.

– *Classificação média (cm)* - Para cada instância, os métodos são ordenados de acordo com o seu *desvio relativo percentual*. O melhor método possui classificação 1, o segundo melhor possui classificação 2 e assim sucessivamente. Em caso de empate, atribui-se a todos os métodos envolvidos a mesma classificação, calculada como a média das classificações individuais. Como exemplo, se os valores forem -0,03, -0,02, -0,02, 0,01, 0,03 e 0,03, as classificações de cada método serão: 1, 2,5, 2,5, 4, 5,5 e 5,5. A *classificação média* de um método é a média das classificações obtidas para todas as instâncias do grupo.

– *Melhor* - É o número de instâncias para as quais o método obteve o melhor (menor) valor entre todos os métodos, incluindo os casos de empate no primeiro lugar.

Quanto menor o valor da medida *drpm*, melhor o método. O valor da medida *cm* varia entre 1 e o número total de métodos utilizados na comparação. O valor 1 indica que o método é melhor em todas as instâncias do grupo. Assim, valores mais próximos de 1 são melhores. Já para a medida *melhor*, quanto maior seu valor, mais competitiva é a heurística.

Na avaliação dos tempos de processamento obtidos, a seguinte medida absoluta será utilizada:

– *Tempo absoluto médio (tam)* - Para cada instância, considera-se a média do tempo de processamento de um algoritmo em um determinado número de execuções.

Nas tabelas apresentadas no presente capítulo, os valores em negrito destacam o melhor resultado obtido pelas buscas locais nas medidas para cada instância ou grupo de instâncias testadas.

Qualidade das Soluções

As Tabelas 3.4 a 3.6 mostram os valores obtidos para as medidas relativas *drpm*, *cm* e *melhor* utilizadas na avaliação da qualidade das soluções encontradas pelas buscas locais para todos os grupos de instâncias testadas.

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	**-0,26**	**1,63**	26
	BL_Teste	-0,24	2,11	**27**
	BL_B	-0,02	2,29	21
	BL_BTMA	0,52	3,98	5
GRM_P (20 instâncias)	BL_Básica	-0,06	2,50	18
	BL_Teste	**-0,11**	**1,85**	**19**
	BL_B	**-0,11**	**1,85**	**19**
	BL_BTMA	0,28	3,80	11
SLM_P (3 instâncias)	BL_Básica	**-0,11**	1,67	1
	BL_Teste	-0,10	**1,33**	**2**
	BL_B	0,07	3,33	1
	BL_BTMA	0,14	4,00	0

Tabela 3.4: Qualidade relativa das buscas locais ($w = 2$).

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	-0,24	1,93	**32**
	BL_Teste	**-0,25**	**1,70**	31
	BL_B	-0,04	2,45	22
	BL_BTMA	0,53	3,93	6
GRM_P (20 instâncias)	BL_Básica	-0,04	2,73	**17**
	BL_Teste	**-0,08**	**1,78**	15
	BL_B	**-0,08**	**1,78**	15
	BL_BTMA	0,19	3,73	11
SLM_P (3 instâncias)	BL_Básica	**-0,06**	**2,00**	1
	BL_Teste	-0,01	**2,00**	**2**
	BL_B	-0,04	**2,00**	1
	BL_BTMA	0,11	4,00	0

Tabela 3.5: Qualidade relativa das buscas locais ($w = 5$).

Para a coluna *drpm*, primeiramente pode-se notar que BL_BTMA obtém os piores resultados para todas as instâncias testadas. Comparando-se os outros métodos entre si, há um grande equilíbrio nos testes. BL_Básica é melhor em cinco grupos de instâncias: $w = 2$ (ORM_P e SLM_P), $w = 5$ (SLM_P) e $w = 10$ (ORM_P e SLM_P). BL_Teste é melhor em quatro grupos de instâncias: $w = 2$ (GRM_P), $w = 5$ (ORM_P e GRM_P) e $w = 10$ (GRM_P), enquanto BL_B é melhor em três grupos de instâncias: $w = 2$, $w = 5$ e $w = 10$ (instâncias GRM_P).

A medida relativa *cm* novamente indica que os piores resultados são obtidos pela BL_BTMA com os valores para todos os grupos de instâncias

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	**-0,30**	**1,60**	**34**
	BL_Teste	-0,21	1,93	30
	BL_B	0,40	2,90	14
	BL_BTMA	0,51	3,58	10
GRM_P (20 instâncias)	BL_Básica	-0,05	2,70	**19**
	BL_Teste	**-0,54**	**1,85**	17
	BL_B	**-0,54**	**1,85**	17
	BL_BTMA	0,26	3,60	16
SLM_P (3 instâncias)	BL_Básica	**-0,28**	**1,67**	**2**
	BL_Teste	-0,11	2,33	0
	BL_B	-0,15	2,00	1
	BL_BTMA	0,54	4,00	0

Tabela 3.6: Qualidade relativa das buscas locais ($w = 10$).

muito próximos ou iguais a 4 (o menor valor apresentado é 3,58 no grupo ORM_P, $w = 10$). Novamente, existe um equilíbrio comparando-se os outros métodos. BL_Básica é melhor em quatro grupos de instâncias: $w = 2$ (ORM_P), $w = 5$ (SLM_P) e $w = 10$ (ORM_P e SLM_P). BL_Teste é melhor em seis grupos de instâncias: $w = 2$ (GRM_P e SLM_P), $w = 5$ (ORM_P, GRM_P e SLM_P) e $w = 10$ (GRM_P), enquanto BL_B é melhor em quatro grupos de instâncias: $w = 2$ (GRM_P), $w = 5$ (GRM_P e SLM_P) e $w = 10$ (GRM_P).

A coluna *melhor* mostra um equilíbrio entre as buscas BL_Básica e BL_Teste. A primeira é superior às demais em cinco grupos de instâncias: $w = 5$ (ORM_P e GRM_P) e $w = 10$ (ORM_P, GRM_P e SLM_P) enquanto a segunda é superior às demais em quatro grupos de instâncias: $w = 2$ (ORM_P, GRM_P e SLM_P) e $w = 5$ (SLM_P). BL_B é superior às demais somente no grupo $w = 2$ (GRM_P). Novamente, BL_BTMA apresenta os piores resultados.

Assim, analisando-se a qualidade das soluções encontradas pelas buscas locais nas três medidas relativas, destacam-se BL_Básica e BL_Teste, com uma pequena diferença favorecendo uma ou outra busca nas instâncias testadas. Isso vem mostrar que o teste realizado é muito eficiente, descartando vizinhos que não melhorariam a solução corrente e evitando, assim, o trabalho desnecessário ao se executar Prim quando troca-se a facilidade candidata a remoção r pela facilidade candidata a inserção i. Para ressaltar esses resultados, as Tabelas 3.7 a 3.15 mostram a quantidade total média de vizinhos e o percentual médio de vizinhos descartados (coluna Desc.(%)) e avaliados (coluna Aval.(%)) pela busca BL_Teste em 15 iterações realizadas para cada instância.

Os resultados mostram que, em média, para quase todas as instâncias testadas, ocorre o descarte de vizinhos, com exceção de duas delas (GRM_P9 com $w = 2$ e $w = 5$) onde todos os vizinhos são avaliados. Em geral, observou-se que, quando p cresce, mantendo-se fixo o número de vértices do grafo ou quando aumenta-se o fator w, diminui o número de vizinhos descartados. Ocorre esse

mesmo padrão de comportamento nas buscas locais BL_B e BL_BTMA. BL_B descarta, em geral, uma quantidade média de vizinhos um pouco maior do que BL_Teste, pois a facilidade vizinha candidata a inserção deve satisfazer o teste que otimiza a busca local e ser adjacente à árvore de Steiner $T(S\backslash\{r\})$. Já BL_BTMA descarta uma quantidade média de vizinhos bem maior do que BL_Teste e BL_B, pois além dos mesmos requisitos apresentados por BL_B, para a facilidade candidata a inserção, o teste $MA - pms - lucroP(i, r) < 0$ deve ser satisfeito. Em média, BL_BTMA descarta 99% dos vizinhos em todas as instâncias testadas.

Por último, BL_B apresenta rendimento próximo ao das duas melhores buscas nas medidas *drpm* e *cm* e um rendimento bem pior na medida *melhor*. Já BL_BTMA apresenta os piores resultados em todas as instâncias testadas nas três medidas relativas em comparação aos outros algoritmos.

Tempos de Processamento

A complexidade de pior caso da BL_Básica e de suas versões é: $O(p^2\,|E|\,n\log n)$. Para a comparação dos tempos de processamento obtidos pelas buscas locais, serão utilizados os tempos médios de execução em segundos (medida *tam*). As Tabelas 3.16 a 3.24 apresentam esses resultados.

Claramente observa-se que BL_BTMA é muito mais rápida do que as demais em todas as instâncias testadas devido à grande quantidade de vizinhos descartados por essa busca. BL_Básica é muito mais lenta do que as demais em todos os problemas, principalmente nas maiores instâncias da classe ORM_P e nas instâncias SLM_P, pois não apresenta o teste que otimiza a busca local, tendo que executar Prim a cada troca de uma facilidade candidata a remoção por uma facilidade candidata a inserção. Comparando-se BL_B com BL_Teste, tem-se que, em geral, a primeira apresenta tempos de processamento menores do que a segunda, porém sem ganhos significativos. Para as buscas locais com descarte de vizinhos, quando aumenta-se o fator w, em geral, aumenta-se o número de vizinhos avaliados, e conseqüentemente, aumenta-se o tempo de processamento de cada instância.

Assim, em termos de tempo de processamento, destaca-se o algoritmo de BL_BTMA, seguido, com uma pequena diferença entre eles, por BL_B e BL_Teste.

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
ORM_P1	1.129,93	1.058,40	93,67	71,53	6,33
ORM_P2	2.850,33	2.654,53	93,13	195,80	6,87
ORM_P3	3.575,73	3.343,53	93,51	232,20	6,49
ORM_P4	6.147,73	4.363,20	70,97	1.784,53	29,03
ORM_P5	9.938,00	4.562,00	45,90	5.376,00	54,10
ORM_P6	2.890,00	2.830,13	97,93	59,86	2,07
ORM_P7	7.458,46	7.271,93	97,50	186,53	2,50
ORM_P8	13.727,40	12.727,00	92,71	1.000,40	7,29
ORM_P9	30.663,80	23.709,00	77,32	6.954,80	22,68
ORM_P10	42.895,20	20.631,26	48,10	22.263,93	51,90
ORM_P11	3.937,60	3.910,93	99,32	26,66	0,68
ORM_P12	7.923,33	7.861,60	99,22	61,73	0,78
ORM_P13	39.199,73	36.690,53	93,60	2.509,20	6,40
ORM_P14	74.429,40	58.667,00	78,82	15.762,40	21,18
ORM_P15	96.581,53	45.285,06	46,89	51.296,46	53,11
ORM_P16	5.310,86	5.285,33	99,52	25,53	0,48
ORM_P17	16.239,46	16.175,33	99,61	64,13	0,39
ORM_P18	62.846,26	59.940,20	95,38	2.906,06	4,62
ORM_P19	135.898,13	116.377,93	85,64	19.520,20	14,36
ORM_P20	187.727,20	98.911,46	52,69	88.815,73	47,31
ORM_P21	8.035,93	8.024,33	99,86	11,60	0,14
ORM_P22	16.089,20	16.031,86	99,64	57,33	0,36
ORM_P23	105.831,66	101.966,80	96,35	3.864,86	3,65
ORM_P24	240.910,86	199.656,00	82,88	41.254,86	17,12
ORM_P25	257.896,33	131.028,93	50,81	126.867,40	49,19
ORM_P26	8.592,20	8.576,80	99,82	15,40	0,18
ORM_P27	20.669,73	20.596,66	99,65	73,06	0,35
ORM_P28	174.513,93	168.481,53	96,54	6.032,40	3,46
ORM_P29	333.694,13	280.870,46	84,17	52.823,66	15,83
ORM_P30	392.624,20	207.826,93	52,93	184.797,26	47,07
ORM_P31	8.720,80	8.703,26	99,80	17,53	0,20
ORM_P32	24.588,46	24.548,80	99,84	39,66	0,16
ORM_P33	241.098,53	235.402,33	97,64	5.696,20	2,36
ORM_P34	488.564,53	413.862,66	84,71	74.701,86	15,29
ORM_P35	12.122,26	12.108,53	99,89	13,73	0,11
ORM_P36	27.591,20	27.541,13	99,82	50,06	0,18
ORM_P37	324.865,93	318.233,46	97,96	6.632,46	2,04
ORM_P38	10.406,53	10.392,46	99,86	14,06	0,14
ORM_P39	30.708,46	30.666,20	99,86	42,26	0,14
ORM_P40	505.554,80	488.991,26	96,72	16.563,53	3,28

Tabela 3.7: Quantidade média de vizinhos em 15 iterações - ORM_P ($w = 2$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
ORM_P1	1.381,13	1.162,20	84,15	218,93	15,85
ORM_P2	3.504,00	2.005,73	57,24	1.498,26	42,76
ORM_P3	3.623,60	2.252,86	62,17	1.370,73	37,83
ORM_P4	7.725,00	2.342,46	30,32	5.382,53	69,68
ORM_P5	9.817,13	1.587,13	16,17	8.230,00	83,83
ORM_P6	2.513,60	2.346,80	93,36	166,80	6,64
ORM_P7	7.890,13	6.853,53	86,86	1.036,60	13,14
ORM_P8	16.350,80	11.264,20	68,89	5.086,60	31,11
ORM_P9	29.603,46	10.933,60	36,93	18.669,86	63,07
ORM_P10	44.743,20	6.378,00	14,25	38.365,20	85,75
ORM_P11	4.264,40	4.194,53	98,36	69,86	1,64
ORM_P12	9.279,60	8.958,46	96,54	321,13	3,46
ORM_P13	39.587,26	22.537,46	56,93	17.049,80	43,07
ORM_P14	80.882,60	27.768,33	34,33	53.114,26	65,67
ORM_P15	135.787,00	14.308,33	10,54	121.478,66	89,46
ORM_P16	5.041,00	4.987,13	98,93	53,86	1,07
ORM_P17	13.458,13	13.298,13	98,81	160,00	1,19
ORM_P18	84.990,40	56.888,86	66,94	28.101,53	33,06
ORM_P19	139.017,53	41.875,20	30,12	97.142,33	69,88
ORM_P20	221.205,26	32.189,26	14,55	189.016,00	85,45
ORM_P21	8.099,66	8.056,73	99,47	42,93	0,53
ORM_P22	17.073,46	16.850,86	98,70	222,60	1,30
ORM_P23	133.262,80	93.423,46	70,10	39.839,33	29,90
ORM_P24	237.441,06	74.697,46	31,46	162.743,60	68,54
ORM_P25	333.129,86	44.780,20	13,44	288.349,66	86,58
ORM_P26	8.843,73	8.806,86	99,58	36,86	0,42
ORM_P27	20.046,20	19.801,26	98,78	244,93	1,22
ORM_P28	214.298,26	150.419,86	70,19	63.878,40	29,81
ORM_P29	313.796,20	96.069,93	30,62	217.726,26	69,38
ORM_P30	491.802,20	77.931,00	15,85	413.871,20	84,15
ORM_P31	8.880,73	8.846,46	99,61	34,26	0,39
ORM_P32	20.586,86	20.494,86	99,55	92,00	0,45
ORM_P33	263.389,66	197.459,60	74,97	65.930,06	25,03
ORM_P34	474.135,06	162.535,06	34,28	311.600,00	65,72
ORM_P35	11.667,13	11.648,00	99,84	19,13	0,16
ORM_P36	29.294,26	29.124,86	99,42	169,40	0,58
ORM_P37	329.428,66	256.243,73	77,78	73.184,93	22,22
ORM_P38	11.443,20	11.405,80	99,67	37,40	0,33
ORM_P39	27.452,93	27.352,60	99,63	100,33	0,37
ORM_P40	455.507,13	335.938,73	73,75	119.568,40	26,25

Tabela 3.8: Quantidade média de vizinhos em 15 iterações - ORM_P ($w = 5$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
ORM_P1	1.355,20	747,26	55,14	607,93	44,86
ORM_P2	3.650,40	637,13	17,45	3.013,26	82,55
ORM_P3	3.188,33	1.170,33	36,71	2.018,00	63,29
ORM_P4	7.282,86	1.069,13	14,68	6.213,73	85,32
ORM_P5	11.339,13	1.048,00	9,24	10.291,13	90,76
ORM_P6	2.610,33	2.133,93	81,75	476,40	18,25
ORM_P7	7.337,46	4.312,00	58,77	3.025,46	41,23
ORM_P8	18.643,53	4.315,93	23,15	14.327,60	76,85
ORM_P9	36.492,73	5.112,00	14,00	31.380,73	85,99
ORM_P10	51.711,40	5.327,33	10,30	46.384,06	89,70
ORM_P11	4.872,60	4.584,46	94,09	288,13	5,91
ORM_P12	12.365,00	10.994,46	88,92	1.370,53	11,08
ORM_P13	49.140,80	10.074,13	20,50	39.066,66	79,50
ORM_P14	93.939,06	15.114,40	16,09	78.824,66	83,91
ORM_P15	150.449,33	11.467,66	7,62	138.981,66	92,38
ORM_P16	4.974,93	4.874,20	97,98	100,73	2,02
ORM_P17	14.515,66	13.794,73	95,03	720,93	4,97
ORM_P18	75.813,73	21.261,66	28,04	54.552,06	71,96
ORM_P19	177.183,20	24.111,06	13,61	153.072,13	86,39
ORM_P20	245.088,66	20.541,33	8,38	224.547,33	91,62
ORM_P21	8.184,13	8.064,86	98,54	119,26	1,46
ORM_P22	21.385,33	20.495,73	95,84	889,60	4,16
ORM_P23	129.565,40	42.742,73	32,99	86.822,66	67,01
ORM_P24	267.686,20	39.290,93	14,68	228.395,26	85,32
ORM_P25	377.673,13	32.902,93	8,71	344.770,20	91,29
ORM_P26	8.466,20	8.379,86	98,98	86,33	1,02
ORM_P27	20.227,73	19.375,60	95,79	852,13	4,21
ORM_P28	209.621,26	66.124,66	31,54	143.496,60	68,46
ORM_P29	382.507,86	53.714,86	14,04	328.793,00	85,96
ORM_P30	631.629,46	64.224,00	10,17	567.405,46	89,83
ORM_P31	9.747,80	9.670,00	99,20	77,80	0,80
ORM_P32	21.347,13	21.064,80	98,68	282,33	1,32
ORM_P33	265.892,80	81.509,33	30,65	184.383,46	69,35
ORM_P34	543.119,73	89.044,13	16,39	454.075,60	83,61
ORM_P35	8.439,46	8.399,33	99,52	40,13	0,48
ORM_P36	28.282,93	27.610,66	97,62	672,26	2,38
ORM_P37	349.121,40	129.600,86	37,12	219.520,53	62,88
ORM_P38	11.655,33	11.581,06	99,36	74,26	0,64
ORM_P39	27.916,26	27.562,13	98,73	354,13	1,27
ORM_P40	476.902,40	165.215,13	34,64	311.687,26	65,36

Tabela 3.9: Quantidade média de vizinhos em 15 iterações - ORM_P ($w = 10$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
GRM_P1	1.425,20	1.304,40	91,52	120,80	8,48
GRM_P2	2.947,06	2.705,46	91,80	241,60	8,20
GRM_P3	6.894,26	5.722,06	83,00	1.172,20	17,00
GRM_P4	7.647,80	3.475,33	45,44	4.172,46	54,56
GRM_P5	9.762,80	2.049,13	20,99	7.713,66	79,01
GRM_P6	7.866,40	739,46	9,40	7.126,93	90,60
GRM_P7	8.301,66	382,46	4,61	7.919,20	95,39
GRM_P8	8.615,86	223,60	2,60	8.392,26	97,40
GRM_P9	6.239,00	0,00	0,00	6.239,00	100,00
GRM_P10	1.572,86	1.525,13	96,97	47,73	3,03
GRM_P11	5.348,86	4.900,33	91,61	448,53	8,39
GRM_P12	8.709,33	7.893,33	90,63	816,00	9,37
GRM_P13	13.314,66	11.867,86	89,13	1.446,80	10,87
GRM_P14	16.273,20	13.040,53	80,14	3.232,66	19,86
GRM_P15	17.428,66	11.568,46	66,38	5.860,20	33,62
GRM_P16	22.439,06	11.085,46	49,40	11.353,60	50,60
GRM_P17	23.661,26	8.815,46	37,26	14.845,80	62,74
GRM_P18	24.789,53	6.597,20	26,61	18.192,33	73,39
GRM_P19	22.885,66	4.103,80	17,93	18.781,86	82,07
GRM_P20	24.877,66	2.114,33	8,50	22.763,33	91,50

Tabela 3.10: Quantidade média de vizinhos em 15 iterações - GRM_P ($w = 2$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
GRM_P1	1.700,33	1.053,73	61,97	646,60	38,03
GRM_P2	3.315,00	1.940,80	58,55	1.374,20	41,45
GRM_P3	5.765,06	2.744,86	47,61	3.020,20	52,39
GRM_P4	8.221,66	1.893,53	23,03	6.328,13	76,97
GRM_P5	6.736,00	930,00	13,81	5.806,00	86,19
GRM_P6	6.860,26	368,33	5,37	6.491,93	94,63
GRM_P7	8.919,93	156,73	1,76	8.763,20	98,24
GRM_P8	7.645,66	42,26	0,55	7.603,40	99,45
GRM_P9	6.403,00	0,00	0,00	6.403,00	100,00
GRM_P10	2.170,46	1.814,26	83,59	356,20	16,41
GRM_P11	5.974,26	4.222,33	70,68	1.751,93	29,32
GRM_P12	10.577,93	6.003,13	56,75	4.574,80	43,25
GRM_P13	12.275,13	5.953,46	48,50	6.321,66	51,50
GRM_P14	14.534,86	6.542,46	45,01	7.992,40	54,99
GRM_P15	20.085,66	6.924,93	34,48	13.160,73	65,52
GRM_P16	19.496,66	5.342,46	27,40	14.154,20	72,60
GRM_P17	21.362,26	4.879,60	22,84	16.482,66	77,16
GRM_P18	23.552,40	4.109,93	17,45	19.442,46	82,55
GRM_P19	25.591,86	2.838,53	11,09	22.753,33	88,91
GRM_P20	26.403,00	1.043,13	3,95	25.359,86	96,05

Tabela 3.11: Quantidade média de vizinhos em 15 iterações - GRM_P ($w = 5$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
GRM_P1	1.451,33	634,46	43,72	816,86	56,28
GRM_P2	3.421,46	1.534,20	44,84	1.887,26	55,16
GRM_P3	6.045,86	2.560,06	42,34	3.485,80	57,66
GRM_P4	8.591,20	1.804,00	21,00	6.787,20	79,00
GRM_P5	6.690,20	812,26	12,14	5.877,93	87,86
GRM_P6	8.791,13	585,00	6,65	8.206,13	93,35
GRM_P7	7.348,26	166,86	2,27	7.181,40	97,73
GRM_P8	8.320,46	62,66	0,75	8.257,80	99,25
GRM_P9	15.945,00	21,00	0,13	15.924,00	99,87
GRM_P10	2.569,60	1.633,86	63,58	935,73	36,42
GRM_P11	6.458,66	3.301,20	51,11	3.157,46	48,89
GRM_P12	10.576,86	4.663,13	44,09	5.913,73	55,91
GRM_P13	13.350,80	5.506,93	41,25	7.843,86	58,75
GRM_P14	18.605,86	6.880,13	36,98	11.725,73	63,02
GRM_P15	18.673,73	5.442,26	29,14	13.231,46	70,86
GRM_P16	20.380,33	4.851,00	23,80	15.529,33	76,20
GRM_P17	24.361,20	5.094,73	20,91	19.266,46	79,09
GRM_P18	21.486,00	3.273,66	15,24	18.212,33	84,76
GRM_P19	26.423,00	2.383,13	9,02	24.039,86	90,98
GRM_P20	28.064,40	1.093,66	3,90	26.970,73	96,10

Tabela 3.12: Quantidade média de vizinhos em 15 iterações - GRM_P ($w = 10$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
SLM_P1	528.941,40	285.950,86	54,06	242.990,53	45,94
SLM_P2	951.414,40	565.581,13	59,45	385.833,26	40,55
SLM_P3	1.447.763,80	862.978,86	59,61	584.784,93	40,39

Tabela 3.13: Quantidade média de vizinhos em 15 iterações - SLM_P ($w = 2$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
SLM_P1	633.183,26	111.967,66	17,68	521.215,60	82,32
SLM_P2	915.941,66	210.897,26	23,03	705.044,40	76,97
SLM_P3	1.131.112,33	228.465,46	20,20	902.646,86	79,80

Tabela 3.14: Quantidade média de vizinhos em 15 iterações - SLM_P ($w = 5$).

Instância	Vizinhos	Descartados	Desc.(%)	Avaliados	Aval.(%)
SLM_P1	508.708,00	275.262,93	54,11	233.445,06	45,89
SLM_P2	959.139,93	596.434,33	62,18	362.705,60	37,82
SLM_P3	1.318.937,20	817.906,40	62,01	501.030,80	37,99

Tabela 3.15: Quantidade média de vizinhos em 15 iterações - SLM_P ($w = 10$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
ORM_P1	0,04	**0,00**	**0,00**	**0,00**
ORM_P2	0,10	0,01	0,01	**0,00**
ORM_P3	0,07	0,02	0,01	**0,00**
ORM_P4	0,23	0,08	0,05	**0,00**
ORM_P5	0,58	0,23	0,19	**0,01**
ORM_P6	0,22	0,02	**0,01**	**0,01**
ORM_P7	0,52	0,04	0,03	**0,01**
ORM_P8	1,22	0,14	0,10	**0,03**
ORM_P9	2,95	0,85	0,83	**0,05**
ORM_P10	8,61	2,60	2,36	**0,09**
ORM_P11	0,39	0,02	**0,01**	**0,01**
ORM_P12	1,30	0,03	0,03	**0,02**
ORM_P13	6,36	0,57	0,52	**0,06**
ORM_P14	19,41	3,45	3,22	**0,14**
ORM_P15	39,00	11,90	12,26	**0,23**
ORM_P16	0,66	0,04	**0,03**	**0,03**
ORM_P17	2,39	0,06	0,05	**0,04**
ORM_P18	21,52	1,22	1,06	**0,13**
ORM_P19	57,22	8,40	8,13	**0,28**
ORM_P20	93,43	38,43	31,80	**0,47**
ORM_P21	1,22	0,06	**0,05**	**0,05**
ORM_P22	4,22	0,08	0,08	**0,06**
ORM_P23	49,18	1,95	2,01	**0,23**
ORM_P24	114,18	23,63	21,27	**0,58**
ORM_P25	179,64	74,81	78,56	**0,94**
ORM_P26	2,23	0,08	**0,07**	**0,07**
ORM_P27	7,42	0,13	0,12	**0,08**
ORM_P28	109,14	3,96	4,71	**0,41**
ORM_P29	250,12	41,66	43,78	**1,02**
ORM_P30	328,84	145,29	147,34	**1,68**
ORM_P31	3,55	0,11	0,10	**0,09**
ORM_P32	11,13	0,15	0,15	**0,12**
ORM_P33	158,45	5,33	5,20	**0,63**
ORM_P34	461,26	79,16	80,46	**1,60**
ORM_P35	4,15	0,14	0,13	**0,12**
ORM_P36	17,28	0,22	0,20	**0,15**
ORM_P37	345,57	8,57	9,63	**0,86**
ORM_P38	7,59	0,17	0,17	**0,14**
ORM_P39	17,64	0,25	0,25	**0,19**
ORM_P40	503,35	26,70	21,40	**1,40**

Tabela 3.16: Tempos de execução em segundos *tam* - ORM_P ($w = 2$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
ORM_P1	0,04	0,01	**0,00**	**0,00**
ORM_P2	0,08	0,05	0,02	**0,00**
ORM_P3	0,08	0,05	0,02	**0,00**
ORM_P4	0,36	0,17	0,09	**0,01**
ORM_P5	0,92	0,30	0,27	**0,02**
ORM_P6	0,15	0,02	0,02	**0,01**
ORM_P7	0,45	0,11	0,06	**0,02**
ORM_P8	1,24	0,50	0,35	**0,03**
ORM_P9	4,81	1,82	1,57	**0,06**
ORM_P10	15,48	3,85	3,75	**0,10**
ORM_P11	0,32	0,03	0,02	**0,01**
ORM_P12	1,45	0,07	0,05	**0,02**
ORM_P13	6,44	2,81	2,20	**0,06**
ORM_P14	22,95	9,26	7,88	**0,16**
ORM_P15	74,74	26,68	22,38	**0,29**
ORM_P16	0,67	0,04	**0,03**	**0,03**
ORM_P17	2,39	0,08	0,07	**0,04**
ORM_P18	21,21	7,89	7,21	**0,17**
ORM_P19	70,87	31,28	35,07	**0,33**
ORM_P20	166,28	69,43	65,38	**0,59**
ORM_P21	1,18	0,06	0,06	**0,05**
ORM_P22	3,91	0,13	0,11	**0,06**
ORM_P23	45,59	16,06	12,89	**0,29**
ORM_P24	140,41	69,84	76,02	**0,69**
ORM_P25	334,31	145,56	146,73	**1,24**
ORM_P26	1,89	0,09	**0,07**	**0,07**
ORM_P27	5,93	0,19	0,14	**0,08**
ORM_P28	91,86	36,93	30,18	**0,48**
ORM_P29	283,19	136,87	151,61	**1,16**
ORM_P30	670,77	286,64	263,79	**1,98**
ORM_P31	2,96	0,12	0,10	**0,09**
ORM_P32	8,04	0,18	0,15	**0,11**
ORM_P33	166,04	53,38	55,88	**0,84**
ORM_P34	428,22	247,81	243,00	**1,91**
ORM_P35	4,64	0,14	0,13	**0,12**
ORM_P36	13,97	0,29	0,24	**0,16**
ORM_P37	289,71	80,17	93,04	**1,23**
ORM_P38	7,85	0,20	0,17	**0,15**
ORM_P39	14,34	0,29	0,26	**0,20**
ORM_P40	485,50	156,13	160,21	**1,97**

Tabela 3.17: Tempos de execução em segundos *tam* - ORM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
ORM_P1	0,02	0,02	**0,00**	**0,00**
ORM_P2	0,11	0,07	0,02	**0,00**
ORM_P3	0,09	0,06	0,02	**0,00**
ORM_P4	0,38	0,17	0,11	**0,01**
ORM_P5	0,86	0,36	0,25	**0,02**
ORM_P6	0,15	0,04	0,02	**0,01**
ORM_P7	0,44	0,22	0,08	**0,02**
ORM_P8	1,76	1,01	0,57	**0,03**
ORM_P9	5,94	2,55	1,71	**0,06**
ORM_P10	17,33	4,67	4,38	**0,10**
ORM_P11	0,32	0,05	0,03	**0,02**
ORM_P12	0,97	0,21	0,10	**0,03**
ORM_P13	9,04	4,70	3,19	**0,07**
ORM_P14	32,78	12,73	12,11	**0,17**
ORM_P15	90,19	30,01	28,02	**0,29**
ORM_P16	0,68	0,04	**0,03**	**0,03**
ORM_P17	2,03	0,19	0,13	**0,04**
ORM_P18	24,71	12,34	12,85	**0,16**
ORM_P19	102,98	45,96	40,56	**0,39**
ORM_P20	251,87	81,35	76,02	**0,67**
ORM_P21	1,25	0,08	0,07	**0,05**
ORM_P22	3,88	0,30	0,18	**0,07**
ORM_P23	67,17	28,97	27,93	**0,34**
ORM_P24	249,32	93,15	90,53	**0,78**
ORM_P25	541,13	168,75	193,96	**1,36**
ORM_P26	2,24	0,10	0,09	**0,07**
ORM_P27	5,78	0,41	0,31	**0,09**
ORM_P28	118,80	66,60	54,17	**0,56**
ORM_P29	435,11	185,42	181,79	**1,32**
ORM_P30	1.001,61	379,25	374,18	**2,29**
ORM_P31	3,34	0,14	0,11	**0,09**
ORM_P32	8,70	0,27	0,22	**0,11**
ORM_P33	222,22	113,79	104,48	**0,86**
ORM_P34	784,10	323,75	286,87	**2,06**
ORM_P35	4,30	0,15	0,13	**0,12**
ORM_P36	14,73	0,59	0,45	**0,16**
ORM_P37	413,42	194,84	177,49	**1,33**
ORM_P38	7,88	0,22	0,20	**0,14**
ORM_P39	21,47	0,48	0,45	**0,19**
ORM_P40	694,46	314,49	317,05	**1,84**

Tabela 3.18: Tempos de execução em segundos *tam* - ORM_P ($w = 10$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
GRM_P1	0,10	0,02	0,03	**0,00**
GRM_P2	0,24	0,05	0,05	**0,01**
GRM_P3	0,40	0,18	0,17	**0,02**
GRM_P4	0,62	0,57	0,58	**0,03**
GRM_P5	1,01	1,09	1,10	**0,04**
GRM_P6	1,33	1,07	1,08	**0,05**
GRM_P7	1,67	1,26	1,28	**0,05**
GRM_P8	1,84	1,37	1,38	**0,06**
GRM_P9	2,07	1,07	1,08	**0,06**
GRM_P10	0,38	0,02	0,02	**0,01**
GRM_P11	0,94	0,15	0,15	**0,02**
GRM_P12	1,69	0,30	0,29	**0,04**
GRM_P13	3,10	0,55	0,55	**0,06**
GRM_P14	3,53	1,21	1,21	**0,08**
GRM_P15	4,50	2,24	2,25	**0,13**
GRM_P16	4,78	4,43	4,44	**0,14**
GRM_P17	7,41	6,25	6,04	**0,17**
GRM_P18	8,76	8,27	8,06	**0,18**
GRM_P19	10,41	8,88	8,74	**0,24**
GRM_P20	17,16	11,18	11,00	**0,23**

Tabela 3.19: Tempos de execução em segundos *tam* - GRM_P ($w = 2$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
GRM_P1	0,07	0,07	0,07	**0,01**
GRM_P2	0,22	0,16	0,16	**0,01**
GRM_P3	0,40	0,37	0,37	**0,02**
GRM_P4	0,57	0,76	0,76	**0,03**
GRM_P5	0,89	0,75	0,72	**0,04**
GRM_P6	1,03	0,84	0,84	**0,04**
GRM_P7	1,48	1,15	1,16	**0,05**
GRM_P8	1,87	1,06	1,07	**0,05**
GRM_P9	1,78	1,01	1,02	**0,06**
GRM_P10	0,31	0,08	0,09	**0,01**
GRM_P11	0,95	0,45	0,46	**0,03**
GRM_P12	1,89	1,27	1,27	**0,05**
GRM_P13	3,21	1,94	1,95	**0,08**
GRM_P14	3,57	2,61	2,64	**0,09**
GRM_P15	4,55	4,22	4,26	**0,11**
GRM_P16	4,78	4,64	4,69	**0,15**
GRM_P17	6,83	5,77	6,06	**0,18**
GRM_P18	7,77	7,04	7,77	**0,19**
GRM_P19	12,49	8,67	10,00	**0,21**
GRM_P20	18,32	11,24	12,01	**0,28**

Tabela 3.20: Tempos de execução em segundos *tam* - GRM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
GRM_P1	0,08	0,07	0,07	**0,01**
GRM_P2	0,24	0,20	0,21	**0,01**
GRM_P3	0,46	0,39	0,39	**0,02**
GRM_P4	0,65	0,76	0,76	**0,03**
GRM_P5	0,93	0,71	0,71	**0,03**
GRM_P6	1,18	1,02	1,02	**0,04**
GRM_P7	1,45	0,94	0,94	**0,05**
GRM_P8	1,90	1,13	1,13	**0,05**
GRM_P9	2,54	2,58	2,66	**0,07**
GRM_P10	0,31	0,19	0,20	**0,01**
GRM_P11	0,87	0,73	0,73	**0,03**
GRM_P12	1,73	1,55	1,63	**0,05**
GRM_P13	2,81	2,21	2,46	**0,08**
GRM_P14	3,52	3,58	4,00	**0,11**
GRM_P15	4,81	4,05	4,09	**0,13**
GRM_P16	5,33	4,99	5,09	**0,14**
GRM_P17	7,92	6,34	6,69	**0,16**
GRM_P18	9,54	6,78	7,03	**0,17**
GRM_P19	11,90	9,50	9,76	**0,20**
GRM_P20	21,33	11,54	11,82	**0,25**

Tabela 3.21: Tempos de execução em segundos *tam* - GRM_P ($w = 10$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
SLM_P1	658,54	248,46	249,98	**2,45**
SLM_P2	1.168,80	512,17	503,11	**3,66**
SLM_P3	1.462,00	1.030,78	837,21	**6,19**

Tabela 3.22: Tempos de execução em segundos *tam* - SLM_P ($w = 2$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
SLM_P1	1.152,48	487,34	482,47	**2,99**
SLM_P2	1.872,48	911,50	834,59	**5,32**
SLM_P3	2.595,91	1.356,90	1.346,67	**8,27**

Tabela 3.23: Tempos de execução em segundos *tam* - SLM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_B	BL_BTMA
SLM_P1	1.732,61	648,40	619,43	**3,36**
SLM_P2	2.943,57	1.102,94	1.018,01	**5,55**
SLM_P3	3.873,29	1.702,27	1.745,37	**8,91**

Tabela 3.24: Tempos de execução em segundos *tam* - SLM_P ($w = 10$).

3.4 Busca Local Concatenada

BL_BTMA obteve os menores tempos de processamento em relação às outras buscas para todas as instâncias testadas. Entretanto, houve perdas na qualidade das soluções encontradas comparando-se com as demais. Por outro lado, BL_Teste obtém soluções de boa qualidade comparadas as soluções encontradas por BL_Básica, com ganhos significativos em tempos de processamento em relação ao algoritmo básico. Porém, BL_Teste apresenta tempos de processamento superiores às buscas BL_B e, principalmente, BL_BTMA. A partir dessas considerações, pode-se concatenar as duas buscas locais: após a construção da solução inicial, executa-se primeiramente a busca local mais rápida (BL_BTMA) até que se encontre um ótimo local, para, em seguida, executar-se a busca local mais lenta (BL_Teste). Uma estratégia híbrida, concatenando duas buscas locais, foi aplicada ao problema de Steiner em [35].

O objetivo é tentar acelerar BL_Teste mantendo-se a mesma qualidade das soluções encontradas por essa busca. Assim, na análise da qualidade das soluções e dos tempos de processamento, BL_Conc (busca local concatenada) será comparada somente às buscas BL_Básica e BL_Teste.

3.4.1 Qualidade das Soluções

As Tabelas 3.25 a 3.27 mostram um grande equilíbrio entre BL_Básica, BL_Teste e BL_Conc nas medidas *drpm* e *cm*.

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	0,0009	**1,79**	**30**
	BL_Teste	0,0138	2,23	28
	BL_Conc	**-0,0147**	1,98	27
GRM_P (20 instâncias)	BL_Básica	0,0378	2,30	18
	BL_Teste	-0,0172	**1,83**	**19**
	BL_Conc	**-0,0206**	1,87	17
SLM_P (3 instâncias)	BL_Básica	0,0001	2,33	1
	BL_Teste	0,0120	2,00	**2**
	BL_Conc	**-0,0121**	**1,67**	0

Tabela 3.25: Qualidade relativa das buscas locais, incluindo BL_Conc - ($w = 2$).

