Otimização Aplicada ao Problema de Circuito Virtual Privado em Redes de Telecomunicações

Patricia Regina de Abreu Lopes

UFRJ

Rio de Janeiro

2010

OTIMIZAZAÇÃO APLICADA AO PROBLEMA DE CIRCUITO VIRTUAL PRIVADO EM REDES DE TELECOMUNICAÇÕES

Patricia Regina de Abreu Lopes

TESE SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE DOUTOR EM CIÊNCIAS EM ENGENHARIA DE SISTEMA E COMPUTAÇÃO.

Examinada por:

Profa. Marcia Helena Costa Fampa, D.Sc.

Prof. Paulo Roberto Oliveira, D.Sc.

Profa. Fernanda Maria Pereira Raupp, D.Sc.

Profa. Luziane Ferreira de Mendonça, D.Sc.

Profa. Nair Maria Maia de Abreu, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

JUNHO DE 2010

Lopes, Patricia Regina de Abreu

Otimização Aplicada ao Problema de Circuito Virtual Privado em Redes de Telecomunicações/ Patricia Regina de Abreu Lopes. - Rio de Janeiro: UFRJ/COPPE, 2010.

XIII, 85 p.: il.; 29,7 cm.

Orientador: Marcia Helena Costa Fampa

Tese (doutorado) - UFRJ/ COPPE/ Programa de Engenharia de Sistema e Computação, 2010.

Referencias Bibliográficas: p. 79-85.

1. Metaheurística GRASP. 2. Métodos Híbridos. 3. Metaheurística ILS. I. Fampa, Marcia Helena Costa II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistema e Computação. III. Título.

A Deus, a minha mãe, meus irmão e ao meu pai, Raimundo Lopes
(*in memorian)*

Agradecimentos

A Deus, pelo término de mais uma etapa da minha vida.

A minha família por todo apoio ao longo da minha vida e do curso de doutorado.

Ao Henrique José Gomes Pereira pelo apoio e compreensão nos momentos difíceis.

A professora e orientadora Marcia Helena Costa Fampa, meu eterno agradecimento pelo seu apoio, carinho, sua amizade, por seu exemplo de vida, aos incentivos nessa longa caminhada, a colaboração para que meu sonho fosse realizado, pelos conhecimentos transmitidos durante o desenvolvimento deste trabalho e durante o curso.

A Nair Maria Maia de Abreu, Luziane Ferreira de Mendonça, Fernanda Maria Pereira Raupp e Paulo Roberto Oliveira pelo carinho, atenção, e por participarem da banca.

Ao professor Nelson Maculan Filho por acreditar no meu potencial, por sua dedicação como professor, por seu apoio e a sua participação na banca de qualificação, meu agradecimento.

Aos professores que ministraram alguma matéria ao longo do meu curso de doutorado contribuindo assim para minha formação.

Ao Mauricio Resende pela colaboração.

A minha irmazinha de coração e orientadora para assuntos aleatórios, Denise Candal por seu apoio nos momentos de crise existencial, por seu carinho, pelas inúmeras horas de risadas e troca de conhecimentos.

Ao Wendel Alexandre Xavier de Melo por sua amizade e pelas inúmeras horas de programação.

Ao meu amigo Alexandre Mazolli, por seu apoio, por suas dicas e ajudas sobre programação.

Aos meus amigos Ana Lucia Sousa, Flavio Signorelli Mendes, Marcelo Signorelli Mendes, Daniel Portinha, Júlio Tadeu Carvalho da Silveira, Lillian Silveira e os amigos de trabalho pelo carinho e apoio.

A todas as pessoas que me ajudaram e ao amigos que fiz ao longo deste caminho.

Ao Jesus Ossian Cunha por manter o Laboratório de otimização funcionando para que eu pudesse trabalhar e colaboração em todos os momentos que precisei.

Aos amigos da secretaria, por toda a sua dedicação e carinho.

A todos aqueles que de uma forma ou de outra me apoiaram o meu muito obrigada.

″O homem nao teria alcancado o possível se inúmeras vezes não tivesse tentado atingir o impossível ″

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Doutor em Ciência (D.Sc.).

OTIMIZAÇÃO APLICADA AO PROBLEMA DE CIRCUITO VIRTUAL PRIVADO EM REDE DE TELECOMUNICAÇÕES

Patricia Regina de Abreu Lopes

Junho/2010

Orientadora : Marcia Helena Costa Fampa

Programa: Engenharia de Sistemas e Computação

São inúmeros os problemas de otimização combinatória no setor de telecomunicações. Com o desenvolvimento de novas técnicas, novos equipamentos e o aumento da busca dos serviços surge a necessidade de otimizar a nova rede. O grande desafio, porém, é a maneira de gerenciar o crescimento acelerado da demanda, mantendo ou mesmo melhorando a qualidade e a confiabilidade dos serviços. Se por um lado os investimentos em infra-estrutura são consideráveis, por outro busca-se aproveitar ao máximo os recursos existentes, visando uma adequada qualidade de serviço ao mesmo tempo em que se tenta adiar novos investimentos.

Este trabalho considera um importante problema que surge nesta área: o Problema de Circuito Virtual Privado (PCVP), que busca minimizar o custo de roteamento de pacotes de dados numa rede de telecomunicações considerando os limites de capacidade de transmissão nesta rede. São propostos inicialmente métodos computacionais de solução para PCVP, baseados nas meta-heurísticas GRASP, VND e ILS. Finalmente, é proposto um método híbrido que utiliza tanto as metodologias heurísticas propostas anteriormente quanto métodos exatos que consideram um modelo de programação inteira apresentado para o problema. Resultados computacionais comparam os diversos métodos propostos.

Abstract of Thesis presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Doctor of Science (D.Sc.).

OPTIMIZATION APPLIED TO THE PRIVATE VIRTUAL CIRCUIT PROBLEM IN TELECOMMUNICATIONS NETWORKS

Patricia Regina de Abreu Lopes

June/2010

Advisor: Marcia Helena Costa Fampa

Department: Computing Systems Engineering

There are many combinatorial optimization problems in the telecommunications area. With the development of new techniques, new equipments and the increase on demand, the necessity to optimize the new telecommunication network appears. The big challenge, however, is how to manage the growth on the demand, keeping or even improving the quality and the confiability of the services. If on one hand the investments in infrastructure are considerable, on the other hand people try to take the maximum advantage of the existing resources, aiming at a proper quality for the service at the same time that they try to postpone new investments.

This work considers an important problem that appears in this area: the Problem of Private Virtual Circuit (PPVC), which aims to minimize the routing cost of data packages in a telecommunication network considering the transmission capacity limits of the network. We initially propose computational methods to solve the PPVC, based on the metaheuristics GRASP, VND and ILS. Finally, we propose a hybrid method, which uses the heuristics proposed before as well as an exact method, that considers a integer programming model presented for the problem. Computational results compare the methods proposed.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas

Abrev	**Descrição**
LCR	Lista de Candidatos Restrita
BL	Busca Local
GRASP	Greedy Randomized Adaptive Search Procedure
ILS	Iterated Local Search
VND	Variable Neighborhood Descent
PCVP	Problema de Circuito Virtual Privado
LB	Local Branching

Capítulo 1

Introdução

Com o passar dos anos os serviços de telecomunicações sofreram uma considerável redução de preços, tendo como conseqüência imediata a popularização dos mesmos. Hoje, falar à distância ou mesmo utilizar os benefícios da tecnologia de telecomunição é algo muito presente no dia-a-dia, até mesmo para uma criança.

O instituto de pesquisas International Data Corporation (IDC) previu em 2002 que o volume de tráfego gerado pelos usuários domésticos de todo o mundo iria dobrar anualmente nos cinco anos seguintes, tendo partido de 180 petabits por dia. De acordo com o estudo, no ano de 2009, mais de 3 milhões de novas conexões em banda larga foram comercializadas só no Brasil, somando um total de 15,06 milhões de acessos, número que aponta um crescimento de 26,4% ante a base instalada no primeiro trimestre do mesmo ano. A tecnologia de maior destaque foi, sem dúvida, a banda larga móvel, somando mais de 1,6 milhões de acessos e superando as expectativas durante o ano, já que o crescimento foi de 82% em relação a 2008, conforme declarou Samuel Rodrigues, analista do mercado de Telecom da IDC Brasil [67].

Estes benefícios terminam sempre por apresentar novos problemas, visto que o aumento da disponibilidade de recursos com baixos custos geram um crescimento na demanda, além de novas formas de utilização que surgem com o avanço da tecnologia, sendo necessária a redefinição da topologia dessas redes. Assim, ao mesmo tempo em que se tornam mais eficientes e com preços acessíveis, os meios de telecomunicações também aumentam em capacidade e complexidade. O grande desafio, porém, é a maneira de

gerenciar o crescimento acelerado da demanda, mantendo ou mesmo melhorando a qualidade e a confiabilidade dos serviços.

Grandes provedores de serviços de Internet são responsáveis por rotear o tráfego na rede, tarefa realizada por meio de protocolos que encaminham pacotes de dados desde a sua origem até o seu destino. Um dos protocolos mais utilizados mundialmente é o *Open Shortest Path First* (OSPF) que roteia o tráfego usando trajetórias de pesos mínimos calculadas sobre a rede provedora, como pode ser observado em [13]. A escolha desses caminhos é uma decisão que deve ser tomada pelos provedores que, por sua vez, buscam satisfazer critérios como minimizar a possibilidade de congestionamento ou a ociosidade da rede.

Se por um lado os investimentos em infraestrutura são grandes nesta área, por outro busca-se aproveitar ao máximo os recursos existentes, visando uma adequada qualidade de serviço ao mesmo tempo em que se tenta adiar novos investimentos.

O Problem de Circuito Virtual Privado (PCVP) está relacionado exatamente a esta questão. Nele, a rede é considerada dada, assim como a capacidade de transmissão dos seus arcos. O objetivo é então transmitir um conjunto de pacotes de dados pela rede, de determinadas origens para destinos, utilizando os recursos já instalados na rede, de forma a otimizar uma certa função objetivo.

O PCVP é, na verdade, uma importante variante de um problema clássico de otimização combinatória, o problema de multifluxo. Os problemas de multifluxo, por sua vez, são problemas de fluxo em redes, que constituem uma subclasse particularmente rica de problemas de otimização e possuem aplicações nas mais diversas áreas. Problemas como distribuição de água ou gás, de planejamento de redes telefônicas e elétricas, de geração de energia hidroelétrica, de controle de tráfego aéreo, de roteamento de dados, alguns problemas de finanças, física, química, logística, etc., podem ser formulados como problemas de otimização de fluxo em redes. Há diversos problemas de fluxo em rede, onde são considerados um ou mais produtos distintos. Dada uma rede, os problemas de fluxo simples (um único produto circulando na rede) consistem em achar o fluxo em cada arco de maneira a enviar o produto de um conjunto de uma ou mais origens até um conjunto de um ou mais destinos, otimizando algum critério, por exemplo minimizar os custos ou

maximizar a quantidade de fluxo enviada. Os problemas de multifluxo são uma generalização dos problemas de fluxo simples e levam em conta K produtos distintos, cada produto tem um conjunto de uma ou mais origens e um conjunto de um ou mais destinos, que compartilham dos mesmos recursos da rede, ou seja, cada um dos produtos compete por um recurso escasso que é a capacidade dos arcos da rede. Há diversos problemas de fluxos simples, como o fluxo a custo mínimo, fluxo máximo e outros, que possuem algoritmos combinatórios polinomiais capazes de gerar a solução ótima inteira (Cook et. al [17] e Ahuja et al. [1]). Por outro lado, o problema de multifluxo, quando há a restrição de integralidade dos fluxos, se torna NP-difícil. (Even et al. [22] mostra que o problema de decisão com apenas dois fluxos é NP-completo). Problemas de multifluxo, há várias décadas, têm recebido grande interesse por parte de pesquisadores, tais como [44, 47, 60, 64]. Atualmente, conferências regulares discutem o tema no meio científico.

Quando consideramos aplicações reais, os problemas de fluxo em redes fazem parte, em geral, da classe de problemas de grande porte e por esta razão são bastante complexos. A dificuldade na resolução dos problemas aumenta à medida que o fluxo nas redes cresce. Com o crescimento, nos últimos anos, das redes de telecomunicações, por exemplo, o problema de fluxo em redes tornou-se bastante importante e isso fez com que aumentasse o estudo dedicado ao desenvolvimento de algoritmos para resolvê-lo.

Na busca pela melhor solução destes problemas, que surge com a necessidade dos usuários, é essencial a utilização de técnicas eficientes para resolvê-los, com tempo razoável.

Para grande parte dos problemas de multifluxo de grande porte torna-se inviável o uso de algoritmos exatos, nos quais a convergência para a solução ótima do problema é demonstrada. Neste caso, métodos heurísticos são usualmente empregados. Esses métodos frequentemente obtém uma solção de boa qualidade ou mesmo a solução ótima, porém geralmente não é dada nenhuma garantia de convergência.

Em [54], por exemplo, é apresentada uma formulação matemática para o PCVP que utiliza uma função objetivo que examina a propagação do atraso e o congestionamento da rede. O problema é modelado com um problema de multifluxo inteiro com função objetivo definida por partes, e os autores aplicam a meta-heurística GRASP com a utilização

do religamento de caminhos na resolução do problema. São aplicadas quatro variações do GRASP com religamento de caminhos para encontrar soluções aproximadas para o problema. As heuristicas apresentadas são variações de um algoritmo guloso e de sua versão melhorada usando a busca local. Os resultados experimentais mostraram que mesmo a heurística gulosa mais simples pode ser melhor que a heurística usada na engenharia de tráfego e que o GRASP com religamento de caminhos mostrou-se bem eficiente. Os autores ressaltam, entretanto, a possibilidade de haver parada na fase de construção do GRASP. Este problema é agravado pela ordem da escolha, totalmente aleatória, do elemento a ser inserido na solução em construção, como detalharemos adiante nesta tese.

Conforme mencionado anteriormente, a escolha dos caminhos a serem utilizados para rotear os pacotes de dados na rede é uma decisão que deve ser tomada pelos provedores que buscam minimizar o custo do roteamento. O protocolo OSPF trata esta questão por meio de uma adequada designação de pesos aos arcos da rede, o que influencia diretamente na determinação dos caminhos mínimos a serem usados. Outro problema encontrado na literatura é, portanto, o problema de atribuição de pesos aos arcos da rede de tecomunicações de forma a poder encontrar o(s) caminho(s) mínimo(s) para ser(em) utilizado(s) pelo protocolo OSPF, este problema é conhecido como *Weight Setting Problem - WSP*, [13, 14, 21, 28, 29]. O WSP é um problema de otimização combinatória não linear e prova-se que é NP-difícil [28].

Em [13], por exemplo, considera-se que os roteadores são os nós de um grafo direcionado e que as ligações entre roteadores constituem o conjunto dos arcos, cada arco possui uma capacidade associada. Também, é dada uma matriz de demandas entre os nós origem e destino. O problema abordado consiste em atribuir um valor positivo w, $w \in [1, w_{max}]$ para cada arco pertencente à rede, de tal maneira que uma função objetivo que avalia o congestionamento na rede possa ser minimizada. Neste trabalho, utiliza-se três metaheurísticas: busca tabu, algoritmo genético e algoritmo genético híbrido. Por fim, apresenta-se uma comparação entre os métodos de resolução descritos.

Em [14], o WSP também é considerado. Neste artigo, é proposto um algoritmo genético híbrido com uma melhoria local. Este procedimento local de melhoria emprega um eficiente algoritmo dinâmico de menor caminho para recompor um caminho mais

curto após a modificação dos pesos da ligação.

Até o presente não há método exato proposto para solucionar o WSP. Na categoria de heurísticas há várias propostas, desde as simples heurísticas construtivas até meta-heurísicas sofisticadas, como as apresentadas nos trabalhos acima citados.

Um outro problema relacionado aos anteriores e oriundo da popularização dos meios de telecomunicações é a previsão do comportamento da rede em situações críticas diversas como falhas de ligações ou picos de demanda, isto é, a sobrevivência (*survivability*) de redes [7]. Este tem sido também objeto de contínuo estudo nos meios acadêmicos, buscando-se a configuração com a melhor relação custo-benefício, porém apresentado uma confiabilidade mínima.

Resumindo, de forma geral, o roteamento de pacotes de dados na Internet passa por dois problemas:

- alocação dos recursos necessários para tratar a demanda de dados entre roteadores com uma confiabilidade mínima,
- gerenciamento dos recursos disponíveis para a transmição do tráfego na rede de forma homogênea, evitando assim a sobrecarga ou a ociosidade de parte dela.

Nesta tese consideramos o segundo problema mencionado acima. Mais especificamente, são propostos métodos computacionais de solução para o Problema de Circuito Virtual Privado (PCVP), no qual objetiva-se definir o roteamento de um conjunto de pacotes de dados, de forma que estes sejam enviados de suas origens para seus destinos, respeitando a capacidade das arestas de ligação da rede e procurando minimizar tanto o congestionamento quanto o atraso na transmissão dos dados. No PCVP considera-se capacidades para as arestas tanto em relação ao número máximo de pacotes que podem ser transmitidos pelas mesmas simultaneamente, quanto capacidades com relação a largura de banda. Neste caso, a soma dos tamanhos dos pacotes transmitidos por uma dada aresta, em Kbits/seg, não deve ultrapassar um dado limite.

Dentre as contribuições específicas desta pesquisa podemos destacar:

- A proposta de um novo algoritmo de construção para o PCVP. Este algoritmo considera uma nova função objetivo para o problema na fase de construção da

meta-heurística GRASP (Greedy Randomized Adaptive Search Procedure) e tem por objetivo evitar a parada prematura da fase de construção observada em [54].

- A proposta de uma nova vizinhança para uma dada solução do PCVP, a ser utilizada no procedimento de busca local das heurísticas propostas.

- Duas novas heurísticas para o PCVP, que consideram a hibridização das meta-heurísticas GRASP e VND (Variable Neighborhood Descent) e GRASP e ILS (Iterated Local Search). O objetivo da hibridização é intensificar a fase de busca local do GRASP.

- Uma análise estatística dos resultados numéricos obtidos com a aplicação das heurísticas propostas. O objetivo da análise é avaliar se há melhoria na qualidade das soluções dos métodos híbridos, quando comparadas às soluções obtidas pelo método GRASP puro. Em outras palavras, observa-se nos resultados se a intensificação da busca local, proveniente das meta-heurísticas VND e ILS é compensada pela melhoria na qualidade média das soluções obtidas.

- Uma proposta de um novo método híbrido para o PCVP, que utiliza tanto procedimento heurístico, quanto exato. Para o desenvolvimento desta metodologia, uma reformulação do modelo matemático apresentado em [54] é feita, de forma a modelar o PCVP como um problema de programação inteira binária. Este modelo é então utilizado na solução de subproblemas do PCVP, provenientes da fixação de algumas variáveis do mesmo, em valores obtidos de forma heurística. A solução dos subproblemas constitui uma nova metodologia de busca local a ser realizada a partir de uma dada solução para o problema. A busca local realizada através da solução exata de problemas de programação inteira binária foi proposta inicialmente por Fishetti e Lodi [27], no ambiente do algoritmo *branch-and-bound*. O objetivo foi realizar uma busca local intensa na vizinhnça de uma dada solução viável para acelerar a convergência do algoritmo. Propomos a utilização da idéia de obter a solução ótima de subproblemas do PCVP para melhorar a qualidade das soluções encontradas através de uma busca local realizada em uma vizinhança

mais abrangente. Apesar de metodologias híbridas terem sido aplicadas a outros problemas de otimização combinatória, nenhum estudo no assunto foi encontrado na literatura para o PCVP.

1.1 Organização do Trabalho

O trabalho apresentado nesta tese está dividido em mais seis Capítulos devidamente descritos abaixo.

No segundo Capítulo o Problema de Circuito Virtual Privado, objeto de estudo desta tese, é formalmente definido e formulado matematicamente através de um modelo de multifluxo inteiro.

No terceiro Capítulo descrevemos as meta-heurísticas utilizadas neste trabalho, o GRASP, o VND, o ILS e algumas de suas variações, e indicamos as referências bibliográficas utilizadas no estudo das mesmas. Procedimento híbridos, que utilizam conjuntamente abordagens das meta-heurísticas GRASP e VND e das meta-heurísticas GRASP e ILS são também descritos.

No quarto Capítulo apresentamos detalhadamente as heurísticas propostas para o PCVP, baseadas nas meta-heurísticas GRASP, VND e ILS e nos procedimentos híbridos descritos no Capítulo três.

No quinto Capítulo apresentamos os resultados numéricos que comparam as diferentes heurísticas propostas entre si com a solução exata obtida para instâncias de pequeno porte. É realizada uma análise estatística dos resultados obtidos.

No sexto Capítulo apresentamos inicialmente uma reformulação do modelo apresentado no capítulo dois para o PCVP, de forma a formulá-lo como um problema de programação inteira binária. Em seguida, descrevemos o método *local branching*, para problemas de programação inteira binária e propomos um método híbrido meta-heurística / exato para o PCVP, que utiliza as heurísticas apresentadas no Capítulo quatro e um algoritmo exato para resolução de subproblemas, baseado na idéia introduzida pelo método *local branching*. Ainda neste Capítulo são apresentados resultados numéricos para este método híbrido.

No sétimo Capítulo apresentamos conclusões e propostas de trabalhos futuros relacionados a esta pesquisa.

Capítulo 2

Problema de Circuito Virtual Privado (PCVP)

Em uma rede de comunicação, os dados enviados de um centro a outro são segmentados em pacotes que podem transitar por diferentes caminhos. Cada pacote constitui uma entidade independente. Estes pequenos pacotes de dados contêm um cabeçalho com um conjunto de informações necessárias para a sua correta transmissão ao longo da rede. Uma dessas informações é o endereço *IP* (*Internet Protocol*) a que se destina. Quando um pacote chega em um nó da rede, ele espera em uma fila antes de ser transmitido para a aresta seguinte. Finalmente, os pacotes são reagrupados no destino de modo a reconstituir a mensagem enviada.

Como cada pacote é tratado de forma individual, a ordem em que os pacotes chegam pode não ser a mesma em que são enviados. O protocolo *IP* apenas envia os pacotes, ficando a tarefa de sua reordenação a cargo de outro protocolo chamado *TCP* (Transmission Control Protocol). Além disso, como o roteamento é uma tarefa muito complexa, a rede mundial é dividida em domínios com regras próprias para rotear tráfego em seu interior e outras regras para rotear tráfego entre domínios. Um domínio é chamado de Sistema Autônomo (Autonomous System (AS)). Há protocolos válidos para atuar dentro de um AS (Interior Gateway Protocolos (IGP)) ou entre unidades de AS (Exterior Gateway Protocols (EGP)).

O problema de roteamento de dados em uma rede de comunicação consiste em obter

o encaminhamento ótimo dos pacotes através das arestas da rede, segundo um ou mais critérios. Este problema e suas variantes são conhecidos também na literatura como *bandwidth packing problems* (BWP). Diversas heuristicas foram propostas para o BWP e suas variantes. Um dos primeiros algoritmos para roteamento de pacotes em redes de comunicação foi proposto por Yee e por Lin [64].

Como mencionado anteriormente, o problema de roteamento de pacotes em redes de comunicação é um caso particular de problemas de multifluxo em redes. Um modelo de programação linear para o problema de fluxo em redes é composto basicamente a partir de dois conjuntos de restrições:

- Conservação do fluxo
- Capacidades das arestas da rede.