Analisando-se a coluna *drpm*, BL_Básica é melhor em três grupos de instâncias: $w = 5$ (SLM_P) e $w = 10$ (ORM_P e SLM_P). BL_Teste também é melhor em três grupos de instâncias: $w = 5$ (ORM_P e GRM_P) e $w = 10$ (GRM_P). BL_Conc é melhor em três grupos de instâncias: $w = 2$ (ORM_P, GRM_P e SLM_P). Na medida relativa *cm*, BL_Básica é melhor em quatro grupos de instâncias: $w = 2$ (ORM_P), $w = 5$ (SLM_P) e $w = 10$ (ORM_P e SLM_P). BL_Teste é melhor em cinco grupos de instâncias: $w = 2$ (GRM_P),

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	-0,0029	2,10	**33**
	BL_Teste	**-0,0081**	**1,83**	30
	BL_Conc	0,011	2,07	29
GRM_P (20 instâncias)	BL_Básica	0,0209	2,42	**17**
	BL_Teste	**-0,0206**	**1,68**	15
	BL_Conc	-0,0003	1,90	13
SLM_P (3 instâncias)	BL_Básica	**-0,0299**	**2,00**	1
	BL_Teste	0,0206	**2,00**	**2**
	BL_Conc	0,0093	**2,00**	**2**

Tabela 3.26: Qualidade relativa das buscas locais, incluindo BL_Conc - ($w = 5$).

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P (40 instâncias)	BL_Básica	**-0,0917**	**1,64**	**34**
	BL_Teste	0,0033	1,96	31
	BL_Conc	0,0884	2,40	23
GRM_P (20 instâncias)	BL_Básica	0,0394	2,40	17
	BL_Teste	**-0,020**	**1,55**	16
	BL_Conc	-0,019	2,05	**18**
SLM_P (3 instâncias)	BL_Básica	**-0,2264**	**1,33**	**2**
	BL_Teste	-0,0559	1,67	1
	BL_Conc	0,2823	3,00	0

Tabela 3.27: Qualidade relativa das buscas locais, incluindo BL_Conc - ($w = 10$).

$w = 5$ (ORM_P, GRM_P e SLM_P) e $w = 10$ (GRM_P). BL_Conc é melhor em dois grupos de instâncias: $w = 2$ (SLM_P) e $w = 5$ (SLM_P).

Na análise da medida *melhor*, observa-se novamente um equilíbrio entre BL_Básica e BL_Teste, seguidas por BL_Conc. BL_Básica é superior às demais em cinco grupos de instâncias: $w = 2$ (ORM_P), $w = 5$ (ORM_P e GRM_P) e $w = 10$ (ORM_P e SLM_P). BL_Teste é superior às demais em três grupos de instâncias: $w = 2$ (GRM_P e SLM_P) e $w = 5$ (SLM_P). BL_Conc é superior às demais em dois grupos de instâncias: $w = 5$ (SLM_P) e $w = 10$ (GRM_P).

Levando-se em consideração todas as medidas, pode-se dizer que, a concatenação das buscas locais (BL_BTMA seguida pela BL_Teste) mantém, em geral, a mesma qualidade das soluções encontradas pelas buscas locais individualmente.

3.4.2 Tempos de Processamento

Os ganhos obtidos pela concatenação das buscas locais ocorreram principalmente nos tempos de processamento, como mostram as Tabelas 3.28 a 3.36 que apresentam os tempos absolutos médios em 15 iterações realizadas para cada instância-algoritmo, incluindo agora os resultados da busca local concatenada.

Instância	BL_Básica	BL_Teste	BL_Conc
ORM_P1	0,04	**0,00**	**0,00**
ORM_P2	0,10	0,01	**0,00**
ORM_P3	0,07	0,02	**0,01**
ORM_P4	0,23	0,08	**0,05**
ORM_P5	0,58	0,23	**0,17**
ORM_P6	0,22	**0,02**	**0,02**
ORM_P7	0,52	0,04	**0,03**
ORM_P8	1,22	0,14	**0,11**
ORM_P9	2,95	0,85	**0,46**
ORM_P10	8,61	2,60	**1,58**
ORM_P11	0,39	**0,02**	**0,02**
ORM_P12	1,30	**0,03**	**0,03**
ORM_P13	6,36	0,57	**0,43**
ORM_P14	19,41	3,45	**1,95**
ORM_P15	39,00	11,90	**8,08**
ORM_P16	0,66	**0,04**	**0,04**
ORM_P17	2,39	**0,06**	**0,06**
ORM_P18	21,52	1,22	**0,84**
ORM_P19	57,22	8,40	**6,28**
ORM_P20	93,43	38,43	**19,96**
ORM_P21	1,22	**0,06**	**0,06**
ORM_P22	4,22	**0,08**	0,10
ORM_P23	49,18	1,95	**1,51**
ORM_P24	114,18	23,63	**16,14**
ORM_P25	179,64	74,81	**54,13**
ORM_P26	2,23	**0,08**	**0,08**
ORM_P27	7,42	**0,13**	**0,13**
ORM_P28	109,14	3,96	**3,19**
ORM_P29	250,12	41,66	**33,05**
ORM_P30	328,84	145,29	**90,25**
ORM_P31	3,55	**0,11**	0,12
ORM_P32	11,13	**0,15**	0,17
ORM_P33	158,45	5,33	**3,22**
ORM_P34	461,26	79,16	**45,65**
ORM_P35	4,15	**0,14**	**0,14**
ORM_P36	17,28	0,22	**0,21**
ORM_P37	345,57	8,57	**7,21**
ORM_P38	7,59	**0,17**	0,20
ORM_P39	17,64	**0,25**	0,30
ORM_P40	503,35	26,70	**18,12**

Tabela 3.28: Tempos de execução em segundos *tam*, incluindo BL_Conc - ORM_P ($w = 2$).

Instância	BL_Básica	BL_Teste	BL_Conc
ORM_P1	0,04	**0,01**	**0,01**
ORM_P2	0,08	0,05	**0,03**
ORM_P3	0,08	0,05	**0,04**
ORM_P4	0,36	0,17	**0,06**
ORM_P5	0,92	0,30	**0,09**
ORM_P6	0,15	**0,02**	**0,02**
ORM_P7	0,45	0,11	**0,09**
ORM_P8	1,24	0,50	**0,29**
ORM_P9	4,81	1,82	**1,19**
ORM_P10	15,48	3,85	**1,35**
ORM_P11	0,32	0,03	**0,02**
ORM_P12	1,45	**0,07**	**0,07**
ORM_P13	6,44	2,81	**1,64**
ORM_P14	22,95	9,26	**3,97**
ORM_P15	74,74	26,68	**6,30**
ORM_P16	0,67	**0,04**	**0,04**
ORM_P17	2,39	**0,08**	**0,08**
ORM_P18	21,21	7,89	**4,35**
ORM_P19	70,87	31,28	**13,10**
ORM_P20	166,28	69,43	**22,43**
ORM_P21	1,18	**0,06**	0,07
ORM_P22	3,91	**0,13**	0,16
ORM_P23	45,59	16,06	**13,64**
ORM_P24	140,41	69,84	**35,99**
ORM_P25	334,31	145,56	**50,59**
ORM_P26	1,89	0,09	**0,08**
ORM_P27	5,93	0,19	**0,17**
ORM_P28	91,86	36,93	**13,22**
ORM_P29	283,19	136,87	**68,38**
ORM_P30	670,77	286,64	**134,31**
ORM_P31	2,96	**0,12**	**0,12**
ORM_P32	8,04	**0,18**	**0,18**
ORM_P33	166,04	53,38	**31,19**
ORM_P34	428,22	247,81	**101,95**
ORM_P35	4,64	**0,14**	**0,14**
ORM_P36	13,97	0,29	**0,28**
ORM_P37	289,71	80,17	**71,15**
ORM_P38	7,85	**0,20**	**0,20**
ORM_P39	14,34	**0,29**	0,30
ORM_P40	485,50	156,13	**111,72**

Tabela 3.29: Tempos de execução em segundos *tam*, incluindo BL_Conc - ORM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_Conc
ORM_P1	0,02	0,02	**0,01**
ORM_P2	0,11	0,07	**0,03**
ORM_P3	0,09	0,06	**0,03**
ORM_P4	0,38	0,17	**0,05**
ORM_P5	0,86	0,36	**0,09**
ORM_P6	0,15	**0,04**	0,05
ORM_P7	0,44	0,22	**0,12**
ORM_P8	1,76	1,01	**0,24**
ORM_P9	5,94	2,55	**0,71**
ORM_P10	17,33	4,67	**0,99**
ORM_P11	0,32	0,05	**0,04**
ORM_P12	0,97	0,21	**0,16**
ORM_P13	9,04	4,70	**0,90**
ORM_P14	32,78	12,73	**5,32**
ORM_P15	90,19	30,01	**8,48**
ORM_P16	0,68	**0,04**	**0,04**
ORM_P17	2,03	0,19	**0,14**
ORM_P18	24,71	12,34	**2,82**
ORM_P19	102,98	45,96	**18,11**
ORM_P20	251,87	81,35	**22,85**
ORM_P21	1,25	**0,08**	**0,08**
ORM_P22	3,88	0,30	**0,24**
ORM_P23	67,17	28,97	**9,70**
ORM_P24	249,32	93,15	**39,08**
ORM_P25	541,13	168,75	**53,74**
ORM_P26	2,24	**0,10**	**0,10**
ORM_P27	5,78	**0,41**	0,42
ORM_P28	118,80	66,60	**33,10**
ORM_P29	435,11	185,42	**68,13**
ORM_P30	1.001,61	379,25	**134,87**
ORM_P31	3,34	**0,14**	**0,14**
ORM_P32	8,70	0,27	**0,24**
ORM_P33	222,22	113,79	**49,66**
ORM_P34	784,10	323,75	**83,84**
ORM_P35	4,30	**0,15**	**0,15**
ORM_P36	14,73	0,59	**0,45**
ORM_P37	413,42	194,84	**96,01**
ORM_P38	7,88	**0,22**	**0,22**
ORM_P39	21,47	0,48	**0,46**
ORM_P40	694,46	314,49	**208,72**

Tabela 3.30: Tempos de execução em segundos *tam*, incluindo BL_Conc - ORM_P ($w = 10$).

Instância	BL_Básica	BL_Teste	BL_Conc
GRM_P1	0,10	**0,02**	**0,02**
GRM_P2	0,24	0,05	**0,03**
GRM_P3	0,40	0,18	**0,10**
GRM_P4	0,62	0,57	**0,29**
GRM_P5	1,01	1,09	**0,61**
GRM_P6	1,33	1,07	**0,67**
GRM_P7	1,67	1,26	**0,75**
GRM_P8	1,84	1,37	**0,79**
GRM_P9	2,07	1,07	**0,49**
GRM_P10	0,38	**0,02**	**0,02**
GRM_P11	0,94	0,15	**0,09**
GRM_P12	1,69	0,30	**0,17**
GRM_P13	3,10	0,55	**0,27**
GRM_P14	3,53	1,21	**0,46**
GRM_P15	4,50	2,24	**1,01**
GRM_P16	4,78	4,43	**2,82**
GRM_P17	7,41	6,25	**2,61**
GRM_P18	8,76	8,27	**3,86**
GRM_P19	10,41	8,88	**4,76**
GRM_P20	17,16	11,18	**4,69**

Tabela 3.31: Tempos de execução em segundos *tam*, incluindo BL_Conc - GRM_P ($w = 2$).

Analisando-se os resultados das tabelas, BL_Conc obtém ganhos significativos em tempos de processamento em relação às buscas BL_Teste e BL_Básica, principalmente nas maiores instâncias testadas.

Assim, levando-se em consideração a qualidade e o tempo, BL_BTMA consegue acelerar BL_Teste sem perder soluções.

3.5 Considerações Finais

Este capítulo apresentou um estudo sobre algoritmos de busca local para o problema das p-medianas conectadas.

Um algoritmo de busca local necessita de uma solução inicial gerada aleatoriamente ou através de uma heurística construtiva. O método escolhido para a geração das soluções iniciais nos algoritmos de busca local localiza as p facilidades nos vértices do grafo, servindo os usuários com a facilidade aberta mais próxima através da heurística Sample Greedy e conecta as p facilidades abertas por uma árvore de Steiner com a heurística Prim.

Em seguida, descreveu-se um algoritmo de busca local básico baseado naquele para o problema das p-medianas proposto em [47]. Algumas idéias como a estratégia de busca com melhoria iterativa e circularidade foram incorporadas ao algoritmo, com o objetivo de acelerar os passos da busca local.

Instância	BL_Básica	BL_Teste	BL_Conc
GRM_P1	0,07	0,07	**0,03**
GRM_P2	0,22	0,16	**0,08**
GRM_P3	0,40	0,37	**0,24**
GRM_P4	0,57	0,76	**0,45**
GRM_P5	0,89	0,75	**0,31**
GRM_P6	1,03	0,84	**0,34**
GRM_P7	1,48	1,15	**0,33**
GRM_P8	1,87	1,06	**0,37**
GRM_P9	1,78	1,01	**0,42**
GRM_P10	0,31	0,08	**0,05**
GRM_P11	0,95	0,45	**0,17**
GRM_P12	1,89	1,27	**0,41**
GRM_P13	3,21	1,94	**0,65**
GRM_P14	3,57	2,61	**1,18**
GRM_P15	4,55	4,22	**2,13**
GRM_P16	4,78	4,64	**3,04**
GRM_P17	6,83	5,77	**2,82**
GRM_P18	7,77	7,04	**3,16**
GRM_P19	12,49	8,67	**6,09**
GRM_P20	18,32	11,24	**5,52**

Tabela 3.32: Tempos de execução em segundos *tam*, incluindo BL_Conc - GRM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_Conc
GRM_P1	0,08	0,07	**0,03**
GRM_P2	0,24	0,20	**0,15**
GRM_P3	0,46	0,39	**0,23**
GRM_P4	0,65	0,76	**0,20**
GRM_P5	0,93	0,71	**0,24**
GRM_P6	1,18	1,02	**0,28**
GRM_P7	1,45	0,94	**0,35**
GRM_P8	1,90	1,13	**0,39**
GRM_P9	2,54	2,58	**0,49**
GRM_P10	0,31	0,19	**0,07**
GRM_P11	0,87	0,73	**0,36**
GRM_P12	1,73	1,55	**0,60**
GRM_P13	2,81	2,21	**0,73**
GRM_P14	3,52	3,58	**2,04**
GRM_P15	4,81	4,05	**1,71**
GRM_P16	5,33	4,99	**2,40**
GRM_P17	7,92	6,34	**2,25**
GRM_P18	9,54	6,78	**3,02**
GRM_P19	11,90	9,50	**3,54**
GRM_P20	21,33	11,54	**3,61**

Tabela 3.33: Tempos de execução em segundos *tam*, incluindo BL_Conc - GRM_P ($w = 10$).

Instância	BL_Básica	BL_Teste	BL_Conc
SLM_P1	658,54	248,46	**151,17**
SLM_P2	1.168,80	512,17	**237,70**
SLM_P3	1.462,00	1.030,78	**515,74**

Tabela 3.34: Tempos de execução em segundos *tam*, incluindo BL_Conc - SLM_P ($w = 2$).

Instância	BL_Básica	BL_Teste	BL_Conc
SLM_P1	1.152,48	487,34	**177,14**
SLM_P2	1.872,48	911,50	**353,95**
SLM_P3	2.595,91	1.356,90	**509,63**

Tabela 3.35: Tempos de execução em segundos *tam*, incluindo BL_Conc - SLM_P ($w = 5$).

Instância	BL_Básica	BL_Teste	BL_Conc
SLM_P1	1.732,61	648,40	**176,43**
SLM_P2	2.943,57	1.102,94	**481,27**
SLM_P3	3.873,29	1.702,27	**734,23**

Tabela 3.36: Tempos de execução em segundos *tam*, incluindo BL_Conc - SLM_P ($w = 10$).

Porém, conectar as p facilidades abertas por uma árvore de Steiner a cada nova troca realizada é uma tarefa muito custosa computacionalmente. Assim, um teste simples foi proposto com o objetivo de evitar, quando possível, a execução da heurística Prim. Candidatos a inserção que passam no teste são investigados e aqueles que não passam no teste são descartados. Incorporando-se essa idéia, três versões da busca local básica foram propostas: a primeira versão incorpora somente o teste; a segunda e a terceira, além de incorporarem o teste priorizam candidatos a inserção que são vizinhos da árvore de Steiner sem o vértice r. Os resultados computacionais mostraram que, levando-se em consideração a qualidade das soluções encontradas, duas buscas se destacaram: a busca local básica e a busca local com teste, confirmando-se assim a efetividade do descarte de vizinhos realizado nessa última busca. Levando-se em consideração os tempos de processamento, duas situações opostas se apresentaram: a busca local pelas bordas com teste da menor aresta é muito mais rápida que as demais e a busca local básica é muito mais lenta do que as demais devido ao alto custo computacional de se executar Prim a cada troca realizada.

Pelo acima apresentado, destacam-se, então, a busca local pelas bordas com teste da menor aresta, que é rápida mas apresenta perda na qualidade das soluções encontradas, e a busca local com teste, de qualidade comparável à busca local básica, porém mais lenta do que às demais buscas locais com descarte de vizinhos. Uma estratégia híbrida foi então implementada,

primeiramente executando-se a busca local mais rápida de qualidade inferior e, em seguida, executando-se a busca local mais lenta de qualidade superior. Os testes computacionais mostraram exatamente que, ao concatená-las, pode-se ganhar em velocidade sem perder a qualidade das soluções encontradas. Portanto, o algoritmo de busca local proposto para o problema das p-medianas conectadas é composto por duas buscas locais concatenadas, incorporando idéias tais como melhoria iterativa, circularidade e descarte de vizinhos com o objetivo de acelerar os passos da busca local, sem perder, no entanto, a qualidade das soluções encontradas.

4
GRASP com Filtro e Reconexão por Caminhos

A metaheurística GRASP *(Greedy Randomized Adaptive Search Procedures)* foi proposta por Feo e Resende [17], onde cada iteração do algoritmo é composta por duas fases: uma fase de construção, na qual uma solução viável é produzida utilizando-se uma heurística construtiva, e uma fase de melhoramento, utilizando-se um algoritmo de busca local, na qual um ótimo local é pesquisado na vizinhança da solução construída. A melhor solução encontrada é mantida como resultado. O pseudo-código da Figura 4.1 ilustra os blocos da metaheurística GRASP para um problema de minimização, na qual *MaxIter* iterações são realizadas, S é uma solução para o problema, $F(.)$ é o valor da função objetivo do problema e *semente* é usada na inicialização do gerador de números aleatórios [46].

```
Procedimento GRASP (MaxIter, semente)
1.  F* ← +∞;
2.  Para it = 1, ..., MaxIter Faça
3.      S ← Fase_Construção(semente);
4.      S' ← Busca_Local(S);
5.      Se (F(S') < F*) Então
6.          F* ← F(S');
7.          S* ← S';
8.      Fim-Se
9.  Fim-Para
10. Retorne S*;
Fim
```

Figura 4.1: Pseudo-código da metaheurística GRASP.

Na fase de construção, responsável pela denominação do método, uma solução viável é iterativamente construída, um elemento a cada vez, utilizando-se as características de um método guloso, a aleatoriedade e a adaptação da função gulosa. Em cada iteração, utiliza-se uma função gulosa característica do problema, que mede o benefício de selecionar cada elemento, para construir uma Lista de Candidatos Restrita (LCR) a partir dos elementos que podem ser adicionados à solução. A heurística é adaptativa porque os benefícios associados com cada elemento são atualizados em cada iteração da fase de construção para refletir as mudanças trazidas pela seleção do elemento anterior.

O componente probabilístico do GRASP é caracterizado pela escolha aleatória de um dos melhores candidatos da LCR, mas não necessariamente o melhor candidato, como acontece em um método puramente guloso. Uma das formas para construir a LCR é definí-la em função de um parâmetro $\alpha \in [0, 1]$, compreendendo todos os elementos que não estão na solução e cujo acréscimo no valor da função objetivo estejam no intervalo $[C_{min}, \alpha(C_{max} - C_{min}) + C_{min}]$, onde C_{min} e C_{max} correspondem, respectivamente, ao menor e ao maior acréscimos no valor da função objetivo dos elementos que ainda não fazem parte da solução. Em um problema de minimização, $\alpha = 0$ implica em uma escolha gulosa e $\alpha = 1$ implica em uma escolha aleatória.

A fase de melhoramento consiste tipicamente de um algoritmo de busca local, já que a solução gerada na fase de construção pode não ser um ótimo local. Assim, partindo-se de cada solução inicial, emprega-se a busca local na tentativa de melhorar a solução.

A metaheurística GRASP possui poucos parâmetros a serem ajustados: um está relacionado ao critério de parada, que é o número máximo de iterações realizadas, e o outro está relacionado a qualidade dos elementos na LCR, influenciada pela escolha do parâmetro α [46].

Os componentes básicos do GRASP para o problema das p-medianas conectadas são: uma heurística construtiva em função do parâmetro α e o algoritmo de busca local concatenada descrito no Capítulo 3 (BL_Conc).

Devido aos altos tempos de processamento apresentados pela busca local nas maiores instâncias, uma estratégia de filtro foi utilizada com o próposito de tentar acelerar as iterações do GRASP. Em geral, a utilização de filtros possui como objetivo evitar a aplicação da busca local a soluções não promissoras, onde provavelmente convergirá para um ótimo local de baixa qualidade.

Uma desvantagem do GRASP em sua versão básica é a independência de cada iteração em relação às demais. Assim, nenhuma informação reunida nas iterações anteriores é utilizada para guiar ou tentar melhorar as soluções encontradas pelas iterações posteriores do algoritmo. Com o objetivo de tentar melhorar a qualidade das soluções, o procedimento de reconexão por caminhos é utilizado como estratégia de intensificação explorando trajetórias entre um ótimo local e um elemento de um conjunto de soluções de alta qualidade, adicionando um mecanismo de memória ao GRASP.

Este capítulo está organizado como se segue: a Seção 4.1 mostra a fase de construção do GRASP; a Seção 4.2 mostra o ambiente de teste, as instâncias e as medidas utilizadas na configuração do GRASP; a Seção 4.3 apresenta o ajuste dos parâmetros da metaheurística; a Seção 4.4 descreve a estratégia de filtro utilizada e resultados computacionais em relação ao GRASP em sua versão básica; a Seção 4.5 apresenta o procedimento de reconexão por caminhos e comparações com o GRASP com filtro que não o utiliza; a Seção 4.6 descreve

a heurística GRASP com filtro e reconexão por caminhos proposta para o problema e, por último, a Seção 4.7 apresenta as conclusões e considerações finais do capítulo.

4.1 Fase de Construção

A Figura 4.2 descreve a fase de construção do GRASP, onde $C(f)$ representa o acréscimo obtido no valor da função objetivo relativo ao custo de atendimento dos usuários ao adicionar a facilidade $f \in V$ à solução S (critério de p-medianas) e $D(f)$ representa o acréscimo obtido no valor da função objetivo relativo ao custo de interconexão das facilidades ao adicionar o caminho mínimo entre a facilidade $f \in V$ e a árvore parcial construída pelo algoritmo executando-se a heurística Prim (critério de Steiner). A função recebe como parâmetros o valor de $\alpha \in [0, 1]$ e a *semente* para inicializar o gerador de números aleatórios.

Função Fase_Construção (α, *semente*)
1. $S \leftarrow \emptyset$;
2. Avalie o acréscimo $C(f), \forall\ f \in V$;
3. $D(f) \leftarrow 0, \forall\ f \in V$;
4. $k \leftarrow 0$;
5. **Enquanto** $(k \neq p)$ **Faça**
6. $\quad C_{min} \leftarrow \min\{C(f) + D(f), f \in V\}$;
7. $\quad C_{max} \leftarrow \max\{C(f) + D(f), f \in V\}$;
8. $\quad LCR \leftarrow \{f \in V \mid C(f) + D(f) \leq C_{min} + \alpha(C_{max} - C_{min})\}$;
9. $\quad$ Selecione f aleatoriamente da LCR;
10. $\quad S \leftarrow S \cup \{f\}$;
11. $\quad V \leftarrow V \setminus \{f\}$;
12. $\quad$ Re-avalie o acréscimo $C(f), \forall\ f \in V$;
13. $\quad k \leftarrow k + 1$;
14. $\quad$ Construa a árvore de Steiner parcial $T_k(S)$;
15. $\quad$ Avalie $D(f)$ em relação à $T_k(S), \forall\ f \in V$;
16. **Fim-Enquanto**
17. $T(S) \leftarrow T_p(S)$;
18. **Retorne** S;

Fim

Figura 4.2: Fase de construção do GRASP.

Na linha 2, avalia-se, para cada facilidade, o acréscimo obtido no valor da função objetivo relativo ao custo de atendimento dos usuários ao adicioná-la à solução (critério de p-medianas). Na linha 3, a distância mínima de cada vértice à arvore parcial é inicializada com zero (critério de Steiner), pois ainda nenhuma facilidade foi escolhida (o conjunto S está vazio). O laço das linhas 5 a 16 é executado iterativamente até que se obtenha p facilidades abertas e a árvore de Steiner conectando as mesmas. A LCR é construída em função

do parâmetro α (linhas 6 a 8), sendo que na primeira iteração, utiliza-se somente o critério de p-medianas para a construção da LCR, pois o conjunto S permanece vazio. Nas $p-1$ iterações restantes, utiliza-se a soma dos critérios de p-medianas e de Steiner para a sua construção. Na linha 9, ocorre a seleção aleatória de uma facilidade da LCR, o que mantém a diversidade das soluções geradas. Na linha 10, a facilidade escolhida é adicionada à solução e, na linha 11, o conjunto de possíveis localizações de facilidades é atualizado excluindo-se a facilidade escolhida anteriormente. Na linha 12, para cada facilidade, reavalia-se o acréscimo obtido no valor da função objetivo referente ao custo de atendimento dos usuários ao adicioná-la à solução. Na linha 14, a árvore de Steiner parcial com k facilidades é construída executando-se a heurística Prim, e na linha 15, avalia-se a menor distância do vértice f à árvore de Steiner parcial. Para as facilidades que são vértices de Steiner na árvore parcial, $D(f) = 0$. Ao final das iterações, o algoritmo conectará os p vértices terminais e possíveis não-terminais através de uma árvore de Steiner (linha 17). Por último, na linha 18, o algoritmo retorna uma solução definida pelo conjunto S formado pelas p facilidades abertas e pelo conjunto $T(S)$ de arestas obtidas quando se conecta estas facilidades por uma árvore.

A linha 1 possui complexidade $O(p)$. As linhas 3, 6, 7 e 8 possuem complexidade $O(n)$, enquanto as linhas 4, 9, 10, 11 e 13 possuem complexidade $O(1)$. Para cada facilidade, avaliar o acréscimo obtido no valor da função objetivo referente ao custo de atendimento ao adicioná-la à solução possui complexidade $O(n)$, correspondente ao número de usuários que serão atendidos pela facilidade. Como existem n possíveis localizações de facilidades candidatas, a complexidade das linhas 2 e 12 é, respectivamente, $O(n^2)$ e $O(pn^2)$. Para a construção da árvore de Steiner parcial (linha 14), no pior caso, p terminais devem ser conectados com complexidade $O(p\,|E|\log n)$, referente à heurística Prim. Para a avaliação do critério de Steiner (linha 15), no máximo p execuções de um algoritmo de caminho mínimo (algoritmo de Dijkstra [15]) devem ser realizadas com complexidade $O(|E|\log n)$, utilizando-se a estrutura de dados *heap* binário [61]. Assim, a linha 15 possui complexidade $O(p\,|E|\log n)$. A complexidade de pior caso do algoritmo é $O(pn^2)$, dominada pela linha 12.

4.2 Ambiente de Teste, Instâncias e Medidas Utilizadas

Para os testes da estratégia de filtro e configuração do procedimento de reconexão por caminhos, utilizou-se um sub-conjunto das instâncias proporcionais apresentadas na Seção 3.3.4, totalizando 36 problemas: ORM_P1 a ORM_P18 e GRM_P1 a GRM_P18 ($w = 5$). Esses testes foram executados em uma máquina Pentium IV 3.2 GHz com 1 Gbyte de memória RAM sob o sistema operacional Linux RedHat 9.0. Na avaliação da qualidade das soluções e

tempos de processamento, as medidas relativas, descritas no Capítulo 3, serão utilizadas. Para cada par instância-algoritmo, foram realizadas cinco execuções com diferentes sementes aleatórias.

Para o ajuste de α no GRASP, importante parâmetro responsável pela qualidade das soluções construídas, utilizou-se um sub-conjunto maior de instâncias proporcionais apresentadas na Seção 3.3.4, totalizando-se 132 problemas: GRM_P1 a GRM_P20 e ORM_P1 a ORM_P24 ($w = 2$, $w = 5$ e $w = 10$). Esse teste foi executado em uma máquina AMD Sempron 1.8 GHz com 512 Mbytes de memória RAM sob o sistema operacional Windows XP Service Pack 2. Para a avaliação da qualidade das soluções e tempos de processamento, considera-se, para cada instância, a média obtida pelo algoritmo (qualidade/tempo) em um determinado número de execuções. Para cada par instância-algoritmo, foram realizadas três execuções com diferentes sementes aleatórias.

O GRASP foi implementado em C e o gerador de números aleatórios utilizado foi o de Matsumoto e Nishimura [53].

4.3 Configuração dos parâmetros no GRASP

Os dois principais parâmetros do GRASP em sua versão básica são: o número máximo de iterações realizadas, representado pela variável *MaxIter* na Figura 4.1, e o valor de $\alpha \in [0, 1]$.

4.3.1 Número de Iterações

No GRASP, o tempo de execução não varia muito entre uma iteração e outra, sendo o tempo total gasto pelo algoritmo previsível e proporcional ao número de iterações realizadas. Aumentando-se o número de iterações, a tendência é de que melhore a qualidade das soluções encontradas pelo GRASP. Assim, quanto maior o número de iterações, maior o tempo gasto pelo algoritmo e melhor será a solução encontrada [46]. Porém, como apresentado no Capítulo 3, o tempo médio de processamento de cada iteração da busca local é elevado para as maiores instâncias testadas, tais como ORM_P25, ORM_P30, ORM_P34, ORM_P37, ORM_P40, SLM_P1, SLM_P2 e SLM_P3. Por isso, o número máximo de iterações realizadas pelo GRASP foi estipulado em 100 para os testes com o filtro e para a configuração do procedimento de reconexão por caminhos e em 200 para o ajuste do parâmetro α, valores relativamente baixos em comparação com outras aplicações GRASP apresentadas na literatura, como, por exemplo em [44].

4.3.2 Valor de α

Um dos aspectos que melhoram o desempenho da busca local é a geração de soluções iniciais de alta qualidade. Por esse motivo, a fase de construção torna-se muito importante no GRASP e, conseqüentemente o valor de α, pois, em geral, este parâmetro influencia fortemente a qualidade das soluções encontradas e os tempos de processamento obtidos pela busca local.

Para o ajuste do parâmetro α, as seguintes configurações foram analisadas: 0 (escolha gulosa), 0,2, 0,4, 0,6, 0,8 e 1 (escolha aleatória). Nas Tabelas 4.1 a 4.12, os valores em negrito representam o menor valor obtido pelas configurações para cada instância.

Analisando-se a qualidade média das soluções encontradas pelo GRASP (Tabelas 4.1 a 4.6), a configuração gulosa ($\alpha = 0$) obteve os piores resultados. Comparando-se as demais configurações entre si, os resultados apresentados foram muito semelhantes. Somando-se o número de instâncias em que os algoritmos encontraram o menor valor entre todos os métodos, incluindo-se os casos de empate no primeiro lugar para as classes testadas (linha *melhor* nas tabelas), as variantes GRASP com $\alpha = 0,0,2,0,4,0,6,0,8$ e 1 obtiveram os seguintes resultados, respectivamente: 65, 120, 118, 112, 117 e 114.

Analisando-se o tempo médio de execução encontrado pelo GRASP (Tabelas 4.7 a 4.12), em geral, como esperado, quanto mais próximo da escolha gulosa, menor o tempo de execução do GRASP devido à geração de soluções iniciais melhores e, conseqüentemente, executando um menor número de trocas durante a busca local. Somando-se o número de instâncias em que os algoritmos encontraram o menor valor entre todos os métodos, incluindo-se os casos de empate no primeiro lugar para as classes testadas (linha *melhor* nas tabelas), as variantes GRASP com $\alpha = 0,0,2,0,4,0,6,0,8$ e 1 obtiveram os seguintes resultados, respectivamente: 93, 22, 9, 1, 3 e 4.

Em termos de qualidade da solução, os resultados apresentados pelas configurações foram semelhantes, com exceção de $\alpha = 0$. Em termos de tempo de processamento, destacaram-se as configurações próximas da escolha gulosa ($\alpha = 0,0,2$ e $0,4$). Poderia utilizar-se tanto $\alpha = 0,2$ quanto $\alpha = 0,4$. Com o objetivo de obter uma maior diversidade na geração das soluções iniciais, optou-se por utilizar $\alpha = 0,4$.

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**6443,00**	**6443,00**	**6443,00**	**6443,00**	**6443,00**	**6443,00**
ORM_P2	**5228,00**	**5228,00**	**5228,00**	**5228,00**	**5228,00**	**5228,00**
ORM_P3	**5368,00**	**5368,00**	**5368,00**	**5368,00**	**5368,00**	**5368,00**
ORM_P4	**5127,00**	**5127,00**	**5127,00**	**5127,00**	**5127,00**	**5127,00**
ORM_P5	**3663,00**	**3663,00**	**3663,00**	**3663,00**	**3663,00**	**3663,00**
ORM_P6	**8185,00**	**8185,00**	**8185,00**	**8185,00**	**8185,00**	**8185,00**
ORM_P7	**6217,00**	**6217,00**	**6217,00**	**6217,00**	**6217,00**	**6217,00**
ORM_P8	**5865,00**	**5865,00**	5865,33	5865,33	5865,33	5865,33
ORM_P9	4703,67	4699,67	**4699,00**	**4699,00**	**4699,00**	**4699,00**
ORM_P10	**3457,00**	**3457,00**	**3457,00**	**3457,00**	**3457,00**	**3457,00**
ORM_P11	7879,00	**7855,00**	**7855,00**	**7855,00**	**7855,00**	**7855,00**
ORM_P12	**7074,00**	**7074,00**	**7074,00**	**7074,00**	**7074,00**	**7074,00**
ORM_P13	5547,00	**5539,00**	**5539,00**	**5539,00**	**5539,00**	**5539,00**
ORM_P14	4981,67	**4981,00**	**4981,00**	**4981,00**	**4981,00**	**4981,00**
ORM_P15	**4477,00**	**4477,00**	**4477,00**	**4477,00**	**4477,00**	**4477,00**
ORM_P16	**8292,00**	**8292,00**	**8292,00**	**8292,00**	**8292,00**	**8292,00**
ORM_P17	**7279,00**	**7279,00**	**7279,00**	**7279,00**	**7279,00**	**7279,00**
ORM_P18	5990,00	**5989,33**	5990,67	5990,33	5991,00	5989,67
ORM_P19	**4787,00**	4788,67	**4787,00**	4788,33	4788,00	4787,67
ORM_P20	4861,00	**4860,00**	**4860,00**	**4860,00**	**4860,00**	**4860,00**
ORM_P21	**9272,00**	**9272,00**	**9272,00**	**9272,00**	**9272,00**	**9272,00**
ORM_P22	**8839,00**	**8839,00**	**8839,00**	**8839,00**	**8839,00**	**8839,00**
ORM_P23	5802,33	5801,33	5801,00	5801,00	**5800,33**	5800,67
ORM_P24	**4985,00**	4985,33	4985,67	4985,33	4985,33	4985,67
melhor	17	20	20	19	20	19

Tabela 4.1: Configuração de α: qualidade média das instâncias ORM_P1 a ORM_P24 ($w = 2$).

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**7184,00**	**7184,00**	**7184,00**	**7184,00**	**7184,00**	**7184,00**
ORM_P2	6580,00	**6572,00**	**6572,00**	**6572,00**	**6572,00**	**6572,00**
ORM_P3	6793,00	**6776,00**	**6776,00**	**6776,00**	**6776,00**	**6776,00**
ORM_P4	6999,00	**6944,00**	**6944,00**	**6944,00**	**6944,00**	**6944,00**
ORM_P5	**5292,00**	**5292,00**	**5292,00**	**5292,00**	**5292,00**	**5292,00**
ORM_P6	8700,00	**8662,00**	**8662,00**	**8662,00**	**8662,00**	**8662,00**
ORM_P7	6990,00	6980,00	**6977,00**	6980,00	**6977,00**	**6977,00**
ORM_P8	7214,00	7206,33	**7203,00**	**7203,00**	**7203,00**	7206,33
ORM_P9	**6436,00**	**6436,00**	**6436,00**	**6436,00**	**6436,00**	**6436,00**
ORM_P10	**5004,00**	**5004,00**	**5004,00**	**5004,00**	**5004,00**	**5004,00**
ORM_P11	**8062,00**	**8062,00**	**8062,00**	**8062,00**	**8062,00**	**8062,00**
ORM_P12	**7666,00**	**7666,00**	**7666,00**	**7666,00**	**7666,00**	**7666,00**
ORM_P13	6696,00	**6693,00**	**6693,00**	**6693,00**	**6693,00**	**6693,00**
ORM_P14	6632,67	**6632,00**	**6632,00**	**6632,00**	**6632,00**	**6632,00**
ORM_P15	6364,00	**6363,00**	**6363,00**	**6363,00**	**6363,00**	**6363,00**
ORM_P16	**8458,00**	**8458,00**	**8458,00**	**8458,00**	**8458,00**	**8458,00**
ORM_P17	**7676,00**	**7676,00**	**7676,00**	**7676,00**	**7676,00**	**7676,00**
ORM_P18	**7259,00**	**7259,00**	**7259,00**	**7259,00**	**7259,00**	**7259,00**
ORM_P19	6540,00	**6537,00**	**6537,00**	**6537,00**	**6537,00**	**6537,00**
ORM_P20	7280,00	**7266,33**	7268,67	7270,67	7270,33	7268,00
ORM_P21	**9473,00**	**9473,00**	**9473,00**	**9473,00**	**9473,00**	**9473,00**
ORM_P22	**9219,00**	**9219,00**	**9219,00**	**9219,00**	**9219,00**	**9219,00**
ORM_P23	**7020,00**	**7020,00**	**7020,00**	**7020,00**	**7020,00**	**7020,00**
ORM_P24	6655,00	**6648,00**	6648,33	6648,67	6648,67	**6648,00**
melhor	12	22	22	21	22	22

Tabela 4.2: Configuração de α: qualidade média das instâncias ORM_P1 a ORM_P24 ($w = 5$).

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**8146,00**	**8146,00**	**8146,00**	**8146,00**	**8146,00**	**8146,00**
ORM_P2	**7706,00**	**7706,00**	**7706,00**	**7706,00**	**7706,00**	**7706,00**
ORM_P3	8296,00	**8265,00**	**8265,00**	**8265,00**	**8265,00**	**8265,00**
ORM_P4	9183,00	**9135,00**	**9135,00**	**9135,00**	**9135,00**	**9135,00**
ORM_P5	**7485,00**	**7485,00**	**7485,00**	**7485,00**	**7485,00**	**7485,00**
ORM_P6	**9387,00**	**9387,00**	**9387,00**	**9387,00**	**9387,00**	**9387,00**
ORM_P7	7882,00	**7878,00**	**7878,00**	7879,00	**7878,00**	7879,00
ORM_P8	8667,00	**8632,00**	**8632,00**	**8632,00**	**8632,00**	**8632,00**
ORM_P9	8214,00	**8174,00**	**8174,00**	**8174,00**	**8174,00**	**8174,00**
ORM_P10	6918,00	**6894,00**	**6894,00**	6894,33	**6894,00**	6894,67
ORM_P11	8405,00	**8383,00**	**8383,00**	**8383,00**	**8383,00**	**8383,00**
ORM_P12	**8426,00**	**8426,00**	**8426,00**	**8426,00**	**8426,00**	**8426,00**
ORM_P13	**7679,00**	**7679,00**	**7679,00**	**7679,00**	**7679,00**	**7679,00**
ORM_P14	8542,00	8505,67	8505,33	8505,67	**8505,00**	8505,33
ORM_P15	8981,00	**8865,67**	8871,00	8869,00	8869,00	8867,33
ORM_P16	8738,00	**8728,00**	**8728,00**	**8728,00**	**8728,00**	**8728,00**
ORM_P17	**8317,00**	**8317,00**	**8317,00**	**8317,00**	**8317,00**	**8317,00**
ORM_P18	8610,00	**8547,00**	**8547,00**	**8547,00**	**8547,00**	**8547,00**
ORM_P19	8371,00	**8320,33**	8329,33	8331,00	8330,67	8331,00
ORM_P20	10492,00	**10407,67**	10452,00	10456,00	10450,00	10468,33
ORM_P21	**9808,00**	**9808,00**	**9808,00**	**9808,00**	**9808,00**	**9808,00**
ORM_P22	**9776,00**	**9776,00**	**9776,00**	**9776,00**	**9776,00**	**9776,00**
ORM_P23	8361,00	**8359,00**	**8359,00**	**8359,00**	**8359,00**	**8359,00**
ORM_P24	8573,00	**8552,00**	8552,67	8555,33	8552,67	8556,33
melhor	9	23	19	17	20	17

Tabela 4.3: Configuração de α: qualidade média das instâncias ORM_P1 a ORM_P24 ($w = 10$).

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	**6403,00**	**6403,00**	**6403,00**	**6403,00**	**6403,00**	**6403,00**
GRM_P2	5493,00	**5489,00**	**5489,00**	**5489,00**	**5489,00**	**5489,00**
GRM_P3	5207,00	**5157,00**	**5157,00**	**5157,00**	**5157,00**	**5157,00**
GRM_P4	5252,00	**5246,00**	**5246,00**	**5246,00**	**5246,00**	**5246,00**
GRM_P5	**5403,00**	**5403,00**	**5403,00**	**5403,00**	**5403,00**	**5403,00**
GRM_P6	**5580,00**	**5580,00**	**5580,00**	**5580,00**	**5580,00**	**5580,00**
GRM_P7	**5766,00**	**5766,00**	**5766,00**	**5766,00**	**5766,00**	**5766,00**
GRM_P8	**5961,00**	**5961,00**	**5961,00**	**5961,00**	**5961,00**	**5961,00**
GRM_P9	**6364,00**	**6364,00**	**6364,00**	**6364,00**	**6364,00**	**6364,00**
GRM_P10	**11475,00**	**11475,00**	**11475,00**	**11475,00**	**11475,00**	**11475,00**
GRM_P11	**9862,00**	**9862,00**	**9862,00**	**9862,00**	**9862,00**	**9862,00**
GRM_P12	8921,00	**8910,00**	**8910,00**	**8910,00**	**8910,00**	**8910,00**
GRM_P13	8455,00	**8352,00**	**8352,00**	**8352,00**	**8352,00**	**8352,00**
GRM_P14	8199,00	8180,00	**8174,00**	8180,67	8176,67	8180,00
GRM_P15	8174,00	**8133,00**	**8133,00**	**8133,00**	**8133,00**	**8133,00**
GRM_P16	8261,00	8244,00	8243,33	**8243,00**	8243,33	8243,67
GRM_P17	8388,00	**8372,00**	**8372,00**	**8372,00**	**8372,00**	**8372,00**
GRM_P18	8531,00	**8530,00**	**8530,00**	**8530,00**	**8530,00**	**8530,00**
GRM_P19	**8701,00**	**8701,00**	**8701,00**	**8701,00**	**8701,00**	**8701,00**
GRM_P20	**9080,00**	**9080,00**	**9080,00**	**9080,00**	**9080,00**	**9080,00**
melhor	10	18	19	19	18	18

Tabela 4.4: Configuração de α: qualidade média das instâncias GRM_P ($w = 2$).

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	7157,00	**7036,00**	**7036,00**	**7036,00**	**7036,00**	**7036,00**
GRM_P2	6763,00	**6662,00**	**6662,00**	**6662,00**	**6662,00**	**6662,00**
GRM_P3	6973,00	**6923,00**	**6923,00**	**6923,00**	**6923,00**	**6923,00**
GRM_P4	7608,00	**7602,00**	**7602,00**	**7602,00**	**7602,00**	**7602,00**
GRM_P5	**8379,00**	**8379,00**	**8379,00**	**8379,00**	**8379,00**	**8379,00**
GRM_P6	**9178,00**	**9178,00**	**9178,00**	**9178,00**	**9178,00**	**9178,00**
GRM_P7	**9984,00**	**9984,00**	**9984,00**	**9984,00**	**9984,00**	**9984,00**
GRM_P8	**10802,00**	**10802,00**	**10802,00**	**10802,00**	**10802,00**	**10802,00**
GRM_P9	12448,00	**12447,00**	**12447,00**	**12447,00**	**12447,00**	**12447,00**
GRM_P10	12254,00	**12202,00**	**12202,00**	**12202,00**	**12202,00**	**12202,00**
GRM_P11	**11086,00**	**11086,00**	**11086,00**	**11086,00**	**11086,00**	**11086,00**
GRM_P12	10779,00	**10767,00**	**10767,00**	**10767,00**	**10767,00**	**10767,00**
GRM_P13	10906,00	**10902,00**	**10902,00**	**10902,00**	**10902,00**	**10902,00**
GRM_P14	11376,00	**11293,00**	**11293,00**	11294,33	11295,67	11295,67
GRM_P15	11890,00	11875,33	**11874,00**	11877,33	**11874,00**	**11874,00**
GRM_P16	12554,00	**12539,00**	**12539,00**	12540,33	12540,33	12541,00
GRM_P17	13284,00	**13276,00**	**13276,00**	13278,33	**13276,00**	**13276,00**
GRM_P18	**14036,00**	**14036,00**	**14036,00**	**14036,00**	**14036,00**	**14036,00**
GRM_P19	**14814,00**	**14814,00**	**14814,00**	**14814,00**	**14814,00**	**14814,00**
GRM_P20	**16422,00**	**16422,00**	**16422,00**	**16422,00**	**16422,00**	**16422,00**
melhor	8	19	20	16	18	18

Tabela 4.5: Configuração de α: qualidade média das instâncias GRM_P ($w = 5$).

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	7939,00	**7884,00**	**7884,00**	**7884,00**	**7884,00**	**7884,00**
GRM_P2	8657,00	**8580,00**	**8580,00**	**8580,00**	**8580,00**	**8580,00**
GRM_P3	9897,00	**9793,00**	**9793,00**	**9793,00**	**9793,00**	**9793,00**
GRM_P4	11508,00	**11487,00**	**11487,00**	**11487,00**	**11487,00**	**11487,00**
GRM_P5	**13273,00**	**13273,00**	**13273,00**	**13273,00**	**13273,00**	**13273,00**
GRM_P6	**15079,00**	**15079,00**	**15079,00**	**15079,00**	**15079,00**	**15079,00**
GRM_P7	16917,00	**16909,00**	**16909,00**	**16909,00**	**16909,00**	**16909,00**
GRM_P8	**18754,00**	**18754,00**	**18754,00**	**18754,00**	**18754,00**	**18754,00**
GRM_P9	**22507,00**	**22507,00**	**22507,00**	**22507,00**	**22507,00**	**22507,00**
GRM_P10	**13083,00**	**13083,00**	**13083,00**	**13083,00**	**13083,00**	**13083,00**
GRM_P11	**13040,00**	**13040,00**	**13040,00**	**13040,00**	**13040,00**	**13040,00**
GRM_P12	13846,00	**13845,00**	**13845,00**	**13845,00**	**13845,00**	**13845,00**
GRM_P13	15044,00	14986,33	**14985,00**	**14985,00**	**14985,00**	**14985,00**
GRM_P14	16375,00	**16355,00**	**16355,00**	**16355,00**	**16355,00**	**16355,00**
GRM_P15	17925,00	**17919,00**	**17919,00**	**17919,00**	**17919,00**	**17919,00**
GRM_P16	19592,00	19579,67	19579,33	**19579,00**	19579,33	**19579,00**
GRM_P17	21346,00	**21317,00**	21318,00	**21317,00**	**21317,00**	**21317,00**
GRM_P18	**23095,00**	**23095,00**	**23095,00**	**23095,00**	**23095,00**	**23095,00**
GRM_P19	**24895,00**	**24895,00**	**24895,00**	**24895,00**	**24895,00**	**24895,00**
GRM_P20	**28566,00**	**28566,00**	**28566,00**	**28566,00**	**28566,00**	**28566,00**
melhor	9	18	18	20	19	20

Tabela 4.6: Configuração de α: qualidade média das instâncias GRM_P ($w = 10$).

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**0,47**	0,85	1,02	1,15	1,22	1,28
ORM_P2	**0,99**	1,54	1,81	1,97	2,13	2,27
ORM_P3	**1,44**	1,73	1,99	2,11	2,22	2,28
ORM_P4	11,17	**7,40**	7,43	7,57	7,73	7,87
ORM_P5	32,35	29,85	**27,26**	27,53	29,54	29,36
ORM_P6	**1,59**	2,72	3,33	3,76	4,14	4,36
ORM_P7	**2,45**	5,00	5,76	6,33	6,85	7,40
ORM_P8	**15,74**	19,45	20,20	21,38	22,51	22,53
ORM_P9	**77,72**	86,62	89,97	90,65	93,59	93,31
ORM_P10	300,30	258,98	**252,49**	264,38	262,20	264,47
ORM_P11	**2,08**	3,81	4,66	5,48	6,25	7,13
ORM_P12	**4,18**	8,04	9,32	10,46	11,43	12,05
ORM_P13	**30,93**	75,68	85,31	91,72	94,68	99,85
ORM_P14	**260,12**	366,94	370,41	375,94	380,09	379,68
ORM_P15	1.187,60	**1.154,76**	1.229,68	1.272,51	1.275,23	1.266,15
ORM_P16	**4,65**	7,19	8,86	10,44	12,04	13,23
ORM_P17	**7,17**	13,76	16,69	18,42	20,47	22,52
ORM_P18	**151,61**	197,63	210,90	224,09	227,61	239,71
ORM_P19	**1.066,82**	1.202,28	1.230,71	1.291,34	1.280,10	1.327,74
ORM_P20	4.771,15	4.717,76	**4.716,19**	4.725,71	4.739,15	4.813,62
ORM_P21	**5,44**	10,79	13,62	16,23	19,11	21,05
ORM_P22	**12,74**	24,02	27,87	31,09	34,46	37,48
ORM_P23	**308,70**	549,93	614,10	638,76	654,95	666,53
ORM_P24	**1.959,88**	3.763,09	4.086,61	4.143,00	4.170,58	4.319,72
melhor	19	2	3	0	0	0

Tabela 4.7: Configuração de α: tempo médio de execução das instâncias ORM_P1 a ORM_P24 ($w = 2$).

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**0,69**	1,38	1,55	1,64	1,70	1,73
ORM_P2	11,68	**5,67**	6,06	6,30	6,78	6,71
ORM_P3	7,03	**6,07**	6,30	6,35	6,53	6,33
ORM_P4	**8,61**	11,14	10,96	11,21	11,11	11,39
ORM_P5	**15,69**	16,90	18,80	18,90	19,85	19,77
ORM_P6	**2,43**	3,77	4,38	4,72	5,04	5,25
ORM_P7	**12,41**	13,92	14,49	15,64	16,03	16,14
ORM_P8	**29,25**	51,66	51,46	52,85	53,67	53,30
ORM_P9	310,39	210,42	**207,84**	219,20	222,88	211,33
ORM_P10	303,96	**227,21**	231,58	244,23	247,27	233,84
ORM_P11	**2,43**	4,73	5,81	6,59	7,48	8,03
ORM_P12	**8,87**	14,91	16,88	17,84	18,72	18,82
ORM_P13	**141,85**	238,09	250,31	247,06	266,17	254,06
ORM_P14	**587,91**	601,83	616,21	619,47	596,50	594,29
ORM_P15	1.931,14	1.185,02	1.161,25	1.174,35	**1.128,74**	1.151,38
ORM_P16	**5,82**	8,34	9,84	11,34	12,89	13,91
ORM_P17	**12,95**	19,15	21,97	24,31	26,31	27,86
ORM_P18	**583,69**	1.020,20	1.019,43	1.027,97	1.035,02	1.026,50
ORM_P19	3.527,33	2.916,49	**2.851,93**	2.855,31	2.948,18	2.883,65
ORM_P20	6.894,01	5.219,73	5.427,66	5.402,04	**5.194,20**	5.328,25
ORM_P21	**6,10**	14,44	17,48	20,12	22,63	24,85
ORM_P22	**20,75**	39,43	44,49	47,50	49,87	52,05
ORM_P23	**3.046,26**	3.904,04	3.913,11	3.916,11	3.934,68	3.830,63
ORM_P24	8.964,07	**7.369,77**	8.006,63	7.639,83	7.596,67	7.945,12
melhor	16	4	2	0	2	0

Tabela 4.8: Configuração de α: tempo médio de execução das instâncias ORM_P1 a ORM_P24 ($w = 5$).

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**1,43**	2,23	2,29	2,35	2,37	2,32
ORM_P2	**3,63**	3,95	4,26	4,25	4,54	4,48
ORM_P3	9,88	**3,85**	4,18	4,28	4,32	4,35
ORM_P4	**8,72**	9,88	10,68	10,51	11,57	11,32
ORM_P5	53,64	23,95	**17,94**	19,20	19,20	18,67
ORM_P6	**3,82**	7,86	8,65	8,73	8,81	9,07
ORM_P7	**12,15**	19,04	20,06	19,71	20,71	21,34
ORM_P8	**40,19**	48,51	49,63	52,17	54,89	54,01
ORM_P9	**93,83**	117,68	125,71	123,41	124,82	126,82
ORM_P10	301,59	214,69	216,49	226,75	216,86	**209,65**
ORM_P11	**3,61**	7,29	8,30	9,30	10,10	10,59
ORM_P12	**23,00**	29,22	30,87	32,85	32,67	34,02
ORM_P13	301,08	**190,95**	200,51	201,85	204,33	209,56
ORM_P14	**678,01**	926,16	899,62	902,13	887,14	862,08
ORM_P15	1.225,15	1.243,00	1.192,30	1.233,70	1.177,55	**1.152,95**
ORM_P16	**4,51**	9,56	11,14	13,13	14,34	15,25
ORM_P17	**27,21**	40,52	44,48	46,88	47,93	49,97
ORM_P18	1.197,37	**931,95**	965,06	1.004,01	988,67	973,76
ORM_P19	**1.917,93**	3.234,97	3.470,85	2.960,18	3.007,70	2.979,12
ORM_P20	**4.909,04**	5.944,87	5.733,55	5.940,10	5.795,23	5.877,47
ORM_P21	**8,56**	20,41	23,75	26,58	28,82	30,07
ORM_P22	**48,05**	93,60	98,03	102,05	102,81	103,92
ORM_P23	3.844,91	**3.429,50**	3.738,92	3.646,77	3.684,44	3.666,73
ORM_P24	8.948,75	8.835,02	8.914,13	8.685,05	9.153,27	**8.526,51**
melhor	16	4	1	0	0	3

Tabela 4.9: Configuração de α: tempo médio de execução das instâncias ORM_P1 a ORM_P24 ($w = 10$).

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	**4,78**	5,74	7,07	7,86	8,27	8,41
GRM_P2	20,59	**17,01**	20,07	21,14	22,50	21,61
GRM_P3	**40,97**	62,70	61,70	64,83	65,02	66,32
GRM_P4	**181,18**	191,69	192,49	203,10	204,20	212,04
GRM_P5	**217,59**	384,33	366,42	380,65	385,04	365,03
GRM_P6	**338,92**	433,34	419,74	447,31	462,14	451,88
GRM_P7	725,05	573,93	**538,45**	559,63	591,74	580,43
GRM_P8	545,38	542,01	**474,57**	524,64	508,27	499,67
GRM_P9	**293,67**	372,09	348,92	359,53	363,49	363,68
GRM_P10	20,59	**17,82**	21,75	21,82	21,80	22,08
GRM_P11	109,00	**62,84**	68,89	73,76	79,47	81,77
GRM_P12	**50,79**	97,66	109,15	112,65	121,40	119,68
GRM_P13	**101,46**	188,40	206,51	216,36	213,86	233,37
GRM_P14	**353,74**	382,72	431,03	420,51	417,32	428,22
GRM_P15	**632,82**	845,09	849,34	874,79	870,21	870,24
GRM_P16	1.915,71	**1.403,08**	1.535,41	1.565,98	1.569,12	1.574,71
GRM_P17	**1.503,46**	1.814,67	1.929,64	2.018,26	2.020,01	1.970,98
GRM_P18	3.648,27	**2.109,44**	2.119,83	2.217,90	2.263,40	2.221,14
GRM_P19	2.411,73	**2.398,63**	2.404,29	2.529,05	2.577,90	2.561,27
GRM_P20	**2.343,01**	2.504,94	2.565,46	2.690,63	2.745,63	2.693,28
melhor	12	6	2	0	0	0

Tabela 4.10: Configuração de α: tempo médio de execução das instâncias GRM_P ($w = 2$).

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	**10,10**	13,71	14,66	15,11	15,08	15,10
GRM_P2	**25,65**	43,65	51,49	55,63	55,58	54,26
GRM_P3	**58,48**	116,48	139,23	142,33	146,04	141,49
GRM_P4	399,75	210,04	221,94	218,89	215,75	**209,72**
GRM_P5	**148,79**	165,66	177,90	184,47	176,02	176,74
GRM_P6	251,47	**176,85**	187,62	186,77	189,25	189,23
GRM_P7	**183,13**	186,71	195,32	198,94	196,85	200,40
GRM_P8	**193,70**	210,20	220,34	222,30	221,75	221,10
GRM_P9	**231,21**	269,20	275,40	279,50	281,94	284,22
GRM_P10	37,68	**32,90**	35,06	34,47	34,40	34,71
GRM_P11	**71,21**	117,29	135,64	140,20	151,00	149,20
GRM_P12	397,29	**300,00**	350,85	348,25	348,83	371,38
GRM_P13	810,00	**591,38**	684,21	652,92	703,30	698,08
GRM_P14	**777,01**	871,67	970,51	933,88	980,46	953,04
GRM_P15	**1.275,39**	1.324,43	1.391,48	1.433,40	1.473,57	1.435,10
GRM_P16	2.317,94	1.643,05	1.711,12	**1.634,44**	1.699,72	1.739,01
GRM_P17	**1.695,34**	1.696,43	1.800,04	1.850,41	1.796,16	1.863,34
GRM_P18	**1.553,37**	1.850,06	2.021,51	2.041,42	2.066,32	2.083,91
GRM_P19	3.665,57	3.005,65	**2.972,64**	3.077,71	3.155,80	3.127,15
GRM_P20	**2.371,96**	2.485,78	2561,56	2.695,46	2.649,14	2.583,37
melhor	13	4	1	1	0	1

Tabela 4.11: Configuração de α: tempo médio de execução das instâncias GRM_P ($w = 5$).

	$\alpha = 0$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	**14,32**	18,31	20,65	20,82	21,12	21,23
GRM_P2	**29,42**	51,70	66,65	63,40	66,12	67,00
GRM_P3	137,07	121,14	121,83	122,01	**120,97**	122,22
GRM_P4	**95,30**	119,17	123,37	126,02	120,97	122,28
GRM_P5	193,72	**132,15**	137,44	138,03	137,63	140,06
GRM_P6	**145,84**	160,71	164,69	167,00	166,85	169,30
GRM_P7	**175,04**	183,36	192,06	192,13	193,05	194,99
GRM_P8	**202,98**	212,55	227,24	228,47	230,63	228,33
GRM_P9	**232,90**	244,98	280,91	282,78	286,53	286,02
GRM_P10	**35,39**	56,84	54,25	51,63	49,61	49,71
GRM_P11	**141,62**	226,43	249,51	267,15	259,66	255,71
GRM_P12	**236,38**	405,95	473,33	486,91	473,89	461,54
GRM_P13	**324,54**	612,21	652,51	668,74	669,89	701,32
GRM_P14	**536,78**	927,72	994,16	1.043,23	1.001,46	963,52
GRM_P15	1.064,86	**1.059,59**	1.130,43	1.159,62	1.151,00	1.130,40
GRM_P16	**946,27**	1.298,26	1.457,37	1.336,11	1.371,48	1.437,10
GRM_P17	**1.253,96**	1.533,02	1.592,81	1.651,05	1.607,05	1.570,54
GRM_P18	**1.461,66**	1.898,23	1.926,82	1.987,33	1.910,28	1.902,66
GRM_P19	**1.738,16**	1.865,76	1.883,49	1.959,56	1.813,52	1.830,02
GRM_P20	**1.373,44**	1.926,96	1.928,98	1.922,69	1.915,16	1.926,77
melhor	17	2	0	0	1	0

Tabela 4.12: Configuração de α: tempo médio de execução das instâncias GRM_P ($w = 10$).

4.4 Estratégia de Filtro

Em virtude dos altos tempos de processamento apresentados pela busca local para as maiores instâncias, uma estratégia de filtro foi utilizada na tentativa de acelerar as iterações do GRASP básico. O filtro é aplicado entre a busca local mais rápida (busca local pelas bordas com teste da menor aresta) e a busca local mais lenta (busca local com teste). Assim, dependendo da solução e da qualidade do ótimo local da primeira busca e da melhor solução encontrada até o momento, executa-se ou não a busca local mais lenta, limitando-se a execução da busca local mais cara somente a ótimos locais da primeira busca suficientemente bons [35].

Seja a solução formada pelo conjunto $S_1^{'}$ de p facilidades abertas e por sua respectiva árvore de Steiner obtida após a execução da busca local mais rápida e $F(S_1^{'})$ o valor de sua função objetivo. Seja também $\lambda > 0$ um parâmetro de corte. Aplica-se a segunda busca local (mais lenta) caso a solução $S_1^{'}$ não tenha sido visitada previamente e $F(S_1^{'}) < (1 + \lambda)\ F^*$, onde F^* é o valor da função objetivo da melhor solução encontrada até o momento. Assim, executa-se a busca local mais lenta quando o ótimo local da busca local mais rápida não foi visitado anteriormente e está suficientemente próximo do valor da função objetivo da melhor solução encontrada até o momento. Como em [35], utilizou-se $\lambda = 1\%$.

Em geral, a utilização do filtro reduz o tempo de processamento às custas de perdas na qualidade média das soluções encontradas. Para verificar quanto o filtro afeta o tempo de processamento e a qualidade da solução, testes foram realizados comparando-se o GRASP sem filtro (GRASPb) com o GRASP com filtro (GRASPf). A Tabela 4.13 apresenta as medidas relativas *drpm* e *cm* utilizadas na análise da qualidade das soluções encontradas pelo GRASPb e GRASPf.

Analisando-se essa tabela, como esperado, o GRASPf perde em qualidade das soluções comparando-se com o GRASPb. Em termos absolutos, dos 180 testes realizados (36 instâncias com cinco execuções cada), o GRASPf manteve o mesmo valor em 170 testes.

	GRASPb	GRASPf
drpm	**-0,0035**	0,0035
cm	**1,43**	1,57

Tabela 4.13: Qualidade relativa na configuração do filtro no GRASP.

A Tabela 4.14 apresenta as medidas relativas *drpm*, *cm* e *trm* utilizadas na análise dos tempos de processamento obtidos pelo GRASPb e GRASPf. Nessa última medida, o GRASPb foi utilizado como algoritmo de referência.

	GRASPb	GRASPf
drpm	37,25	**-37,25**
cm	1,00	**2,00**
trm	1,00	**0,50**

Tabela 4.14: Tempo relativo na configuração do filtro no GRASP.

Na medida *cm*, observa-se que, em todos os testes ocorreram uma diminuição nos tempos de processamento. Na medida *trm*, o GRASPf gastou, em média, 50% do tempo de execução do GRASPb nas instâncias testadas. Em geral, as três medidas relativas mostraram que houve ganhos em tempos de processamento do GRASPf em relação ao GRASPb, com perdas na qualidade média das soluções encontradas.

4.5 Reconexão por Caminhos

O procedimento de reconexão por caminhos, proposto por Glover [19], foi primeiramente utilizado em busca tabu e *scatter search* como estratégia de intensificação explorando trajetórias entre uma solução corrente e soluções de elite. Iniciando-se de uma ou mais soluções, caminhos ou trajetórias que conduzem a soluções de elite são gerados e explorados em busca de melhorias. Um caminho é construído selecionando movimentos que introduzem atributos na solução atual que estão presentes nas soluções de elite. Reconexão por caminhos pode ser visto como uma estratégia que busca atributos presentes em soluções de alta qualidade para serem incorporados a uma outra solução.

O procedimento de reconexão por caminhos foi primeiramente utilizado no contexto do GRASP por Laguna e Martí [33] como estratégia de intensificação aplicada a cada ótimo local. Em geral, duas estratégias básicas têm sido utilizadas [46, 49]:

- aplicá-lo em todos os pares de soluções de elite periodicamente durante as iterações do GRASP ou como um passo de pós-otimização após todas as iterações terem sido realizadas;
- aplicá-lo como estratégia de intensificação após cada ótimo local produzido pela fase de busca local.

A utilização de reconexão por caminhos como estratégia de intensificação tem sido mais efetiva do que simplesmente utilizá-lo como passo de pós-otimização. Em geral, o melhor é combinar intensificação com pós-otimização [46, 49], porém, optou-se por utilizá-lo apenas como estratégia de intensificação.

Sejam duas soluções quaisquer formadas pelos conjuntos S_1 e S_2 compostos por p facilidades abertas e por suas respectivas árvores de Steiner. O

procedimento como estratégia de intensificação atua em um par de soluções (S_1, S_2), onde S_1 é um ótimo local produzido pela busca local em cada iteração do GRASP e S_2 é uma solução de elite escolhida de um conjunto P composto por *MaxElite* soluções de alta qualidade encontradas durante as iterações do GRASP. Inicia-se a trajetória da melhor solução entre S_1 e S_2, pois permite investigar com maior detalhe a vizinhança da melhor solução. Quando investigando em um único sentido, essa estratégia tem apresentado bons resultados na literatura [46, 49].

A Figura 4.3 ilustra o pseudo-código do algoritmo que recebe como parâmetros o par de soluções S_1 (solução inicial) e S_2 (solução alvo ou guia), além da primeira e segunda facilidade mais próxima de cada usuário u, ϕ_1^u e ϕ_2^u, respectivamente.

O algoritmo inicia na linha 2 calculando a diferença simétrica $\Delta(S_1, S_2)$ entre as soluções, o que corresponde ao conjunto de facilidades diferentes encontradas em S_1 e S_2. O valor $|\Delta(S_1, S_2)|$ corresponde a quantidade de trocas de facilidades necessárias para alcançar a solução alvo S_2 partindo-se da solução inicial S_1. A linha 5 do algoritmo consiste em construir, para cada usuário u, uma lista com todas as facilidades ordenadas pela distância ao usuário. Na linha 6, o conjunto de usuários afetados V_a é inicializado com todos os usuários. Na linha 7, as estruturas de dados auxiliares *ganho*, *perda* e *extra* são inicializadas. A idéia do procedimento é gerar um caminho de soluções entre S_1 e S_2 e caso o algoritmo obtenha sucesso, retorna-se a melhor solução encontrada no caminho; caso contrário, retorna-se a melhor solução entre S_1 e S_2 (linha 31). O procedimento termina quando a solução alvo é alcançada, isto é, quando $\Delta(S, S_2) = \emptyset$. Enquanto essa condição não for satisfeita (laço das linhas 8 a 30), em cada iteração, ocorre a atualização eficiente das estruturas de dados auxiliares (linhas 9 a 11). Em seguida, examina-se todas as trocas de facilidades possíveis de serem realizadas na solução corrente S selecionando aquela que maximiza a soma dos lucros obtidos no custo de atendimento dos usuários e no custo de interconexão das facilidades (linha 12). Essa função será explicada com maior detalhe posteriormente. A melhor troca é realizada atualizando-se a solução S (linhas 13 e 14) e as facilidades ainda disponíveis no conjunto da diferença simétrica (linha 15). Quando necessário, a melhor solução S^* é também atualizada (linhas 16 a 19). Nas linhas 20 a 25 ocorre a atualização do conjunto de usuários afetados. As informações presentes nas estruturas auxiliares encontram-se desatualizadas em relação à troca ocorrida e são corrigidas executando-se a função *Desfaz_Atualiza_Estruturas* somente para os usuários afetados (linhas 26 a 28). Por último, na linha 29, ocorre a atualização da primeira e da segunda facilidade mais próxima de cada usuário afetado, levando-se em consideração a troca efetuada.

A Figura 4.4 apresenta a função responsável pela escolha da troca mais

Procedimento `Reconexão_Caminhos`$(S_1, S_2, \phi_1, \phi_2)$
1. $S \leftarrow S_1$;
2. Calcular a diferença simétrica $\Delta(S, S_2)$;
3. $F^* \leftarrow \min\{F(S), F(S_2)\}$;
4. $S^* \leftarrow \text{argmin}\{F(S), F(S_2)\}$;
5. Para cada $u \in V$, ordenar as facilidades de acordo com a distância ao usuário;
6. $V_a \leftarrow V$;
7. $ganho(i) \leftarrow 0, perda(r) \leftarrow 0, extra(i, r) \leftarrow 0, \forall\ i \notin S, \forall\ r \in S$;
8. **Enquanto** $(\Delta(S, S_2) \neq \emptyset)$ **Faça**
9. **Para Todo** $(u \in V_a)$ **Faça**
10. *Atualiza_Estruturas(ganho, perda, extra, u, ϕ_1^u, ϕ_2^u)*;
11. **Fim-Para-Todo**
12. $(i, r, melhor) \leftarrow$ *Melhor_Troca(S, S_2, ganho, perda, extra)*;
13. $S \leftarrow S \setminus \{r\}$;
14. $S \leftarrow S \cup \{i\}$;
15. $\Delta(S, S_2) \leftarrow \Delta(S, S_2) \setminus \{r\}$;
16. **Se** $(F(S) < F^*)$ **Então**
17. $F^* \leftarrow F(S)$;
18. $S^* \leftarrow S$;
19. **Fim-Se**
20. $V_a \leftarrow \emptyset$;
21. **Para Todo** $(u \in V)$ **Faça**
22. **Se** $(\phi_1^u = r$ **ou** $\phi_2^u = r$ **ou** $d_{ui} < d_2^u)$ **Então**
23. $V_a \leftarrow V_a \cup \{u\}$;
24. **Fim-Se**
25. **Fim-Para-Todo**
26. **Para Todo** $(u \in V_a)$ **Faça**
27. *Desfaz_Atualiza_Estruturas(ganho, perda, extra, u, ϕ_1^u, ϕ_2^u)*;
28. **Fim-Para-Todo**
29. Atualizar a primeira e segunda facilidade mais próxima de cada usuário $u \in V_a$;
30. **Fim-Enquanto**
31. **Retorne** S^*;

Fim

Figura 4.3: Pseudo-código do procedimento de reconexão por caminhos.

lucrativa (gulosa) realizada em cada passo do procedimento. A função recebe como parâmetros a solução corrente composta pelo conjunto S de p facilidades abertas e pela árvore de Steiner $T(S)$ correspondente, a solução alvo composta pelo conjunto S_2 de p facilidades abertas, além das estruturas *ganho*, *perda* e *extra*.

```
Função Melhor_Troca(S, S2, ganho, perda, extra)
1.  melhor ← −∞;
2.  Para Todo (i1 ∉ S2 \ S) Faça
3.      Para Todo (r1 ∈ S \ S2) Faça
4.          lucroP(i1, r1) ← ganho(i1) − perda(r1) + extra(i1, r1);
5.          lucroS(i1, r1) ← FT(S) − FT(S');
6.          lucro(i1, r1) ← lucroP(i1, r1) + lucroS(i1, r1);
7.          Se (lucro(i1, r1) > melhor) Então
8.              melhor ← lucro(i1, r1);
9.              i ← i1; r ← r1;
10.         Fim-Se
11.     Fim-Para-Todo
12. Fim-Para-Todo
13. Retorne i, r, melhor;
Fim
```

Figura 4.4: Função que encontra a melhor troca no algoritmo de reconexão por caminhos.

A função é composta por dois laços aninhados. O externo (linhas 2 a 12) percorre as facilidades pertencentes ao conjunto $S_2 \setminus S$ e o interno (linhas 3 a 11) percorre as facilidades pertencentes ao conjunto $S \setminus S_2$. Ambos são executados até que se obtenha a melhor troca possível. Os lucros obtidos no custo de atendimento dos usuários e no custo de inteconexão das facilidades ao se trocar r_i por i_i são calculados nas linhas 4 e 5, respectivamente. O lucro total é calculado na linha 6 e a melhor troca é armazenada nas linhas 7 a 10. Na linha 13, retorna-se as facilidades i e r da melhor troca junto com seu lucro.

Dois aspectos importantes do procedimento são: gerenciamento do conjunto P e seleção de uma solução de elite presente em P em cada iteração.

Observações empíricas mostraram que, quanto maior o caminho percorrido entre as soluções, maior a probabilidade que uma solução melhor seja encontrada [48]. Assim, deve-se evitar aplicar reconexão por caminhos em soluções similares, sendo importante priorizar tanto a qualidade quanto a diversidade das soluções presentes em P. No início do algoritmo, P está vazio. Cada ótimo local é considerado candidato a ser inserido em P se é diferente de cada solução atualmente presente no conjunto. Quando *MaxElite* soluções estão presentes (P está cheio), o ótimo local candidato substitui o pior elemento de P se um dos critérios for satisfeito:

– o custo do ótimo local candidato é melhor do que o custo da melhor solução de elite presente em P, priorizando, assim, a qualidade;

– o custo do ótimo local candidato é melhor do que o custo da pior solução de elite presente em P e é diferente de todas as soluções do conjunto, priorizando, assim, tanto a qualidade quanto a diversidade.

Se o algoritmo encontra uma solução melhor do que a melhor solução encontrada até o momento, esta torna-se também um candidato a inserção em P de acordo com as condições mencionadas acima.

A partir da segunda iteração do GRASP, ocorre a seleção de uma solução de elite presente em P. Uma maneira simples seria selecioná-la aleatoriamente. Porém, pode-se selecionar uma solução semelhante ao ótimo local candidato, diminuindo as chances de encontrar uma boa solução no caminho entre S_1 e S_2. A diferença simétrica representa o tamanho da trajetória explorada entre as soluções. Assim, a seleção de um elemento de P no algoritmo favorece soluções de elite com uma maior diferença simétrica em relação ao ótimo local candidato. Isto é, quanto maior a diferença simétrica entre o ótimo local e a solução de elite, maior a chance de seleção do elemento de P. Esses dois métodos foram comparados em [48] e a seleção com probabilidade proporcional à diferença simétrica apresentou resultados melhores do que a seleção aleatória.

Reconexão por caminhos pode ser visto como uma busca local aplicada à solução S_1, com três principais diferenças:

1. na reconexão por caminhos, o número de movimentos permitidos é muito menor do que na busca local; somente um sub-conjunto das facilidades são candidatos a inserção $(S_2 \setminus S_1)$ e a remoção $(S_1 \setminus S_2)$. À medida que o algoritmo progride, esses subconjuntos tornam-se menores;

2. a busca local termina quando encontra um mínimo local, não aceitando movimentos que piorem a solução corrente. A reconexão por caminhos termina quando encontra a solução alvo, podendo aceitar movimentos que piorem a solução corrente;

3. a busca local utiliza melhoria iterativa selecionando a primeira solução aprimorante na vizinhança da solução corrente e a reconexão por caminhos utiliza descida mais rápida selecionando a melhor solução na vizinhança restrita da solução corrente.

4.5.1 Configuração do Tamanho do Conjunto de Soluções de Elite

Um dos parâmetros a serem ajustados no procedimento de reconexão por caminhos é o tamanho máximo do conjunto de soluções de elite (*MaxElite*). As seguintes configurações foram avaliadas: GRASP com filtro sem reconexão por caminhos (GRASPf) e GRASP com filtro e reconexão por caminhos com $MaxElite = 5, 10, 15$ e 20 (GRASPf_RC5, GRASPf_RC10, GRASPf_RC15 e GRASPf_RC20, respectivamente).

As medidas *drpm* e *cm* serão utilizadas para analisar a qualidade das soluções encontradas, como mostra a Tabela 4.15.

	drpm	*cm*
GRASPf	0,002038	3,18
GRASPf_RC5	-0,000312	3,01
GRASPf_RC10	**-0,001341**	**2,83**
GRASPf_RC15	-0,000578	2,96
GRASPf_RC20	0,000193	3,01

Tabela 4.15: Qualidade relativa na configuração de *MaxElite* no GRASP.

Analisando-se a Tabela 4.15, em ambas as medidas, o melhor resultado foi obtido pelo GRASPf_RC10. A adição do procedimento de reconexão por caminhos não pode piorar a qualidade da solução encontrada pelo GRASPf. Assim, dos 180 testes realizados (36 instâncias com cinco execuções cada), GRASPf_RC5, GRASPf_RC10, GRASPf_RC15 e GRASPf_RC20 melhoraram a qualidade da solução encontrada pelo GRASPf em 5, 7, 6 e 5 testes, respectivamente.

Para a análise dos tempos de processamento, os valores obtidos pelos algoritmos nas medidas *drpm*, *cm* e *trm* são mostrados na Tabela 4.16. Nessa última medida, utilizou-se o algoritmo GRASPf como referência.

	drpm	*cm*	*trm*
GRASPf	**-8,75**	**1,51**	**1,00**
GRASPf_RC5	2,04	3,33	1,13
GRASPf_RC10	1,93	3,00	1,13
GRASPf_RC15	2,19	3,46	1,13
GRASPf_RC20	2,59	3,69	1,14

Tabela 4.16: Tempo relativo na configuração de *MaxElite* no GRASP.

Analisando-se a Tabela 4.16, como esperado, os menores tempos de processamento foram obtidos pelo GRASPf, pois não existe a necessidade de controle e gerenciamento do conjunto de soluções de elite. Os algoritmos com reconexão por caminhos não apresentaram uma diferença significativa entre eles, porém, em geral, quanto maior o número de soluções a serem gerenciadas no conjunto P, maior o tempo de processamento do algoritmo. Na medida

trm, o tempo de processamento mais elevado foi obtido pelo GRASPf_RC20, que excedeu 14%, em média, o tempo de execução do GRASPf nas instâncias testadas.

Levando-se em consideração a qualidade das soluções e os tempos de processamento em relação ao GRASPf, os melhores resultados foram obtidos pelo GRASPf_RC10. Assim, o tamanho do conjunto de soluções de elite foi configurado em dez.