Definimos o Problema de Circuito Virtual Privado (PCVP) como um serviço de gerenciamento de uma rede *frame relay* (comutador de pacotes) que envia fluxos através de um conjunto de circuitos virtuais privados de uma grande rede, a seus clientes, isto é, um serviço que utiliza uma eficiente tecnologia de comunicação de dados para transmitir de maneira rápida e barata a informação digital através de uma rede de dados, dividindo essas informações em pacotes (*frames*), a um ou mais destinos de uma ou mais origens. Cada produto (demanda) a ser roteado está associado a um par de nós (origem-destino).

Durante o uso dos circuitos virtuais privados decisões são tomadas automaticamente sem nenhum conhecimento dos pedidos futuros. Essas decisões podem causar ineficiência ao longo do tempo, por exemplo, o congestionamento da rede. Por este motivo, torna-se necessária a redistribuição das demandas, ou seja, dada uma demanda que possui uma rota, denominada rota preferida, o comutador (switch) moverá a demanda de sua rota atual para a nova rota, assim que este movimento possa ser executado. Esta nova rota passará a ser a rota preferida da mesma. Este procedimento de roteamento também é utilizado para o caso de falha em alguma parte da rede.

Para formular o PCVP, são introduzidos alterações no modelo do problema de multifluxo clássico, os critérios utilizados para a minimização do atraso médio por mensagem e do congestionamento, considerando-se a capacidade das arestas com relação ao número

de pacotes enviados e à largura da banda. Considera-se também que um pacote não pode ser dividido no seu envio.

Observe que, considerar a propagação do atraso, e não apenas o número de arestas contidos na rota percorrida da origem ao destino é particularmente importante nas redes internacionais, aonde as distâncias entre nós da rede variam consideravelmente.

O PCVP é formulado então como um problema de multifluxo inteiro com restrições adicionais e uma função objetivo híbrida, que examina a propagação do atraso e o congestionamento da rede. Dado um conjunto de pacotes com diferentes tamanhos, o objetivo do PCVP é, portanto, transmitir os pacotes através da rede de forma a minimizar a função objetivo e respeitando as restrições de conservação de fluxo e capacidade das arestas. Os fluxos na rede são simultâneos e os pacotes compartilham os recursos da rede.

A seguir formularemos o PCVP.

2.1 Formulação do Problema de Circuito Virtual Privado

Consideremos $G = (V; E)$ um grafo não direcionado que representa uma rede com a tecnologia de envio de demanda em pacotes, onde $V = \{1, ..., n\}$ é o conjunto dos nós da rede que possuem condutores (switches) e $E = \{(i, j)|i, j \in V, i < j\}$ o conjunto de m arestas que conectam os nós da rede.

Para cada aresta $(i, j) \in E$, $i < j$, denotamos b_{ij} como o fluxo máximo permitido na aresta, em Kbits/seg, c_{ij} como o número máximo de pacotes que podem ser roteadas simultaneamente através da mesma e d_{ij} a propagação do atraso associado à aresta.

Cada produto $k \in K$, $K = \{1, ..., p\}$ corresponde a um pacote a ser roteado associado a um par de nós origem-destino e a uma exigência de capacidade r_k (tamanho do pacote em Kbits/seg).

Tem-se como objetivo minimizar a propagação de atraso e/ou o congestionamento da rede. Observe que o atraso normalmente é associado ao congestionamento, porém ele

pode ter também como causa a grande distância e a baixa velocidade de transmissão.

Desta forma, minimizar o atraso pode não diminuir o congestionamento e minimizar o congestionamento pode não diminuir o atraso. O objetivo passa a ser a busca por um equilíbrio entre o atraso e o congestionamento.

Para cada aresta $(i, j) \in E$, o valor máximo de fluxo b_{ij} não pode ser excedido e o número de pacotes distribuidos não pode ser maior que c_{ij}.

Para modelar o PCVP, associa-se a cada aresta $(i, j) \in E$ e a cada pacote $k \in K$, duas variáveis x_{ij}^k e x_{ji}^k, tais que $x_{ij}^k = 1$ ($x_{ji}^k = 1$) se e somente se a aresta $(i, j) \in E$ é usada para transmitir o produto k do nó i (j) para o nó j (i) e $x_{ij}^k = 0$ ($x_{ji}^k = 0$) caso contrário.

Então a formulação para o PCVP é dada por:

$$\text{Minimizar } \phi(x) = \sum_{(i,j)\in E, i<j} \phi_{ij}\ (x_{ij}^1, ..., x_{ij}^p, x_{ji}^1, ..., x_{ji}^p)$$

sujeito a:

$$\begin{aligned}
&\sum_{k\in K} r_k(x_{ij}^k + x_{ji}^k) \leq b_{ij}, \forall (i,j) \in E, i < j \\
&\sum_{k\in K} (x_{ij}^k + x_{ji}^k) \leq c_{ij}, \forall (i,j) \in E, i < j \\
&\sum_{(i,j)\in E} x_{ij}^k - \sum_{(i,j)\in E} x_{ji}^k = a_i^k, \forall i \in V, \forall k \in K \\
&x_{ij}^k \in \{0,1\}, \forall (i,j) \in E, \forall k \in K.
\end{aligned} \tag{2.1}$$

A primeira restrição $\sum_{k\in K} r_k(x_{ij}^k + x_{ji}^k) \leq b_{ij}$ representa o limite de fluxo permitido em cada aresta. É comum abordar este problema permitindo-se a relaxação desta restrição. Neste caso, sua violação é penalizada na função objetivo, através da função de penalidade introduzida pela componente do congestionamento da aresta.

A segunda restrição $\sum_{k\in K} (x_{ij}^k + x_{ji}^k) \leq c_{ij}$ limita o número de pacotes que atravessam cada aresta. Ao contrário da primeira, tal restrição, em geral, não pode ser relaxada.

A terceira restrição $\sum_{(i,j)\in E} x_{ij}^k - \sum_{(i,j)\in E} x_{ji}^k = a_i^k$ representa a conservação do fluxo, determinando que todo fluxo que for enviado por um nó (origem) deverá chegar em

outro nó, até atingir o nó destino.

Observe que a terceira restrição $\sum_{(i,j)\in E} x_{ij}^k - \sum_{(i,j)\in E} x_{ji}^k = a_i^k$ e a quarta restrição $x_{ij}^k \in \{0,1\}$ juntas determinam que o fluxo não poderá ser divido, ou seja, o pacote deverá ser enviado integralmente, até o nó destino.

Para cada pacote definimos como, $a_i^k = 1$, se o nó i for a origem do produto k, $a_i^k = -1$, se o nó i for o destino do produto k, e para todos os outros nós consideramos que $a_i^k = 0$, ou seja, todos os outros nós poderão ser nós de passagem entre a origem e o destino.

A parcela da função objetivo ϕ_{ij}, associada a cada aresta $(i,j) \in E$ com $i < j$ é definida como uma combinação linear das componentes do atraso e do congestionamento da aresta.

A primeira componente de ϕ_{ij}, componente do atraso, onde ρ_k modela o atraso, é definida como $\phi_{ij}^d\ (x_{ij}^1, ..., x_{ij}^p, x_{ji}^1, ..., x_{ji}^p) = d_{ij} \sum \rho_k (x_{ij}^k + x_{ji}^k)$.

Se $\rho_k = 1$ esta componente leva a minimização do número de "saltos", ou seja, o número de nós no caminho utilizado para rotear o produto k, ponderada pela propagação do atraso em cada aresta.

Se $\rho_k = r_k$, a minimização leva em consideração leva em consideração a capacidade exigida para transmitir o pacote k através de cada aresta, ponderando o atraso em cada aresta.

Seja $y_{ij} = \sum_{k\in K} r_k (x_{ij}^k + x_{ji}^k)$ o total do fluxo que atravessa a aresta $(i,j) \in E$ com $i < j$. A segunda componente de ϕ_{ij}, chamada de componente de congestionamento, depende da utilização das taxas u_{ij}, que serão definidas como $u_{ij} = \dfrac{y_{ij}}{b_{ij}}$ para cada aresta $(i,j) \in E$ com $i < j$. Ela é representada pela função linear por partes proposta por Fortz e Trorup (2000) [28]. A componente penaliza cada vez mais os fluxos que se aproximam ou que violam os limites da capacidade.

$$\phi_{ij}^{b}(x_{ij}^{1},...,x_{ij}^{p},x_{ji}^{1},...,x_{ji}^{p}) = b_{ij} * \begin{cases} u_{ij} & u_{ij} \in [0,\frac{1}{3}) \\ 3.u_{ij} - \frac{2}{3} & u_{ij} \in [\frac{1}{3},\frac{2}{3}) \\ 10.u_{ij} - \frac{16}{3} & u_{ij} \in [\frac{2}{3},\frac{9}{10}) \\ 70.u_{ij} - \frac{178}{3} & u_{ij} \in [\frac{9}{10},1) \\ 500.u_{ij} - \frac{1468}{3} & u_{ij} \in [1,\frac{11}{10}) \\ 5000.u_{ij} - \frac{16318}{3} & u_{ij} \in [\frac{11}{10},\infty) \end{cases} \tag{2.2}$$

A função de penalidade será zero se a aresta estiver ociosa. Observe que, dado que a função penaliza fortemente fluxos que excedem a capacidade, a solução deve ser tal que, se possível, nenhuma aresta apresente taxa de utilização u_{ij} acima do valor 1. Busca-se assim, um perfil de fluxos o mais homogeneo possível, sem sobrecarregar nenhuma aresta.

Na função custo associada à cada aresta (i,j) da rede faz-se uso dos pesos $(1-\delta)$ e δ, os quais correspondem respectivamente à propagação do atraso e à componente do congestionamento, $\delta \in [0,1]$.

$$\phi_{ij}(x_{ij}^{1},...,x_{ij}^{p},x_{ji}^{1},...,x_{ji}^{p}) = (1-\delta).\phi_{ij}^{d}(x_{ij}^{1},...,x_{ij}^{p},x_{ji}^{1},...,x_{ji}^{p}) + \delta.\phi_{ij}^{b}(x_{ij}^{1},...,x_{ij}^{p},x_{ji}^{1},...,x_{ji}^{p})$$

Se $\delta > 0$, a componente de congestionamento está presente na função objetivo e a restrição de capacidade, $\sum_{k \in K} r_k(x_{ij}^{k} + x_{ji}^{k}) \leq b_{ij}$, poderá ser relaxada.

Capítulo 3

Meta-heurísticas

3.1 Introdução

Tomemos como base para o estudo realizado neste Capítulo o problema de otimização combinatória definido como:

$$(P) \qquad \text{Minimizar } f(x), \qquad (3.1)$$
$$\text{sujeito a: } x \in X,$$

onde $f : X \rightarrow \Re$ é uma função real e X é um conjunto finito de soluções viáveis. Objetiva-se encontrar uma solução ótima para (P), $x^* \in X$, tal que $f(x^*) \leq f(x)$, $\forall x \in X$. Definimos x^* como uma solução ótima global de (P).

Uma característica encontrada em muitos problemas combinatoriais é a presença de soluções ótimas locais. Dizemos que x^* é uma solução ótima local para (P), se $x^* \in X$ e $f(x^*) \leq f(x)$, $\forall x \in N(x^*)$, onde $N(x^*)$ é um conjunto de soluções denotado por vizinhança de x^*.

Definiremos como vizinhança $N(x)$ de uma solução x, o conjunto de soluções que podem ser alcançadas a partir de x, por uma operação simples. Considerando-se que uma dada solução é composta por elementos, tal operação pode ser a remoção ou adição de um elemento à solução ou mesmo o intercâmbio de dois elementos da solução. Desta forma, se uma solução x não for pior do que qualquer outra solução em sua vizinhança $N(x)$ então x é definido como um ótimo local com respeito a esta vizinhança [55].