4.5.2 Comparação entre o GRASP com Filtro e o GRASP com Filtro e Reconexão por Caminhos

Uma hipótese a ser analisada é descobrir se o tempo extra gasto na execução do algoritmo de reconexão por caminhos, se utilizado pelo GRASP com filtro para iterações adicionais produziria resultados melhores. Com o objetivo de testá-la, comparações serão realizadas entre o GRASP com filtro sem reconexão por caminhos (GRASPf) e o GRASP com filtro com reconexão por caminhos e $MaxElite = 10$ (GRASPf_RC). Para isso, serão utilizados gráficos que comparam experimentalmente diferentes algoritmos aleatórios ou diferentes versões do mesmo algoritmo aleatório [1]. Os gráficos mostram a distribuição de probabilidade empírica da variável aleatória tempo gasto para encontrar um valor alvo. Para cada algoritmo, fixa-se o valor alvo e faz-se 200 execuções independentes. O algoritmo termina quando uma solução de valor menor ou igual ao valor alvo é encontrada. Associa-se ao i-ésimo menor tempo de execução t_i à probabilidade $prob_i = (i - 1/2)/200$, para $i = 1, 2, \ldots, 200$. As seguintes instâncias (com seus respectivos valores alvo entre parênteses) foram utilizadas para comparação: GRM_P11 (11086), GRM_P17 (13276), ORM_P9 (6436) e ORM_P15 (6363) para $w = 5$.

As Figuras 4.5 e 4.6 mostram que as curvas do GRASPf e do GRASPf_RC são próximas para as instâncias GRM_P11 e ORM_P9, respectivamente. Os ganhos obtidos nas instâncias menores com a adição do procedimento são pequenos ou quase nulos nas instâncias consideradas.

Quando aumenta-se o tamanho da instância, os ganhos com o procedimento de reconexão por caminhos tornam-se significativos, como mostram as Figuras 4.7 e 4.8.

Os gráficos ilustram que a probabilidade de encontrar uma solução com valor pelo menos tão bom quanto o valor alvo aumenta do GRASPf para o GRASPf_RC em ambas as instâncias. Observando-se a Figura 4.7, a probabilidade de encontrar o valor alvo em menos do que 140 segundos é de 100% para o GRASPf_RC e aproximadamente 73% para o GRASPf. A figura mostra claramente que o GRASPf_RC é mais rápido do que o GRASPf para encontrar o valor alvo 13276. Já na Figura 4.8, o ganho não é tão significativo

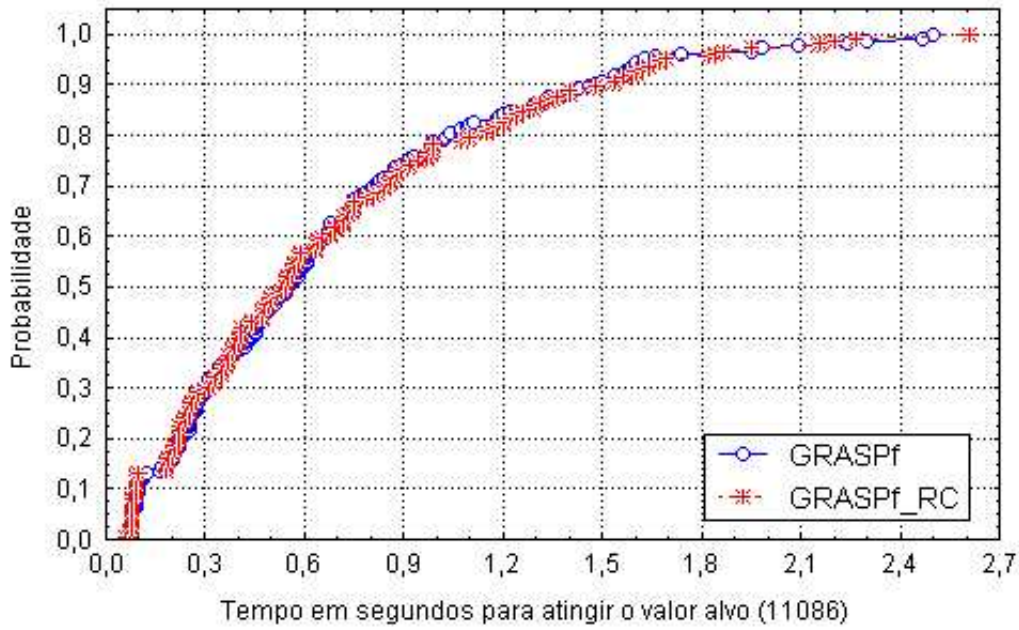

Figura 4.5: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 11086 para a instância GRM_P11 ($w = 5$).

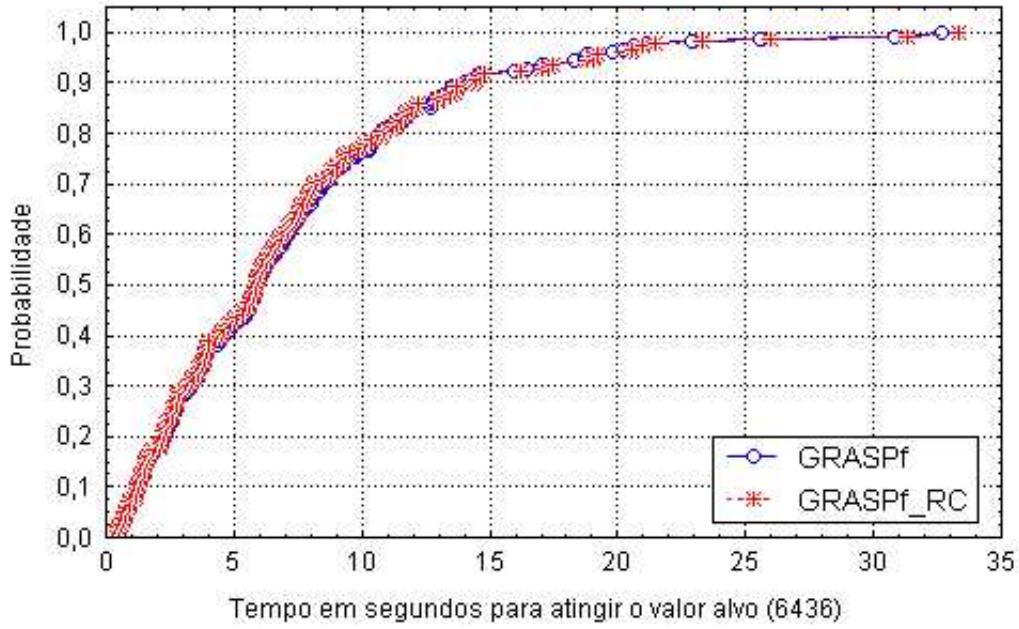

Figura 4.6: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 6436 para a instância ORM_P9 ($w = 5$).

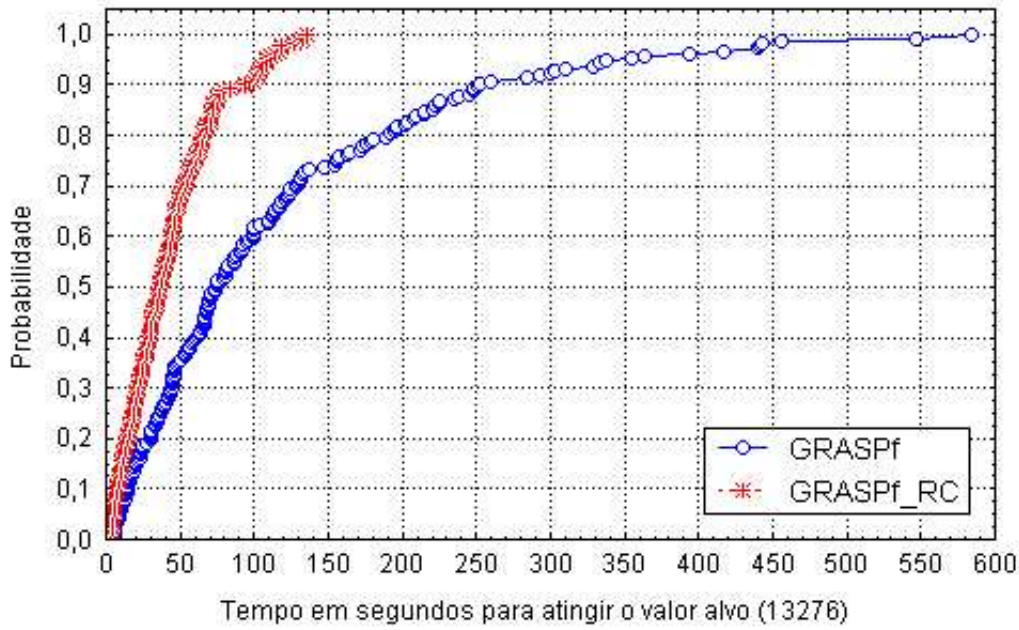

Figura 4.7: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 13276 para a instância GRM_P17 ($w = 5$).

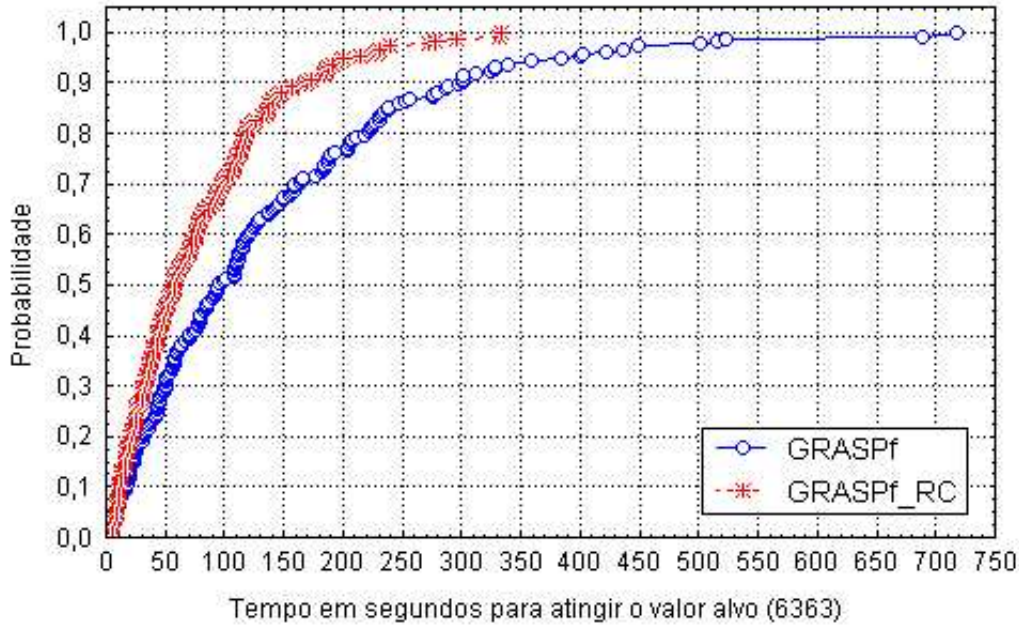

Figura 4.8: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 6363 para a instância ORM_P15 ($w = 5$).

quanto ao anteriormente apresentado, mas o GRASPf_RC se mostra mais rápido do que o GRASPf para encontrar o valor alvo 6363. Para o primeiro algoritmo, a probabilidade de encontrar o valor alvo em menos do que 340 segundos é de 100% enquanto que para o segundo é aproximadamente 94%.

4.6 Algoritmo GRASP com Filtro e Reconexão por Caminhos

A heurística GRASP com filtro e reconexão por caminhos é mostrada na Figura 4.9.

Procedimento GRASP_Filtro_Reconexão_Caminhos (α, *semente*)
1. $P \leftarrow \emptyset$;
2. $F^* \leftarrow +\infty$;
3. **Para** $it = 1, \ldots, MaxIter$ **Faça**
4. $S \leftarrow$ Fase_Construção(α, *semente*);
5. $S_1^{'} \leftarrow$ BL_BTMA(S);
6. **Se** ($S_1^{'}$ não foi visitado previamente) **e** ($F(S_1^{'}) < (1+\lambda) \times F^*$) **Então**
7. $S^{'} \leftarrow$ BL_Teste($S_1^{'}$);
8. **Fim-Se**
9. **Senão**
10. $S^{'} \leftarrow S_1^{'}$;
11. **Fim-Senão**
12. $S_1 \leftarrow S^{'}$;
13. Atualizar o conjunto de soluções de elite P com S_1;
14. **Se** ($it \geq 2$) **Então**
15. Escolher uma solução $S_2 \in P$;
16. **Se** ($F(S_1) \leq F(S_2)$) **Então**
17. $S^{''} \leftarrow$ Reconexão_por_Caminhos(S_1, S_2);
18. **Fim-Se**
19. **Senão**
20. $S^{''} \leftarrow$ Reconexão_por_Caminhos(S_2, S_1);
21. **Fim-Senão**
22. **Se** ($F(S^{''}) < F^*$) **Então**
23. $F^* \leftarrow F(S^{''})$;
24. $S^* \leftarrow S^{''}$;
25. **Fim-Se**
26. **Fim-Se**
27. **Senão**
28. $F^* \leftarrow F(S^{'})$;
29. $S^* \leftarrow S^{'}$;
30. **Fim-Senão**
31. **Fim-Para**
32. **Retorne** S^***;**

Fim

Figura 4.9: Algoritmo GRASP com filtro e reconexão por caminhos.

O conjunto de soluções de elite P com tamanho máximo igual a dez está inicialmente vazio (linha 1). Executa-se o procedimento um número máximo de iterações (parâmetro *MaxIter*). Em cada iteração (laço das linhas 3 a 31), uma solução inicial é gerada pela fase de construção na linha 4. Em seguida, executa-se a busca local mais rápida (busca local pelas bordas com teste da menor aresta) na solução gerada pela fase de construção (linha 5). A linha 6 é responsável pelo teste da estratégia de filtro e caso ambas as condições sejam satisfeitas, executa-se a busca local mais lenta (busca local com teste) no ótimo local encontrado pela primeira busca (linha 7); caso contrário, se pelo menos uma das condições não for satisfeita, a busca local com teste não é executada. O ótimo local encontrado pela busca local mais rápida ou pela busca local mais lenta é um candidato a inserção no conjunto de soluções de elite. Caso satisfaça as condições de diversidade e qualidade apresentadas na Seção 4.5, torna-se um novo elemento do conjunto P (linha 13). Na linha 14, o procedimento de reconexão por caminhos começa a atuar a partir da segunda iteração. Primeiramente, uma solução é escolhida do conjunto de soluções de elite privilegiando-se elementos com uma maior diferença simétrica em relação ao ótimo local candidato (linha 15). Na linha 16, determina-se qual será a solução inicial e a solução alvo entre o ótimo local candidato e o elemento de P. A melhor solução será a solução inicial e a pior solução será a solução alvo (guia). Assim, de acordo com esse teste, executa-se o procedimento de reconexão por caminhos entre a solução inicial e a solução alvo nas linhas 17 ou 20. Caso o procedimento obtenha sucesso, retorna-se a melhor solução encontrada no caminho; caso contrário, retorna-se a melhor solução entre S_1 e S_2. Durante cada passo, se o procedimento encontra uma solução melhor do que a melhor solução encontrada até o momento, esta também torna-se uma candidata a inserção em P. Quando necessário, a melhor solução S^* é atualizada (linhas 22 a 25). Na primeira iteração, atualiza-se a melhor solução encontrada nas linhas 27 a 30. A melhor solução encontrada pelo GRASP com filtro e reconexão por caminhos é retornada na linha 32.

4.7 Considerações Finais

Este capítulo apresentou a heurística GRASP com filtro e reconexão por caminhos para o problema das p-medianas conectadas.

O algoritmo é composto por uma fase de construção cuja RCL é construída em função de dois critérios: critério de p-medianas e de Steiner. Comparando-se diversos valores de α, escolheu-se a configuração $\alpha = 0,4$, próxima da escolha gulosa para garantir qualidade média, mas suficientemente grande para garantir diversidade. A fase de busca local utilizada é aquela proposta no Capítulo 3.

Com o objetivo de tentar acelerar as iterações do GRASP, uma estratégia de filtro apresentada na literatura foi testada, atuando entre a busca local pelas bordas com teste da menor aresta e a busca local com teste. Em geral, a utilização do filtro reduz o tempo de processamento do algoritmo às custas de uma pequena perda na qualidade das soluções.

Com a finalidade de melhorar a qualidade das soluções encontradas pelo GRASP, o procedimento de reconexão por caminhos foi utilizado como estratégia de intensificação. Gráficos de distribuição de probabilidade empírica do tempo gasto para atingir o valor alvo foram mostrados com o objetivo de comparar o GRASP com filtro com e sem reconexão por caminhos. Os resultados mostraram que os ganhos são significativos em instâncias maiores. Isto pode ser explicado pelo fato de que, nessas instâncias, o caminho percorrido entre o ótimo local e o elemento do conjunto de soluções de elite tende a ser maior (maior diferença simétrica), aumentando-se a probabilidade de encontrar uma solução melhor do que a solução inicial e a solução alvo.

5
VNS com Filtro e Reconexão por Caminhos

A metaheurística VNS *(Variable Neighborhood Search)* foi proposta por Mladenović e Hansen [40] e possui como idéia básica a mudança de vizinhanças realizada da seguinte maneira: aplica-se um algoritmo de busca local, explorando vizinhanças próximas da solução corrente até que nenhuma solução melhor seja encontrada. Na tentativa de fugir de ótimos locais, aumenta-se a vizinhança, buscando-se soluções mais distantes da solução corrente.

As vizinhanças são exploradas em ordem crescente de distância da solução corrente. Para a construção de diferentes estruturas de vizinhanças, define-se uma métrica de distância entre duas soluções quaisquer, criando-se as vizinhanças baseadas nessa métrica [27].

Essa metaheurística foi escolhida devido às recentes aplicações de sucesso apresentadas na literatura [2, 11]. A solução inicial do VNS para o problema das p-medianas conectadas será gerada, como no GRASP com filtro e reconexão por caminhos, pela heurística construtiva descrita na Seção 4.1 com $\alpha = 0,4$ e o algoritmo de busca local utilizado será aquele descrito no Capítulo 3 (BL_Conc).

A implementação de uma metaheurística envolve um balanceamento entre tempo de processamento e qualidade da solução. Com o objetivo de diminuir os tempos de processamento obtidos pelo VNS, a estratégia de filtro apresentada na Seção 4.4 será utilizada entre a busca local pelas bordas com teste da menor aresta e a busca local com teste. Por outro lado, para melhorar a qualidade das soluções encontradas, uma simples modificação no VNS será incorporada, perturbando-se a solução corrente mais de uma vez e escolhendo a melhor solução para aplicação da busca local (parâmetro b) e o procedimento de reconexão por caminhos será utilizado como estratégia de intensificação, como apresentado na Seção 4.5.

Este capítulo está organizado como se segue: a Seção 5.1 apresenta a descrição do algoritmo VNS em sua versão básica; a Seção 5.2 mostra o ambiente de teste, as instâncias e as medidas utilizadas na configuração do VNS; a Seção 5.3 apresenta o ajuste dos parâmetros da metaheurística; a Seção 5.4 descreve os resultados computacionais do VNS com o filtro em relação ao VNS em sua versão básica. A Seção 5.5 mostra a configuração do parâmetro b e a Seção 5.6 apresenta a configuração do procedimento de reconexão por caminhos e

resultados computacionais em relação ao VNS que não o utiliza. A Seção 5.7 descreve a heurística VNS com filtro e reconexão por caminhos proposta para o problema e, por último, a Seção 5.8 apresenta as conclusões e considerações finais do capítulo.

5.1 Algoritmo VNS Básico

Sejam duas soluções quaisquer formadas pelos conjuntos S e $S^{'}$ compostos por p facilidades abertas e por suas respectivas árvores de Steiner. Define-se a distância entre as duas soluções igual a k, se e somente se, S e $S^{'}$ diferem em k facilidades. Assim, a função de distância simétrica ρ pode ser definida como:

$$\rho(S, S^{'}) = |S \backslash S^{'}| = |S^{'} \backslash S|, \forall\ S, S^{'}. \tag{5-1}$$

As estruturas de vizinhança utilizadas são induzidas por ρ, isto é, k facilidades pertencentes à solução corrente são substituídas por k facilidades da sua vizinhança ($k \leq p$). Seja $N_k,\ k = 1, \ldots, k_{max},\ k_{max} \leq p$, o conjunto de estruturas de vizinhanças selecionadas e $N_k(S)$ a vizinhança da solução S. Mais formalmente, $S^{'} \in N_k(S) \Leftrightarrow \rho(S, S^{'}) = k$.

A Figura 5.1 apresenta o algoritmo VNS em sua versão básica para o problema das p-medianas conectadas.

```
Procedimento VNS_Básico(α, semente)
1.  F* ← +∞;
2.  S ← Fase_Construção(α, semente);
3.  Enquanto (critério de parada) Faça
4.      k ← 1;
5.      Enquanto (k ≤ k_max) Faça
6.          Gerar uma solução aleatória S' ∈ N_k(S);
7.          S' ← BL_Conc (S');
8.          Se (F(S') < F*) Então
9.              F* ← F(S');
10.             S* ← S';
11.             S ← S';
12.             k ← 1;
13.         Fim-Se
14.         Senão
15.             k ← k + 1;
16.         Fim-Senão
17.     Fim-Enquanto
18. Fim-Enquanto
19. Retorne S*;
Fim
```

Figura 5.1: Algoritmo VNS básico.

A solução inicial é gerada na linha 2 como descrito na Seção 4.1. Executa-se o VNS (laço das linhas 3 a 18) enquanto o critério de parada não for satisfeito. O critério de parada pode ser o tempo de processamento, o número de iterações ou mesmo o número de iterações sem melhoria no valor da função objetivo. O algoritmo inicia explorando a vizinhança que possui a menor distância em relação à solução corrente (linha 4). Enquanto a maior estrututura de vizinhança não for alcançada (parâmetro k_{max}) no laço das linhas 5 a 17, os seguintes passos são executados. Na linha 6, gera-se aleatoriamente uma solução $S^{'} \in N_k(S)$ na $k-$ésima vizinhança de S. Uma perturbação aleatória é utilizada com a finalidade de evitar ciclagem, o que poderia ocorrer se qualquer regra determinística fosse utilizada. A solução $S^{'}$ é então utilizada como solução inicial para o algoritmo de busca local (BL_Conc), na linha 7. A execução das linhas 6 e 7 constitui uma iteração no VNS. Se o valor da função objetivo do ótimo local é melhor do que o valor da função objetivo da melhor solução encontrada até o momento (linha 8), atualiza-se a melhor solução, a solução corrente e retorna-se à vizinhança N_1 (linhas 9 a 12). Caso contrário, aumenta-se a distância entre a solução corrente S e a nova solução $S^{'}$ que será gerada aleatoriamente (linha 15). A melhor solução encontrada pelo algoritmo é retornada na linha 19.

5.2 Ambiente de Teste, Instâncias e Medidas Utilizadas

A heurística VNS foi implementada em C com o parâmetro de otimização -O3 e executada em uma máquina Pentium IV 3.2 GHz com 1 Gbyte de memória RAM sob o sistema operacional Linux RedHat 9.0. O gerador de números aleatórios utilizado foi o de Matsumoto e Nishimura [53].

Para os testes da estratégia de filtro, ajuste do parâmetro b e configuração do procedimento de reconexão por caminhos, utilizou-se um sub-conjunto das instâncias proporcionais apresentadas na Seção 3.3.4, totalizando 36 problemas: ORM_P1 a ORM_P18 e GRM_P1 a GRM_P18 ($w = 5$). Para a avaliação da qualidade das soluções e dos tempos de processamento, as medidas relativas, descritas no Capítulo 3, foram utilizadas.

Para o ajuste do parâmetro k_{max}, utilizou-se um sub-conjunto maior de instâncias proporcionais apresentadas na Seção 3.3.4, totalizando-se 120 problemas: GRM_P1 a GRM_P20 e ORM_P1 a ORM_P20 ($w = 2$, $w = 5$ e $w = 10$). Para a avaliação da qualidade das soluções e dos tempos de processamento, considera-se, respectivamente, para cada instância, a média em termos de qualidade e tempo obtida pelo algoritmo em um determinado número de execuções.

Para todos os testes e ajuste de parâmetros, foram realizadas cinco execuções com diferentes sementes aleatórias para cada par instância-algoritmo.

5.3 Configuração dos Parâmetros do VNS

Os parâmetros a serem ajustados no VNS em sua versão básica são: critério de parada do algoritmo e número total de vizinhanças utilizadas, representado pela variável k_{max} na Figura 5.1.

O critério de parada escolhido para os testes de configuração do VNS foi o número de iterações realizadas. Uma iteração do VNS corresponde à execução de uma perturbação aleatória seguida de uma busca local. Quanto maior o número de iterações, maior a probabilidade de encontrar uma solução melhor. Porém, devido aos tempos de processamento elevados apresentados por BL_Conc nas maiores instâncias no Capítulo 3, o número de iterações foi estipulado em 100, um valor baixo em relação ao encontrado em algumas aplicações de VNS na literatura, como em [26].

O parâmetro que influencia a qualidade das soluções encontradas e os tempos de processamento obtidos pelo algoritmo VNS é o número total de vizinhanças utilizadas $k_{max} \leq p$. Com o objetivo de obter um compromisso entre qualidade e tempo, testes foram realizados com valores de $k_{max} = 4, 8, 10, 15, 20$ e p para o sub-conjunto de instâncias testadas. Nas Tabelas 5.3 a 5.14, os valores em negrito representam o menor valor obtido pelas configurações para cada instância.

Analisando-se a qualidade média das soluções encontradas pelas configurações do VNS (Tabelas 5.3 a 5.8), quanto maior o valor de k_{max}, maior a possibilidade de fuga dos ótimos locais e, conseqüentemente, aumenta-se a probabilidade de encontrar soluções melhores. Quando o valor de k_{max} é menor, aumenta-se a chance de convergência prematura nos ótimos locais. Somando-se o número de instâncias em que os algoritmos encontraram o menor valor entre todos os métodos, incluindo-se os casos de empate no primeiro lugar para as classes testadas (linha *melhor* nas tabelas), as variantes de VNS com $k_{max} = 4, 8, 10, 15, 20$ e p obtiveram os seguintes resultados, respectivamente: 88, 98, 96, 107, 106 e 102, de um total de 120 instâncias.

Já na análise dos tempos médios de processamento obtidos pelas configurações do VNS (Tabelas 5.9 a 5.14), ocorre o inverso. Valores menores de k_{max} obtêm resultados melhores, isto é, quanto menor o valor do parâmetro, mais rápida é a heurística. Somando-se o número de instâncias em que os algoritmos encontraram o menor valor entre todos os métodos, incluindo-se os casos de empate no primeiro lugar para as classes testadas (linha *melhor* nas tabelas), as variantes de VNS com $k_{max} = 4, 8, 10, 15, 20$ e p obtiveram os seguintes resultados, respectivamente: 106, 8, 4, 3, 1 e 0, de um total de 120 instâncias.

A configuração $k_{max} = 4$ obteve os melhores tempos de processamento, porém os piores resultados em termos de qualidade das soluções. Já a con-

figuração $k_{max} = p$ obteve soluções de qualidade comparável às melhores configurações ($k_{max} = 15$ e 20), porém, apresentou os piores tempos de processamento. Analisando-se qualidade e tempo, as configurações que obtiveram os melhores resultados foram $k_{max} = 15$ e 20. Optou-se por utilizar a configuração $k_{max} = 15$.

5.4 Estratégia de Filtro

A heurística GRASP, proposta no Capítulo 4, utilizou uma estratégia de filtro com o objetivo de acelerar as iterações do algoritmo. Com a mesma finalidade, o VNS utilizará a mesma estratégia entre a busca local pelas bordas com teste da menor aresta e a busca local com teste, explicada na Seção 4.4, pois BL_Conc apresentou tempos de processamento elevados nas maiores instâncias testadas, conforme mostrado no Capítulo 3.

Em geral, a utilização do filtro reduz o tempo de processamento às custas de perdas na qualidade média das soluções encontradas. Para verificar essa hipótese, testes foram realizados comparando-se o VNS sem filtro (VNSb) com o VNS com filtro (VNSf). A Tabela 5.1 apresenta as medidas relativas *drpm* e *cm* utilizadas na análise da qualidade das soluções encontradas pelo VNSb e VNSf.

	VNSb	VNSf
drpm	**-0,0019**	0,0019
cm	**1,42**	1,58

Tabela 5.1: Qualidade relativa na configuração do filtro no VNS.

Analisando-se a qualidade das soluções encontradas nas medidas *drpm* e *cm*, como esperado, o VNSf perde em qualidade das soluções comparando-se com o VNSb. Em números absolutos, das 180 execuções realizadas (36 instâncias com cinco execuções cada), VNSf piora a solução encontrada pelo VNSb somente em 13 execuções.

A Tabela 5.2 apresenta as medidas relativas *drpm*, *cm* e *trm* utilizadas na análise dos tempos de processamento obtidos pelo VNSb e VNSf. Nessa última medida, o VNSb foi utilizado como algoritmo de referência.

	VNSb	VNSf
drpm	39,07	**-39,07**
cm	1,97	**1,03**
trm	1,00	**0,47**

Tabela 5.2: Tempo relativo na configuração do filtro no VNS.

As três medidas relativas mostraram que, ao se utilizar o filtro no VNS, ganhos em tempos de processamento foram obtidos em relação ao VNSb. Na medida *trm*, o VNSf gastou, em média, 47% do tempo de execução do VNSb.

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**6443,00**	**6443,00**	**6443,00**	**6443,00**	**6443,00**	**6443,00**
ORM_P2	**5228,00**	**5228,00**	**5228,00**	**5228,00**	**5228,00**	**5228,00**
ORM_P3	**5368,00**	**5368,00**	**5368,00**	**5368,00**	**5368,00**	**5368,00**
ORM_P4	5127,00	5127,00	**5126,20**	5127,00	5127,00	5127,00
ORM_P5	**3663,00**	**3663,00**	**3663,00**	**3663,00**	**3663,00**	**3663,00**
ORM_P6	**8185,00**	**8185,00**	**8185,00**	**8185,00**	**8185,00**	**8185,00**
ORM_P7	**6217,00**	**6217,00**	**6217,00**	**6217,00**	**6217,00**	**6217,00**
ORM_P8	5869,20	5866,60	5866,00	**5865,00**	**5865,00**	**5865,00**
ORM_P9	**4699,00**	**4699,00**	4699,40	**4699,00**	4699,40	4699,20
ORM_P10	**3457,00**	**3457,00**	**3457,00**	**3457,00**	**3457,00**	**3457,00**
ORM_P11	**7855,00**	**7855,00**	**7855,00**	**7855,00**	**7855,00**	**7855,00**
ORM_P12	**7074,00**	**7074,00**	**7074,00**	**7074,00**	**7074,00**	**7074,00**
ORM_P13	**5539,00**	**5539,00**	**5539,00**	**5539,00**	**5539,00**	**5539,00**
ORM_P14	4981,40	**4981,00**	4981,40	**4981,00**	**4981,00**	**4981,00**
ORM_P15	4477,60	**4477,00**	**4477,00**	**4477,00**	**4477,00**	**4477,00**
ORM_P16	**8292,00**	**8292,00**	**8292,00**	**8292,00**	**8292,00**	**8292,00**
ORM_P17	**7279,00**	**7279,00**	**7279,00**	**7279,00**	**7279,00**	**7279,00**
ORM_P18	5989,00	5987,60	5989,00	**5987,20**	5989,00	5990,00
ORM_P19	4787,20	**4787,00**	4788,00	4787,20	4787,20	4787,80
ORM_P20	4860,80	4860,60	**4860,00**	**4860,00**	**4860,00**	**4860,00**
melhor	13	16	15	18	16	16

Tabela 5.3: Configuração de k_{max}: qualidade média das instâncias ORM_P1 a ORM_P20 ($w = 2$).

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**7184,00**	**7184,00**	**7184,00**	**7184,00**	**7184,00**	**7184,00**
ORM_P2	**6572,00**	**6572,00**	**6572,00**	**6572,00**	**6572,00**	**6572,00**
ORM_P3	**6776,00**	**6776,00**	**6776,00**	**6776,00**	**6776,00**	**6776,00**
ORM_P4	**6944,00**	6944,00	**6944,00**	**6944,00**	**6944,00**	**6944,00**
ORM_P5	**5292,00**	**5292,00**	**5292,00**	**5292,00**	**5292,00**	**5292,00**
ORM_P6	**8662,00**	**8662,00**	**8662,00**	**8662,00**	**8662,00**	**8662,00**
ORM_P7	6984,80	6982,60	**6980,80**	**6980,80**	**6980,80**	**6980,80**
ORM_P8	7209,20	7205,00	**7203,00**	**7203,00**	**7203,00**	**7203,00**
ORM_P9	**6436,00**	**6436,00**	**6436,00**	**6436,00**	**6436,00**	**6436,00**
ORM_P10	5004,40	**5004,00**	**5004,00**	**5004,00**	**5004,00**	**5004,00**
ORM_P11	**8062,00**	**8062,00**	**8062,00**	**8062,00**	**8062,00**	**8062,00**
ORM_P12	**7666,00**	**7666,00**	**7666,00**	**7666,00**	**7666,00**	**7666,00**
ORM_P13	**6693,00**	**6693,00**	**6693,00**	**6693,00**	**6693,00**	**6693,00**
ORM_P14	6633,60	6633,60	6632,40	**6632,00**	**6632,00**	**6632,00**
ORM_P15	6369,60	6365,20	6364,60	6364,00	6363,80	**6363,60**
ORM_P16	**8458,00**	**8458,00**	**8458,00**	**8458,00**	**8458,00**	**8458,00**
ORM_P17	**7676,00**	**7676,00**	**7676,00**	**7676,00**	**7676,00**	**7676,00**
ORM_P18	**7259,00**	**7259,00**	**7259,00**	**7259,00**	**7259,00**	**7259,00**
ORM_P19	6538,80	6538,20	6538,60	6537,20	**6537,00**	6537,80
ORM_P20	7282,00	7275,40	7277,60	**7269,00**	7273,80	7271,80
melhor	13	14	16	18	18	18

Tabela 5.4: Configuração de k_{max}: qualidade média das instâncias ORM_P1 a ORM_P20 ($w = 5$).

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**8146,00**	**8146,00**	**8146,00**	**8146,00**	**8146,00**	**8146,00**
ORM_P2	**7706,00**	**7706,00**	**7706,00**	**7706,00**	**7706,00**	**7706,00**
ORM_P3	**8265,00**	**8265,00**	**8265,00**	**8265,00**	**8265,00**	**8265,00**
ORM_P4	**9135,00**	**9135,00**	**9135,00**	**9135,00**	**9135,00**	**9135,00**
ORM_P5	7498,60	**7485,00**	**7485,00**	**7485,00**	**7485,00**	**7485,00**
ORM_P6	**9387,00**	**9387,00**	**9387,00**	**9387,00**	**9387,00**	**9387,00**
ORM_P7	7879,60	7879,00	**7878,00**	**7878,00**	**7878,00**	**7878,00**
ORM_P8	**8632,00**	**8632,00**	**8632,00**	**8632,00**	**8632,00**	**8632,00**
ORM_P9	**8174,00**	**8174,00**	8175,40	**8174,00**	**8174,00**	**8174,00**
ORM_P10	6900,80	6905,20	6898,80	**6895,20**	6896,20	6895,40
ORM_P11	**8383,00**	**8383,00**	**8383,00**	**8383,00**	**8383,00**	**8383,00**
ORM_P12	**8426,00**	**8426,00**	**8426,00**	**8426,00**	**8426,00**	**8426,00**
ORM_P13	**7679,00**	**7679,00**	**7679,00**	**7679,00**	**7679,00**	**7679,00**
ORM_P14	8507,80	8506,40	8505,60	8506,60	**8505,20**	8505,40
ORM_P15	8922,60	8901,80	8885,40	8906,00	8876,80	**8870,60**
ORM_P16	**8728,00**	**8728,00**	**8728,00**	**8728,00**	**8728,00**	**8728,00**
ORM_P17	**8317,00**	**8317,00**	**8317,00**	**8317,00**	**8317,00**	**8317,00**
ORM_P18	**8547,00**	**8547,00**	**8547,00**	**8547,00**	**8547,00**	**8547,00**
ORM_P19	8328,00	8329,80	8319,00	**8316,60**	8322,60	8325,00
ORM_P20	10504,40	10499,80	10497,60	10477,60	10457,60	**10449,40**
melhor	13	14	14	17	16	17

Tabela 5.5: Configuração de k_{max}: qualidade média das instâncias ORM_P1 a ORM_P20 ($w = 10$).

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	**6403,00**	**6403,00**	**6403,00**	**6403,00**	**6403,00**	**6403,00**
GRM_P2	5489,80	**5489,00**	**5489,00**	**5489,00**	**5489,00**	**5489,00**
GRM_P3	**5157,00**	**5157,00**	**5157,00**	**5157,00**	**5157,00**	**5157,00**
GRM_P4	**5246,00**	**5246,00**	**5246,00**	**5246,00**	**5246,00**	**5246,00**
GRM_P5	**5403,00**	**5403,00**	**5403,00**	**5403,00**	**5403,00**	**5403,00**
GRM_P6	**5580,00**	**5580,00**	**5580,00**	**5580,00**	**5580,00**	**5580,00**
GRM_P7	**5766,00**	**5766,00**	**5766,00**	**5766,00**	**5766,00**	**5766,00**
GRM_P8	**5961,00**	**5961,00**	**5961,00**	**5961,00**	**5961,00**	**5961,00**
GRM_P9	**6364,00**	**6364,00**	**6364,00**	**6364,00**	**6364,00**	**6364,00**
GRM_P10	**11475,00**	**11475,00**	**11475,00**	**11475,00**	**11475,00**	**11475,00**
GRM_P11	**9862,00**	**9862,00**	**9862,00**	**9862,00**	**9862,00**	**9862,00**
GRM_P12	**8910,00**	**8910,00**	**8910,00**	**8910,00**	**8910,00**	**8910,00**
GRM_P13	**8352,00**	**8352,00**	**8352,00**	**8352,00**	**8352,00**	**8352,00**
GRM_P14	8181,60	8186,80	8187,80	8184,00	**8178,60**	8179,60
GRM_P15	8137,20	**8133,00**	**8133,00**	**8133,00**	**8133,00**	8134,60
GRM_P16	**8243,80**	8244,40	8244,20	8244,00	8244,00	8244,20
GRM_P17	**8372,00**	**8372,00**	**8372,00**	**8372,00**	**8372,00**	**8372,00**
GRM_P18	**8530,00**	**8530,00**	**8530,00**	**8530,00**	**8530,00**	**8530,00**
GRM_P19	**8701,00**	**8701,00**	**8701,00**	**8701,00**	**8701,00**	**8701,00**
GRM_P20	**9080,00**	**9080,00**	**9080,00**	**9080,00**	**9080,00**	**9080,00**
melhor	17	18	18	18	19	17

Tabela 5.6: Configuração de k_{max}: qualidade média das instâncias GRM_P ($w = 2$).

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	**7036,00**	**7036,00**	**7036,00**	**7036,00**	**7036,00**	**7036,00**
GRM_P2	**6662,00**	**6662,00**	**6662,00**	**6662,00**	**6662,00**	**6662,00**
GRM_P3	**6923,00**	**6923,00**	**6923,00**	**6923,00**	**6923,00**	**6923,00**
GRM_P4	**7602,00**	**7602,00**	**7602,00**	**7602,00**	**7602,00**	**7602,00**
GRM_P5	**8379,00**	**8379,00**	**8379,00**	**8379,00**	**8379,00**	**8379,00**
GRM_P6	**9178,00**	**9178,00**	**9178,00**	**9178,00**	**9178,00**	**9178,00**
GRM_P7	**9984,00**	**9984,00**	**9984,00**	**9984,00**	**9984,00**	**9984,00**
GRM_P8	**10802,00**	**10802,00**	**10802,00**	**10802,00**	**10802,00**	**10802,00**
GRM_P9	**12447,00**	**12447,00**	**12447,00**	**12447,00**	**12447,00**	**12447,00**
GRM_P10	**12202,00**	**12202,00**	**12202,00**	**12202,00**	**12202,00**	**12202,00**
GRM_P11	**11086,00**	**11086,00**	**11086,00**	**11086,00**	**11086,00**	**11086,00**
GRM_P12	10776,60	10769,40	**10767,00**	**10767,00**	**10767,00**	**10767,00**
GRM_P13	10904,40	**10902,00**	10902,80	**10902,00**	10902,80	10902,80
GRM_P14	**11293,00**	**11293,00**	11294,60	11293,80	**11293,00**	11293,80
GRM_P15	11876,60	**11874,40**	11876,40	11876,20	**11874,40**	11874,60
GRM_P16	12542,60	**12539,00**	12541,40	12541,40	**12539,00**	**12539,00**
GRM_P17	13278,80	13277,40	13277,40	**13276,00**	13277,40	13277,40
GRM_P18	**14036,00**	**14036,00**	**14036,00**	**14036,00**	**14036,00**	**14036,00**
GRM_P19	**14814,00**	**14814,00**	**14814,00**	**14814,00**	**14814,00**	**14814**
GRM_P20	**16422,00**	**16422,00**	**16422,00**	**16422,00**	**16422,00**	**16422,00**
melhor	15	18	15	17	18	16

Tabela 5.7: Configuração de k_{max}: qualidade média das instâncias GRM_P ($w = 5$).

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	**7884,00**	**7884,00**	**7884,00**	**7884,00**	**7884,00**	**7884,00**
GRM_P2	**8580,00**	**8580,00**	**8580,00**	**8580,00**	**8580,00**	**8580,00**
GRM_P3	**9793,00**	**9793,00**	**9793,00**	**9793,00**	**9793,00**	**9793,00**
GRM_P4	**11487,00**	**11487,00**	**11487,00**	**11487,00**	**11487,00**	**11487,00**
GRM_P5	**13273,00**	**13273,00**	**13273,00**	**13273,00**	**13273,00**	**13273,00**
GRM_P6	**15079,00**	**15079,00**	**15079,00**	**15079,00**	**15079,00**	**15079,00**
GRM_P7	**16909,00**	**16909,00**	**16909,00**	**16909,00**	**16909,00**	**16909,00**
GRM_P8	**18754,00**	**18754,00**	**18754,00**	**18754,00**	**18754,00**	**18754,00**
GRM_P9	22510,40	**22507,00**	**22507,00**	**22507,00**	**22507,00**	**22507,00**
GRM_P10	**13083,00**	**13083,00**	**13083,00**	**13083,00**	**13083,00**	**13083,00**
GRM_P11	**13040,00**	**13040,00**	**13040,00**	**13040,00**	**13040,00**	**13040,00**
GRM_P12	**13845,00**	**13845,00**	**13845,00**	**13845,00**	**13845,00**	**13845,00**
GRM_P13	14989,00	**14985,80**	14986,60	**14985,80**	**14985,80**	**14985,80**
GRM_P14	**16355,00**	**16355,00**	**16355,00**	**16355,00**	**16355,00**	**16355,00**
GRM_P15	**17919,00**	**17919,00**	**17919,00**	**17919,00**	**17919,00**	**17919,00**
GRM_P16	**19579,40**	19579,60	**19579,40**	19579,60	19579,60	19579,60
GRM_P17	21318,80	21317,60	21318,20	**21317,00**	**21317,00**	21317,60
GRM_P18	**23095,00**	**23095,00**	**23095,00**	**23095,00**	**23095,00**	**23095,00**
GORM_P19	**24895,00**	**24895,00**	**24895,00**	**24895,00**	**24895,00**	**24895,00**
GRM_P20	**28566,00**	**28566,00**	**28566,00**	**28566,00**	**28566,00**	**28566,00**
melhor	17	18	18	19	19	18

Tabela 5.8: Configuração de k_{max}: qualidade média das instâncias GRM_P ($w = 10$).

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**0,42**	0,57	0,56	0,57	0,57	0,58
ORM_P2	**0,81**	1,17	1,13	1,13	1,13	1,16
ORM_P3	**0,88**	1,25	1,26	1,25	1,26	1,27
ORM_P4	**3,44**	4,62	4,62	4,96	5,11	5,25
ORM_P5	**8,99**	13,49	14,14	16,04	16,89	19,00
ORM_P6	**1,16**	1,51	1,51	1,53	1,50	1,56
ORM_P7	**2,29**	3,28	3,33	3,35	3,33	3,48
ORM_P8	**9,11**	13,17	12,95	13,69	13,43	14,06
ORM_P9	**38,80**	52,13	52,49	54,71	57,93	60,96
ORM_P10	**109,20**	130,58	137,51	143,25	151,24	168,19
ORM_P11	**1,48**	1,68	1,64	1,65	1,66	1,68
ORM_P12	**3,63**	4,10	4,19	4,19	4,20	4,29
ORM_P13	**45,07**	50,21	50,22	52,45	53,75	58,38
ORM_P14	**178,57**	190,15	197,15	202,34	232,35	246,72
ORM_P15	**495,19**	545,82	556,65	601,72	631,07	830,17
ORM_P16	**2,77**	3,02	3,00	3,01	2,95	3,12
ORM_P17	**4,42**	5,40	5,84	5,84	5,82	6,06
ORM_P18	**54,82**	63,12	67,98	70,59	70,74	86,74
ORM_P19	**379,33**	455,96	423,01	500,28	496,54	536,38
ORM_P20	1.446,04	**1.397,37**	1.456,86	1.579,04	1.641,58	1.891,17
melhor	19	1	0	0	0	0

Tabela 5.9: Configuração de k_{max}: tempo médio de execução das instâncias ORM_P1 a ORM_P20 ($w = 2$).

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**0,77**	0,80	0,80	0,79	0,81	0,80
ORM_P2	**2,32**	2,76	2,92	2,83	2,84	2,85
ORM_P3	3,33	**3,28**	3,36	3,31	3,32	3,35
ORM_P4	**5,01**	5,28	5,37	5,56	5,65	5,64
ORM_P5	**9,06**	9,62	9,36	9,57	10,05	9,92
ORM_P6	**1,75**	1,81	1,80	1,80	1,81	1,80
ORM_P7	**7,49**	7,92	7,75	7,75	7,78	7,76
ORM_P8	**25,03**	25,42	26,35	27,14	28,21	28,19
ORM_P9	**86,81**	104,32	104,96	111,63	112,38	123,23
ORM_P10	**99,21**	103,56	104,32	107,52	110,89	124,87
ORM_P11	**1,77**	1,82	1,83	1,82	1,82	1,83
ORM_P12	**6,46**	6,94	7,17	7,14	7,24	7,30
ORM_P13	**107,73**	114,27	120,67	121,12	125,43	132,41
ORM_P14	**226,32**	249,90	262,28	278,83	277,06	337,02
ORM_P15	510,44	520,47	**499,81**	523,92	525,86	587,38
ORM_P16	**2,65**	2,86	2,87	2,87	2,87	2,87
ORM_P17	**5,60**	6,43	6,75	6,78	6,79	6,82
ORM_P18	**260,52**	272,41	275,17	296,80	325,43	350,96
ORM_P19	**1.003,81**	1.042,71	1.097,13	1.121,39	1.158,49	1.180,73
ORM_P20	2.045,04	**1.975,54**	2.076,64	2.045,24	2.082,70	2.144,66
melhor	17	2	1	0	0	0

Tabela 5.10: Configuração de k_{max}: tempo médio de execução das instâncias ORM_P1 a ORM_P20 ($w = 5$).

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
ORM_P1	**1,24**	1,49	1,48	1,46	1,86	1,54
ORM_P2	**2,01**	2,58	2,58	2,53	2,88	2,69
ORM_P3	**1,89**	2,48	2,52	2,50	2,97	2,66
ORM_P4	**4,78**	6,16	6,25	6,52	7,62	7,45
ORM_P5	**9,32**	12,07	12,26	12,32	12,67	12,92
ORM_P6	**3,80**	4,83	4,79	4,73	4,96	5,02
ORM_P7	**10,56**	13,57	13,72	13,55	14,99	14,28
ORM_P8	**27,33**	32,74	33,16	33,24	36,20	34,96
ORM_P9	**64,09**	72,76	72,94	74,91	87,64	82,24
ORM_P10	**125,93**	131,51	135,93	130,52	149,11	151,44
ORM_P11	**3,62**	4,02	4,02	3,93	4,05	4,30
ORM_P12	**15,68**	18,27	18,36	18,03	17,10	20,00
ORM_P13	**116,16**	116,84	122,48	121,31	116,68	135,75
ORM_P14	**362,66**	409,96	403,55	450,89	443,08	566,81
ORM_P15	654,51	644,83	646,33	**638,13**	658,91	766,57
ORM_P16	**3,68**	3,98	3,98	3,73	4,01	4,25
ORM_P17	**16,97**	18,05	19,08	17,83	18,86	20,10
ORM_P18	**317,97**	326,43	337,52	366,36	366,75	426,73
ORM_P19	**1.048,48**	1.091,12	1.238,73	1.141,21	1.334,11	1.340,77
ORM_P20	**1.886,63**	2.055,35	2.211,74	2.160,39	2.470,19	2.323,34
melhor	19	0	0	1	0	0

Tabela 5.11: Configuração de k_{max}: tempo médio de execução das instâncias ORM_P1 a ORM_P20 ($w = 10$).

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	**1,17**	1,18	1,19	1,45	1,46	1,19
GRM_P2	**2,56**	3,02	3,16	3,90	3,96	3,19
GRM_P3	**8,17**	8,94	9,19	12,12	12,25	9,81
GRM_P4	**21,92**	31,32	31,84	39,46	40,41	35,02
GRM_P5	**50,01**	63,91	65,79	71,42	74,41	74,04
GRM_P6	**57,63**	70,46	70,69	79,16	84,94	91,86
GRM_P7	**83,34**	99,43	103,22	109,18	114,23	113,48
GRM_P8	**79,74**	82,28	88,11	95,80	100,14	97,98
GRM_P9	**62,34**	64,34	64,99	70,14	69,71	71,25
GRM_P10	**3,42**	3,63	3,67	3,90	3,86	3,75
GRM_P11	**10,14**	10,90	11,79	12,70	12,48	12,16
GRM_P12	**13,73**	17,03	16,57	19,41	18,80	18,34
GRM_P13	**23,53**	26,38	28,42	31,81	34,05	32,73
GRM_P14	**51,19**	60,66	60,12	68,82	68,22	67,71
GRM_P15	**94,98**	112,04	118,80	130,28	133,16	134,73
GRM_P16	**187,04**	230,55	218,84	253,00	273,75	283,70
GRM_P17	**276,79**	300,78	303,94	327,17	337,85	371,94
GRM_P18	**308,50**	323,77	322,37	353,72	358,43	417,53
GRM_P19	**311,23**	379,34	400,37	433,24	446,63	434,05
GRM_P20	**365,20**	430,35	448,85	468,14	467,97	473,94
melhor	20	0	0	0	0	0

Tabela 5.12: Configuração de k_{max}: tempo médio de execução das instâncias GRM_P ($w = 2$).

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	2,69	2,64	**2,63**	**2,63**	2,64	2,69
GRM_P2	**7,90**	8,82	9,06	9,08	9,07	8,95
GRM_P3	**17,66**	20,81	22,02	24,24	24,25	23,97
GRM_P4	**27,14**	31,46	30,61	34,49	34,24	34,64
GRM_P5	**27,73**	29,17	29,56	30,42	30,64	31,18
GRM_P6	32,20	32,19	**31,93**	32,69	33,40	34,11
GRM_P7	**34,38**	34,59	35,02	34,90	35,32	36,30
GRM_P8	**38,78**	39,57	39,31	39,03	39,88	39,76
GRM_P9	**47,05**	47,77	48,13	49,16	48,23	49,93
GRM_P10	5,56	5,55	5,56	5,56	**5,53**	5,62
GRM_P11	**18,58**	18,96	19,10	19,07	19,02	19,40
GRM_P12	**39,58**	46,74	50,53	53,01	53,10	53,78
GRM_P13	90,32	**86,96**	97,13	99,23	96,69	97,86
GRM_P14	**109,55**	133,48	137,92	142,88	141,86	147,30
GRM_P15	**167,63**	183,96	189,24	201,31	205,75	208,59
GRM_P16	**187,71**	214,19	231,60	234,91	230,44	257,93
GRM_P17	**209,09**	222,07	220,40	236,09	250,54	265,92
GRM_P18	**207,29**	237,79	237,73	246,59	279,43	288,64
GRM_P19	**412,28**	464,12	485,28	504,09	541,83	518,99
GRM_P20	**345,17**	380,27	374,55	412,22	454,20	446,87
melhor	16	1	2	1	1	0

Tabela 5.13: Configuração de k_{max}: tempo médio de execução das instâncias GRM_P ($w = 5$).

	$k_{max} = 4$	$k_{max} = 8$	$k_{max} = 10$	$k_{max} = 15$	$k_{max} = 20$	$k_{max} = p$
Instância	Média	Média	Média	Média	Média	Média
GRM_P1	3,64	**3,60**	4,67	**3,60**	4,26	3,65
GRM_P2	**9,09**	10,52	14,20	10,83	12,98	11,20
GRM_P3	**13,51**	16,64	24,01	20,22	23,77	23,28
GRM_P4	**19,96**	21,05	27,04	24,04	27,78	28,07
GRM_P5	**28,07**	28,62	32,03	30,50	30,37	31,08
GRM_P6	**33,59**	36,01	37,76	37,63	35,97	37,00
GRM_P7	**39,15**	39,58	43,24	44,08	41,77	43,45
GRM_P8	**45,71**	45,79	48,49	49,88	47,24	49,03
GRM_P9	55,58	**54,38**	60,07	58,01	56,91	59,40
GRM_P10	**9,29**	9,67	10,07	10,08	9,67	10,10
GRM_P11	**36,53**	39,33	40,34	40,75	39,16	42,13
GRM_P12	67,63	**66,83**	70,67	77,67	73,63	80,89
GRM_P13	**101,76**	109,55	108,34	113,30	117,56	128,80
GRM_P14	**127,43**	147,91	163,08	165,08	162,85	179,80
GRM_P15	**161,96**	176,36	192,55	186,63	172,74	202,50
GRM_P16	**182,97**	198,85	208,61	217,94	217,68	271,43
GRM_P17	**221,97**	242,82	249,37	278,29	263,31	291,36
GRM_P18	**293,68**	327,98	323,76	342,50	346,65	365,00
GRM_P19	294,98	296,93	**289,13**	317,18	290,65	335,16
GRM_P20	337,02	**332,47**	342,99	349,87	343,46	355,16
melhor	15	4	1	1	0	0

Tabela 5.14: Configuração de k_{max}: tempo médio de execução das instâncias GRM_P ($w = 10$).

5.5 Configuração do Parâmetro b

Na linha 6 do algoritmo VNS em sua versão básica (Figura 5.1), uma perturbação aleatória é realizada na vizinhança de ordem k da solução corrente. Uma possível modificação seria encontrar a melhor solução perturbada entre b soluções geradas aleatoriamente na $k-$ésima vizinhança da solução corrente, aplicando, em seguida, a busca local a essa solução.

Para mostrar a influência da quantidade de vezes que a perturbação é executada (parâmetro b) na qualidade das soluções e nos tempos de processamento, testes computacionais foram realizados comparando-se o VNS com filtro com $b = 1, 2, 4, 8$ e 10. O valor $b = 1$ corresponde à configuração padrão utilizada no algoritmo da Figura 5.1.

Para a análise da qualidade das soluções, as medidas relativas *drpm*, *cm* e *melhor* foram utilizadas, como mostra a Tabela 5.15.

	b				
	1	2	4	8	10
drpm	0,000234	-0,000748	**-0,002189**	-0,000971	0,003674
cm	2,96	2,94	**2,92**	2,93	3,25
melhor	163	**169**	167	166	160

Tabela 5.15: Qualidade relativa na configuração de b.

Nas medidas *drpm* e *cm*, os melhores resultados foram obtidos por $b = 4$, enquanto na medida *melhor*, a configuração $b = 2$ obteve o melhor resultado. As configurações $b = 1$ e $b = 10$ obtiveram os piores resultados em todas as medidas relativas.

A Tabela 5.16 apresenta as medidas *drpm*, *cm* e *trm* utilizadas na análise dos tempos de processamento obtidos pelo VNS com filtro e $b = 1, 2, 4, 8$ e 10. Nessa última medida, a configuração padrão $b = 1$ foi tomada como algoritmo de referência.

	b				
	1	2	4	8	10
drpm	**-1,98**	-0,83	-0,13	-1,68	4,63
cm	**2,78**	2,83	3,00	2,86	3,53
trm	**1,00**	1,02	1,02	1,01	1,07

Tabela 5.16: Tempo relativo na configuração de b.

Em todas as medidas, os melhores resultados foram obtidos por $b = 1$. Porém, os resultados foram muito semelhantes em todas as configurações, mostrando que a operação de perturbação é relativamente barata em comparação ao tempo de processamento total do algoritmo. A maior perturbação testada ($b = 10$) torna, em média, o algoritmo 7% mais lento do que a configuração $b = 1$ para as instâncias analisadas.

Os resultados das tabelas mostram que, perturbar a solução mais de uma vez na $k-$ésima vizinhança da solução corrente e executar a busca local na melhor solução encontrada pode levar a resultados promissores. Os resultados obtidos pelas configurações $b = 2$, $b = 4$ e $b = 8$ são muito semelhantes em termos de qualidade e tempo. Optou-se por utilizar $b = 2$.

5.6 VNS e Reconexão por Caminhos

O procedimento de reconexão por caminhos foi também utilizado no VNS como estratégia de intensificação conectando cada ótimo local produzido pela busca local com um elemento de um conjunto de soluções de elite armazenados durante as iterações do VNS (ver Seção 4.5).

5.6.1 Configuração do Tamanho do Conjunto de Soluções de Elite

O parâmetro a ser ajustado no procedimento de reconexão por caminhos é o tamanho máximo do conjunto de soluções de elite (*MaxElite*). As seguintes configurações foram analisadas: VNS com filtro sem reconexão por caminhos (VNSf) e VNS com filtro e reconexão por caminhos com $MaxElite = 5, 10, 15$ e 20 (VNSf_RC5, VNSf_RC10, VNSf_RC15 e VNSf_RC20, respectivamente).

As medidas *drpm* e *cm* serão utilizadas para analisar a qualidade das soluções encontradas, como mostra a Tabela 5.17.

	drpm	*cm*
VNSf	0,001637	3,25
VNSf_RC5	-0,001101	**2,82**
VNSf_RC10	**-0,001499**	2,97
VNSf_RC15	-0,0002	2,94
VNSf_RC20	0,001165	3,01

Tabela 5.17: Qualidade relativa na configuração de *MaxElite* no VNS.

Analisando-se a Tabela 5.17, nas medidas *drpm* e *cm*, o melhor resultado foi obtido pelo VNSf_RC10 e VNSf_RC5, respectivamente. Em termos absolutos, dos 180 testes realizados (36 instâncias com cinco execuções cada), VNSf_RC5, VNSf_RC10, VNSf_RC15 e VNSf_RC20 melhoraram a qualidade da solução encontrada pelo VNSf em 13, 14, 12 e 12 testes, respectivamente.

Para a análise dos tempos de processamento, os valores obtidos pelos algoritmos nas medidas *drpm*, *cm* e *trm* são mostrados na Tabela 5.18. Na medida *trm*, o VNSf é o algoritmo de referência.

Analisando-se as medidas na Tabela 5.18, os melhores resultados foram obtidos pelo VNSf, pois não existe a necessidade do controle e gerenciamento do conjunto de soluções de elite. Os algoritmos com reconexão por caminhos não apresentaram uma diferença significativa entre eles.

	drpm	*cm*	*trm*
VNSf	**-8,39**	**1,81**	**1,00**
VNSf_RC5	1,95	3,00	1,13
VNSf_RC10	1,70	3,03	1,13
VNSf_RC15	1,84	3,25	1,13
VNSf_RC20	2,91	3,75	1,14

Tabela 5.18: Tempo relativo na configuração de *MaxElite* no VNS.

Levando-se em consideração as medidas de qualidade e tempo, destacaram-se os algoritmos VNS_RC5 e VNS_RC10, pois apresentaram menores tempos de processamento e qualidade superiores às demais configurações com um maior número de soluções de elite. Optou-se pela configuração $MaxElite = 10$ para armazenar um número maior de soluções.

5.6.2 Comparação entre o VNS com Filtro e o VNS com Filtro e Reconexão por Caminhos

Para mostrar a efetividade do procedimento de reconexão por caminhos no VNS, comparações serão realizadas através de gráficos de distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo, executando-se o algoritmo com e sem o procedimento. As seguintes instâncias com seus respectivos valores alvo entre parênteses serão analisadas: GRM_P11 (11086), GRM_P18 (14036), ORM_P9 (6436) e ORM_P18 (7259), para $w = 5$. Nas figuras, o VNS com filtro com parâmetros $k_{max} = 15$ e $b = 2$ será denominado de VNSf e o VNS com filtro e reconexão por caminhos com os parâmetros $k_{max} = 15$, $b = 2$ e $MaxElite = 10$ será denominado de VNSf_RC.

As Figuras 5.2 e 5.3 mostram que as curvas do VNSf e do VNSf_RC praticamente se equivalem para essas instâncias. No primeiro gráfico, o VNSf chega a ser um pouco melhor do que o VNSf_RC para encontrar o valor alvo 11086 na instância GRM_P11. A probabilidade de encontrar um valor menor ou igual ao alvo em menos de 1,1 segundo é de aproximadamente 89% (resp. 81%) para o VNSf (resp. VNSf_RC). Em instâncias menores, os ganhos com a utilização do procedimento de reconexão por caminhos no VNS não são significativos.

Quando aumenta-se o tamanho da instância, a utilização do procedimento de reconexão por caminhos torna-se uma boa estratégia, como mostram as Figuras 5.4 e 5.5.

Os gráficos ilustram que a probabilidade de encontrar um valor pelo menos tão bom quanto o valor alvo aumenta do VNSf para o VNSf_RC para as duas instâncias testadas. Na Figura 5.4, a probabilidade de encontrar um valor alvo menor ou igual a 7259 em menos do que 40 segundos é de aproximadamente 82% para o VNSf e de aproximadamente 91% para o

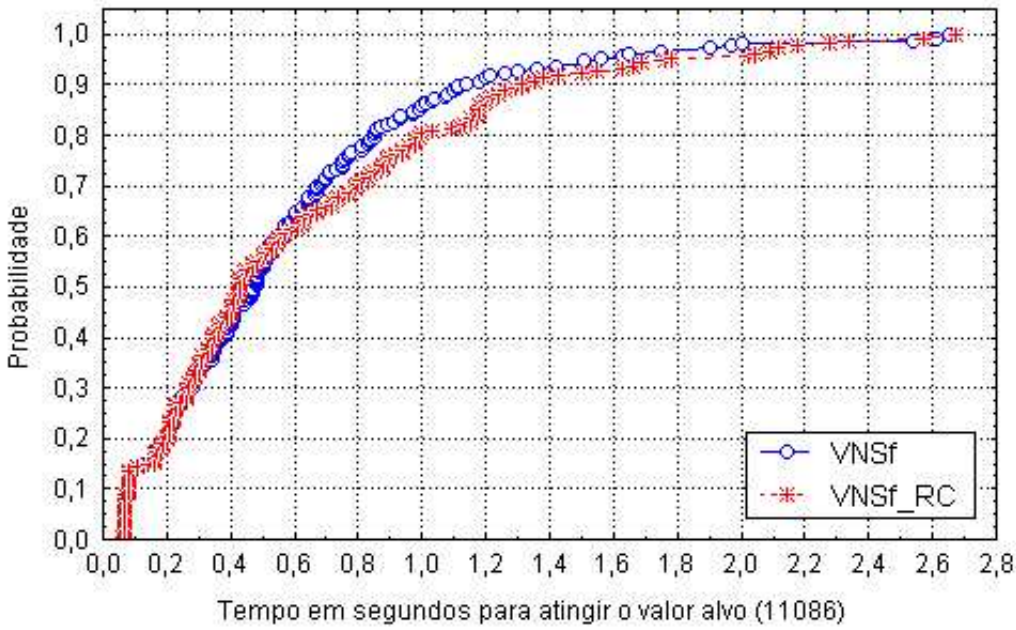

Figura 5.2: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 11086 para a instância GRM_P11 ($w = 5$).

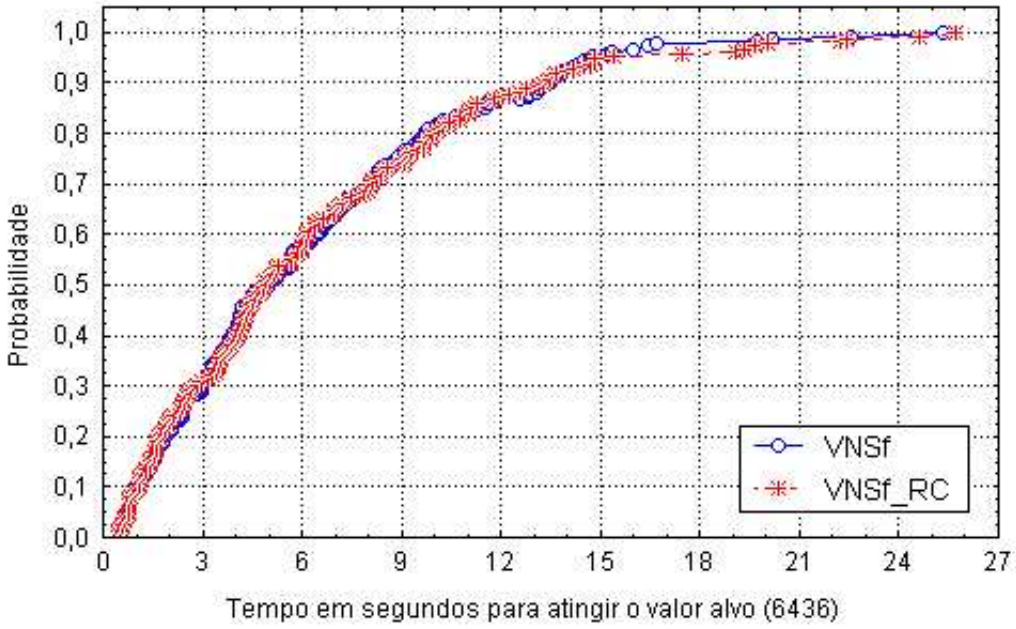

Figura 5.3: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 6436 para a instância ORM_P9 ($w = 5$).

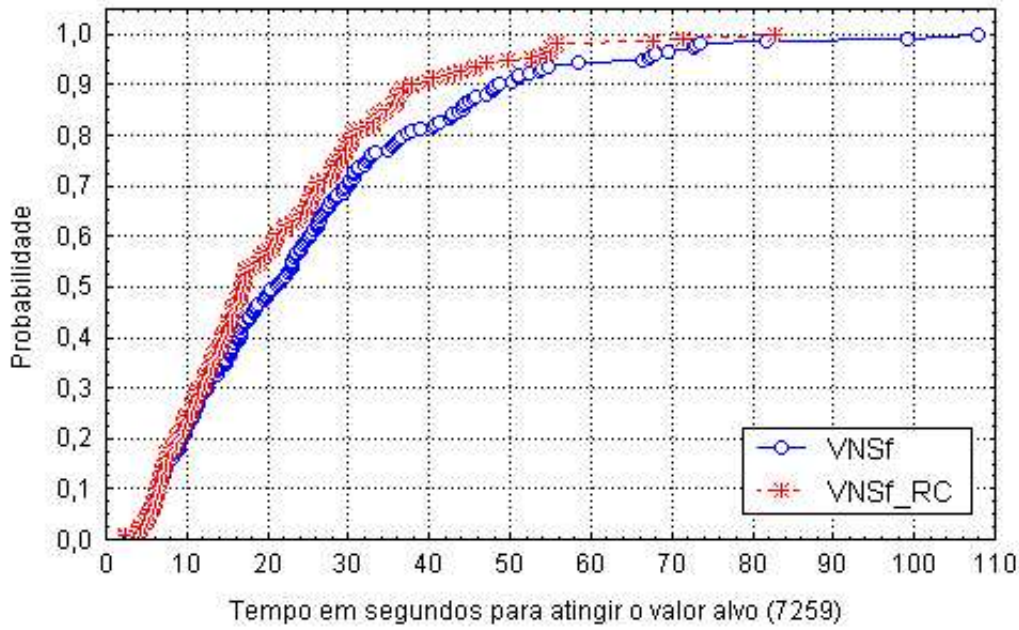

Figura 5.4: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 7259 para a instância ORM_P18 ($w = 5$).

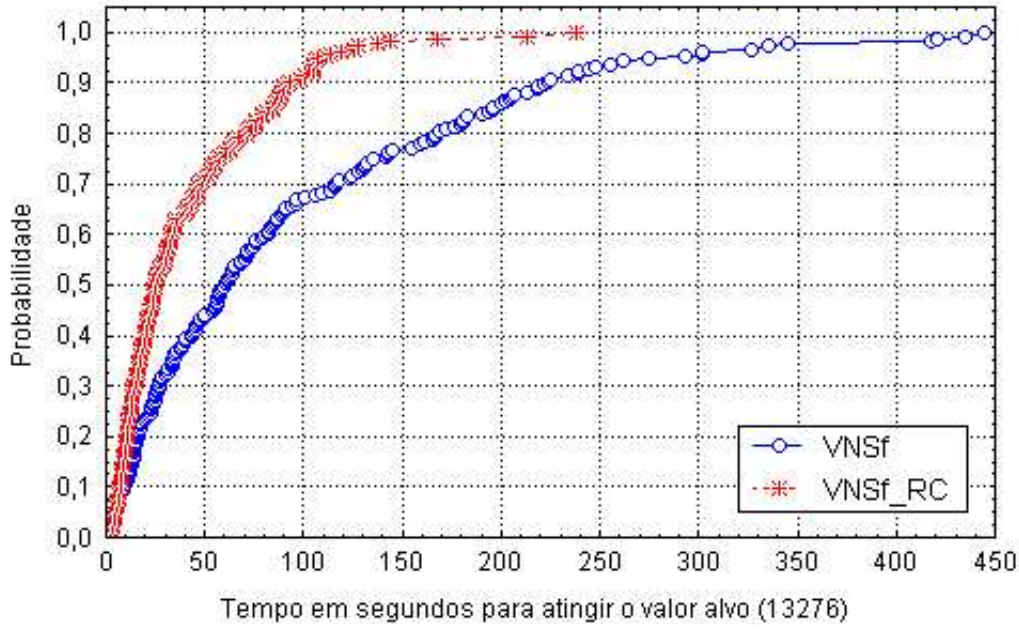

Figura 5.5: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 13276 para a instância GRM_P17 ($w = 5$).

VNSf_RC. Na Figura 5.5, o ganho do VNSf_RC em relação ao VNSf é mais significativo. A probabilidade de encontrar um valor alvo menor ou igual a 13276 em menos do que 150 segundos é de aproximadamente 77% para o VNSf e de quase 100% para o VNSf_RC.

5.7 Algoritmo VNS com Filtro e Reconexão por Caminhos

A heurística VNS com filtro e reconexão por caminhos é mostrada na Figura 5.6.

O conjunto de soluções de elite P com tamanho máximo igual a dez está inicialmente vazio (linha 1). A solução inicial S do VNS é gerada na linha 3. Executa-se o laço das linhas 4 a 43 enquanto o critério de parada não for satisfeito. O critério de parada escolhido para a configuração do VNS foi o número de iterações realizadas ($niter = 100$). O algoritmo inicia explorando a vizinhança que possui a menor distância em relação à solução corrente (linha 5). Enquanto a maior estrututura de vizinhança não for alcançada ($k_{max} = 15$) no laço das linhas 6 a 42, os seguintes passos são executados. Primeiramente, a solução S é perturbada aleatoriamente $b = 2$ vezes na sua k-ésima vizinhança (linhas 7 a 13) e a melhor solução perturbada serve como entrada para a busca local mais rápida na linha 14. Na linha 15 ocorre o teste da estratégia de filtro e caso ambas as condições sejam satisfeitas, executa-se a busca local mais lenta no ótimo local encontrado pela primeira busca (linha 16). O ótimo local encontrado pela segunda busca é um candidato a inserção no conjunto de soluções de elite e caso satisfaça as condições de qualidade e diversidade, torna-se um novo elemento de P (linha 22). Caso pelo menos uma das condições não sejam satisfeitas na linha 15, o ótimo local da primeira busca também pode ser inserido em P (linha 22). Na linha 23, o procedimento de reconexão por caminhos começa a atuar a partir da segunda iteração. Primeiramente, uma solução é escolhida do conjunto de soluções de elite privilegiando-se elementos com uma maior diferença simétrica em relação ao ótimo local candidato (linha 24). Na linha 25, determina-se qual será a solução inicial e a solução alvo entre o ótimo local candidato e o elemento de P. A melhor solução será a solução inicial e a pior será a solução alvo. Assim, de acordo com esse teste, executa-se reconexão por caminhos entre a solução inicial e a solução alvo nas linhas 26 ou 29. Caso o procedimento obtenha sucesso, retorna-se a melhor solução encontrada no caminho; caso contrário, retorna-se a melhor solução entre S_1 e S_2. Durante cada passo, se o procedimento encontra uma solução melhor do que a melhor solução encontrada até o momento, esta também torna-se uma candidata a inserção em P. Se o valor da função objetivo do procedimento de reconexão por caminhos for melhor do que o valor da função objetivo da melhor solução encontrada até o momento (linha 31), atualiza-se a melhor

```
Procedimento VNS_Filtro_Reconexão_Caminhos(α, semente)
1.  P ← ∅;
2.  F* ← +∞;
3.  S ← Fase_Construção(α, semente);
4.  Para it = 1, ..., niter Faça
5.      k ← 1;
6.      Enquanto (k ≤ k_max) Faça
7.          F' ← +∞;
8.          Para j = 1, ..., b Faça
9.              Gerar uma solução aleatória S'' ∈ N_k(S);
10.             Se (F(S'') < F') Então
11.                 F' ← F(S''), S' ← S'';
12.             Fim-Se
13.         Fim-Para
14.         S'_1 ← BL_BTMA(S');
15.         Se (S'_1 não foi visitado previamente) e
                (F(S'_1) < (1 + λ) × F*) Então
16.             S' ← BL_Teste(S'_1);
17.         Fim-Se
18.         Senão
19.             S' ← S'_1;
20.         Fim-Senão
21.         S_1 ← S';
22.         Atualizar o conjunto de soluções de elite P com S_1;
23.         Se (it ≥ 2) Então
24.             Escolher uma solução S_2 ∈ P;
25.             Se (F(S_1) ≤ F(S_2)) Então
26.                 S'' ← Reconexão_por_Caminhos(S_1, S_2);
27.             Fim-Se
28.             Senão
29.                 S'' ← Reconexão_por_Caminhos(S_2, S_1);
30.             Fim-Senão
31.             Se (F(S'') < F*) Então
32.                 F* ← F(S''), S* ← S'';
33.                 S ← S'';
34.                 k ← 1;
35.             Fim-Se
36.             Senão
37.                 k ← k + 1;
38.             Fim-Senão
39.         Senão
40.             F* ← F(S'), S* ← S';
41.         Fim-Senão
42.     Fim-Enquanto
43. Fim-Para
44. Retorne S*;
Fim
```

Figura 5.6: Algoritmo VNS com filtro e reconexão por caminhos.

solução, a solução corrente e retorna-se à vizinhança N_1 (linhas 32 a 34). Caso contrário, aumenta-se a distância entre a solução corrente S e as soluções geradas aleatoriamente no processo de perturbação (linha 37). Na primeira iteração, atualiza-se a melhor solução nas linhas 39 a 41. A melhor solução encontrada pelo algoritmo é retornada na linha 44.

5.8 Considerações Finais

Este capítulo propôs a heurística VNS com filtro e reconexão por caminhos para o problema das p-medianas conectadas.

A solução inicial do VNS é gerada pela heurística construtiva do GRASP com filtro e reconexão por caminhos, apresentada na Seção 4.1, e o algoritmo de busca local utilizado foi descrito no Capítulo 3.

Com o objetivo de diminuir os tempos de processamento apresentados pelo VNS em sua versão básica, uma estratégia de filtro foi incorporada ao algoritmo entre a busca local pelas bordas com teste da menor aresta e a busca local com teste. Resultados computacionais comparando-se o VNS com e sem o filtro mostraram ganhos em termos de tempo de processamento com uma pequena perda na qualidade das soluções encontradas.

Uma simples modificação que se mostrou efetiva no algoritmo foi perturbar aleatoriamente a solução corrente um número determinado de vezes (parâmetro b) e utilizar a melhor solução perturbada como entrada para o algoritmo de busca local. Testes computacionais foram realizados com valores de $b = 1, 2, 4, 8$ e 10. Como essa é uma operação relativamente barata, os tempos de processamento mantiveram-se praticamente idênticos à configuração padrão ($b = 1$) com melhoras na qualidade das soluções encontradas em quase todas as configurações. De acordo com os testes realizados, optou-se por utilizar $b = 2$.

O procedimento de reconexão por caminhos foi utilizado do mesmo modo ao apresentado no Capítulo 4. Comparações realizadas entre o VNS com filtro com e sem o procedimento através de gráficos de distribuição de probabilidade empírica do tempo gasto para atingir o valor alvo mostraram que, em instâncias menores, o VNS não se beneficia do procedimento. Porém, quando aumenta-se o tamanho da instância, a utilização do procedimento é uma boa estratégia, conduzindo a resultados melhores do que o VNS que não o utiliza.

6
Comparações entre as Heurísticas GRASP e VNS

O objetivo desse capítulo é comparar a heurística GRASP com filtro e reconexão por caminhos, apresentada no Capítulo 4, com a heurística VNS com filtro e reconexão por caminhos, apresentada no Capítulo 5, em termos da qualidade das soluções encontradas e dos tempos de processamento obtidos para diversas classes de instâncias. Três diferentes tipos de comparações serão realizadas.

Primeiramente, as heurísticas serão comparadas obtendo-se a qualidade das soluções encontradas em três diferentes tempos de processamento para cada instância, possibilitando estudar o comportamento dos algoritmos em tempos distintos de execução.

Em seguida, gráficos de distribuição de probabilidade empírica do tempo gasto para encontrar um valor alvo serão utilizados para a comparação das heurísticas.

O modelo baseado em árvore geradora mínima restrita por grau para o problema das p-medianas conectadas, apresentado na Seção 2.4.2, possibilitou a obtenção de soluções ótimas para algumas instâncias testadas. Assim, por último, comparações em termos absolutos serão realizadas entre as heurísticas nas instâncias em que o ótimo é conhecido. Para as instâncias em que não se conhece o ótimo, a melhor solução encontrada pelas heurísticas GRASP e VNS será a melhor solução conhecida para aquela instância. Essas comparações mostraram a necessidade de incorporar um passo adicional em ambos os algoritmos. O passo de pós-otimização consiste em executar um algoritmo ótimo para o problema de Steiner tendo como entrada as p facilidades abertas da melhor solução encontrada pelas heurísticas, após todas as iterações terem sido realizadas.

Este capítulo está organizado como se segue. A Seção 6.1 mostra o ambiente de teste e as instâncias utilizadas na comparação das heurísticas. A Seção 6.2 compara a qualidade das soluções encontradas pelas heurísticas fixando-se três diferentes tempos de processamento para cada instância. A Seção 6.3 apresenta gráficos de distribuição de probabilidade do tempo gasto para atingir o valor alvo para ambas as heurísticas. A Seção 6.4 mostra as comparações em termos absolutos entre as heurísticas para todas as instâncias e descreve o passo de pós-otimização, assim como seus resultados computacionais. Por

último, a Seção 6.5 apresenta as conclusões e considerações finais do capítulo.