Teoricamente é possível encontrar uma solução ótima global para (P) utilizando algoritmos de enumeração, porém na prática a estratégia de enumeração completa das soluções se torna inviável, pois o número de combinações cresce exponencialmente com o tamanho do problema [49].

Muitos problemas gerados a partir de aplicações reais são muito grandes também para o uso de algoritmos exatos, que garantidamente forneçam a sua solução ótima global. Por exemplo alguns problemas ligados ao setor de telecomunicações, como o que estudamos nesta tese. A estes problemas usualmente emprega-se métodos heurísticos para a obtenção de soluções de boa qualidade.

Nas últimas décadas foram desenvolvidas heurísticas bastante gerais para problemas de otimização combinatória que utilizam uma combinação de procedimentos aleatórios e gulosos na obtenção de uma solução para o problema. Estas heurísticas ficaram conhecidas como meta-heurísticas e têm sido bastante eficientes na resolução de diversos problemas. As meta-heurísticas são conhecidas por sua capacidade de escapar de ótimos locais de baixa qualidade,[29].

O nosso objetivo neste Capítulo é fazer um estudo das principais meta-heurísticas que serão utilizadas no desenvolvimento dos métodos computacionais propostos nesta tese. A meta-heurística GRASP será apresentada mais detalhadamente, uma vez que ela é a base de nossa proposta. Apresentaremos não apenas a metodologia básica do GRASP, mas também todas as propostas de melhorias no algoritmo básico que encontramos na literatura. Em seguida, descreveremos mais suscintamente as meta-heurísticas VND e ILS, que serão utilizadas em nosso trabalho para gerar métodos heurísticos híbridos, que mixam procedimentos de diferentes meta-heurísticas. Conforme exporemos a seguir, estas metodologias híbridas têm gerado heurísticas de boa qualidade para diversos problemas de otimização combinatória. Em nossa proposta iremos comparar um algoritmo puro GRASP para resolver o PCVP com duas heurísticas híbridas, na primeira substituimos a busca local do GRASP por um procedimento baseado na meta-heurística VND e na segunda, a substituimos por um procedimento baseado na meta-heurística ILS.

3.2 Meta-heurística GRASP

A meta-heurística GRASP (Greedy Randomized Adaptive Search Procedures) foi baseada nos trabalhos de Lin & Kernighan (1973) [39] e Hard & Shogan (1977) [33] e proposta por Thomas Feo e Maurício Resende no final dos anos 80, [23]. Consiste na combinação de uma heurística construtiva aleatorizada, encarregada de produzir uma solução, com um método de busca local, que explora a vizinhança da solução construida, retornando um mínimo local. A busca local estocástica é um procedimento iterativo cujas iterações são independentes [61]. A melhor solução produzida durante as iterações será o resultado obtido pelo algoritmo.

Uma heurística construtiva é um procedimento que seleciona sequencialmente elementos da solução de um conjunto finito, com o objetivo de obter, ao final, uma solução viável. Tornar aleatória a seleção do elemento que é incluído na solução, fazendo com que a heurística construtiva seja probabilística, tem por finalidade possibilitar a geração de soluções distintas. O resultado disto é que a heurística construtiva aleatorizada tem o poder de evitar que o processo de busca fique preso a ótimos locais.

A meta-heurística GRASP tem sido usada para obter soluções de qualidade para muitos problemas de otimização combinatória [3, 24, 25, 39, 45, 49, 38]. Tem se destacado em relação a outras meta-heurísticas por sua facilidade de implementação e de ajuste dos parâmetros. A eficiência do GRASP se deve basicamente a uma implementação eficiente da estrutura de dados que assegurem iterações rápidas [54].

Vários melhoramentos tem sido propostos para a meta-heurística GRASP, como a hibridização do método com outras meta-heurísticas. Em geral, verifica-se que abordagens híbridas possibilitam a obtenção de soluções de melhor qualidade comparadas às obtidas pelo algoritmo básico [3, 20].

O religamento de caminhos é uma das melhorias propostas, que ao ser incorporada à meta-heurística GRASP tem permitido a obtenção de soluções de melhor qualidade comparadas com aquelas obtidas pelo algoritmo puro [3], permitindo à meta-heurística GRASP intensificar a busca em regiões onde soluções de qualidade tenham sido encontradas.

Um fator importante na meta-heurística GRASP pura ou mesmo nas abordagens híbridas, é a seqüência de números aleatórios utilizada durante a sua execução. Cada número aleatório dessa seqüência é usado para iniciar a fase construtiva do método em uma iteração.

A fase construtiva do GRASP é realizada por um algoritmo semi-guloso, onde a escolha de cada elemento inserido na solução é feita de forma aleatória, porém controlada por uma função gulosa, onde esta mede o benefício associado a seleção de cada elemento. Dessa forma, para um número fixo de iterações, a qualidade da solução obtida pelo algoritmo dependerá da seqüência de números aleatórios usada. Da mesma forma, fixando-se a qualidade de solução que deseja-se obter, o tempo de execução do algoritmo também dependerá da seqüência de números aleatórios usada. Com isto concluimos que a meta-heurística GRASP depende da semente inicial utilizada para a geração de números aleatórios diferentes, [24, 45, 50, 48, 52].

3.2.1 Algoritmo GRASP Básico

O pseudo-código da Figura 3.1 ilustra o algoritmo básico do GRASP para o problema de minimização (3.1), onde os seguintes parâmetros são utilizados: *MaxIter*, que é o número máximo de iterações a serem realizadas, *Sem*, que é a semente inicial para o gerador de números pseudo aleatórios e α, que é um número que pertence ao intervalo $[0, 1]$ e é utilizado na construção de uma solução a cada iteração do algoritmo.

A seguir, serão descritas as etapas do algoritmo GRASP básico, isto é, a fase de construção e de busca local.

3.2.2 Fase de Construção (Construção Randômica)

A fase de construção tem como objetivo prover soluções viáveis distintas, produzidas utilizando-se um procedimento de construção, tal como uma heurística construtiva. As soluções obtidas serão exploradas na fase de busca local.

Na fase de construção, uma solução viável é iterativamente construida, inserindo-se na solução um elemento de cada vez. A cada iteração da fase construtiva, são avaliados

```
Prodedimento GRASP (MaxIter, Sem, α)
  1  Leia Dados de Entrada ();
  2  Para k = 1, ...,MaxIter faça
  3  Início
  4    x̂ ← Construção Randômica ( Sem,α);
  5    x̃ ← Busca Local (x̂);
  6    x̄ ← Atualização da Solução(x̃);
  7  Fim;
  8  Retorna x̄
Fim GRASP;
```

Figura 3.1: Pseudo código GRASP.

apenas elementos que podem ser adicionados à solução, isto é, não devem ser considerados os elementos que uma vez na solução fazem com que esta seja inviável. Ao conjunto destes elementos, aqueles que podem ser adicionados à solução, dá-se o nome de *elementos candidatos* (conjunto C).

Para fazer a escolha de qual elemento será adicionado à solução devemos montar uma lista com os melhores elementos do conjunto C. Ordenamos, então, estes elementos candidatos em uma *lista restrita de candidatos* (LRC). Os elementos membros da LRC são escolhidos com base em uma função gulosa, que "avalia" o benefício de cada elemento do conjunto C fazer parte da solução.

A escolha do elemento que é adicionado à solução é realizada de maneira aleatória dentre os membros da LRC, isto é, todos os elementos desta lista tem a mesma probabilidade de serem considerados na solução que está sendo iterativamente construída nesta fase.

Cada vez que é selecionado um elemento da LRC para fazer parte da solução, a viabilidade da solução parcialmente construida deve ser verificada frente às restrições do problema. Sendo a solução construida viável, a fase de construção se encerra. No caso de a solução ser ainda inviável, um novo elemento deve ser adicionado. A cada adição de um elemento à solução, o conjunto C e a LRC devem ser reavaliados.

A meta-heurística GRASP é, portanto, adaptativa, uma vez que os benefícios associados a cada um dos elementos do conjunto C são atualizados a cada iteração da fase de construção para refletir as mudanças trazidas pela seleção do elemento na iteração anterior.

A componente probabilística do GRASP é caracterizada pela escolha aleatória de um dos elementos da lista de candidatos LRC, que não é necessariamente o melhor. Note que em uma heurística gulosa construtiva o elemento que sempre é selecionado é aquele com a melhor avaliação. É importante ressaltar que nesse caso a heurística é determinística, ou seja, se reaplicada nas mesmas condições provém a mesma solução.

3.2.3 Lista Restrita de Candidatos

Exitem duas estratégias básicas para construção da LRC.

A primeira consiste em um esquema de *cardinalidade*, onde para um valor inteiro k pré-estabelecido, coloca-se na LRC os k melhores elementos da lista de candidatos [45]. Este esquema para a construção da LRC não tem sido muito usado na literatura.

A outra abordagem associa a cada elemento candidato c do conjunto C, o valor do seu benefício $h(c)$, dado por uma função gulosa $h(\cdot)$. Considera-se $h_{max} = max\{h(c)|c \in C\}$, $h_{min} = min\{h(c)|c \in C\}$ e um parâmetro $\alpha \in [0, 1]$. Em um problema de minimização, uma LRC baseada em valores será determinada por LRC $= \{c \in C|h(c) \leq \ h_{min} + \alpha(h_{max} - h_{min}\}$. Note que h_{min} e h_{max} são respectivamente o melhor e o pior valor da função gulosa correspondente a todos os elementos candidatos a fazer parte da solução, numa dada iteração da fase de construção. A função gulosa aplicada em c pode medir, por exemplo, o aumento no valor da função objetivo do problema ao se acrescentar o elemento c à solução. O tamanho da LRC, neste caso, é definido em função do parâmetro α. Ele controla a qualidade dos elementos que são considerados para formar a LRC. O valor da função gulosa quando, aplicada a estes elementos, deve estar no intervalo $[h_{min},\ \alpha(h_{max} - h_{min}) + h_{min}]$.

Observa-se que para um problema de minimização $\alpha = 0$ implica em uma escolha puramente gulosa, enquanto $\alpha = 1$, implica em uma escolha aleatória, isto é, neste caso a LRC contém todos os elementos possíveis.

Analogamente, em um esquema baseado em cardinalidade, no caso $k = 1$ o algoritmo semi-guloso é transformado em um algoritmo guloso, enquanto que para $k = |C|$ são construídas soluções aleatórias. Dessa forma, os parâmetros α e k determinam se a fase construtiva da meta-heurística GRASP será mais semelhante a um algoritmo de construção aleatória ou a um algoritmo guloso.

O parâmetro α pode ser escolhido de diversas maneiras:

Fixo - selecionado pelo usuário, em geral, próximo ao guloso para garantir qualidade média, mas longe o suficiente para gerar diversidade. As primeiras implementações da meta-heurística GRASP usavam valores fixos para α, os quais eram determinados através de experimentos. Para cada problema tratado ou até mesmo para cada classe de problemas testes, um valor diferente de α era estabelecido [45]. Porém, de acordo com [45], excluindo o caso da construção randômica, o GRASP com α fixo pode nunca convergir para um mínimo global.

Aleatório - selecionado de forma aleatória no intervalo [0,1] , discretizado ou contínuo; Resende et al. [54] propuseram o uso de um α gerado aleatoriamente no intervalo [0,1] em cada iteração do algoritmo, de forma a contornar o problema de convergência para o ótimo local, encontrado quando α é fixo.

Auto-ajustável (GRASP Reativo) - o parâmetro α é ajustado em tempo de execução de acordo com informações de soluções visitadas anteriormente.

Este procedimento será mais detalhado a frente.

3.2.4 Algoritmo de Construção Básico

O pseudo-código da Figura 3.2 descreve a fase de construção GRASP para um problema de minimização, onde $h(\mathbf{.})$ é a função gulosa, α é o parâmetro responsável pelo tamanho da LRC e C o conjunto de elementos candidatos.