6.1 Ambiente de Teste e Instâncias Utilizadas

As heurísticas GRASP e VNS foram implementadas em C com o parâmetro de otimização -O3 e executadas em uma máquina Pentium IV 3.2 GHz com 1 Gbyte de memória RAM sob o sistema operacional Linux RedHat 9.0. O gerador de números aleatórios utilizado foi o de Matsumoto e Nishimura [53].

Para a execução do modelo baseado em árvore geradora mínima restrita por grau, utilizou-se a mesma máquina e a versão 8.0 do resolvedor de programação linear inteira ILOG CPLEX. A restrição de eliminação de sub-rotas foi substituída pela restrição de Miller-Tucker-Zemlin utilizada no problema do caixeiro viajante [38].

Para a comparação das heurísticas, dois grupos de instâncias serão utilizados. O primeiro grupo são as instâncias proporcionais das classes ORM_P, GRM_P e SLM_P ($w = 2$, $w = 5$ e $w = 10$), apresentadas na Seção 3.3.4, totalizando-se 189 problemas. O segundo grupo, denominado de instâncias não proporcionais, foi obtido a partir do grupo de instâncias proporcionais ORM_P e GRM_P ($w = 2$) da seguinte maneira. Dado um grafo $G = (V, E)$ com custos associados às arestas, o custo de servir cada usuário i a partir de cada facilidade j é dado pelo valor do caminho mais curto d_{ij} no grafo G, como no grupo de instâncias proporcionais. Porém, o custo de instalação c_{ij} de cada aresta é escolhido aleatoriamente no intervalo $[(w \times a_{ij}) - (w \times a_{ij})/2, (w \times a_{ij}) + (w \times a_{ij})/2]$, onde a_{ij} é o custo associado à aresta (i, j) nas instâncias proporcionais ORM_P e GRM_P com $w = 2$. As classes de instâncias não-proporcionais serão denominadas de ORM_NP e GRM_NP, totalizando 60 problemas.

6.2 Tempo de Processamento Limitado

Nesta seção, as heurísticas serão comparadas com base nas soluções encontradas em três tempos de processamento distintos para cada instância: um tempo menor de execução (tp), um tempo intermediário de execução (ti) e um tempo maior de execução (tg). Como ambas as heurísticas utilizam a mesma busca local, adotou-se o seguinte procedimento para a fixação dos tempos. Para cada instância das classes proporcionais ORM_P, GRM_P e SLM_P ($w = 2$, $w = 5$ e $w = 10$), utilizou-se o tempo médio de processamento de cada iteração da busca local concatenada em 15 execuções realizadas, obtido dos experimentos no Capítulo 3 (medida tam). Fixou-se então $tp = 10 \times tam$, $ti = 40 \times tam$ e $tg = 70 \times tam$. Os tempos tp, ti e tg das classes ORM_NP

e GRM_NP serão os mesmos das respectivas classes proporcionais ORM_P e GRM_P ($w = 2$).

Como exemplo, o valor de *tam* para a instância ORM_P19 ($w = 2$) é $6,28$ segundos. Assim, executam-se as heurísticas para essa instância com a mesma semente do gerador de números aleatórios fixando-se os seguintes tempos: $tp = 62,80$ segundos, $ti = 251,20$ segundos e $tg = 439,60$ segundos. Para as instâncias em que o valor de *tam* é muito pequeno, fixou-se $tam = 1,00$ segundo. Por exemplo, o valor de *tam* para a instância ORM_P1 ($w = 2$) é $0,003$ segundos. Assim, executam-se as heurísticas para essa instância com a mesma semente com os tempos: $tp = 10,00$ segundos, $ti = 40,00$ segundos e $tg = 70,00$ segundos.

Para cada instância, heurística e tempo de processamento, cinco execuções diferentes foram realizadas. As Tabelas 6.1 a 6.9 mostram os valores obtidos para as medidas relativas *drpm*, *cm* e *melhor* para todos os grupos de instâncias (proporcionais e não proporcionais) e tempos de processamento testados. As heurísticas GRASP e VNS acrescidas da estratégia de filtro e do procedimento de reconexão por caminhos serão denominadas, respectivamente, de GRASPf_RC e VNSf_RC. Os valores em negrito destacam o melhor resultado obtido pelas heurísticas em cada medida para cada grupo de instâncias nos tempos *tp*, *ti* e *tg*.

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P	GRASPf_RC	0,003	1,60	33
(40 instâncias)	VNSf_RC	**-0,003**	**1,40**	**40**
GRM_P	GRASPf_RC	0,004	1,63	19
(20 instâncias)	VNSf_RC	**-0,004**	**1,38**	**20**
SLM_P	GRASPf_RC	**-0,020**	**1,00**	**2**
(3 instâncias)	VNSf_RC	0,020	2,00	**2**

Tabela 6.1: Qualidade relativa das heurísticas no tempo *tp* para as instâncias proporcionais ($w = 2$).

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P	GRASPf_RC	**-0,005**	**1,42**	**38**
(40 instâncias)	VNSf_RC	0,005	1,58	37
GRM_P	GRASPf_RC	0,0006	**1,45**	19
(20 instâncias)	VNSf_RC	**-0,0006**	1,55	**20**
SLM_P	GRASPf_RC	**-0,10**	**1,00**	**3**
(3 instâncias)	VNSf_RC	0,10	2,00	0

Tabela 6.2: Qualidade relativa das heurísticas no tempo *tp* para as instâncias proporcionais ($w = 5$).

Primeiramente serão analisados os resultados obtidos pelas heurísticas nas instâncias proporcionais. As Tabelas 6.1 a 6.3 apresentam os resultados obtidos pelo GRASPf_RC e VNSf_RC no menor tempo de execução (*tp*).

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P	GRASPf_RC	**-0,027**	**1,44**	**38**
(40 instâncias)	VNSf_RC	0,027	1,56	35
GRM_P	GRASPf_RC	0,00006	1,55	**20**
(20 instâncias)	VNSf_RC	**-0,00006**	**1,45**	18
SLM_P	GRASPf_RC	**-0,37**	**1,00**	**3**
(3 instâncias)	VNSf_RC	0,37	2,00	0

Tabela 6.3: Qualidade relativa das heurísticas no tempo *tp* para as instâncias proporcionais ($w = 10$).

As medidas *drpm* e *cm* mostram que, em média, o algoritmo GRASPf_RC obteve resultados melhores do que a heurística VNSf_RC no tempo *tp* para as instâncias testadas. Na medida *drpm*, o GRASPf_RC foi melhor do que o VNSf_RC em cinco grupos de instâncias: ORM_P ($w = 5$ e $w = 10$) e SLM_P ($w = 2$, $w = 5$ e $w = 10$). Na medida *cm*, o GRASPf_RC foi melhor do que o VNSf_RC em seis grupos de instâncias: ORM_P ($w = 5$ e $w = 10$), GRM_P ($w = 5$) e SLM_P ($w = 2$, $w = 5$ e $w = 10$). Na medida *melhor*, das 189 instâncias proporcionais testadas, GRASPf_RC (resp. VNSf_RC) obteve o menor valor em 175 problemas (resp. 172 problemas).

Sejam agora os resultados obtidos pelas heurísticas no tempo intermediário de execução (*ti*), como mostram as Tabelas 6.4 a 6.6.

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P	GRASPf_RC	0,002	1,54	34
(40 instâncias)	VNSf_RC	**-0,002**	**1,46**	**39**
GRM_P	GRASPf_RC	0,0009	1,53	**20**
(20 instâncias)	VNSf_RC	**-0,0009**	**1,47**	**20**
SLM_P	GRASPf_RC	**-0,0007**	**1,33**	**3**
(3 instâncias)	VNSf_RC	0,0007	1,67	**3**

Tabela 6.4: Qualidade relativa das heurísticas no tempo *ti* para as instâncias proporcionais ($w = 2$).

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P	GRASPf_RC	**-0,002**	**1,46**	**40**
(40 instâncias)	VNSf_RC	0,002	1,54	38
GRM_P	GRASPf_RC	0,0004	**1,47**	**20**
(20 instâncias)	VNSf_RC	**-0,0004**	1,53	**20**
SLM_P	GRASPf_RC	**-0,044**	**1,00**	**3**
(3 instâncias)	VNSf_RC	0,044	2,00	0

Tabela 6.5: Qualidade relativa das heurísticas no tempo *ti* para as instâncias proporcionais ($w = 5$).

Analisando-se os resultados obtidos no tempo *ti*, praticamente o mesmo padrão de comportamento ocorreu nas medidas *drpm* e *cm* em relação ao menor tempo de execução (*tp*). Na medida *drpm*, GRASPf_RC foi melhor do

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P	GRASPf_RC	**-0,009**	**1,42**	**39**
(40 instâncias)	VNSf_RC	0,009	1,58	38
GRM_P	GRASPf_RC	**-0,0001**	**1,50**	**20**
(20 instâncias)	VNSf_RC	0,0001	**1,50**	**20**
SLM_P	GRASPf_RC	**-0,18**	**1,00**	**3**
(3 instâncias)	VNSf_RC	0,18	2,00	0

Tabela 6.6: Qualidade relativa das heurísticas no tempo ti para as instâncias proporcionais ($w = 10$).

que o VNSf_RC em seis grupos de instâncias: ORM_P ($w = 5$ e $w = 10$), GRM_P ($w = 10$) e SLM_P ($w = 2$, $w = 5$ e $w = 10$). Na medida *cm*, o GRASPf_RC foi melhor do que o VNSf_RC em seis grupos de instâncias: ORM_P ($w = 5$ e $w = 10$), GRM_P ($w = 5$) e SLM_P ($w = 2$, $w = 5$ e $w = 10$). Na medida *melhor*, dos 189 problemas testados, GRASPf_RC e VNSf_RC obteve o menor valor em 182 e 178 problemas, respectivamente. Executando-se os algoritmos em um tempo maior em relação a tp, as heurísticas convergem para as mesmas soluções em alguns grupos de instâncias, como, por exemplo, GRM_P ($w = 2$, $w = 5$ e $w = 10$).

Sejam os resultados obtidos pelas heurísticas no maior tempo de execução (tg), como mostram as Tabelas 6.7 a 6.9.

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P	GRASPf_RC	0,001	1,54	35
(40 instâncias)	VNSf_RC	**-0,001**	**1,46**	**38**
GRM_P	GRASPf_RC	0,0003	1,53	**20**
(20 instâncias)	VNSf_RC	**-0,0003**	**1,47**	**20**
SLM_P	GRASPf_RC	**0,00**	**1,5**	**3**
(3 instâncias)	VNSf_RC	**0,00**	**1,5**	**3**

Tabela 6.7: Qualidade relativa das heurísticas no tempo tg para as instâncias proporcionais ($w = 2$).

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P	GRASPf_RC	**-0,0012**	**1,47**	**39**
(40 instâncias)	VNSf_RC	0,0012	1,53	38
GRM_P	GRASPf_RC	0,0004	1,55	**20**
(20 instâncias)	VNSf_RC	**-0,0004**	**1,45**	**20**
SLM_P	GRASPf_RC	**-0,031**	**1,00**	**3**
(3 instâncias)	VNSf_RC	0,031	2,00	1

Tabela 6.8: Qualidade relativa das heurísticas no tempo tg para as instâncias proporcionais ($w = 5$).

Analisando-se as tabelas do tempo tg, ocorreu uma pequena melhora dos resultados obtidos pelo VNSf_RC em relação ao GRASPf_RC nas medidas *drpm* e *cm*, comparando-se com os tempos de execução anteriores (tp e ti). Nas

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_P	GRASPf_RC	**-0,006**	**1,44**	**40**
(40 instâncias)	VNSf_RC	0,006	1,56	39
GRM_P	GRASPf_RC	0,0002	1,53	**20**
(20 instâncias)	VNSf_RC	**-0,0002**	**1,47**	**20**
SLM_P	GRASPf_RC	**-0,14**	**1,00**	**3**
(3 instâncias)	VNSf_RC	0,14	2,00	0

Tabela 6.9: Qualidade relativa das heurísticas no tempo tg para as instâncias proporcionais ($w = 10$).

medidas *drpm* e *cm*, GRASPf_RC foi melhor do que VNSf_RC nos mesmos quatro grupos de instâncias: ORM_P ($w = 5$ e $w = 10$) e SLM_P ($w = 5$ e $w = 10$). Na medida *melhor*, das 189 instâncias testadas, GRASPf_RC (resp. VNSf_RC) obteve o menor valor em 183 problemas (resp. 179 problemas). Quando aumenta-se o tempo de execução, a tendência é de que, os algoritmos apresentem praticamente os mesmos resultados, convergindo para as mesmas soluções na maioria das instâncias testadas.

Sejam agora os resultados obtidos pelas heurísticas nas instâncias não proporcionais nos tempos tp, ti e tg (Tabelas 6.10 a 6.12).

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_NP	GRASPf_RC	0,005	1,56	36
(40 instâncias)	VNSf_RC	**-0,005**	**1,44**	**37**
GRM_NP	GRASPf_RC	0,006	1,55	19
(20 instâncias)	VNSf_RC	**-0,006**	**1,45**	**20**

Tabela 6.10: Qualidade relativa das heurísticas no tempo tp para as instâncias não proporcionais.

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_NP	GRASPf_RC	0,004	1,57	32
(40 instâncias)	VNSf_RC	**-0,004**	**1,43**	**40**
GRM_NP	GRASPf_RC	**-0,002**	**1,48**	**20**
(20 instâncias)	VNSf_RC	0,002	1,52	17

Tabela 6.11: Qualidade relativa das heurísticas no tempo ti para as instâncias não proporcionais.

Classe	Algoritmo	*drpm*	*cm*	*melhor*
ORM_NP	GRASPf_RC	0,004	1,60	33
(40 instâncias)	VNSf_RC	**-0,004**	**1,40**	**40**
GRM_NP	GRASPf_RC	**-0,002**	**1,45**	**19**
(20 instâncias)	VNSf_RC	0,002	1,55	18

Tabela 6.12: Qualidade relativa das heurísticas no tempo tg para as instâncias não proporcionais.

Analisando-se os resultados no menor tempo de execução (tp), em média, VNSf_RC apresentou resultados melhores do que GRASPf_RC nas medidas

drpm e *cm* em ambas as classes. Na medida *melhor*, das 60 instâncias testadas, GRASPf_RC (resp. VNSf_RC) obteve o menor valor em 55 problemas (resp. 57 problemas). Aumentando-se o tempo de execução dos algoritmos (tempo ti), o VNSf_RC continua sendo melhor na classe ORM_NP, porém, apresenta resultados piores do que o GRASPf_RC na classe GRM_NP. Na medida *melhor*, dos 60 problemas, GRASPf_RC (resp. VNSf_RC) obteve o menor valor em 52 problemas (resp. 57 problemas). Por último, no maior tempo de execução (tg), as heurísticas mantiveram o mesmo padrão de comportamento do tempo ti nas medidas *drpm* e *cm*. Na medida *melhor*, dos 60 problemas, GRASPf_RC (resp. VNSf_RC) obteve o menor valor em 52 problemas (resp. 58 problemas).

6.3 Tempo para Atingir um Valor Alvo

Esta seção compara as heurísticas através de gráficos que mostram a distribuição de probabilidade empírica da variável aleatória tempo gasto para encontrar um valor alvo. Como explicado na Seção 4.5.2, primeiramente fixa-se o valor alvo e faz-se 200 execuções independentes para cada heurística. Em cada execução, termina-se o algoritmo quando uma solução de valor menor ou igual ao valor alvo é encontrada. O i-ésimo menor tempo de execução é associado à probabilidade $prob_i = (i - 1/2)/200$ para $i = 1, 2, \ldots, 200$. Através desses gráficos, pode-se comparar experimentalmente diferentes algoritmos aleatórios ou diferentes versões do mesmo algoritmo aleatório [1].

As seguintes instâncias proporcionais e não proporcionais, com seus respectivos valores alvo entre parênteses, foram avaliadas: ORM_P18 (5990) para $w = 2$, ORM_P30 (7530) para $w = 5$, ORM_P20 (10600) para $w = 10$, GRM_NP17 (7370) e ORM_NP37 (6195). Instâncias maiores foram escolhidas porque nesses problemas os algoritmos apresentaram diferenças significativas. Executando-se as heurísticas em instâncias menores, em geral, as curvas de ambas se equivalem, como mostra a Figura 6.1 (instância GRM_P5 com $w = 5$ e valor alvo 8379). Pelo gráfico, em um tempo inferior à $3,5$ segundos, as heurísticas encontram um valor menor ou igual ao alvo 8379 com 100% de probabilidade.

Analisando-se as instâncias proporcionais maiores, no problema ORM_P18 com $w = 2$ (Figura 6.2), a probabilidade de encontrar uma solução com valor pelo menos tão bom quanto o valor alvo 5990 aumenta do GRASPf_RC para o VNSf_RC. A probabilidade de encontrar o valor alvo em menos de 20 segundos é aproximadamente 90% para o VNSf_RC e aproximadamente 50% para o GRASPf_RC.

Quando aumenta-se o fator w nas instâncias proporcionais, o GRASPf_RC é, em geral, superior ao VNSf_RC, como mostram as Figuras 6.3 e 6.4. No primeiro (resp. segundo) gráfico, a probabilidade de encontrar o

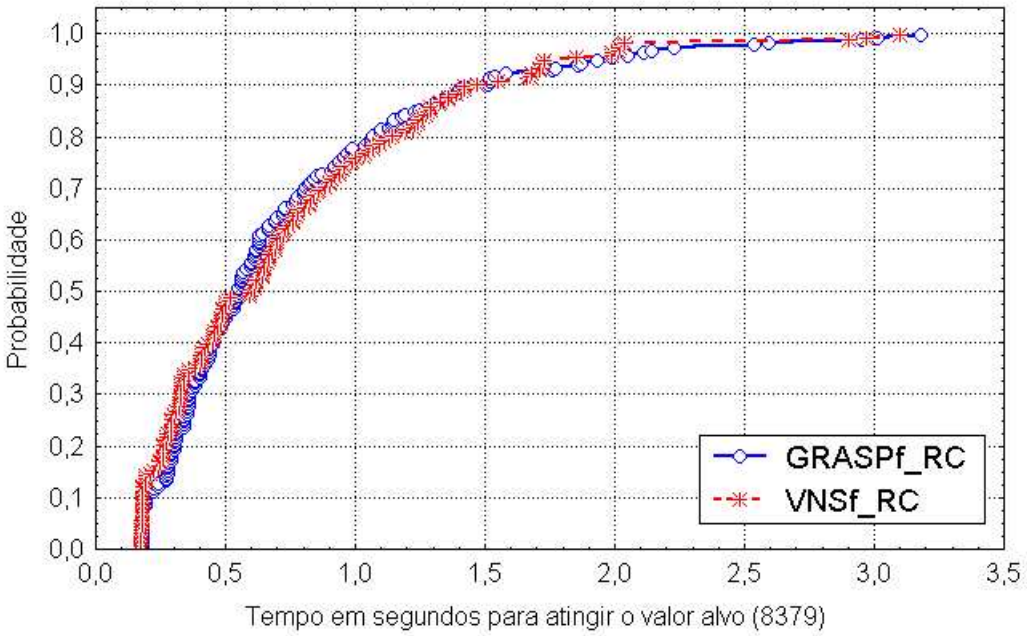

Figura 6.1: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 8379 para a instância GRM_P5 ($w = 5$).

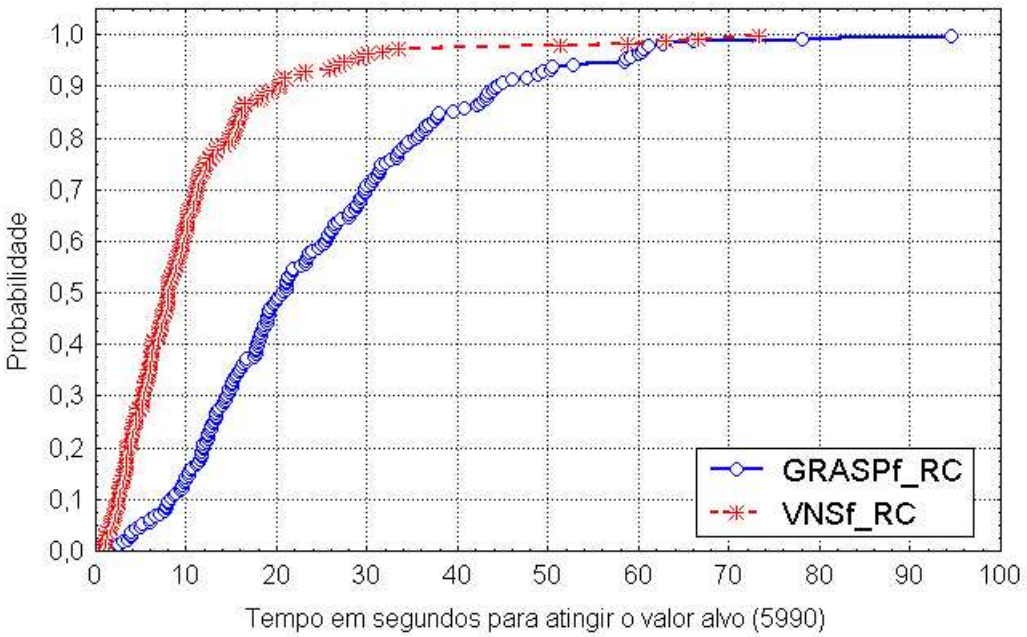

Figura 6.2: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 5990 para a instância ORM_P18 ($w = 2$).

valor alvo em menos de 250 segundos (resp. 95 segundos) é de 100% para o GRASPf_RC e de aproximadamente 85% (resp. 64%) para o VNSf_RC.

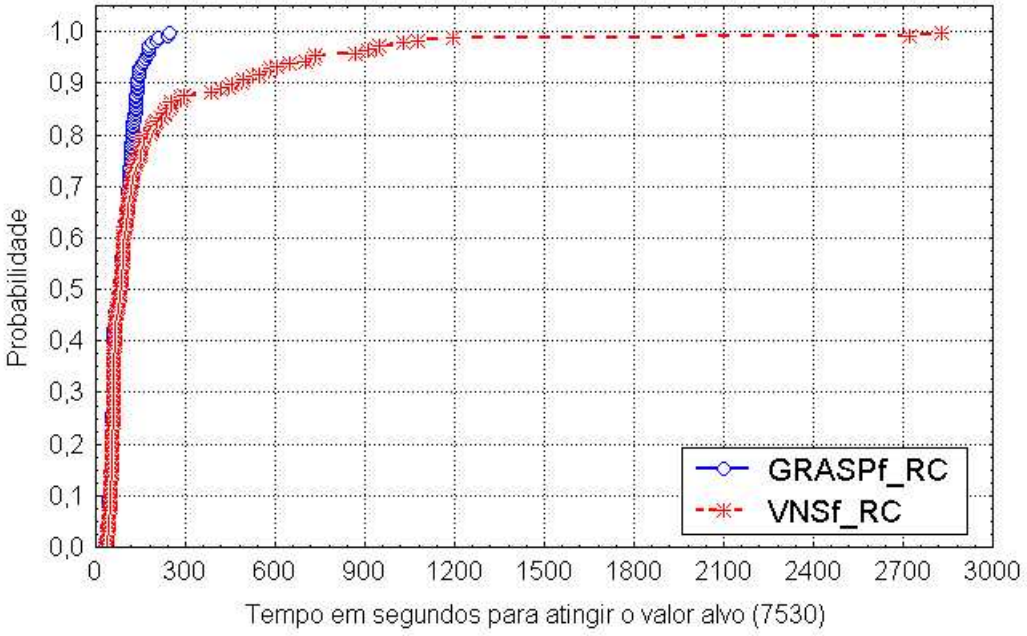

Figura 6.3: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 7530 para a instância ORM_P30 ($w = 5$).

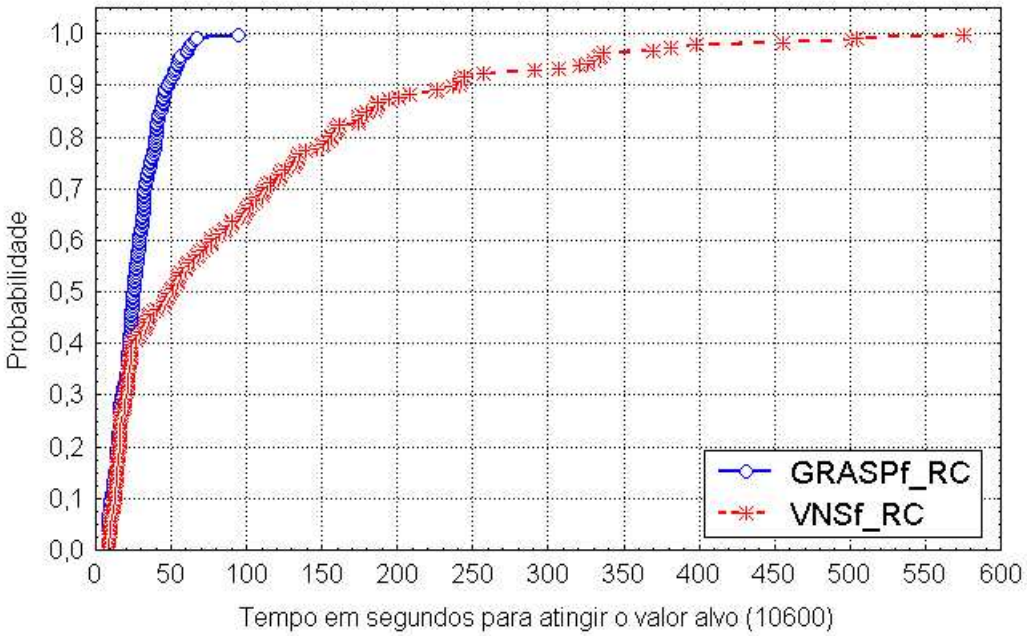

Figura 6.4: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 10600 para a instância ORM_P20 ($w = 10$).

Analisando-se agora as duas instâncias não proporcionais testadas, observou-se que, GRASPf_RC é superior ao VNSf_RC no problema GRM_NP17 e que, VNSf_RC é superior ao GRASPf_RC na instância ORM_NP37. Na Figura 6.5, a probabilidade de encontrar o valor alvo em menos de 100 segun-

dos é de aproximadamente 90% para o GRASPf_RC e de aproximadamente 71% para o VNSf_RC, enquanto que, na Figura 6.6, um valor menor ou igual ao alvo 6195 é encontrado em um tempo inferior à 100 segundos pelo VNSf_RC e GRASPf_RC com 95% e 45% de probabilidade, respectivamente.

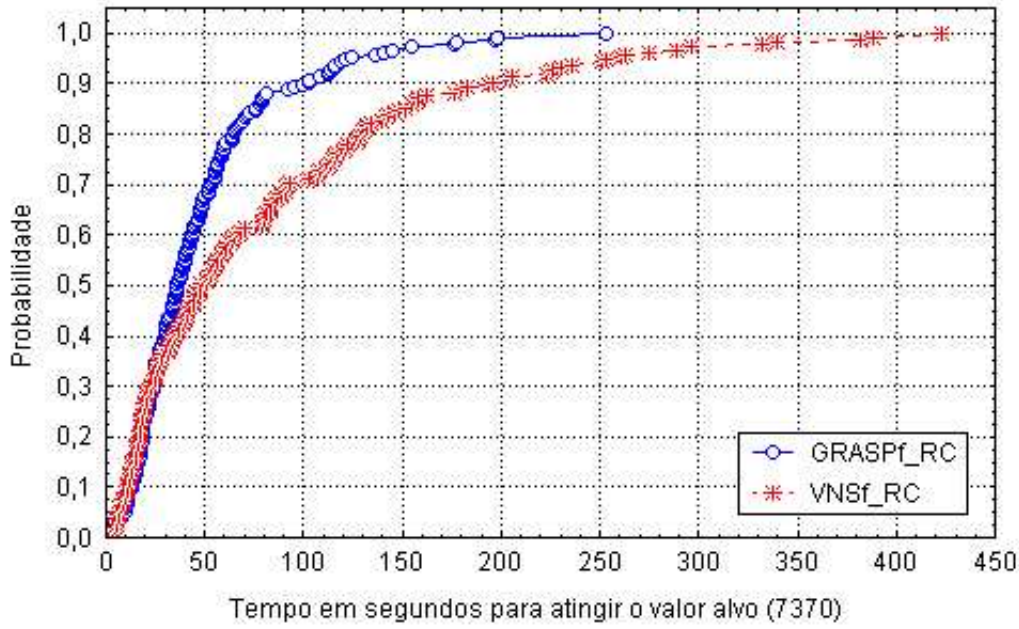

Figura 6.5: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 7370 para a instância GRM_NP17.

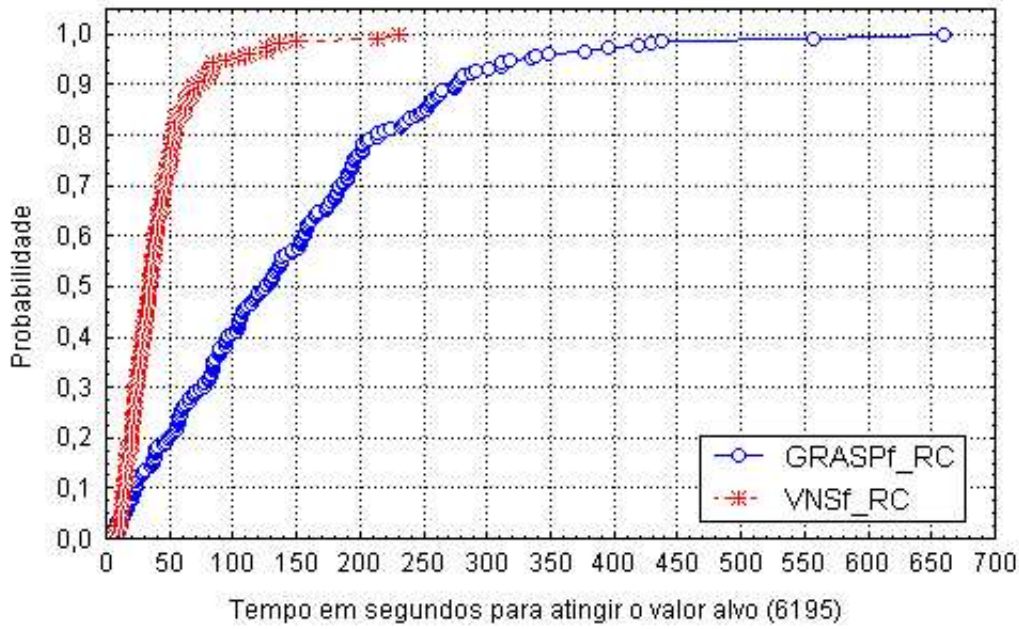

Figura 6.6: Distribuição de probabilidade empírica do tempo gasto para encontrar o valor alvo 6195 para a instância ORM_NP37.

6.4
Soluções Ótimas e Passo de Pós-Otimização

O modelo baseado em árvore geradora mínima restrita por grau possibilitou a obtenção de soluções ótimas para 49 das 189 instâncias proporcionais testadas, como mostram as Tabelas 6.13 e 6.14. As colunas indicam o nome do problema (Instância), o número de vértices do grafo (n), o número de arestas do grafo ($|E|$), o número de facilidades a serem instaladas (p), o valor da solução ótima do problema (F_O), o tempo gasto em segundos para encontrar a solução ótima (Tempo (s)) e o número de nós explorados pelo algoritmo *branch and bound* (Nós).

Observando-se as Tabelas 6.13 e 6.14, pode-se notar que a solução ótima foi encontrada somente para instâncias proporcionais relativamente pequenas em termos de número de vértices e facilidades a serem instaladas, o que mostra a dificuldade para a resolução exata de instâncias até mesmo menores. Gastou-se mais de oito dias de tempo de processamento para encontrar a solução ótima da instância ORM_P27 ($w = 2$) e quase dez dias de tempo de processamento para encontrar a solução ótima da instância GRM_P12 ($w = 2$). O problema ORM_P35 ($w = 2$) não foi resolvido devido à falta de memória. A solução ótima do problema ORM_P12 com $w = 2$ (instância com asterisco na Tabela 6.13) foi encontrada inicializando-se o resolvedor de programação linear inteira com o valor da solução obtida pelas heurísticas GRASPf_RC e VNSf_RC no tempo tg (7074). Mesmo utilizando o valor da solução obtida pelas heurísticas como limite inicial, gastou-se aproximadamente três dias de tempo de processamento para encontrar a solução ótima do problema.

A obtenção de soluções ótimas para esses problemas possibilitou realizar comparações em termos absolutos com as heurísticas. Para cada instância, a melhor solução obtida pelo GRASPf_RC e pelo VNSf_RC em cinco execuções com o tempo limitado a tg (em segundos) será utilizada para a comparação com a solução ótima. Para as instâncias em que não se conhece o ótimo, a melhor solução encontrada pelo algoritmo GRASPf_RC ou pela heurística VNSf_RC será a melhor solução conhecida. As Tabelas 6.15 a 6.25 apresentam esses resultados para as instâncias proporcionais e não proporcionais, onde as colunas F_G^* e F_V^* correspondem ao valor da melhor solução encontrada pelos algoritmos GRASPf_RC e VNSf_RC, respectivamente. As colunas $\%_G = 100 \times \frac{F_G^* - F_O}{F_O}$ e $\%_V = 100 \times \frac{F_V^* - F_O}{F_O}$ indicam, respectivamente, o desvio relativo percentual do valor das soluções obtidas pelos algoritmos GRASPf_RC e VNSf_RC em relação ao valor ótimo de cada instância proporcional. Os valores em negrito representam o melhor resultado obtido pelo GRASPf_RC ou VNSf_RC nas instâncias em que não ocorreram empates.

Comparando-se a qualidade das soluções e os tempos de processamento do algoritmo exato (Tabelas 6.13 e 6.14) e das heurísticas (Tabelas 6.15

a 6.20) nas instâncias proporcionais, observa-se que, em poucos segundos, o GRASPf_RC e o VNSf_RC encontraram a solução ótima na maioria das instâncias testadas, enquanto o algoritmo exato gastou horas ou até mesmo dias de tempo de processamento para encontrá-la. Porém, em alguns problemas como, por exemplo, GRM_P1 e ORM_P27 ($w = 2$), as heurísticas não conseguiram encontrar a solução ótima.

	Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
$w = 2$	ORM_P1	100	198	5	6443	1.072,39	59.893
	ORM_P2	100	193	10	5228	4.792,11	140.621
	ORM_P3	100	198	10	5368	9.366,33	372.665
	ORM_P4	100	196	20	5123	108.305,75	5.477.384
	ORM_P6	200	786	5	8180	58.447,19	1.016.864
	ORM_P7	200	779	10	6217	166.072,09	2.098.362
	ORM_P11	300	1.772	5	7855	5.072,67	22.906
	ORM_P12*	300	1.758	10	7074	266.225,22	910.891
	ORM_P16	400	3.153	5	8292	3.843,13	2.350
	ORM_P17	400	3.142	10	7279	40.988,16	64.417
	ORM_P21	500	4.909	5	9272	10.817,77	9.453
	ORM_P22	500	4.896	10	8839	169.399,73	209.789
	ORM_P26	600	7.068	5	10022	41.627,77	45.071
	ORM_P27	600	7.072	10	8553	720.582,59	460.940
	ORM_P31	700	9.601	5	10185	102.431,89	50.702
$w = 5$	ORM_P1	100	198	5	7184	1.599,26	54.990
	ORM_P2	100	193	10	6572	28.168,31	1.067.082
	ORM_P3	100	198	10	6776	76.090,34	1.749.654
	ORM_P11	300	1.772	5	8062	14.966,63	86.414
	ORM_P16	400	3.153	5	8458	10.551,84	3.675
	ORM_P21	500	4.909	5	9473	54.611,31	94.832
	ORM_P26	600	7.068	5	10169	179.345,39	237.600
$w = 10$	ORM_P1	100	198	5	8146	1.310,67	33.641
	ORM_P2	100	193	10	7706	7.315,97	219.336
	ORM_P3	100	198	10	8265	73.774,47	1.732.743
	ORM_P11	300	1.772	5	8383	29.226,83	56.750
	ORM_P16	400	3.153	5	8728	56.704,55	16.187
	ORM_P21	500	4.909	5	9808	72.600,92	47.605

Tabela 6.13: Valor da solução ótima para algumas instâncias da classe proporcional ORM_P ($w = 2$, $w = 5$ e $w = 10$).

Comparando-se as heurísticas entre si (Tabelas 6.15 a 6.25), observa-se que as mesmas obtiveram praticamente os mesmos resultados no maior tempo de execução (tg). Nas instâncias proporcionais (189 problemas), GRASPf_RC e VNSf_RC encontraram os mesmos resultados em 173 instâncias. GRASPf_RC foi superior ao VNSf_RC em dez instâncias e VNSf_RC foi melhor do que GRASPf_RC em seis instâncias. Nas instâncias não proporcionais (60 problemas), GRASPf_RC e VNSf_RC encontraram os mesmos resultados em

49 instâncias. GRASPf_RC foi superior ao VNSf_RC em duas instâncias e VNSf_RC foi melhor do que GRASPf_RC em oito instâncias.

Em relação às 49 instâncias proporcionais cujo valor ótimo é conhecido, os resultados apresentados pelas heurísticas são idênticos. GRASPf_RC e VNSf_RC conseguiram encontrar a solução ótima em 42 problemas. Nas sete instâncias restantes, as heurísticas ficaram, no máximo, a 0,54% do valor ótimo em GRM_P10 com $w = 2$.

Analisando-se o conjunto de facilidades abertas encontradas na solução ótima e comparando-se com as facilidades abertas encontradas pelas heurísticas nas sete instâncias, foi observado que, em algumas delas, as mesmas facilidades foram abertas pelo algoritmo exato e pela heurística, mas a árvore de Steiner obtida pela heurística Prim ao conectá-las não era ótima. A partir dessas observações, propõe-se então um passo adicional de pós-otimização, que consiste em utilizar um algoritmo exato para o problema de Steiner usando como entrada as facilidades abertas na melhor solução encontrada pelos algoritmos GRASPf_RC e VNSf_RC após todas as iterações terem sido realizadas.

	Instância	n	$\|E\|$	p	F_O	Tempo (s)	Nós
	GRM_P1	100	4.950	5	6391	6.496,75	84.575
	GRM_P2	100		10	5475	45.788,98	154.709
	GRM_P3	100		15	5157	33.666,48	95.137
$w = 2$	GRM_P4	100		20	5246	269.193,56	1.559.511
	GRM_P10	150	11.175	5	11413	15.056,66	17.124
	GRM_P11	150		10	9823	83.297,28	51.512
	GRM_P12	150		15	8867	856.170,36	361.340
	GRM_P1	100	4.950	5	7036	1.383,08	2.104
	GRM_P2	100		10	6662	35.836,39	86.685
	GRM_P3	100		15	6923	19.166,23	57.150
$w = 5$	GRM_P4	100		20	7602	20.264,56	175.112
	GRM_P10	150	11.175	5	12202	7.655,94	5.596
	GRM_P11	150		10	11086	79.751,42	53.927
	GRM_P12	150		15	10767	319.858,98	187.331
	GRM_P1	100	4.950	5	7884	1.466,13	2.613
	GRM_P2	100		10	8580	22.530,55	76.213
	GRM_P3	100		15	9793	2.949,95	7.896
$w = 10$	GRM_P4	100		20	11487	16.681,17	191.562
	GRM_P10	150	11.175	5	13083	8.693,39	6.717
	GRM_P11	150		10	13040	28.675,19	21.631
	GRM_P12	150		15	13845	373.131,34	312.704

Tabela 6.14: Valor da solução ótima para algumas instâncias da classe proporcional GRM_P ($w = 2$, $w = 5$ e $w = 10$).