3.2.5 Fase de Busca Local

Geralmente é possível melhorar a solução construida na fase de construção do GRASP, realizando-se a partir dela uma busca local.

Procedimento `Construção Randômica` (`sem`, α)

1 $\hat{x} = \emptyset$;
2 `Inicializa o conjunto` C;
3 **Enquanto** $C \neq \emptyset$ **faça**
4 **Início**
5 $h_{min} = min\{h(t)|t \in C\}$;
6 $h_{max} = max\{h(t)|t \in C\}$;
7 `LRC` $= \{s \in C \mid h(s) \leq h_{min} + \alpha(h_{max} - h_{min})\}$;
8 `Seleciona s, aleatoriamente da LRC`;
9 $\hat{x} = \hat{x} \bigcup \{s\}$;
10 `Atualize o conjunto candidato` C;
11 **Fim**;
12 **Retorna** $\hat{x}$

Fim `Construção Randômica`;

Figura 3.2: Pseudo código Construção Randômica.

Em um procedimento de busca local a partir de x procura uma solução melhor do que x no conjunto de soluções $x^1, x^2, \ldots, x^k$, pertencentes à vizinhança $N(x)$. Estas são soluções que podem ser obtidas através da aplicação de uma modificação elementar em x, chamada de movimento.

A notação de *movimentos de trocas - (p,q)*, por exemplo, será usada indicando que dada uma solução x, a sua vizinhança será gerada removendo-se p elementos de x e inserindo-se q novos elementos em x.

Por exemplo, se $x^0 = (1, 2, 3)$ e $N_1(x^0)$ é a vizinhança gerada por movimentos de permutação de dois elementos, isto é, *movimentos de trocas - (2,2)*, obteremos como vizinhança de x^0 o conjunto $N(x^0) = \{(2, 1, 3), (1, 3, 2), (3, 2, 1)\}$. Vizinhanças simples tal como *movimentos de trocas - (2,2)* são frequentemente usadas.

Em um problema de minimização, a vizinhança $N(x^k)$ de x^k é analisada na k-ésima iteração e procura-se encontrar uma solução $x^{k+1} \in N(x^k)$, tal que $f(x^{k+1}) < f(x^k)$. Quando uma solução melhora a solução corrente, ela passa a ser a nova solução e a sua

vizinhança passa a ser analisada. Caso contrário, a busca termina e a solução corrente é um ótimo local.

Logo, a vizinhança N para o problema P em relação a solução x^k, define um subconjunto de soluções $N(x^k)$, onde x^k é dito um ótimo local, se não existe solução melhor que x^k em $N(x^k)$.

A eficiência de um procedimento de busca local depende de vários fatores, tais como a estrutura da vizinhança, a técnica usada para explorar a vizinhança, a função que deve ser otimizada e a solução inicial, [45]. Sua idéia básica consiste em usar diferentes soluções iniciais como pontos de partida para a busca local.

A busca na vizinhança pode ser implementada usando-se uma estratégia de *best-improving* ou de *first-improving*. No caso da estratratégia *best-improving* todos os vizinhos são investigados e a solução inicial é substituida pela melhor solução encontrada na vizinhança. Na estratégia de *first-improving* troca-se a solução inicial pelo primeiro vizinho que tenha custo menor.

De acordo com [54], observa-se, na prática, que ambas as estratégias são rápidas porém a *first-improving* é executada com tempo computacional menor. Outra observação se refere à convergência prematura para um mínimo local não global, o qual ocorre mais provavelmente com a estratégia *best-improving*.

Na Figura 3.3 podemos observar o procedimento de busca local.

Procedimento `Busca Local` $(\hat{x})$

1 **Enquanto** `existe` $x \in N(\hat{x})$ `tal que` $f(x) < f(\hat{x})$ **faça**
2 **Início**
3 `Seleciona` $x \in N(\hat{x})$ `tal que` $f(x) < f(\hat{x})$;
4 $\hat{x} = x$;
5 `Busca Local (`$\hat{x}$`) ;`
6 **Fim**;
7 **Retorna** $(\hat{x})$;

Fim `Busca Local`;

Figura 3.3: Pseudo código Busca Local.

Uma solução x é dita como pertencente a *bacia de atração* de um ótimo local quando, a partir de uma busca local iniciada em x, é possível atingir este ótimo local. Partindo de uma solução na *bacia de atração* de um ótimo global, a busca local irá encontrar este ótimo global [3, 45].

Uma solução na bacia de atração de um ótimo global será produzida, caso um número grande de soluções geradas aleatoriamente seja usado para iniciar a busca local. Porém soluções produzidas aleatoriamente são, em geral, de baixa qualidade. Além disso, o número de movimentos necessários para que soluções geradas aleatoriamente e que estejam na bacia de atração de um ótimo global atinjam o ótimo, possivelmente será muito elevado ou até mesmo exponencial ao tamanho do problema [45].

Por outro lado, algoritmos gulosos geralmente produzem soluções de melhor qualidade do que as geradas aleatoriamente. O uso de soluções gulosas como ponto de partida para uma busca local, em geral, levará a boas soluções, porém, sub-ótimas, isto é, soluções de qualidade inferior à qualidade dos ótimos globais, [45].

Observando-se então, que a diversidade das soluções produzidas por um algoritmo guloso é muito pequena; a meta-heurística GRASP propõe um algoritmo semi-guloso para produzir as soluções iniciais usadas em cada iteração, garantindo assim a diversidade de soluções e ao mesmo tempo um controle na qualidade das soluções produzidas.

3.2.6 Aplicação da Estatística ao GRASP

De acordo com [54, 29], a meta-heurística GRASP pode ser vista como uma técnica de amostragem repetitiva, isto é, um processo de escolha da amostra, a qual é a parte inicial de qualquer estudo estatístico.

Amostragem é definida como uma escolha criteriosa de elementos a serem submetidos ao estudo estatístico e para que os resultados sejam representativos, deve-se tomar o cuidado com a escolha do conjunto de dados da amostra (LRC), onde este deve apresentar características tão próximas quanto possível da população (conjunto C).

Podemos observar que uma solução produzida em uma iteração do método GRASP é uma amostra de alguma distribuição de frequência desconhecida, onde a média e a variância são de natureza restritiva, isto é, são funções restritivas da LRC pois os valores

da média e da variância dependem dos elementos da LRC [3, 54].

Usando o conceito da estatística, as medidas de tendência central são usadas para indicar um valor que tende a tipificar, ou a representar melhor, um conjunto de dados. Dentre estas, temos a media aritmética ($\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$), onde todos os dados tem a mesma importância, e a variância ($s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n-1}$), que é a média dos quadrados dos desvios dos valores a contar da média.

Por exemplo, caso a LRC seja formada por apenas um elemento, apenas uma solução será produzida, a variância será zero e a média será o valor da solução gulosa. Caso a LRC seja formada por um número maior de elementos, então várias soluções diferentes serão produzidas e a variância será diferente de zero, ou seja, maior do que do exemplo anterior, e o valor da média deve ser pior. A estratégia gulosa tem um papel menor nesse caso, uma vez que a média dos valores dos elementos contidos na LRC deve ser pior que o valor do melhor elemento.

3.3 Melhoramentos para a fase de Construção

Nesta Seção serão mostradas algumas técnicas que podem ser aplicadas na fase de construção da meta-heurística GRASP. Essas técnicas têm como finalidade melhorar a qualidade das soluções construidas.

3.3.1 Função Tendência

Na fase de construção do GRASP básico o próximo elemento a ser inserido na solução é selecionado aleatoriamente entre os elementos candidatos na LRC. Cada elemento da LRC possui a mesma probabilidade de ser selecionado.

Bresina propõe em [50] uma seleção do elemento da LRC, onde a cada elemento da lista de canditatos restrita é atribuída uma probabilidade de seleção. Como podemos observar na Tabela 3.1, uma família de distribuições de probabilidade é proposta.

Primeiro, o valor da função gulosa de cada elemento $c \in C$ é calculado, uma vez que

é necessário eleger os membros de C que farão parte da lista LRC. Daí, os elementos da LRC devem ser ordenados de acordo com o valor da função gulosa. Seja $rank(c), c \in C$ a posição relativa do elemento c na lista LRC de acordo com a ordenação dada pela função gulosa. A cada elemento da LRC é associado um valor, $bias(rank(c))$, onde $bias(.)$ é calculado de acordo com a função de tendência, cujas expressões propostas no trabalho de Bresina [12] estão colocadas na Tabela 3.1. A probabilidade de seleção de um elemento c da lista de candidatos é então determinada por:

$$p(c) = \frac{bias(rank(c))}{\sum_{c' \in \text{LRC}} bias(rank(c'))}, c \in \text{LRC} \tag{3.2}$$

Tabela 3.1: Opções para Função Tendência.

1	Logarítmica	$bias(r) = 1/log(r+1)$
2	Linear	$bias(r) = 1/r$
3	Polinomial	$bias(r) = 1/r^n$
4	Exponencial	$bias(r) = 1/e^r$
5	Randômica	$bias(r) = 1$

Suponha, por exemplo, que a função de Bresina exponencial é utilizada numa LRC com dois elementos c_1 e c_2, tal que $h(c_1) < h(c_2)$. Neste caso $rank(c_1) = 1$ e $rank(c_2) = 2$. A probabilidade de escolha do elemento c_1 é dada por $\frac{\frac{1}{e}}{\frac{1}{e} + \frac{1}{e^2}}$ e a probabilidade de escolha do elemento c_2 é dada por $\frac{\frac{1}{e^2}}{\frac{1}{e} + \frac{1}{e^2}}$. O elemento c_1 tem, portanto, uma maior probabilidade de ser escolhido.

3.3.2 GRASP Reativo

O GRASP Reativo foi proposto com o intuito de incorporar um mecanismo de aprendizado à fase de construção da meta-heurística GRASP, a qual não utiliza mecanismo de memória.

No procedimento GRASP Reativo o parâmetro α é ajustado em tempo de execução de acordo com informações de soluções visitadas anteriormente.

Em cada iteração do algoritmo, o valor do parâmetro α é escolhido aleatoriamente. Porém, ao invés de escolher esse parâmetro com uma distribuição uniforme, ele é escolhido a partir de um conjunto de valores discretos $\{\alpha_1, \alpha_2, ..., \alpha_m\}$, onde a cada α_k associa-se uma probabilidade de escolha $p(\alpha_k)$. Utilizando a definição clássica de probabilidade, suponha que um evento E possa acontecer de h maneiras diferentes, em um total de m modos possíveis, igualmente prováveis. Então, a probabilidade de ocorrência do evento (denominado sucesso) é definida por: $p = P\{E\} = \dfrac{h}{m}$. Quando o evento tem resultados igualmente prováveis temos $p = P\{E\} = \dfrac{1}{m}$.

O procedimento atualiza em tempo de execução as probabilidades $\{p(\alpha_1), p(\alpha_2), ..., p(\alpha_m)\}$ para favorecer valores de α que tenham produzido soluções de qualidade, isto é, a partir da segunda iteração as probabilidades não serão mais iguais para cada valor de α.

Seja f^* o valor da função objetivo na melhor solução encontrada até o momento e a_i a média dos valores nas soluções encontradas usando α_i na fase de construção. A probabilidade $p(\alpha_i)$ será definida como $p(\alpha_i) = \dfrac{q_i}{\sum_{j=1}^{m} q_j}$, onde $q_i = \dfrac{f^*}{a_i}$, $i = 1, ..., m$. Este procedimento será realizado periodicamente pelo algoritmo, permitindo que valores de α_i que tenham produzido soluções boas nas iterações anteriores tenham uma maior probabilidade de serem escolhidos [45, 54, 48].

Um procedimento de GRASP Reativo pode ser visto como uma estratégia que usa uma memória de longo prazo para escolher valores do parâmetro α da LRC [52]. Pode-se observar em [52, 3, 50, 45, 54] que o procedimento GRASP Reativo permitiu melhorar o algoritmo GRASP básico em termos de robustez e de qualidade de solução.

3.3.3 Princípio da Proximidade de Otimalidade

O princípio da proximidade da otimalidade (*Proximate Optimality Principle*) foi proposto em [54, 31] para a meta-heurística busca tabu. Este define que "Soluções boas em um nível, possivelmente estarão próximas a soluções boas em um nível adjacente".

Este conceito foi adaptado a meta-heurística GRASP [3, 48], isto é, durante a fase de construção, é proposto que seja aplicado um procedimento de busca local à solução parcial, porém não deve ser aplicado a cada passo da fase construtiva com a finalidade de preservar a variabilidade de soluções e devido ao alto custo computacional. Podemos observar em [11] que este procedimento é realizado após 40% e 80% dos movimentos da fase de construção.

3.3.4 Memória e aprendizado

A meta-heurística GRASP básico não utiliza memória de longo prazo. A cada iteração do algoritmo são armazenadas apenas a solução corrente e a melhor solução encontrada até o momento. Com isto o GRASP não utiliza informações colhidas nas iterações anteriores.

Fleurent e Glover propuseram uma utilização da memória de longo prazo na fase construtiva. Esta é usada para modificar as probabilidades de escolha de cada elemento da LRC [3, 54, 48, 11].

Este procedimento consiste em ter um conjunto de soluções elite, ou seja, um conjunto das P melhores soluções encontradas até o momento.

Tomemos a função de intensificação *I(c)* que mede o número de vezes em que um elemento c aparece nas soluções do conjunto elite. Se o elemento c for inserido em uma solução S, e esta tornar S mais semelhante às soluções do conjunto de elite, $I(c)$ obterá um valor alto.

É proposto que a função *I(c)* seja usada na fase construtiva da meta-heurística GRASP, definindo-se uma função $E(c) = F(g(c), I(c))$, dependentes das funções gulosa e de intensificação a ser utilizada na determinação da LCR.

Por exemplo, seja $E(c) = \lambda g(c) + I(c)$, onde λ é um parâmetro do algoritmo. O esquema de intensificação proposto incentiva a seleção de elementos da LRC que possuam valores de $E(c)$ altos. Para isso, a cada elemento c da LRC são atribuidas probabilidades de escolha dadas por $p(c) = \dfrac{E(c)}{\sum_{j \in LRC} E(j)}$.

Uma das sugestões para o parâmetro λ é a sua variação durante a execução do algoritmo para favorecer mais a parcela gulosa ou a parcela de intensificação [11].

O método de memória e aprendizagem é aplicado ao problema de *Job Shop Scheduling*, [11].

3.3.5 Método do ruído

A idéia deste método é a de introduzir algum ruído nos custos originais do problema. O método do ruído foi introduzido por Charon e Hudry [54, 3] e posteriormente adaptado ao GRASP, sendo utilizado em casos em que o algoritmo de construção não é muito sensível à randomização e em casos em que não existe um algoritmo guloso para ser randomizado.

Ribeiro et al. [57] introduz uma abordagem GRASP para o problema de Steiner, onde são aplicados métodos de perturbação de custo que incorporam os mecanismos de intensificação e diversificação, originalmente propostos no contexto da busca tabu.

Três estratégias de perturbação de custos são propostas, chamadas de D, I e U. Neste processo, os pesos originais (sem perturbação) são usados nas três primeiras iterações do algoritmo, combinados com três heurísticas de construção. As três estratégias de perturbação de custo são usadas de forma cíclica nas iterações seguintes.

Seja w_e o peso de uma aresta e. A cada iteração i, o peso modificado w_e^i de cada aresta e é selecionado aleatoriamente de uma distribuição uniforme, entre w_e e $r_i(e)$. w_e, onde o coeficiente $r_i(e)$ depende da estratégia de perturbação de custo aplicada na iteração i. Seja $t_{i-1}(e)$ o número de ótimos locais onde a aresta e aparece após $i-1$ iterações do algoritmo GRASP. As estratégia de perturbação de custo D, I e U são descritas a seguir:

- Estratégia D - é uma estratégia de diversificação, com $r_i(e) = 1,25 + 0,75\dfrac{t_{i-1}(e)}{(i-1)}$. Portanto, os valores elevados de $r_i(e)$ são atribuídos a arestas que aparecem mais frequentemente nos ótimos locais encontrados anteriormente.

- Estratégia I - é uma estratégia de intensificação, com $r_i(e) = 2 - 0,75\dfrac{t_{i-1}(e)}{(i-1)}$. Portanto, arestas que aparecem poucas vezes nos ótimos locais encontrados anteriormente são fortemente penalizados.

- Estratégia U - é uma estratégia de penalização uniforme, independente de informações de frequência, onde $r_i(e) = 2$

A caracterização de uma abordagem de oscilação estratégica alterna entre os métodos de intencificação (I) e diversificação (D).

De acordo com [57], o algoritmo GRASP com perturbação de custo e religamento de caminhos está entre as melhores heurísticas disponíveis para o problema de Steiner em grafos.

Em [57], Ribeiro at al. também comentam o método usado no problema da árvore de Steiner com prêmios. Neste, uma solução nova é construída em cada iteração usando os prêmios dos nós atualizados por uma função de perturbação, de acordo com a estrutura da solução atual. Dois esquemas de perturbação são usados:

- Perturbação por eliminação: para permitir uma diversificação na busca, nós persistentes, isto é, nós que estão presentes na solução construída na iteração anterior e que permanecem na solução após a busca local, são proibidos de fazer parte da solução corrente. Isso é obtido alterando-se para zero os prêmios desses nós. Um parâmetro θ controla a fração de nós persistentes que terão seus prêmios temporariamente alterados.

- Perturbação por mudança de prêmios: ruídos são introduzidos nos prêmios dos nós para alterar o valor da função objetivo. Para cada nó i, é gerado um fator de perturbação $\beta(i)$ no intervalo $[1-\gamma, 1+\gamma]$, onde γ é um parâmetro de implementação. O prêmio associado ao nó i é temporariamente alterado para $\pi'(i) = \pi(i)\beta(i)$, onde $\pi(i)$ é o prêmio original.

3.3.6 GRASP em heurísticas híbridas

Como vimos na Seção anterior, podemos associar o método de religamento de caminhos ao GRASP, hibridizando-o e obtendo uma melhora considerável em relação ao GRASP puro.

Abordagens híbridas do GRASP possibilitam em geral a obtenção de soluções de qualidade superior àquelas obtidas pelo método puro, o que pode ser visto em [3, 26, 57].

Grande parte das estratégias híbridas desenvolvidas a partir de um algoritmo GRASP baseiam-se na substituição da fase de busca local por outro algoritmo de busca. Em [35],

Laguna e González-Velarde (1991) propuseram a hibridização do método GRASP com uma algoritmo de busca tabu; além deles, Delmaire et al (1999), Colomé e Serra (1998) e Liu et at. (2000) também propuseram um GRASP em que a busca local é feita pela meta-heurística busca tabu [3, 48, 16].

Outra hibridização do GRASP foi proposta por Ahuja et al., onde a fase de construção do GRASP foi utilizada como mecanismo de geração da população inicial para o algoritmo genético [6]. Antony et al. propuseram uma estratégia onde a população inicial é formada por soluções construidas aleatoriamente e por soluções construidas por heurísticas GRASP [8]. Lourenço et al. propuseram o GRASP no algoritmo genético para implementar um tipo de cruzamento (*crossover*) chamado *perfect offspring* [40], mais detalhes podem ser encontrados em [3, 45, 48].

Ribeiro e Souza propuseram o GRASP combinado a uma estratégia de VND (*Variable Neighborhood Descent*)) para explorar diferentes vizinhanças,[50, 56]. Festa et al., [26], estudaram variantes da estratégia híbrida de GRASP com VNS (*Variable Neighborhood Search*). As estratégias de VNS e VND associada ao GRASP também foram propostas por Hansen e Mladenovic [41] e por Martins *et al.* [41]. Mais detalhes podem ser encontrados em [3, 29].

3.4 Mecanismos usados para acelerar o tempo computacional do GRASP

Com foi visto anteriormente, a eficiência do GRASP se deve basicamente a uma implementação eficiente da estrutura de dados que assegurem iterações rápidas [54]. A independência das iterações do GRASP faz com que este não aprenda com o histórico das soluções encontradas previamente. Algumas técnicas de implementação podem ser aplicadas ao método GRASP para reduzir o seu tempo computacional e introduzir uma aprendizagem, como a tabela *hash* e estratégias de filtragem.

3.4.1 Tabela Hash

Uma das operações mais comuns realizadas em tarefas computacionais é a operação de *busca*. Nela, uma informação em particular é procurada entre uma grande quantidade de dados. A tabela *Hash* é uma estrutura de dados usada para implementar dicionários (conjunto dinâmico com operações de inserção, remoção e consulta).

Os dados estão estruturados na forma de *registros*. Cada registro contém uma *chave* e a informação propriamente dita. O objetivo de uma busca é, dada uma chave, encontrar todos os registros cujas chaves são iguais a chave dada.

Um registro de um cadastro de contas bancárias pode ter como chave, por exemplo, o CPF ou o número da conta, além de informações não discriminatórias, tais como nome do titular, data de nascimento, etc.

A estrutura chamada *Hash* é uma forma de armazenar os registros em um tipo de tabela diminuindo o espaço de busca. Quando se deseja acessar um registro específico dessa tabela, aplica-se uma função na chave desejada. O resultado dessa função é um endereço da tabela.

O objetivo da tabela *Hash* é conseguir buscar uma chave em tempo $O(1)$, ou seja, a transformação daria exatamente o endereço desejado. Na prática porém, esse tempo pode levar até $O(n)$ no pior caso. Apesar disso, a tabela *Hash* se mostra extremamente prática e eficiente.

A primeira parte da estrutura *Hash* é uma função, chamada *função de Hash*, que realiza a transformação da chave em um endereço da tabela. Como nenhuma função de *Hash* é injetora, duas ou mais chaves diferentes podem ser colocadas em um mesmo endereço da tabela, ocorrendo assim uma colisão. A segunda parte da estrutura é a *resolução de colisão*, que lida com os casos em que chaves diferentes são mapeadas para um mesmo endereço da tabela.

Uma forma de resolver o problema de colisão é colocar em cada posição da tabela uma lista encadeada. Quando uma chave é mapeada para uma posição da tabela, ela começa a encabeçar a lista. Cada chave seguinte mapeada para o mesmo endereço é posta no final da lista.

No caso do armazenamento em listas simplesmente encadeadas, as inserções são efetuadas de forma imediata nos seus extremos. A inserção no fim da lista requer um ponteiro adicional para o último elemento. Desta forma, a resolução de possíveis colisões pode ser implementada armazenando-se as novas chaves no início das listas para evitar a manutenção desse ponteiro.

Martin, Resende, Ribeiro e Pardalos (2000) usaram a tabela *hash* na meta-heurística GRASP para armazenar todas as soluções usadas como soluções iniciais para busca local. Após cada fase construtiva, a tabela *hash* é consultada para verificar se a solução gerada é nova. Em caso positivo, a solução é inserida na tabela *hash* e o procedimento de busca local é iniciado a partir dessa solução. Caso contrário, o procedimento de busca local não é executado e uma nova iteração é iniciada [3, 45].

3.4.2 Estratégias de Filtragem

As estratégias de filtragem também podem ser usadas para acelerar a execução do método GRASP. Essas estratégias consistem em aplicar a busca local apenas a soluções obtidas durante a fase construtiva, que sejam consideradas promissoras. Para isso, a cada iteração, submete-se a solução construída a um critério de qualidade definido em função do valor da melhor solução encontrada até o momento [3]. Mais detalhes desta metodologia podem ser encontrados em [46].