Instância	n	p	tg (s)	F_O	F_G^*	$\%_G$	F_V^*	$\%_V$
ORM_P1	100	5	70,00	6443	6443	0,00	6443	0,00
ORM_P2	100	10	70,00	5228	5228	0,00	5228	0,00
ORM_P3	100	10	70,00	5368	5368	0,00	5368	0,00
ORM_P4	100	20	70,00	5123	5123	0,00	5123	0,00
ORM_P5	100	33	70,00	——	3663	——	3663	——
ORM_P6	200	5	70,00	8180	8185	0,06	8185	0,06
ORM_P7	200	10	70,00	6217	6217	0,00	6217	0,00
ORM_P8	200	20	70,00	——	5865	——	5865	——
ORM_P9	200	40	70,00	——	4699	——	4699	——
ORM_P10	200	67	110,60	——	3457	——	3457	——
ORM_P11	300	5	70,00	7855	7855	0,00	7855	0,00
ORM_P12	300	10	70,00	7074	7074	0,00	7074	0,00
ORM_P13	300	30	70,00	——	5539	——	5539	——
ORM_P14	300	60	136,50	——	4981	——	4981	——
ORM_P15	300	100	565,60	——	4477	——	4477	——
ORM_P16	400	5	70,00	8292	8292	0,00	8292	0,00
ORM_P17	400	10	70,00	7279	7279	0,00	7279	0,00
ORM_P18	400	40	70,00	——	**5982**	——	5984	——
ORM_P19	400	80	439,60	——	4787	——	4787	——
ORM_P20	400	133	1.397,20	——	4860	——	4860	——
ORM_P21	500	5	70,00	9272	9272	0,00	9272	0,00
ORM_P22	500	10	70,00	8839	8839	0,00	8839	0,00
ORM_P23	500	50	105,70	——	5799	——	5799	——
ORM_P24	500	100	1.129,80	——	4983	——	**4982**	——
ORM_P25	500	167	3.789,10	——	4808	——	4808	——
ORM_P26	600	5	70,00	10022	10022	0,00	10022	0,00
ORM_P27	600	10	70,00	8553	8555	0,02	8555	0,02
ORM_P28	600	60	223,30	——	5599	——	**5598**	——
ORM_P29	600	120	2.313,50	——	**5078**	——	5079	——
ORM_P30	600	200	6.317,50	——	5133	——	5133	——
ORM_P31	700	5	70,00	10185	10185	0,00	10185	0,00
ORM_P32	700	10	70,00	——	9509	——	9509	——
ORM_P33	700	70	225,40	——	5920	——	**5919**	——
ORM_P34	700	140	3.195,50	——	5099	——	5099	——
ORM_P35	800	5	70,00	——	10485	——	10485	——
ORM_P36	800	10	70,00	——	10157	——	10157	——
ORM_P37	800	80	504,70	——	6403	——	**6402**	——
ORM_P38	900	5	70,00	——	11156	——	11156	——
ORM_P39	900	10	70,00	——	9591	——	9591	——
ORM_P40	900	90	1.268,40	——	6438	——	**6437**	——

Tabela 6.15: Comparações absolutas entre as heurísticas para as instâncias proporcionais da classe ORM_P ($w = 2$).

Instância	n	p	tg (s)	F_O	F_G^*	$\%_G$	F_V^*	$\%_V$
ORM_P1	100	5	70,00	7184	7184	0,00	7184	0,00
ORM_P2	100	10	70,00	6572	6572	0,00	6572	0,00
ORM_P3	100	10	70,00	6776	6776	0,00	6776	0,00
ORM_P4	100	20	70,00	——	6944	——	6944	——
ORM_P5	100	33	70,00	——	5292	——	5292	——
ORM_P6	200	5	70,00	——	8662	——	8662	——
ORM_P7	200	10	70,00	——	6977	——	**6967**	——
ORM_P8	200	20	70,00	——	7203	——	7203	——
ORM_P9	200	40	83,30	——	6436	——	6436	——
ORM_P10	200	67	94,50	——	5004	——	5004	——
ORM_P11	300	5	70,00	8062	8062	0,00	8062	0,00
ORM_P12	300	10	70,00	——	7666	——	7666	——
ORM_P13	300	30	114,80	——	6693	——	6693	——
ORM_P14	300	60	277,90	——	6632	——	6632	——
ORM_P15	300	100	441,00	——	6363	——	6363	——
ORM_P16	400	5	70,00	8458	8458	0,00	8458	0,00
ORM_P17	400	10	70,00	——	7676	——	7676	——
ORM_P18	400	40	304,50	——	7259	——	7259	——
ORM_P19	400	80	917,00	——	6537	——	6537	——
ORM_P20	400	133	1.570,10	——	**7256**	——	7258	——
ORM_P21	500	5	70,00	9473	9473	0,00	9473	0,00
ORM_P22	500	10	70,00	——	9219	——	9219	——
ORM_P23	500	50	954,80	——	7020	——	7020	——
ORM_P24	500	100	2.519,30	——	6648	——	6648	——
ORM_P25	500	167	3.541,30	——	6897	——	6897	——
ORM_P26	600	5	70,00	10169	10169	0,00	10169	0,00
ORM_P27	600	10	70,00	——	8893	——	8893	——
ORM_P28	600	60	925,40	——	6791	——	6791	——
ORM_P29	600	120	4.786,60	——	6739	——	6739	——
ORM_P30	600	200	9.401,70	——	**7485**	——	7486	——
ORM_P31	700	5	70,00	——	10332	——	10332	——
ORM_P32	700	10	70,00	——	9821	——	9821	——
ORM_P33	700	70	2.183,30	——	7224	——	7224	——
ORM_P34	700	140	7.136,50	——	6771	——	6771	——
ORM_P35	800	5	70,00	——	10610	——	10610	——
ORM_P36	800	10	70,00	——	10452	——	10452	——
ORM_P37	800	80	4.980,50	——	7897	——	7897	——
ORM_P38	900	5	70,00	——	11300	——	11300	——
ORM_P39	900	10	70,00	——	9839	——	9839	——
ORM_P40	900	90	7.820,40	——	7836	——	7836	——

Tabela 6.16: Comparações absolutas entre as heurísticas para as instâncias proporcionais da classe ORM_P ($w = 5$).

Instância	n	p	tg (s)	F_O	F_G^*	$\%_G$	F_V^*	$\%_V$
ORM_P1	100	5	70,00	8146	8146	0,00	8146	0,00
ORM_P2	100	10	70,00	7706	7706	0,00	7706	0,00
ORM_P3	100	10	70,00	8265	8265	0,00	8265	0,00
ORM_P4	100	20	70,00	——	9135	——	9135	——
ORM_P5	100	33	70,00	——	7485	——	7485	——
ORM_P6	200	5	70,00	——	9387	——	9387	——
ORM_P7	200	10	70,00	——	7878	——	7878	——
ORM_P8	200	20	70,00	——	8632	——	8632	——
ORM_P9	200	40	70,00	——	8174	——	8174	——
ORM_P10	200	67	70,00	——	6894	——	6894	——
ORM_P11	300	5	70,00	8383	8383	0,00	8383	0,00
ORM_P12	300	10	70,00	——	8426	——	8426	——
ORM_P13	300	30	70,00	——	7679	——	7679	——
ORM_P14	300	60	372,40	——	8505	——	8505	——
ORM_P15	300	100	593,60	——	8864	——	8864	——
ORM_P16	400	5	70,00	8728	8728	0,00	8728	0,00
ORM_P17	400	10	70,00	——	8317	——	8317	——
ORM_P18	400	40	197,40	——	8547	——	8547	——
ORM_P19	400	80	1.267,70	——	8313	——	8313	——
ORM_P20	400	133	1.599,50	——	**10386**	——	10397	——
ORM_P21	500	5	70,00	9808	9808	0,00	9808	0,00
ORM_P22	500	10	70,00	——	9776	——	9776	——
ORM_P23	500	50	679,00	——	8359	——	8359	——
ORM_P24	500	100	2.735,60	——	8548	——	8548	——
ORM_P25	500	167	3.761,80	——	9610	——	9610	——
ORM_P26	600	5	70,00	——	10414	——	10414	——
ORM_P27	600	10	70,00	——	9317	——	9317	——
ORM_P28	600	60	2.317,00	——	8077	——	8077	——
ORM_P29	600	120	4.769,10	——	8552	——	8552	——
ORM_P30	600	200	9.440,90	——	10735	——	10735	——
ORM_P31	700	5	70,00	——	10556	——	10556	——
ORM_P32	700	10	70,00	——	10304	——	10304	——
ORM_P33	700	70	3.476,20	——	8666	——	8666	——
ORM_P34	700	140	5.868,80	——	8551	——	8551	——
ORM_P35	800	5	70,00	——	10811	——	10811	——
ORM_P36	800	10	70,00	——	10898	——	10898	——
ORM_P37	800	80	6.720,70	——	9575	——	9575	——
ORM_P38	900	5	70,00	——	11525	——	11525	——
ORM_P39	900	10	70,00	——	10220	——	10220	——
ORM_P40	900	90	14.610,40	——	9380	——	9380	——

Tabela 6.17: Comparações absolutas entre as heurísticas para as instâncias proporcionais da classe ORM_P ($w = 10$).

Instância	n	p	$tg\ (s)$	F_O	F_G^*	$\%_G$	F_V^*	$\%_V$
GRM_P1	100	5	70,00	6391	6403	0,19	6403	0,19
GRM_P2	100	10	70,00	5475	5489	0,26	5489	0,26
GRM_P3	100	15	70,00	5157	5157	0,00	5157	0,00
GRM_P4	100	20	70,00	5246	5246	0,00	5246	0,00
GRM_P5	100	25	70,00	——	5403	——	5403	——
GRM_P6	100	30	70,00	——	5580	——	5580	——
GRM_P7	100	35	70,00	——	5766	——	5766	——
GRM_P8	100	40	70,00	——	5961	——	5961	——
GRM_P9	100	50	70,00	——	6364	——	6364	——
GRM_P10	150	5	70,00	11413	11475	0,54	11475	0,54
GRM_P11	150	10	70,00	9823	9862	0,40	9862	0,40
GRM_P12	150	15	70,00	8867	8910	0,48	8910	0,48
GRM_P13	150	20	70,00	——	8352	——	8352	——
GRM_P14	150	25	70,00	——	8174	——	8174	——
GRM_P15	150	30	70,70	——	8133	——	8133	——
GRM_P16	150	35	197,40	——	8243	——	8243	——
GRM_P17	150	40	182,70	——	8372	——	8372	——
GRM_P18	150	45	270,20	——	8530	——	8530	——
GRM_P19	150	50	333,20	——	8701	——	8701	——
GRM_P20	150	60	328,30	——	9080	——	9080	——

Tabela 6.18: Comparações absolutas entre as heurísticas para as instâncias proporcionais da classe GRM_P ($w = 2$).

Instância	n	p	$tg\ (s)$	F_O	F_G^*	$\%_G$	F_V^*	$\%_V$
GRM_P1	100	5	70,00	7036	7036	0,00	7036	0,00
GRM_P2	100	10	70,00	6662	6662	0,00	6662	0,00
GRM_P3	100	15	70,00	6923	6923	0,00	6923	0,00
GRM_P4	100	20	70,00	7602	7602	0,00	7602	0,00
GRM_P5	100	25	70,00	——	8379	——	8379	——
GRM_P6	100	30	70,00	——	9178	——	9178	——
GRM_P7	100	35	70,00	——	9984	——	9984	——
GRM_P8	100	40	70,00	——	10802	——	10802	——
GRM_P9	100	50	70,00	——	12447	——	12447	——
GRM_P10	150	5	70,00	12202	12202	0,00	12202	0,00
GRM_P11	150	10	70,00	11086	11086	0,00	11086	0,00
GRM_P12	150	15	70,00	10767	10767	0,00	10767	0,00
GRM_P13	150	20	70,00	——	10902	——	10902	——
GRM_P14	150	25	82,60	——	11293	——	11293	——
GRM_P15	150	30	149,10	——	11874	——	11874	——
GRM_P16	150	35	212,80	——	12539	——	12539	——
GRM_P17	150	40	197,40	——	13276	——	13276	——
GRM_P18	150	45	221,20	——	14036	——	14036	——
GRM_P19	150	50	426,30	——	14814	——	14814	——
GRM_P20	150	60	386,40	——	16422	——	16422	——

Tabela 6.19: Comparações absolutas entre as heurísticas para as instâncias proporcionais da classe GRM_P ($w = 5$).

Instância	n	p	$tg\ (s)$	F_O	F_G^*	$\%_G$	F_V^*	$\%_V$
GRM_P1	100	5	70,00	7884	7884	0,00	7884	0,00
GRM_P2	100	10	70,00	8580	8580	0,00	8580	0,00
GRM_P3	100	15	70,00	9793	9793	0,00	9793	0,00
GRM_P4	100	20	70,00	11487	11487	——	11487	——
GRM_P5	100	25	70,00	——	13273	——	13273	——
GRM_P6	100	30	70,00	——	15079	——	15079	——
GRM_P7	100	35	70,00	——	16909	——	16909	——
GRM_P8	100	40	70,00	——	18754	——	18754	——
GRM_P9	100	50	70,00	——	22507	——	22507	——
GRM_P10	150	5	70,00	13083	13083	0,00	13083	0,00
GRM_P11	150	10	70,00	13040	13040	0,00	13040	0,00
GRM_P12	150	15	70,00	13845	13845	0,00	13845	0,00
GRM_P13	150	20	70,00	——	14985	——	14985	——
GRM_P14	150	25	142,80	——	16355	——	16355	——
GRM_P15	150	30	119,70	——	17919	——	17919	——
GRM_P16	150	35	168,00	——	19579	——	19579	——
GRM_P17	150	40	157,50	——	21317	——	21317	——
GRM_P18	150	45	211,40	——	23095	——	23095	——
GRM_P19	150	50	247,80	——	24895	——	24895	——
GRM_P20	150	60	252,70	——	28566	——	28566	——

Tabela 6.20: Comparações absolutas entre as heurísticas para as instâncias proporcionais da classe GRM_P ($w = 10$).

Instância	n	p	$tg\ (s)$	F_O	F_G^*	$\%_G$	F_V^*	$\%_V$
SLM_P1	700	233	10.581,90	——	4851	——	4851	——
SLM_P2	800	267	16.639,00	——	5317	——	5317	——
SLM_P3	900	300	36.101,80	——	5409	——	5409	——

Tabela 6.21: Comparações absolutas entre as heurísticas para as instâncias proporcionais da classe SLM_P ($w = 2$).

Instância	n	p	$tg\ (s)$	F_O	F_G^*	$\%_G$	F_V^*	$\%_V$
SLM_P1	700	233	12.399,80	——	6961	——	6961	——
SLM_P2	800	267	24.776,50	——	**7840**	——	7841	——
SLM_P3	900	300	35.674,10	——	**7880**	——	7885	——

Tabela 6.22: Comparações absolutas entre as heurísticas para as instâncias proporcionais da classe SLM_P ($w = 5$).

Instância	n	p	$tg\ (s)$	F_O	F_G^*	$\%_G$	F_V^*	$\%_V$
SLM_P1	700	233	12.350,10	——	**9767**	——	9788	——
SLM_P2	800	267	33.688,90	——	**11304**	——	11319	——
SLM_P3	900	300	51.396,10	——	**11273**	——	11289	——

Tabela 6.23: Comparações absolutas entre as heurísticas para as instâncias proporcionais da classe SLM_P ($w = 10$).

Instância	n	p	$tg\ (s)$	F_G^*	F_V^*
ORM_NP1	100	5	70,00	6447	6447
ORM_NP2	100	10	70,00	5196	5196
ORM_NP3	100	10	70,00	5235	5235
ORM_NP4	100	20	70,00	4935	**4930**
ORM_NP5	100	33	70,00	3377	3377
ORM_NP6	200	5	70,00	8098	8098
ORM_NP7	200	10	70,00	6153	6153
ORM_NP8	200	20	70,00	5627	**5623**
ORM_NP9	200	40	70,00	4364	4364
ORM_NP10	200	67	110,60	3362	3362
ORM_NP11	300	5	70,00	7832	7832
ORM_NP12	300	10	70,00	6979	6979
ORM_NP13	300	30	70,00	5342	5342
ORM_NP14	300	60	136,50	4781	4781
ORM_NP15	300	100	565,60	4120	4120
ORM_NP16	400	5	70,00	8293	8293
ORM_NP17	400	10	70,00	7246	7246
ORM_NP18	400	40	70,00	5838	**5835**
ORM_NP19	400	80	439,60	4546	4546
ORM_NP20	400	133	1.397,20	4428	4428
ORM_NP21	500	5	70,00	9238	9238
ORM_NP22	500	10	70,00	8800	8800
ORM_NP23	500	50	105,70	5629	**5628**
ORM_NP24	500	100	1.129,80	4678	4678
ORM_NP25	500	167	3.789,10	4538	4538
ORM_NP26	600	5	70,00	9996	9996
ORM_NP27	600	10	70,00	8514	8514
ORM_NP28	600	60	223,30	5442	**5438**
ORM_NP29	600	120	2.313,50	4801	4801
ORM_NP30	600	200	6.317,50	4700	4700
ORM_NP31	700	5	70,00	10158	10158
ORM_NP32	700	10	70,00	9474	9474
ORM_NP33	700	70	225,40	5723	**5718**
ORM_NP34	700	140	3.195,50	4759	4759
ORM_NP35	800	5	70,00	10463	10463
ORM_NP36	800	10	70,00	10116	10116
ORM_NP37	800	80	504,70	6190	**6189**
ORM_NP38	900	5	70,00	11129	11129
ORM_NP39	900	10	70,00	9551	9551
ORM_NP40	900	90	1.268,40	6213	6213

Tabela 6.24: Comparações absolutas entre as heurísticas para as instâncias não proporcionais da classe ORM_NP.

Instância	n	p	tg (s)	F_G^*	F_V^*
GRM_NP1	100	5	70,00	6156	6156
GRM_NP2	100	10	70,00	5185	5185
GRM_NP3	100	15	70,00	4825	4825
GRM_NP4	100	20	70,00	4720	4720
GRM_NP5	100	25	70,00	4675	4675
GRM_NP6	100	30	70,00	4675	4675
GRM_NP7	100	35	70,00	4685	4685
GRM_NP8	100	40	70,00	4717	4717
GRM_NP9	100	50	70,00	4806	4806
GRM_NP10	150	5	70,00	11334	11334
GRM_NP11	150	10	70,00	9710	9710
GRM_NP12	150	15	70,00	**8751**	8759
GRM_NP13	150	20	70,00	8135	8135
GRM_NP14	150	25	70,00	7719	7719
GRM_NP15	150	30	70,70	7527	7527
GRM_NP16	150	35	197,40	**7427**	7431
GRM_NP17	150	40	182,70	7362	**7360**
GRM_NP18	150	45	270,20	7341	7341
GRM_NP19	150	50	333,20	7346	7346
GRM_NP20	150	60	328,30	7405	7405

Tabela 6.25: Comparações absolutas entre as heurísticas para as instâncias não proporcionais da classe GRM_NP.

6.4.1 Passo de Pós-Otimização

O algoritmo de pós-otimização poderia ser uma heurística de busca local ou mesmo uma heurística baseada em uma metaheurística para o problema de Steiner. Um algoritmo exato foi utilizado porque o número de terminais é relativamente pequeno em comparação com instâncias testadas na literatura para o problema de Steiner [61], não demandando, assim, um custo computacional alto. Além disso, obtém-se também um parâmetro para testar a eficiência da heurística construtiva Prim escolhida para conectar as p facilidades abertas. A pós-otimização é executada por um algoritmo do tipo *branch and ascent* [61].

As Tabelas 6.26 a 6.44 apresentam a melhor solução encontrada pelos algoritmos aplicando-se o *branch and ascent* à solução obtida em cada uma das cinco execuções das heurísticas limitadas ao tempo tg. As heurísticas GRASPf_RC e VNSf_RC acrescidas do procedimento de pós-otimização serão denominadas, respectivamente, de GRASPf_RC_PO e VNSf_RC_PO. As heurísticas apresentaram os mesmos resultados para as classes de instâncias GRM_P ($w = 2$, $w = 5$ e $w = 10$) e, por isso, os resultados serão mostrados em uma mesma tabela. A coluna F_P indica o valor da função objetivo relacionado ao custo de atendimento dos usuários realizado pelas p facilidades

abertas referente a melhor solução encontrada pelos algoritmos; A coluna F_T indica o valor da função objetivo relacionado ao custo de interconexão das facilidades abertas referente a melhor solução encontrada pelos algoritmos; F_{OT} representa o valor da árvore de Steiner obtida pelo algoritmo exato no passo de pós-otimização; Tempo (s) indica o tempo gasto em segundos na execução do algoritmo exato e F_{PO} representa o valor da melhor solução encontrada pelas heurísticas, levando-se em consideração, os possíveis ganhos obtidos com a pós-otimização. Os valores em negrito nas colunas F_T e F_{OT} representam uma melhoria na árvore de Steiner obtida pela pós-otimização e, conseqüentemente, uma melhoria na melhor solução encontrada pelos algoritmos (coluna F_{PO}).

A coluna Tempo (s) indica que o algoritmo exato do passo de pós-otimização executa em menos de um segundo para todas as instâncias testadas. A escolha de Prim se mostrou bastante acertada para integrar o algoritmo de busca local utilizado pelas heurísticas, pois na grande maioria dos problemas, a heurística construtiva conseguiu encontrar soluções com valor ótimo obtido pelo algoritmo *branch and ascent*.

Adicionando-se o passo de pós-otimização, as heurísticas encontraram a solução ótima nas instâncias ORM_P27 e GRM_P1 ($w = 2$). Nas instâncias ORM_P6, GRM_P2, GRM_P10, GRM_P11 e GRM_P12 ($w = 2$), em que a solução ótima é conhecida e as heurísticas GRASPf_RC e VNSf_RC não conseguiram encontrar o ótimo, houve uma melhoria somente na instância GRM_P10, com o desvio relativo percentual caindo de $0,54\%$ para $0,46\%$. Nessas cinco instâncias, GRASPf_RC_PO e VNSf_RC_PO ficaram, no máximo, a 0,48% do valor ótimo na instância GRM_P12.

Nos problemas em que não se conhece o ótimo, GRASPf_RC_PO conseguiu melhorar a melhor solução encontrada por GRASPf_RC em 13 instâncias proporcionais: ORM_P23, ORM_P28, ORM_P29, ORM_P33, ORM_P34 e ORM_P37 ($w = 2$), ORM_P7, ORM_P27 e ORM_P36 ($w = 5$), ORM_P32 e ORM_P39 ($w = 2$ e $w = 5$). Nas instâncias não proporcionais, ocorreram 16 melhorias: ORM_P8, ORM_P14, ORM_P16, ORM_P18, ORM_P19, ORM_P22, ORM_P23, ORM_P27, ORM_P28, ORM_P29, ORM_P33, ORM_P36, ORM_P37 e ORM_P40 ($w = 2$), GRM_P1 e GRM_P10 ($w = 2$). Já o VNSf_RC_PO conseguiu melhorar a melhor solução encontrada pelo VNSf_RC em 12 instâncias proporcionais: ORM_P23, ORM_P28, ORM_P29, ORM_P33 e ORM_P37 ($w = 2$), ORM_P7, ORM_P27 e ORM_P36 ($w = 5$), ORM_P32 e ORM_P39 ($w = 2$ e $w = 5$). Nas instâncias não proporcionais, ocorreram 16 melhorias: ORM_P14, ORM_P16, ORM_P18, ORM_P19, ORM_P22, ORM_P23, ORM_P27, ORM_P28, ORM_P29, ORM_P33, ORM_34, ORM_P36, ORM_P37 e ORM_P40 ($w = 2$), GRM_P1 e GRM_P10 ($w = 2$). Com a pós-otimização, das 189 (resp. 60) instâncias proporcionais (resp. não

proporcionais) testadas, GRASPf_RC_PO e VNSf_RC_PO encontraram os mesmos resultados em 174 (resp. 53) instâncias. GRASPf_RC foi superior a VNSf_RC em dez (resp. duas) instâncias e VNSf_RC foi melhor do que GRASPf_RC em cinco (resp. cinco) instâncias.

Instância	F_O	F_G^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
ORM_P1	6443	6443	5891	552	552	0,00	6443
ORM_P2	5228	5228	4224	1004	1004	0,00	5228
ORM_P3	5368	5368	4276	1092	1092	0,00	5368
ORM_P4	5123	5123	3661	1462	1462	0,01	5123
ORM_P5	——	3663	2269	1394	1394	0,00	3663
ORM_P6	8180	8185	7867	318	318	0,00	8185
ORM_P7	6217	6217	5689	528	528	0,01	6217
ORM_P8	——	5865	4751	1114	1114	0,01	5865
ORM_P9	——	4699	3285	1414	1414	0,01	4699
ORM_P10	——	3457	2193	1264	1264	0,00	3457
ORM_P11	7855	7855	7717	138	138	0,01	7855
ORM_P12	7074	7074	6658	416	416	0,01	7074
ORM_P13	——	5539	4613	926	926	0,02	5539
ORM_P14	——	4981	3447	1534	1534	0,02	4981
ORM_P15	——	4477	2781	1696	1696	0,02	4477
ORM_P16	8292	8292	8172	120	120	0,03	8292
ORM_P17	7279	7279	7009	270	270	0,02	7279
ORM_P18	——	5982	4980	1002	1002	0,03	5982
ORM_P19	——	4787	3223	1564	1564	0,06	4787
ORM_P20	——	4860	2828	2032	2032	0,05	4860
ORM_P21	9272	9272	9138	134	134	0,03	9272
ORM_P22	8839	8839	8579	260	260	0,02	8839
ORM_P23	——	5799	4805	**994**	**992**	0,08	**5797**
ORM_P24	——	4983	3475	1508	1508	0,09	4983
ORM_P25	——	4808	2994	1814	1814	0,10	4808
ORM_P26	10022	10022	9924	98	98	0,05	10022
ORM_P27	8553	8555	8329	**226**	**224**	0,11	**8553**
ORM_P28	——	5599	4653	**946**	**944**	0,10	**5597**
ORM_P29	——	5079	3497	**1582**	**1580**	0,12	**5077**
ORM_P30	——	5133	3131	2002	2002	0,11	5133
ORM_P31	10185	10185	10087	98	98	0,05	10185
ORM_P32	——	9509	9301	**208**	**200**	0,16	**9501**
ORM_P33	——	5921	4835	**1086**	**1082**	0,13	**5917**
ORM_P34	——	5100	3608	**1492**	**1490**	0,16	**5098**
ORM_P35	——	10485	10401	84	84	0,07	10485
ORM_P36	——	10157	9953	204	204	0,08	10157
ORM_P37	——	6404	5222	**1182**	**1180**	0,18	**6402**
ORM_P38	——	11156	11060	96	96	0,18	11156
ORM_P39	——	9591	9423	**168**	**158**	0,11	**9581**
ORM_P40	——	6438	5342	1096	1096	0,22	6438

Tabela 6.26: Resultados da pós-otimização para a heurística GRASPf_RC nas instâncias proporcionais da classe ORM_P ($w = 2$).

Instância	F_O	F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
ORM_P1	6443	6443	5891	552	552	0,00	6443
ORM_P2	5228	5228	4224	1004	1004	0,00	5228
ORM_P3	5368	5368	4276	1092	1092	0,00	5368
ORM_P4	5123	5123	3661	1462	1462	0,01	5123
ORM_P5	——	3663	2269	1394	1394	0,00	3663
ORM_P6	8180	8185	7867	318	318	0,00	8185
ORM_P7	6217	6217	5689	528	528	0,01	6217
ORM_P8	——	5865	4751	1114	1114	0,01	5865
ORM_P9	——	4699	3285	1414	1414	0,01	4699
ORM_P10	——	3457	2193	1264	1264	0,00	3457
ORM_P11	7855	7855	7717	138	138	0,01	7855
ORM_P12	7074	7074	6658	416	416	0,01	7074
ORM_P13	——	5539	4613	926	926	0,02	5539
ORM_P14	——	4981	3441	1540	1540	0,02	4981
ORM_P15	——	4477	2781	1696	1696	0,02	4477
ORM_P16	8292	8292	8172	120	120	0,03	8292
ORM_P17	7279	7279	7009	270	270	0,02	7279
ORM_P18	——	5984	4980	1004	1004	0,03	5984
ORM_P19	——	4787	3235	1552	1552	0,06	4787
ORM_P20	——	4860	2828	2032	2032	0,05	4860
ORM_P21	9272	9272	9138	134	134	0,03	9272
ORM_P22	8839	8839	8579	260	260	0,02	8839
ORM_P23	——	5799	4797	**1002**	**1000**	0,08	**5797**
ORM_P24	——	4982	3512	1470	1470	0,09	4982
ORM_P25	——	4808	2994	1814	1814	0,10	4808
ORM_P26	10022	10022	9924	98	98	0,05	10022
ORM_P27	8553	8555	8329	**226**	**224**	0,11	**8553**
ORM_P28	——	5598	4652	**946**	**944**	0,09	**5596**
ORM_P29	——	5079	3491	**1588**	**1586**	0,13	**5077**
ORM_P30	——	5133	3153	1980	1980	0,11	5133
ORM_P31	10185	10185	10087	98	98	0,05	10185
ORM_P32	——	9509	9301	**208**	**200**	0,16	**9501**
ORM_P33	——	5919	4825	**1094**	**1090**	0,21	**5915**
ORM_P34	——	5099	3583	1516	1516	0,17	5099
ORM_P35	——	10485	10401	84	84	0,07	10485
ORM_P36	——	10157	9953	204	204	0,08	10157
ORM_P37	——	6402	5186	**1216**	**1214**	0,32	**6400**
ORM_P38	——	11156	11060	96	96	0,18	11156
ORM_P39	——	9591	9423	**168**	**158**	0,11	**9581**
ORM_P40	——	6437	5339	1098	1098	0,22	6437

Tabela 6.27: Resultados da pós-otimização para a heurística VNSf_RC nas instâncias proporcionais da classe ORM_P ($w = 2$).

Instância	F_O	F_G^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
ORM_P1	7184	7184	6059	1125	1125	0,01	7184
ORM_P2	6572	6572	4992	1580	1580	0,00	6572
ORM_P3	6776	6776	5021	1755	1755	0,00	6776
ORM_P4	——	6944	4549	2395	2395	0,00	6944
ORM_P5	——	5292	2907	2385	2385	0,00	5292
ORM_P6	——	8662	7872	790	790	0,00	8662
ORM_P7	——	6977	5822	**1155**	**1130**	0,00	**6952**
ORM_P8	——	7203	5148	2055	2055	0,01	7203
ORM_P9	——	6436	4221	2215	2215	0,00	6436
ORM_P10	——	5004	3054	1950	1950	0,01	5004
ORM_P11	8062	8062	7717	345	345	0,01	8062
ORM_P12	——	7666	6796	870	870	0,01	7666
ORM_P13	——	6693	5238	1455	1455	0,01	6693
ORM_P14	——	6632	4392	2240	2240	0,00	6632
ORM_P15	——	6363	3708	2655	2655	0,02	6363
ORM_P16	8458	8458	8188	270	270	0,02	8458
ORM_P17	——	7676	7021	655	655	0,01	7676
ORM_P18	——	7259	5444	1815	1815	0,02	7259
ORM_P19	——	6537	4362	2175	2175	0,03	6537
ORM_P20	——	7256	3951	3305	3305	0,04	7256
ORM_P21	9473	9473	9138	335	335	0,04	9473
ORM_P22	——	9219	8594	625	625	0,03	9219
ORM_P23	——	7020	5325	1695	1695	0,05	7020
ORM_P24	——	6648	4418	2230	2230	0,06	6648
ORM_P25	——	6897	4037	2860	2860	0,07	6897
ORM_P26	10169	10169	9924	245	245	0,00	10169
ORM_P27	——	8893	8338	**555**	**550**	0,05	**8888**
ORM_P28	——	6791	5146	1645	1645	0,07	6791
ORM_P29	——	6739	4694	2045	2045	0,09	6739
ORM_P30	——	7485	4035	3450	3450	0,12	7485
ORM_P31	——	10332	10087	245	245	0,04	10332
ORM_P32	——	9821	9306	**515**	**495**	0,16	**9801**
ORM_P33	——	7224	5364	1860	1860	0,10	7224
ORM_P34	——	6771	4721	2050	2050	0,13	6771
ORM_P35	——	10610	10405	205	205	0,06	10610
ORM_P36	——	10452	9972	**480**	**475**	0,09	**10447**
ORM_P37	——	7897	5772	2125	2125	0,14	7897
ORM_P38	——	11300	11060	240	240	0,18	11300
ORM_P39	——	9839	9429	**410**	**405**	0,11	**9834**
ORM_P40	——	7836	5891	1945	1945	0,16	7836

Tabela 6.28: Resultados da pós-otimização para a heurística GRASPf_RC nas instâncias proporcionais da classe ORM_P ($w = 5$).

Instância	F_O	F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
ORM_P1	7184	7184	6059	1125	1125	0,01	7184
ORM_P2	6572	6572	4992	1580	1580	0,00	6572
ORM_P3	6776	6776	5021	1755	1755	0,00	6776
ORM_P4	——	6944	4549	2395	2395	0,00	6944
ORM_P5	——	5292	2907	2385	2385	0,00	5292
ORM_P6	——	8662	7872	790	790	0,00	8662
ORM_P7	——	6977	5822	**1155**	**1130**	0,00	**6952**
ORM_P8	——	7203	5148	2055	2055	0,01	7203
ORM_P9	——	6436	4221	2215	2215	0,00	6436
ORM_P10	——	5004	3054	1950	1950	0,01	5004
ORM_P11	8062	8062	7717	345	345	0,01	8062
ORM_P12	——	7666	6796	870	870	0,01	7666
ORM_P13	——	6693	5238	1455	1455	0,01	6693
ORM_P14	——	6632	4392	2240	2240	0,00	6632
ORM_P15	——	6363	3708	2655	2655	0,02	6363
ORM_P16	8458	8458	8188	270	270	0,02	8458
ORM_P17	——	7676	7021	655	655	0,01	7676
ORM_P18	——	7259	5444	1815	1815	0,02	7259
ORM_P19	——	6537	4362	2175	2175	0,03	6537
ORM_P20	——	7258	3893	3365	3365	0,06	7258
ORM_P21	9473	9473	9138	335	335	0,04	9473
ORM_P22	——	9219	8594	625	625	0,03	9219
ORM_P23	——	7020	5325	1695	1695	0,05	7020
ORM_P24	——	6648	4418	2230	2230	0,06	6648
ORM_P25	——	6897	4037	2860	2860	0,07	6897
ORM_P26	10169	10169	9924	245	245	0,00	10169
ORM_P27	——	8893	8333	**560**	**555**	0,06	**8888**
ORM_P28	——	6791	5146	1645	1645	0,07	6791
ORM_P29	——	6739	4694	2045	2045	0,09	6739
ORM_P30	——	7486	4016	3470	3470	0,12	7486
ORM_P31	——	10332	10087	245	245	0,04	10332
ORM_P32	——	9821	9306	**515**	**495**	0,16	**9801**
ORM_P33	——	7224	5374	1850	1850	0,10	7224
ORM_P34	——	6771	4711	2060	2060	0,13	6771
ORM_P35	——	10610	10405	205	205	0,06	10610
ORM_P36	——	10452	9972	**480**	**475**	0,09	**10447**
ORM_P37	——	7897	5812	2085	2085	0,14	7897
ORM_P38	——	11300	11060	240	240	0,18	11300
ORM_P39	——	9839	9429	**410**	**405**	0,11	**9834**
ORM_P40	——	7836	5891	1945	1945	0,16	7836

Tabela 6.29: Resultados da pós-otimização para a heurística VNSf_RC nas instâncias proporcionais da classe ORM_P ($w = 5$).

Instância	F_O	F_G^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
ORM_P1	8146	8146	6846	1300	1300	0,00	8146
ORM_P2	7706	7706	6606	1100	1100	0,00	7706
ORM_P3	8265	8265	5765	2500	2500	0,00	8265
ORM_P4	——	9135	5115	4020	4020	0,00	9135
ORM_P5	——	7485	3345	4140	4140	0,00	7485
ORM_P6	——	9387	8037	1350	1350	0,00	9387
ORM_P7	——	7878	6468	1410	1410	0,00	7878
ORM_P8	——	8632	6692	1940	1940	0,00	8632
ORM_P9	——	8174	5344	2830	2830	0,01	8174
ORM_P10	——	6894	3204	3690	3690	0,01	6894
ORM_P11	8383	8383	7753	630	630	0,00	8383
ORM_P12	——	8426	7046	1380	1380	0,01	8426
ORM_P13	——	7679	6249	1430	1430	0,00	7679
ORM_P14	——	8505	5165	3340	3340	0,01	8505
ORM_P15	——	8864	4024	4840	4840	0,02	8864
ORM_P16	8728	8728	8188	540	540	0,01	8728
ORM_P17	——	8317	7127	1190	1190	0,01	8317
ORM_P18	——	8547	6567	1980	1980	0,02	8547
ORM_P19	——	8313	5013	3300	3300	0,03	8313
ORM_P20	——	10386	4316	6070	6070	0,04	10386
ORM_P21	9808	9808	9138	670	670	0,04	9808
ORM_P22	——	9776	8706	1070	1070	0,02	9776
ORM_P23	——	8359	6179	2180	2180	0,04	8359
ORM_P24	——	8548	5148	3400	3400	0,00	8548
ORM_P25	——	9610	4400	5210	5210	0,07	9610
ORM_P26	——	10414	9924	490	490	0,06	10414
ORM_P27	——	9317	8557	760	760	0,03	9317
ORM_P28	——	8077	6037	2040	2040	0,04	8077
ORM_P29	——	8552	5252	3300	3300	0,07	8552
ORM_P30	——	10735	4445	6290	6290	0,11	10735
ORM_P31	——	10556	10126	430	430	0,04	10556
ORM_P32	——	10304	9434	870	870	0,05	10304
ORM_P33	——	8666	6366	2300	2300	0,05	8666
ORM_P34	——	8551	5201	3350	3350	0,12	8551
ORM_P35	——	10811	10491	320	320	0,05	10811
ORM_P36	——	10898	10028	870	870	0,06	10898
ORM_P37	——	9575	6845	2730	2730	0,12	9575
ORM_P38	——	11525	11105	420	420	0,08	11525
ORM_P39	——	10220	9480	740	740	0,08	10220
ORM_P40	——	9380	6840	2540	2540	0,12	9380

Tabela 6.30: Resultados da pós-otimização para a heurística GRASPf_RC nas instâncias proporcionais da classe ORM_P ($w = 10$).