3.5 Variable Neighborhood Descent - VND

O Método de Descida em Vizinhança Variável, conhecido como VND (**Variable Neighborhood Descent**) é uma estratégia de refinamento, ou seja, é um método de busca local proposto por Mladenovic e Hansen (1997) que consiste em explorar o espaço de soluções através de trocas sistemáticas de estruturas de vizinhança, aceitando somente soluções de melhora da solução corrente e retornando à primeira estrutura quando uma solução melhor é encontrada [42]. O uso do VND está, em geral, subordinado ao uso de outras meta-heurísticas.

A mudança na estrutura de vizinhanças no VND é executada de maneira deter-

minística. Definindo-se $N_s^v = \{N_s^1, N_s^2, \ldots, N_s^r\}$ como o conjunto das vizinhanças de uma dada solução s, esta meta-heurística terá os seguintes procedimentos:

- Explora o espaço de soluções através de trocas sistemáticas de estruturas de vizinhança.
- Explora vizinhanças gradativamente mais "distantes".
- Sempre que há melhora em uma certa vizinhança, retorna-se à vizinhança "menos distante", ou seja, $N_s^{(1)}$ (o menor índice está diretamente associado ao tipo de estrutura de vizinhança dita "menos distante").

A troca sistemática de estrutura de vizinhança busca melhorar a qualidade da solução obtida nesta busca local, baseando-se nas seguintes afirmações:

- Não há nada que assegure que um ótimo local associado a um tipo de estrutura $N_s^{(k)}$ corresponda a um ótimo local com respeito a qualquer um dos demais tipos de estrutura, além disso, um ótimo global corresponde a um ótimo local para todos os demais tipos de estrutura de vizinhança.
- Observa-se que para uma variedade de problemas, ótimos locais com respeito a um tipo de estrutura de vizinhança encontram-se nas proximidades, em relação aos demais tipos de estruturas de vizinhança.
- Empiricamente observa-se que um ótimo local frequentemente fornece algum tipo de informação com respeito ao ótimo global. Este é o caso em que os ótimos locais e o ótimo global compartilham uma parcela relativamente considerável de variáveis com mesmo valor, portanto sugere-se uma investigação sistemática da vizinhança de um ótimo local até a obtenção de uma nova solução de melhor valor.

O comportamento desta meta-heurística está esquematizado na Figura 3.4, onde explora-se vizinhanças gradativamentes mais "distantes".

Como se observa na Figura 3.4, são consideradas r diferentes estruturas de vizinhança $\{N_s^{(1)}, N_s^{(2)}, \ldots, N_s^{(r)}\}$. A busca é iniciada a partir de uma solução s, usando-se a primeira

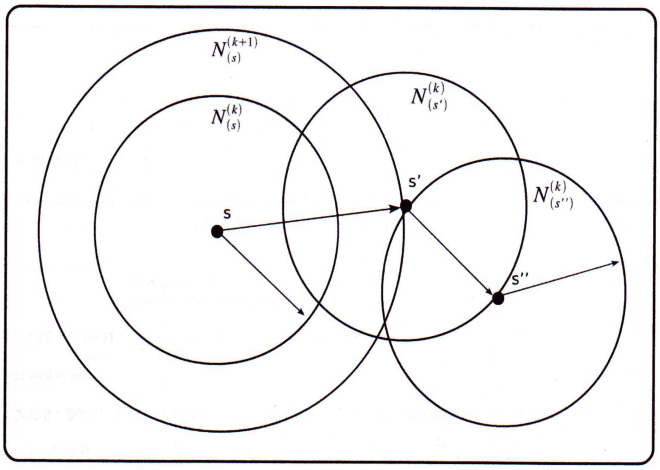

Figura 3.4: Comportamento VND.

estrutura de vizinhança $N_s^{(1)}$. Caso encontre um vizinho s' na vizinhança $N_s^{(1)}$ que seja melhor que s, atualiza-se s e repete-se o procedimento; caso contrário, faz-se a busca local na vizinhança $N_s^{(2)}$. Se a solução melhorar nesta estrutura, o VND move para a nova solução e retorna a usar $N_s^{(1)}$ outra vez, caso contrário, continua-se com $N_s^{(3)}$ e assim por diante. Se a última estrutura $N_s^{(r)}$ for aplicada e nenhuma melhoria adicional for possível, a solução representa um ótimo local em relação a todas as estruturas de vizinhança e o VND termina. Na Figura 3.5 temos o pseudocódigo do algoritmo VND.

Finalmente, observamos que a ordem em que as vizinhanças são exploradas é relevante no algoritmo, uma vez que as primeiras vizinhanças consideradas são mais exploradas que as últimas durante o procedimento.

3.6 Iterated Local Search

A meta-heurística ILS (Iterated Local Search) foi proposta por Lourenço, Martin e Stutze em 2002 [29]. Esta meta-heurística consiste, assim como o GRASP, na aplicação iterativa de um procedimento de busca local em uma solução inicial $\bar{x}$. A solução inicial é obtida através de uma heurística de construção ou de um procedimento aleatório. A busca local tenta, inicialmente, melhorar a solução construida, e posteriormente, melhorar ótimos locais perturbados. Desta forma, o desempenho do ILS depende da busca local e da

```
Procedimento VND
  1  Solução inicial s;
  2  Número de estruturas diferentes de vizinhança r;
  3  k ⟵ 1 (Tipo de estrutura de vizinhança)
  4  Enquanto k ≤ r faça
  5    Encontra o melhor vizinho s' ∈ N_k(s)
  6    se f(s') < f(s)
  7       então s ⟵ s'; k ⟵ 1;
  8    senão k ⟵ k + 1;
  9  Fim;
 10  Retorna(s);
Fim VND;
```

Figura 3.5: Pseudo código VND.

perturbação sofrida por um dado ótimo local. Depende ainda de um critério de aceitação, que estabelece se uma dada solução deve ou não substituir a solução corrente.

O ILS é um método de busca local que procura focar a busca não no espaço completo de soluções, mas em um subespaço definido por soluções que são ótimos locais. Ele baseia-se na simples idéia de que uma solução produzida por um procedimento de busca local pode ser melhorada gerando-se novas soluções de partida. Em outras palavras, pode-se dizer que o ILS é um método de busca local baseado na ideia de multi-start, o qual trabalha com várias soluções de partida.

Ao longo dos anos esta simples idéia vem sendo redescoberta pelos mais diversos trabalhos presentes na literatura. No método multi-start tradicional uma solução inicial é sempre gerada de forma totalmente aleatória, portanto não possui qualquer relação com as outras soluções de início, enquanto que o método ILS tenta melhorar a solução produzida pelo procedimento de busca local, provocando perturbações nos mínimos locais obtidos.

O objetivo no ILS é explorar S^*, um subconjunto do espaço de soluções do problema, S, contendo apenas ótimos locais, caminhando-se de um ótimo local s^* para outro

Procedimento `Iterated Local Search`

1 $s^0 \longleftarrow$ *Gera Solução Inicial ()*

2 $s^* \longleftarrow$ *Busca Local* (s^0);

3 **Enquanto** `Critério de Parada não satisfeito` **faça**;

4 $s^{*\prime} \longleftarrow$ *Perturbação (histórico,* s^**)* ;

5 $s^{*\prime} \longleftarrow$ *Busca Local* (s')

6 $s^* \longleftarrow$ *Critério de Aceitação (histórico,* s^*,$s^{*\prime}$);

7 **Fim Enquanto**;

8 **Retorna** (s^*);

Fim `Iterated Local Search`;

Figura 3.6: Pseudo código Iterated Local Search.

próximo a ele. Através da Figura 3.7 é possível observar o comportamento esperado com a aplicação da meta-heurística ILS. Partindo de um mínimo local s^*, aplica-se então o procedimento de perturbação obtendo uma solução s' pertencente a S. Em seguida, aplica-se um procedimento de busca local em s' e encontra-se um novo mínimo $s^{*\prime}$, o qual espera-se que seja uma solução com custo menor que as demais soluções já visitadas até o momento. Se $s^{*\prime}$ passar no teste de aceitação, ele se torna o próximo elemento da caminhada em S^*; senão, retorna-se a s^*.

Esse procedimento pode encontrar regiões promissoras no espaço de soluções. Para isso, a intensidade das perturbações não pode ser nem tão baixa e nem tão alta. Se a intensidade for baixa, s' pode pertencer na região de atração de s^* e com isso poucas novas soluções de S^* serão exploradas. Por outro lado, se a intensidade for muito alta, s' será uma solução aleatória e o algoritmo se comportará como um procedimento de múltiplos reinícios aleatórios.

Observa-se que o procedimento de perturbação da meta-heurística ILS dá-se efetivamente sempre que se está em um mínimo local.

Além dos procedimentos mais comuns exigidos por métodos baseados em busca local, outros três fazem parte do ILS, descritos a seguir:

- Perturbação, modifica uma solução s^* gerando uma solução intermediária s';

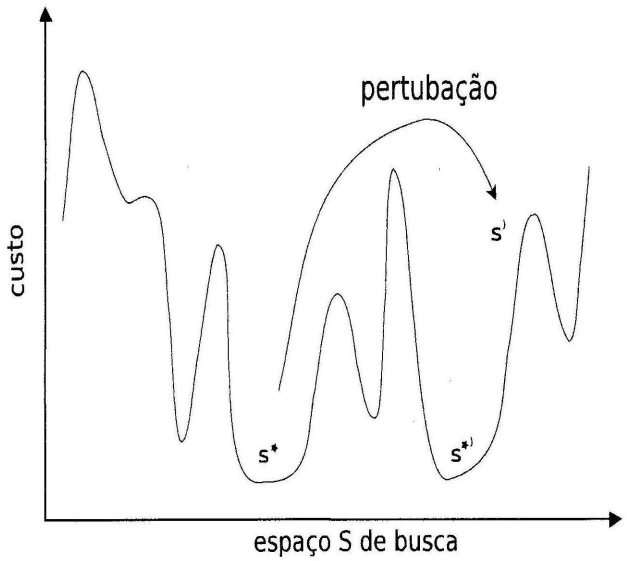

Figura 3.7: Comportamento ILS.

- Histórico, armazena informações com respeito à evolução alcançada ao longo do processo. Este item é pouco empregado na prática, pois geralmente é computacionalmente custoso;

- Critério de aceitação, o qual decide em que solução a próxima perturbação será aplicada.

Na Figura 3.8 temos o pseudocódigo do algoritmo Iterated Local Search.

Finalmente, observamos que perturbações determinísticas podem conduzir a formação de ciclos. Em geral torna-se a perturbação aleatória ou adaptativa para evitar a ciclagem.

3.6.1 Perturbação

No procedimento de perturbação, sucessivos movimentos aleatórios baseados na estrutura de vizinhança são empregados. A Figura 3.9 esquematiza o procedimento empregado.

Definimos que, perturbar uma dada solução é caminhar de forma aleatória de um vizinho para outro dentro de uma ou mais vizinhanças, sem que a solução construida seja examinada, ou seja, não estamos preocupados se a nova solução irá melhorar ou

Procedimento `Iterated Local Search`

1 $s^0 \longleftarrow$ *Gera Solução Inicial ()*

2 $s^* \longleftarrow$ *Busca Local* (s^0);

3 **Enquanto** `Critério de Parada não satisfeito` **faça**;

4 s**'** $\longleftarrow$ *Perturbação (histórico,* s^**)* ;

5 s^***'** $\longleftarrow$ *Busca Local (*s**'***)*

6 $s^* \longleftarrow$ *Critério de Aceitação (histórico,* s^*,s^***'***)*;

7 **Fim Enquanto**;

8 **Retorna** (s^*);

Fim `Iterated Local Search`;

Figura 3.8: Pseudo código Iterated Local Search.