Instância	F_O	F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
ORM_P1	8146	8146	6846	1300	1300	0,00	8146
ORM_P2	7706	7706	6606	1100	1100	0,00	7706
ORM_P3	8265	8265	5765	2500	2500	0,00	8265
ORM_P4	——	9135	5115	4020	4020	0,00	9135
ORM_P5	——	7485	3345	4140	4140	0,00	7485
ORM_P6	——	9387	8037	1350	1350	0,00	9387
ORM_P7	——	7878	6468	1410	1410	0,00	7878
ORM_P8	——	8632	6692	1940	1940	0,00	8632
ORM_P9	——	8174	5344	2830	2830	0,01	8174
ORM_P10	——	6894	3204	3690	3690	0,01	6894
ORM_P11	8383	8383	7753	630	630	0,00	8383
ORM_P12	——	8426	7046	1380	1380	0,01	8426
ORM_P13	——	7679	6249	1430	1430	0,00	7679
ORM_P14	——	8505	5165	3340	3340	0,01	8505
ORM_P15	——	8864	4024	4840	4840	0,02	8864
ORM_P16	8728	8728	8188	540	540	0,01	8728
ORM_P17	——	8317	7127	1190	1190	0,01	8317
ORM_P18	——	8547	6567	1980	1980	0,02	8547
ORM_P19	——	8313	5013	3300	3300	0,03	8313
ORM_P20	——	10397	4387	6010	6010	0,07	10397
ORM_P21	9808	9808	9138	670	670	0,04	9808
ORM_P22	——	9776	8706	1070	1070	0,02	9776
ORM_P23	——	8359	6159	2200	2200	0,04	8359
ORM_P24	——	8548	5148	3400	3400	0,00	8548
ORM_P25	——	9610	4400	5210	5210	0,08	9610
ORM_P26	——	10414	9924	490	490	0,06	10414
ORM_P27	——	9317	8557	760	760	0,03	9317
ORM_P28	——	8077	6037	2040	2040	0,04	8077
ORM_P29	——	8552	5252	3300	3300	0,07	8552
ORM_P30	——	10735	4445	6290	6290	0,13	10735
ORM_P31	——	10556	10126	430	430	0,04	10556
ORM_P32	——	10304	9434	870	870	0,05	10304
ORM_P33	——	8666	6366	2300	2300	0,05	8666
ORM_P34	——	8551	5201	3350	3350	0,12	8551
ORM_P35	——	10811	10491	320	320	0,05	10811
ORM_P36	——	10898	10028	870	870	0,06	10898
ORM_P37	——	9575	6825	2750	2750	0,12	9575
ORM_P38	——	11525	11105	420	420	0,08	11525
ORM_P39	——	10220	9480	740	740	0,08	10220
ORM_P40	——	9380	6840	2540	2540	0,12	9380

Tabela 6.31: Resultados da pós-otimização para a heurística VNSf_RC nas instâncias proporcionais da classe ORM_P ($w = 10$).

Instância	F_O	F_G^*/F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
GRM_P1	6391	6403	5703	**700**	**688**	0,09	**6391**
GRM_P2	5475	5489	4707	782	782	0,02	5489
GRM_P3	5157	5157	3971	1186	1186	0,01	5157
GRM_P4	5246	5246	3648	1598	1598	0,03	5246
GRM_P5	——	5403	3391	2012	2012	0,02	5403
GRM_P6	——	5580	3152	2428	2428	0,01	5580
GRM_P7	——	5766	2898	2868	2868	0,02	5766
GRM_P8	——	5961	2663	3298	3298	0,02	5961
GRM_P9	——	6364	2260	4104	4104	0,02	6364
GRM_P10	11413	11475	10935	**540**	**530**	0,06	**11465**
GRM_P11	9823	9862	9046	816	816	0,07	9862
GRM_P12	8867	8910	7664	1246	1246	0,07	8910
GRM_P13	——	8352	6636	1716	1716	0,07	8352
GRM_P14	——	8174	6070	2104	2104	0,08	8174
GRM_P15	——	8133	5623	2510	2510	0,08	8133
GRM_P16	——	8243	5325	2918	2918	0,08	8243
GRM_P17	——	8372	5042	3330	3330	0,08	8372
GRM_P18	——	8530	4816	3714	3714	0,06	8530
GRM_P19	——	8701	4581	4120	4120	0,05	8701
GRM_P20	——	9080	4164	4916	4916	0,06	9080

Tabela 6.32: Resultados da pós-otimização para as heurísticas GRASPf_RC e VNSf_RC nas instâncias proporcionais da classe GRM_P ($w = 2$).

6.5 Considerações Finais

Nesse capítulo, as heurísticas GRASP e VNS com filtro e reconexão por caminhos (GRASPf_RC e VNSf_RC, respectivamente) foram comparadas de três formas diferentes.

A qualidade das soluções encontradas pelas heurísticas para as 249 instâncias testadas (189 proporcionais e 60 não proporcionais) foram analisadas em três tempos de processamento distintos: tp, ti e tg. Utilizando-se na avaliação as medidas *drpm*, *cm* e *melhor*, em geral, o GRASPf_RC se mostrou superior ao VNSf_RC nas instâncias proporcionais. Já nas instâncias não proporcionais, o VNSf_RC apresentou resultados melhores. A diferença nas medidas torna-se menor quando aumenta-se o tempo de processamento, com as heurísticas convergindo praticamente para as mesmas soluções nas instâncias proporcionais e não proporcionais nos maiores tempos de processamento (ti e tg).

Em seguida, gráficos do tempo gasto para atingir o valor alvo foram utilizados na comparação. Dos problemas analisados, GRASPf_RC apresentou resultados melhores do que VNSf_RC nas instâncias proporcionais com fator w maior ($w = 5$ e $w = 10$) e nas instâncias não proporcionais GRM_NP.

Instância	F_O	F_G^*/F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
GRM_P1	7036	7036	6181	855	855	0,01	7036
GRM_P2	6662	6662	4707	1955	1955	0,02	6662
GRM_P3	6923	6923	4018	2905	2905	0,01	6923
GRM_P4	7602	7602	3707	3895	3895	0,02	7602
GRM_P5	——	8379	3464	4915	4915	0,02	8379
GRM_P6	——	9178	3263	5915	5915	0,01	9178
GRM_P7	——	9984	3059	6925	6925	0,02	9984
GRM_P8	——	10802	2832	7970	7970	0,01	10802
GRM_P9	——	12447	2357	10090	10090	0,03	12447
GRM_P10	12202	12202	11297	905	905	0,04	12202
GRM_P11	11086	11086	9046	2040	2040	0,07	11086
GRM_P12	10767	10767	7672	3095	3095	0,06	10767
GRM_P13	——	10902	6742	4160	4160	0,05	10902
GRM_P14	——	11293	6118	5175	5175	0,06	11293
GRM_P15	——	11874	5734	6140	6140	0,06	11874
GRM_P16	——	12539	5479	7060	7060	0,05	12539
GRM_P17	——	13276	5181	8095	8095	0,06	13276
GRM_P18	——	14036	4901	9135	9135	0,06	14036
GRM_P19	——	14814	4674	10140	10140	0,06	14814
GRM_P20	——	16422	4222	12200	12200	0,07	16422

Tabela 6.33: Resultados da pós-otimização para as heurísticas GRASPf_RC e VNSf_RC nas instâncias proporcionais da classe GRM_P ($w = 5$).

Instância	F_O	F_G^*/F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
GRM_P1	7884	7884	6224	1660	1660	0,01	7884
GRM_P2	8580	8580	4790	3790	3790	0,02	8580
GRM_P3	9793	9793	4093	5700	5700	0,01	9793
GRM_P4	11487	11487	3757	7730	7730	0,02	11487
GRM_P5	——	13273	3513	9760	9760	0,02	13273
GRM_P6	——	15079	3309	11770	11770	0,01	15079
GRM_P7	——	16909	3079	13830	13830	0,01	16909
GRM_P8	——	18754	2854	15900	15900	0,02	18754
GRM_P9	——	22507	2397	20110	20110	0,03	22507
GRM_P10	13083	13083	11423	1660	1660	0,02	13083
GRM_P11	13040	13040	9190	3850	3850	0,03	13040
GRM_P12	13845	13845	7775	6070	6070	0,06	13845
GRM_P13	——	14985	7025	7960	7960	0,05	14985
GRM_P14	——	16355	6395	9960	9960	0,04	16355
GRM_P15	——	17919	5919	12000	12000	0,06	17919
GRM_P16	——	19579	5549	14030	14030	0,05	19579
GRM_P17	——	21317	5297	16020	16020	0,05	21317
GRM_P18	——	23095	5035	18060	18060	0,05	23095
GRM_P19	——	24895	4775	20120	20120	0,06	24895
GRM_P20	——	28566	4316	24250	24250	0,06	28566

Tabela 6.34: Resultados da pós-otimização para as heurísticas GRASPf_RC e VNSf_RC nas instâncias proporcionais da classe GRM_P ($w = 10$).

Instância	F_O	F_G^*/F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
SLM_P1	——	4851	3065	1786	1786	0,19	4851
SLM_P2	——	5317	3291	2026	2026	0,23	5317
SLM_P3	——	5409	3281	2128	2128	0,32	5409

Tabela 6.35: Resultados da pós-otimização para a heurística GRASPf_RC nas instâncias proporcionais da classe SLM_P ($w = 2$).

Instância	F_O	F_G^*/F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
SLM_P1	——	4851	3051	1800	1800	0,19	4851
SLM_P2	——	5317	3297	2020	2020	0,23	5317
SLM_P3	——	5409	3325	2084	2084	0,32	5409

Tabela 6.36: Resultados da pós-otimização para a heurística VNSf_RC nas instâncias proporcionais da classe SLM_P ($w = 2$).

Instância	F_O	F_G^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
SLM_P1	——	6961	3986	2975	2975	0,15	6961
SLM_P2	——	7840	4205	3635	3635	0,23	7840
SLM_P3	——	7880	4315	3565	3565	0,28	7880

Tabela 6.37: Resultados da pós-otimização para a heurística GRASPf_RC nas instâncias proporcionais da classe SLM_P ($w = 5$).

Instância	F_O	F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
SLM_P1	——	6961	3986	2975	2975	0,16	6961
SLM_P2	——	7841	4196	3645	3645	0,23	7841
SLM_P3	——	7885	4325	3560	3560	0,29	7885

Tabela 6.38: Resultados da pós-otimização para a heurística VNSf_RC nas instâncias proporcionais da classe SLM_P ($w = 5$).

Instância	F_O	F_G^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
SLM_P1	——	9767	4297	5470	5470	0,16	9767
SLM_P2	——	11304	4574	6730	6730	0,19	11304
SLM_P3	——	11273	4683	6590	6590	0,30	11273

Tabela 6.39: Resultados da pós-otimização para a heurística GRASPf_RC nas instâncias proporcionais da classe SLM_P ($w = 10$).

Instância	F_O	F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
SLM_P1	——	9788	4268	5520	5520	0,16	9788
SLM_P2	——	11319	4569	6750	6750	0,19	11319
SLM_P3	——	11294	4654	6640	6640	0,30	11294

Tabela 6.40: Resultados da pós-otimização para a heurística VNSf_RC nas instâncias não proporcionais da classe SLM_P ($w = 10$).

Instância	F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
ORM_NP1	6447	5893	554	554	0,00	6447
ORM_NP2	5196	4128	1068	1068	0,00	5196
ORM_NP3	5235	4368	867	867	0,00	5235
ORM_NP4	4935	3458	1477	1477	0,01	4935
ORM_NP5	3377	1900	1477	1477	0,01	3377
ORM_NP6	8098	7824	274	274	0,01	8098
ORM_NP7	6153	5641	512	512	0,01	6153
ORM_NP8	5629	4591	**1038**	**1034**	0,02	**5625**
ORM_NP9	4364	3107	1257	1257	0,01	4364
ORM_NP10	3362	1999	1363	1363	0,01	3362
ORM_NP11	7832	7717	115	115	0,00	7832
ORM_NP12	6979	6658	321	321	0,02	6979
ORM_NP13	5342	4485	857	857	0,02	5342
ORM_NP14	4781	3432	**1349**	**1348**	0,04	**4780**
ORM_NP15	4120	2440	1680	1680	0,04	4120
ORM_NP16	8293	8172	**121**	**115**	0,02	**8287**
ORM_NP17	7246	7010	236	236	0,03	7246
ORM_NP18	5838	4962	**876**	**868**	0,04	**5830**
ORM_NP19	4546	3179	**1367**	**1366**	0,06	**4545**
ORM_NP20	4428	2650	1778	1778	0,05	4428
ORM_NP21	9238	9138	100	100	0,02	9238
ORM_NP22	8800	8583	**217**	**214**	0,03	**8797**
ORM_NP23	5629	4712	**917**	**916**	0,08	**5628**
ORM_NP24	4678	3364	1314	1314	0,09	4678
ORM_NP25	4538	2749	1789	1789	0,11	4538
ORM_NP26	9996	9917	79	79	0,03	9996
ORM_NP27	8514	8307	**207**	**205**	0,09	**8512**
ORM_NP28	5443	4650	**793**	**790**	0,10	**5440**
ORM_NP29	4801	3407	**1394**	**1393**	0,14	**4800**
ORM_NP30	4700	2898	1802	1802	0,15	4700
ORM_NP31	10158	10086	72	72	0,06	10158
ORM_NP32	9474	9297	177	177	0,10	9474
ORM_NP33	5723	4806	**917**	**915**	0,17	**5721**
ORM_NP34	4759	3446	1313	1313	0,18	4759
ORM_NP35	10463	10400	63	63	0,06	10463
ORM_NP36	10116	9934	**182**	**178**	0,14	**10112**
ORM_NP37	6190	5168	**1022**	**1014**	0,20	**6182**
ORM_NP38	11129	11060	69	69	0,11	11129
ORM_NP39	9551	9423	128	128	0,19	9551
ORM_NP40	6213	5293	**920**	**919**	0,67	**6212**

Tabela 6.41: Resultados da pós-otimização para a heurística GRASPf_RC nas instâncias não proporcionais da classe ORM_NP ($w = 2$).

Instância	F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
ORM_NP1	6447	5893	554	554	0,00	6447
ORM_NP2	5196	4128	1068	1068	0,00	5196
ORM_NP3	5235	4368	867	867	0,00	5235
ORM_NP4	4930	3381	1549	1549	0,01	4930
ORM_NP5	3377	1900	1477	1477	0,01	3377
ORM_NP6	8098	7824	274	274	0,01	8098
ORM_NP7	6153	5641	512	512	0,01	6153
ORM_NP8	5623	4533	1090	1090	0,02	5623
ORM_NP9	4364	3107	1257	1257	0,01	4364
ORM_NP10	3362	1999	1363	1363	0,01	3362
ORM_NP11	7832	7717	115	115	0,00	7832
ORM_NP12	6979	6658	321	321	0,02	6979
ORM_NP13	5342	4485	857	857	0,02	5342
ORM_NP14	4781	3432	**1349**	**1348**	0,04	**4780**
ORM_NP15	4120	2409	1711	1711	0,05	4120
ORM_NP16	8293	8172	**121**	**115**	0,02	**8287**
ORM_NP17	7246	7010	236	236	0,03	7246
ORM_NP18	5836	4948	**888**	**880**	0,04	**5828**
ORM_NP19	4546	3188	**1358**	**1357**	0,07	**4545**
ORM_NP20	4428	2650	1778	1778	0,07	4428
ORM_NP21	9238	9138	100	100	0,02	9238
ORM_NP22	8800	8583	**217**	**214**	0,03	**8797**
ORM_NP23	5628	4698	**930**	**929**	0,08	**5627**
ORM_NP24	4678	3371	1307	1307	0,08	4678
ORM_NP25	4538	2749	1789	1789	0,11	4538
ORM_NP26	9996	9917	79	79	0,03	9996
ORM_NP27	8514	8307	**207**	**205**	0,09	**8512**
ORM_NP28	5439	4634	**805**	**803**	0,17	**5437**
ORM_NP29	4801	3423	**1378**	**1377**	0,12	**4800**
ORM_NP30	4700	2898	1802	1802	0,15	4700
ORM_NP31	10158	10086	72	72	0,06	10158
ORM_NP32	9474	9297	177	177	0,10	9474
ORM_NP33	5718	4803	**915**	**913**	0,21	**5716**
ORM_NP34	4759	3465	**1294**	**1293**	0,18	**4758**
ORM_NP35	10463	10400	63	63	0,06	10463
ORM_NP36	10116	9934	**182**	**178**	0,14	**10112**
ORM_NP37	6190	5159	**1031**	**1023**	3,46	**6182**
ORM_NP38	11129	11060	69	69	0,11	11129
ORM_NP39	9551	9423	128	128	0,19	9551
ORM_NP40	6213	5295	**918**	**917**	0,64	**6212**

Tabela 6.42: Resultados da pós-otimização para a heurística VNSf_RC nas instâncias não proporcionais da classe ORM_NP ($w = 2$).

Instância	F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
GRM_NP1	6156	5703	**453**	**434**	0,03	**6137**
GRM_NP2	5185	4498	687	687	0,02	5185
GRM_NP3	4825	3986	839	839	0,03	4825
GRM_NP4	4720	3708	1012	1012	0,02	4720
GRM_NP5	4675	3418	1257	1257	0,01	4675
GRM_NP6	4675	3161	1514	1514	0,02	4675
GRM_NP7	4685	2915	1770	1770	0,03	4685
GRM_NP8	4717	2688	2029	2029	0,02	4717
GRM_NP9	4806	2233	2573	2573	0,03	4806
GRM_NP10	11334	10841	**493**	**477**	0,09	**11318**
GRM_NP11	9710	8752	958	958	0,12	9710
GRM_NP12	8751	7640	1111	1111	0,11	8751
GRM_NP13	8135	6801	1334	1334	0,09	8135
GRM_NP14	7719	6181	1538	1538	0,06	7719
GRM_NP15	7527	5748	1779	1779	0,10	7527
GRM_NP16	7427	5413	2014	2014	0,06	7427
GRM_NP17	7362	5123	2239	2239	0,08	7362
GRM_NP18	7341	4865	2476	2476	0,08	7341
GRM_NP19	7346	4632	2714	2714	0,07	7346
GRM_NP20	7405	4123	3282	3282	0,08	7405

Tabela 6.43: Resultados da pós-otimização para a heurística GRASPf_RC nas instâncias não proporcionais da classe GRM_NP ($w = 2$).

Instância	F_V^*	F_P	F_T	F_{OT}	Tempo (s)	F_{PO}
GRM_NP1	6156	5703	**453**	**434**	0,03	**6137**
GRM_NP2	5185	4498	687	687	0,02	5185
GRM_NP3	4825	3986	839	839	0,03	4825
GRM_NP4	4720	3708	1012	1012	0,02	4720
GRM_NP5	4675	3418	1257	1257	0,01	4675
GRM_NP6	4675	3161	1514	1514	0,02	4675
GRM_NP7	4685	2915	1770	1770	0,03	4685
GRM_NP8	4717	2688	2029	2029	0,02	4717
GRM_NP9	4806	2233	2573	2573	0,03	4806
GRM_NP10	11334	10841	**493**	**477**	0,09	**11318**
GRM_NP11	9710	8752	958	958	0,12	9710
GRM_NP12	8759	7613	1146	1146	0,07	8759
GRM_NP13	8135	6801	1334	1334	0,09	8135
GRM_NP14	7719	6181	1538	1538	0,06	7719
GRM_NP15	7527	5748	1779	1779	0,10	7527
GRM_NP16	7431	5416	2015	72015	0,08	7431
GRM_NP17	7360	5126	2234	2234	0,06	7360
GRM_NP18	7341	4865	2476	2476	0,08	7341
GRM_NP19	7346	4632	2714	2714	0,07	7346
GRM_NP20	7405	4123	3282	3282	0,08	7405

Tabela 6.44: Resultados da pós-otimização para a heurística VNSf_RC nas instâncias não proporcionais da classe GRM_NP ($w = 2$).

Já o algoritmo VNSf_RC apresentou resultados melhores do que GRASPf_RC nas instâncias proporcionais com $w = 2$ e nas instâncias não proporcionais ORM_NP.

Por último, 49 soluções ótimas para problemas de pequeno porte foram obtidas e os resultados encontrados pelas heurísticas no tempo tg foram comparados com essas instâncias. Para os problemas que não possuem o ótimo, a melhor solução obtida pelos algoritmos no tempo tg foi a melhor solução encontrada. Levando-se em consideração os casos de empate no primeiro lugar, dos 249 problemas, GRASPf_RC apresentou a melhor solução em 234 instâncias e VNSf_RC apresentou a melhor solução em 236 instâncias. Em relação aos problemas com o ótimo conhecido, as heurísticas apresentaram os mesmos resultados, encontrando a solução ótima em 42 problemas, ficando, no máximo, a 0,54% do ótimo nas sete instâncias onde o mesmo não foi alcançado. Investigações adicionais mostraram que, em algumas instâncias, as facilidades abertas encontradas pelo algoritmo exato coincidem com as facilidades abertas encontradas pelas heurísticas. Assim, a árvore de Steiner obtida pelos algoritmos aproximados não é ótima, o que sugeriu um passo adicional de pós-otimização.

O passo de pós-otimização consistiu na execução de um algoritmo ótimo para o problema de Steiner, tendo como vértices teminais as facilidades abertas da melhor solução encontrada pelo GRASPf_RC ou pelo VNSf_RC após todas as iterações terem sido realizadas nas cinco execuções no tempo tg. O algoritmo exato encontrou a árvore de Steiner ótima da melhor solução encontrada pelas heurísticas em menos de um segundo para todas as instâncias testadas. Os resultados da pós-otimização mostraram a eficiência da heurística Prim que obteve a grande maioria das solução ótimas. Acrescentando o passo de pós-otimização, GRASPf_RC e VNSf_RC encontraram duas novas soluções ótimas, totalizando 44 problemas e, ficando, no máximo a 0,48% do valor ótimo nas cinco instâncias onde o mesmo não foi alcançado. Nas instâncias em que não se conhece o ótimo, GRASP e VNS, acrescidas da pós-otimização, conseguiram melhorar a melhor solução encontrada pelos algoritmos sem o procedimento em 29 e 28 instâncias, respectivamente. Levando-se em consideração os casos de empate no primeiro lugar e incluindo a pós-otimização, dos 249 problemas, GRASPf_RC apresentou a melhor solução em 239 instâncias e VNSf_RC apresentou a melhor solução em 237 instâncias.

7
Conclusões e Trabalhos Futuros

Esta tese apresentou heurísticas para o problema das p-medianas conectadas. O estudo de modelos que unem as áreas de localização de facilidades e de projeto de redes é um tema recente de pesquisa, como observado na revisão bibliográfica.

Duas formulações de programação linear inteira foram apresentadas para o problema: um modelo baseado em fluxos em redes com várias comodidades e um modelo baseado em árvore geradora mínima restrita por grau. Comparações entre as duas formulações foram realizadas em 168 instâncias testes utilizando-se um resolvedor de programação inteira. As instâncias testadas são pequenas em termos de números de vértices e quantidade de facilidades a serem instaladas. Ambos os modelos encontraram a solução ótima para os 168 problemas, porém, o modelo por árvore resolveu à otimalidade as instâncias em um tempo de processamento bem menor do que o modelo por fluxos. Generalizando-se o problema das p-medianas conectadas, definiu-se também o problema de localização de facilidades não-capacitadas conectadas e um modelo baseado em árvore geradora mínima restrita por grau para o mesmo.

Uma estratégia de busca local híbrida foi proposta para o problema das p-medianas conectadas. Primeiramente executou-se uma busca local rápida, de qualidade inferior e, logo em seguida, uma busca local lenta, de qualidade superior. A estratégia de melhoria iterativa e a técnica de circularidade foram utilizadas com o objetivo de acelerar as iterações da busca local. Ganhos consideráveis foram obtidos em termos de tempo de processamento, incorporando-se um simples teste para evitar a execução da heurística construtiva Prim que possui um custo computacional muito alto.

Duas heurísticas também foram propostas e comparadas: um algoritmo GRASP e VNS acrescidos de uma estratégia de filtro e do procedimento de reconexão por caminhos. Ambos utilizaram a mesma heurística construtiva para gerar uma solução inicial e a mesma estratégia de busca local híbrida. Com o objetivo de tentar diminuir os elevados tempos de processamento apresentados pela busca local nas maiores instâncias, uma estratégia de filtro foi utilizada em ambos os algoritmos entre a busca local mais rápida e a busca local mais lenta. Testes computacionais mostraram que o filtro reduz o tempo de processamento às custas de perdas na qualidade média das soluções. Com

o objetivo de tentar melhorar a qualidade das soluções, utilizou-se reconexão por caminhos como estratégia de intensificação após a execução da busca local em ambas as heurísticas. Gráficos de distribuição de probabalidade mostraram a efetividade do procedimento tanto no GRASP quanto no VNS.

As comparações entre as heurísticas foram realizadas através de três testes diferentes: utilizando-se tempos distintos de execução, gráficos do tempo gasto para atingir o valor alvo e através de soluções ótimas encontradas pela formulação por árvore em 49 instâncias proporcionais. Em geral, nos três testes, os resultados mostraram uma pequena superioridade do GRASP em relação ao VNS. As heurísticas obtiveram a solução ótima em 42 problemas e uma análise minuciosa revelou a necessidade de incorporar um passo adicional em ambas as heurísticas. A adição do passo de pós-otimização se mostrou efetiva pois possibilitou as heurísticas encontrarem duas novas soluções ótimas e melhorarem a melhor solução conhecida em diversas instâncias.

A busca local concatenada obteve ganhos significativos nos tempos de processamento em relação à versão básica nas instâncias testadas. Porém, para os maiores problemas, a estratégia híbrida apresentou tempos de processamento elevados. Duas possíveis extensões poderiam ser estudadas com o objetivo de diminuir os altos tempos de processamento apresentados pelas heurísticas nas maiores instâncias: incorporar estimativas para o problema de Steiner na vizinhança dos algoritmos de busca local estudados, como apresentado em [51], com a finalidade de restringir ainda mais os vizinhos a serem avalidos, evitando-se, assim, a execução da heurística Prim a cada nova troca entre uma facilidade que não está na solução com uma facilidade que faz parte da solução. A outra vertente seria o desenvolvimento de implementações paralelas em *clusters* das heurísticas apresentadas nessa tese.

A formulação por árvore para o problema das p-medianas conectadas possibilitou a obtenção de soluções ótimas somente para as instâncias pequenas. Uma possível extensão a ser investigada é melhorar e integrar as formulações propostas para o problema em algoritmos do tipo *branch and bound* e *branch and price*, de modo a resolver instâncias de maior porte. Outra alternativa interessante é estudar a utilização da relaxação lagrangeana para a obtenção de limites inferiores para o problema ou mesmo para a construção de heurísticas lagrangeanas.

Finalmente, pode-se desenvolver algoritmos aproximados para o problema de localização de facilidades não-capacitadas conectadas, além de estudar extensões e variantes de ambos os modelos. Nos dois problemas apresentados na tese, as facilidades podem servir uma quantidade ilimitada de demanda. Uma possível variante introduziria limites na quantidade de demanda que as facilidades podem servir, isto é, as facilidades seriam consideradas capacitadas.

Referências Bibliográficas

[1] AIEX, R. M.; RESENDE, M. G. C. ; RIBEIRO, C. C.. **Probability Distribution of Solution Time in GRASP: An Experimental Investigation**. Journal of Heuristics, 8:343–373, 2002.

[2] ALOISE, D. J.; ALOISE, D.; ROCHA, C. T. M.; FILHO, J. C. R.; MOURA, L. S. S. ; RIBEIRO, C. C.. **Scheduling Workover Rigs for Onshore Oil Production**. Discrete Applied Mathematics, 154:695–702, 2006.

[3] ANDREWS, M.; ZHANG, L.. **The Access Network Design Problem**. Em: PROCEEDINGS OF THE 39TH ANNUAL IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, p. 40–49, Palo Alto, 1998.

[4] BANDELT, H. J.; LABBÉ, M.. **How Bad Can A Voting Locating Be?** Social Choice and Welfare, 3:125–145, 1986.

[5] BEASLEY, J. E.. **A Note on Solving Large p-Median Problems**. European Journal of Operational Research, 21:270–273, 1985.

[6] BEASLEY, J. E.. **An SST-Based Algorithm for the Steiner Problem in Graphs**. Networks, 19:1–16, 1989.

[7] BERMAN, O.; INGCO, D. I. ; ODONI, A. R.. **Improving the Location of Minisum Facilities Through Network Modification**. Annals of Operations Research, 40:1–16, 1992.

[8] BHADURY, J.; CHANDRASEKHARAN, R. ; GEWALI, L.. **Computational Complexity of Integrated Models of Network Design and Facility Location**. Southwest Journal of Pure and Applied Mathematics, 1:30–43, 2000.

[9] CAMPBELL, J. F.. **A Survey of Network Hub Location**. Studies in Locational Analysis, 6:31–49, 1994.

[10] CANUTO, S. A.; RESENDE, M. G. C. ; RIBEIRO, C. C.. **Local Search with Perturbations for the Prize-Collecting Steiner Tree Problem in Graphs**. Networks, 38:50–58, 2001.

[11] CAPOROSSI, G.; HANSEN, P.. **Variable Neighbourhood Search for Extremal Graphs: 1 The AutoGraphiX system**. Discrete Mathematics, 212:29–44, 2000.

[12] CORNUÉJOLS, G.; NEMHAUSER, G. L. ; WOLSEY, L. A.. **The Uncapacitated Facility Location Problem**. Em: Mirchandani, P. B.; Francis, R. L., editores, DISCRETE LOCATION THEORY, p. 119–171. Wiley-Interscience, 1990.

[13] CURRENT, J.; MIN, H. ; SCHILLING, D.. **Multiobjective Analysis of Facility Location Decisions**. European Journal of Operational Research, 49:295–307, 1990.

[14] CURRENT, J.; DASKIN, M. S. ; SCHILLING, D.. **Discrete Network Location Models**. Em: Drezner, Z.; Hamacher, H., editores, FACILITY LOCATION THEORY: APPLICATIONS AND METHODS, p. 81–118. Springer-Verlag, 2002.

[15] DIJKSTRA, E. W.. **A Note on Two Problems in Connexion With Graphs**. Numerische Mathematik, 1:269–271, 1959.

[16] DREZNER, Z.; WESOLOWSKY, G. O.. **Network Design: Selection and Design of Links and Facility Location**. Transportation Research Part A, 37:241–256, 2003.

[17] FEO, T. A.; RESENDE, M. G. C.. **A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem**. Operations Research Letters, 8:67–71, 1989.

[18] GALVÃO, R. D.; REVELLE, C. S.. **A Lagrangean Heuristic for the Maximal Covering Problem**. European Journal of Operational Research, 18:114–123, 1996.

[19] GLOVER, F.. **Tabu Search and Adaptive Memory Programming - Advances, Applications and Challenges**. Em: Barr, R. S.; Helgason, R. V. ; Kennington, J. L., editores, INTERFACES IN COMPUTER SCIENCE AND OPERATIONS RESEARCH, p. 1–75. Kluwer Academic Publishers, 1996.

[20] GUHA, S.; KHULLER, S.. **Connected Facility Location Problems**. Em: Pardalos, P. M.; Du, D., editores, NETWORK DESIGN: CONNECTIVITY AND FACILITIES LOCATION, p. 179–190. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1997.

[21] GUPTA, A.; KLEINBERG, J.; KUMAR, A.; RASTOGI, R. ; YENER, B.. **Provisioning a Virtual Private Network: A Network Design**

Problem for Multicommodity Flow. Em: PROCEEDINGS OF THE 33RD ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, p. 389–398, Hersonissos, 2001.

[22] GUPTA, A.; KUMAR, A. ; ROUGHGARDEN, T.. **A Constant-Factor Approximation Algorithm for Multicommodity Rent-or-Buy Problem**. Em: PROCEEDINGS OF THE 43RD ANNUAL IEEE SYMPOSIUM ON FOUNDATIONS OF COMPUTER SCIENCE, p. 333–342, Washington, 2002.

[23] GUPTA, A.; KUMAR, A. ; ROUGHGARDEN, T.. **Simpler and Better Approximation Algorithms for Network Design**. Em: PROCEEDINGS OF THE 35TH ANNUAL ACM SYMPOSIUM ON THEORY OF COMPUTING, p. 365–372, San Diego, 2003.

[24] HAKIMI, S. L.. **Optimum Locations of Switching Centers and Absolute Centers and Medians of a Graph**. Operations Research, 12:450–459, 1964.

[25] HAKIMI, S. L.. **Optimum Distribution of Switching Centers in a Communication Network and Some Related Graph-Theoretic Problems**. Operations Research, 13:462–475, 1965.

[26] HANSEN, P.; MLADENOVIĆ, N.. **Variable Neighbourhood Search for the P-Median**. Location Science, 5:207–226, 1997.

[27] HANSEN, P.; MLADENOVIĆ, N.. **Variable Neighbourhood Search: Principles and Applications**. European Journal of Operational Research, 130:449–467, 2001.

[28] KARP, R.. **Reducibility Among Combinatorial Problems**. Em: Miller, R.; Thatcher, J., editores, COMPLEXITY OF COMPUTER COMPUTATIONS, p. 85–103. Plenum Press, 1972.

[29] KARIV, O.; HAKIMI, S. L.. **An Algorithmic Approach to Network Location Problems. II: The p-Medians**. SIAM Journal on Applied Mathematics, 37:539–560, 1979.

[30] KHULLER, S.; ZHU, A.. **The General Steiner Tree-Star Problem**. Information Processing Letters, 84:215–220, 2002.

[31] KUEHN, A. A.; HAMBURGER, M. J.. **A Heuristic Program for Locating Warehouses**. Management Science, 9:643–666, 1963.

[32] LABBÉ, M.; LAPORTE, G.; MARTÍN, I. R. ; GONZÁLEZ, J. J. S.. **The Median Cycle Problem**. Relatório técnico, Université Libre de Bruxelles, Bruxelas, 2001.

[33] LAGUNA, M.; MARTÍ, R.. **GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization**. INFORMS Journal on Computing, 11:44–52, 1999.

[34] MAGNANTI, T. L.; WONG, R. T.. **Network Design and Transportation Planning: Models and Algorithms**. Transportation Science, 18:1–55, 1984.

[35] MARTINS, S. L.; PARDALOS, P. M.; RESENDE, M. G. C. ; RIBEIRO, C. C.. **A Parallel GRASP for the Steiner Tree Problem in Graphs Using a Hybrid Local Search Strategy**. Journal of Global Optimization, 17:267–283, 2000.

[36] MELKOTE, S.; DASKIN, M. S.. **An Integrated Model of Facility Location and Transportation Network Design**. Transportation Research Part A, 35:515–538, 2001.

[37] MELKOTE, S.; DASKIN, M. S.. **Capacited Facility Location Network Design Problems**. European Journal of Operational Research, 129:481–495, 2001.

[38] MILLER, C. E.; TUCKER, A. W. ; ZEMLIN, R. A.. **Integer Programming Formulation of Traveling Salesman Problems**. Journal of ACM, 7:326–329, 1960.

[39] MIN, H.; JAYARAMAN, V. ; SRIVASTAVA, R.. **Combined Location-Routing Problems: A Synthesis and Future Research Directions**. European Journal of Operational Research, 108:1–15, 1998.

[40] MLADENOVIĆ, N.; HANSEN, P.. **Variable Neighbourhood Search**. Computers and Operations Research, 24:1097–1100, 1997.

[41] OWEN, S. H.; DASKIN, M. S.. **Strategic Facility Location: A Review**. European Journal of Operational Research, 111:423–447, 1998.

[42] PEETERS, D.; THOMAS, I.. **The Effect of Spatial Structure on p-Median Results**. Transportation Science, 29:366–373, 1995.

[43] POLZIN, T.; DANESHMAND, S. V.. **A Comparison of Steiner Tree Relaxations**. Discrete Applied Mathematics, 112:241–261, 2001.

[44] PRAIS, M.; RIBEIRO, C. C.. **Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment**. INFORMS Journal on Computing, 12:164–176, 2000.

[45] RAVI, R.; SINHÁ, A.. **Approximation Algorithms for Problems Combining Facility Location and Network Design**. Operations Research, 54:73–81, 2006.

[46] RESENDE, M. G. C.; RIBEIRO, C. C.. **Greedy Randomized Adaptive Search Procedures (GRASP)**. Em: Glover, F.; Kochenberger, G., editores, HANDBOOK OF METAHEURISTICS, p. 219–249. Kluwer Academic Publishers, 2002.

[47] RESENDE, M. G. C.; WERNECK, R. F.. **On the Implementation of a Swap-Based Local Search Procedure for the p-Median Problem**. Em: PROCEEDINGS OF THE FIFTH WORKSHOP ON ALGORITHM ENGINEERING AND EXPERIMENTS, p. 119–127, Baltimore, 2003.

[48] RESENDE, M. G. C.; WERNECK, R. F.. **A Hybrid Heuristic for the p-Median Problem**. Journal of Heuristics, 10:59–88, 2004.

[49] RESENDE, M. G. C.; RIBEIRO, C. C.. **GRASP with Path Relinking: Recent Advances and Applications**. Em: Ibaraki, T.; Nonobe, K. ; Yagiura, M., editores, METAHEURISTICS: PROGRESS AS REAL PROBLEM SOLVERS, p. 29–63. Kluwer Academic Publishers, 2005.

[50] REVELLE, C.; SWAIN, R.. **Central Facilities Location**. Geographical Analysis, 2:30–42, 1970.

[51] RIBEIRO, C. C.; SOUZA, M. C.. **Tabu Search for the Steiner Problem in Graphs**. Networks, 36:138–146, 2000.

[52] RIBEIRO, C. C.; UCHOA, E. ; WERNECK, R. F.. **A Hybrid GRASP with Perturbations for the Steiner Problem in Graphs**. INFORMS Journal on Computing, 14:228–246, 2002.

[53] RIBEIRO, C. C.; SOUZA, R. C. ; VIEIRA, C. E. C.. **A Comparative Computational Study of Random Number Generators**. Pacific Journal of Optimization, 1:565–578, 2005.

[54] ROLLA, S. R.. **Algoritmo para o Problema de Planejamento de Redes com Localização de Facilidades Capacitadas**. Dissertação de Mestrado, Universidade Federal de Minas Gerais, Departamento de Ciência da Computação, 1999.

[55] SENNE, E. L. F.; LORENA, L. A. N.. **Langrangean/Surrogate Heuristics for p-Median Problems**. Em: Laguna, M.; González-Velarde, J. L., editores, COMPUTING TOOLS FOR MODELING, OPTIMIZATION AND SIMULATION: INTERFACES IN COMPUTER

SCIENCE AND OPERATIONS RESEARCH, p. 115–130. Kluwer Academic Publishers, 2000.

[56] SWAMY, C.; KUMAR, A.. **Primal-Dual Algorithms for Connected Facility Location Problems**. Algorithmica, 40:245–269, 2004.

[57] TAKAHASHI, H.; MATSUYAMA, A.. **An Approximate Solution for the Steiner Problem in Graphs**. Mathematica Japonica, 24:573–577, 1980.

[58] TEITZ, M. B.; BART, P.. **Heuristic Methods for Estimating the Generalized Vertex Median of a Weighted Graph**. Operations Research, 16:955–961, 1968.

[59] TOREGAS, C.; SWAIN, R.; REVELLE, C. ; BERGMAN, L.. **The Location of Emergency Service Facilities**. Operations Research, 19:1363–1373, 1971.

[60] VOβ, S.. **Steiner's Problem in Graphs: Heuristics Methods**. Discrete Applied Mathematics, 40:45–72, 1992.

[61] WERNECK, R. F.. **Problema de Steiner em Grafos: Algoritmos Primais, Duais e Exatos**. Dissertação de Mestrado, Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2001.

[62] WHITAKER, R.. **A Fast Algorithm for the Greedy Interchange of Large-Scale Clustering and Median Location Problems**. Information Systems and Operational Research, 21:95–108, 1983.

[63] WINTER, P.. **Steiner Problems in Networks: A Survey**. Networks, 40:45–72, 1987.

[64] WONG, R. T.. **A Dual Ascent Approach for Steiner Tree Problems on a Directed Graph**. Mathematical Programming, 28:271–287, 1984.