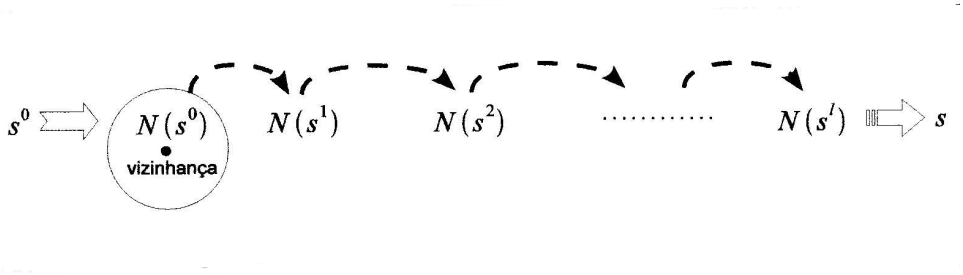

Figura 3.9: Comportamento Procedimento Perturbação.

piorar a função objetivo. A Figura 3.10 apresenta o algoritmo que descreve o método de perturbação proposto.

O procedimento apresentado toma como solução inicial s_0 e considera o nível P de perturbação, onde P representa o número de vizinhanças sucessivas nas quais poderemos caminhar.

3.6.2 Memória Local das Perturbações

Nesta Seção apresentamos uma variação de ILS, que emprega um mecanismo para o qual adota-se o termo Memória Local de Perturbações. Este consiste em uma memória de curto prazo, a qual impede que uma perturbação realize um movimento reverso a outro

Procedimento `Perturbação ILS`

1 `Solução inicial` s_0;
2 P `o nível de perturbação`;
3 $s \longleftarrow s_0$;
4 **Para** $i = 1, ..., P$ **faça**
5 $s' \longleftarrow$ `randomic` $\{N_s\}$
6 $s \longleftarrow s'$
7 **Fim**;

Fim `Perturbação ILS`;

Figura 3.10: Pseudo código Perturbação ILS.

que já havia sido realizado naquela sessão, o que levaria a uma exclusão mútua entre eles.

No algoritmo da Figura 3.11, podemos observar que a cada iteração, uma pertubação é incluida em um conjunto tabu T. Além disso observa-se também, que esta memória é mantida apenas localmente, ou seja, é mantida apenas até o término de uma Seção de pertubações, depois é completamente eliminado.

Considera-se T uma Lista, que comporta-se exatamente como uma fila circular, ou seja, os movimentos mais antigos são eliminados a medida em que novos movimentos são adicionados e a fila se encontra cheia.

Procedimento Perturbação com Memória ILS

1 Solução inicial s_0;

2 P o nível de perturbação;

3 $T \longleftarrow \{$ movimentos proibidos $\}$;

4 $s \longleftarrow s_0$;

5 **Para** $i = 1, ..., P$ **faça**;

6 $s' \longleftarrow$ randomic(s), $s' = s \oplus m \in \{N_s\}$ e $m \notin \{T\}$;

7 $s \longleftarrow s'$;

8 $T \longleftarrow T \cup m$;

9 **Fim**;

Fim Perturbação com Memória ILS;

Figura 3.11: Pseudo código Perturbação com Memória ILS.

Capítulo 4

Meta-heurísticas aplicadas ao PCVP

Este Capítulo descreve a metodologia utilizada para a resolução do Problema Circuito Virtual Privado.

Estudaremos a aplicação da meta-heurística GRASP ao PCVP, procedimento este que denominaremos GRASP-PCVP, a adaptação da meta-heurística VND ao GRASP-PCVP, procedimento este que denominaremos GRASP-VND-PCVP e a adaptação da meta-heurística ILS ao GRASP-PCVP, procedimento este que denominaremos GRASP-ILS-PCVP. Apresentaremos os procedimentos utilizados em cada fase do GRASP, do VND e do ILS para obter-se uma solução para o problema.

4.1 GRASP aplicado ao PCVP

Como foi mencionado no Capítulo 2, representaremos a rede de telecomunicações do problema PCVP por um grafo não direcionado $G = (V, E)$. Consideraremos $V = \{1, ..., n\}$, o conjunto dos nós da rede e $E = \{(i, j)|i, j \in V, i < j\}$ o conjunto de m arestas que conectam os nós da rede.

Para cada aresta (i, j), denotamos b_{ij} como o fluxo máximo permitido em Kbits/seg na aresta, c_{ij} como o máximo de pacotes que podem ser roteados simultaneamente através da aresta e d_{ij} a propagação do atraso associado a aresta.

Cada produto $k \in K$, $K = \{1, ..., p\}$ corresponde a um pacote a ser roteado associado a um par de nós origem-destino e a uma exigência de capacidade r_k (tamanho do pacote

em Kbits/seg) nos arcos que comporão a sua rota.

Nesta Seção descreveremos detalhadamente os procedimentos que constituirão a primeira heurística apresentada para solucionar o PCVP. Esta heurística é baseada na metaheurística GRASP e será denotada por GRASP-PCVP.

Observamos que o objetivo das heurísticas apresentadas é gerar uma solução viável para o PCVP, a qual será representada por um conjunto de rotas, ou seja, um conjunto de caminhos no grafo G, onde cada caminho corresponde a uma rota que leva um pacote de sua origem para seu destino.

A primeira heurística apresentada, o GRASP-PCVP, assemelha-se a que foi apresentada em [54] e será utilizada como base para as nossas comparações. A fase de construção, no entanto, difere um pouco da que foi apresentada no artigo, uma vez que encontramos lentidão na convergência do algoritmo em algumas instâncias de teste, com a proposta original. Observamos que esta lentidão se deve ao fato do algoritmo ser reiniciado toda vez que uma solução parcialmente construida inviabiliza o roteamento de todos os pacotes, conforme será explicado posteriormente. Destacamos a mudança na fase de construção proposta, como a primeira contribuição desta tese.

Como visto no Capítulo 3, o GRASP é um processo iterativo, em que cada iteração consiste em duas fases, uma fase de construção, em que uma solução viável é produzida, e uma fase de busca local, em que procuramos um ótimo local na vizinhança da solução construida. A seguir detalharemos os principais procedimentos das fases do algoritmo.

4.1.1 Fase Construção

O procedimento de *Construção* que podemos ver na Figura 3.2 do Capítulo 3, descreve também a fase de construção do GRASP-PCVP, a qual construirá de forma iterativa uma solução viável para o problema, adicionando um elemento de cada vez.

No caso do PCVP, consideramos um elemento de uma dada solução como uma rota associada a um único pacote (r_k), ou seja, um caminho no grafo G que leva um pacote de sua origem ao seu destino.

A cada iteração da fase de construção, a lista de candidatos C é inicialmente composta por todos os pacotes (r_k) que ainda não foram roteados, na solução já parcialmente

construida. Cria-se então uma lista de candidatos restrita, ou LCR, com um número fixo de elementos n_c. Em cada iteração, farão parte da LCR os n_c pacotes que não foram roteados que possuam as maiores demandas, ou seja, os n_c pacotes com maiores tamanhos r_k.

Propomos neste trabalho que a seleção do elemento da LCR que fará parte da solução seja baseada na proposta de Bresina,[12]. Um elemento l é selecionado aleatoriamente desta lista com probabilidade:

$$\pi(l) = \frac{r_l}{\sum_{k \in RLC} r_k}.$$

Observamos que ao dar prioridade a pacotes de maior tamanho inicialmente, procuramos evitar que estes pacotes não tenham mais caminhos viáveis para chegar aos seus destinos no momento em que forem ser roteados. Isso poderia acontecer se pacotes menores, roteados anteriormente na fase de construção, tivessem utilizado de forma não eficiente as capacidades das arestas na rede.

Uma vez que um determinado pacote l é selecionado da LCR, a rota que o leva de sua origem ao seu destino é calculada por um algoritmo de caminho mais curto, nos experimentos computacionais deste trabalho utilizamos para tal o algoritmo de *Dijkstral*, [58].

Nesta etapa do procedimento, o objetivo é calcular o caminho mais curto da origem de l ao seu destino num grafo $G' = (V', E')$ o qual é redefinido a cada iteração do algoritmo de construção. Neste grafo o conjunto de arestas E' é composto por todas as arestas (i, j) de E que ainda tenham capacidade de transmitir mais um pacote, ou seja, pelas arestas (i, j), nas quais o número de pacotes transmitidos na solução parcialmente construida é menor que c_{ij}. Já V' contem todos os nós de V adjacentes a pelo menos uma aresta de E'. Observamos que ao considerar G' ao invés de G, impedimos a construção de soluções que violem a restrição $\sum_{k \in K}(x_{ij}^k + x_{ji}^k) \leq c_{ij}$. Lembramos que a restrição que limita a capacidade das arestas com relação ao tamanho dos pacotes em b_{ij}, foi relaxada e não precisa ser verificada.

A distância entre dois nós i e j de G', ou seja, o peso ou o custo associado à aresta (i, j), com $i < j$, foi definida em [54] como o acréscimo na parcela da função objetivo, ϕ_{ij},

gerada pelo roteamento do pacote l com uso da aresta (i,j). A função ϕ_{ij} foi definida no Capítulo 2 e será apresentada novamente abaixo para maior clareza.

$$\begin{aligned}\phi_{ij}(x_{ij}^1,...,x_{ij}^p,x_{ji}^1,...,x_{ji}^p) &= (1-\delta).\phi_{ij}^d(x_{ij}^1,...,x_{ij}^p,x_{ji}^1,...,x_{ji}^p) \\ &+ \delta\phi_{ij}^b(x_{ij}^1,...,x_{ij}^p,x_{ji}^1,...,x_{ji}^p),\end{aligned} \tag{4.1}$$

onde

$$\phi_{ij}^d\ (x_{ij}^1,...,x_{ij}^p,x_{ji}^1,...,x_{ji}^p) = d_{ij}\sum \rho_k(x_{ij}^k + x_{ji}^k)$$

e

$$\phi_{ij}^b(x_{ij}^1,...,x_{ij}^p,x_{ji}^1,...,x_{ji}^p) = b_{ij}\begin{cases} u_{ij} & u_{ij} \in [0,\frac{1}{3}) \\ 3.u_{ij} - \frac{2}{3} & u_{ij} \in [\frac{1}{3},\frac{2}{3}) \\ 10.u_{ij} - \frac{16}{3} & u_{ij} \in [\frac{2}{3},\frac{9}{10}) \\ 70.u_{ij} - \frac{178}{3} & u_{ij} \in [\frac{9}{10},1) \\ 500.u_{ij} - \frac{1468}{3} & u_{ij} \in [1,\frac{11}{10}) \\ 5000.u_{ij} - \frac{16318}{3} & u_{ij} \in [\frac{11}{10},\infty). \end{cases} \tag{4.2}$$

O impacto na parcela da função objetivo do problema (ϕ_{ij}) causado por, adicionar à solução parcial, a rota do pacote l, supondo que esta rota contém a aresta (i,j), é medido através do incremento $\Delta\phi_{ij}$.

Mais precisamente, dado $\underline{K} \subseteq K$ o conjunto de pacotes previamente roteados, seja $\underline{K}_{ij} \subseteq \underline{K}$ o subconjunto dos pacotes que já estão sendo roteados através da aresta (i,j).

Analogamente, definimos $\overline{K} = \underline{K} \cup \{l\} \subseteq K$ como o novo conjunto de pacotes a serem roteados e $\overline{K}_{ij} = \underline{K}_{ij} \cup \{l\} \subseteq \overline{K}$ para toda aresta $(i,j) \in E'$. A definição deste conjunto supõe que o pacote l é roteado por cada uma das arestas do grafo, para assim, medir o impacto de sua utilização na rota de l, na função objetivo do problema.

Definimos também $\underline{x}_{ij}^h = 1$ ($\underline{x}_{ji}^h = 1$) se o pacote $h \in \underline{K}$ é roteado através da aresta (i,j) de i (j) para j (i). E $\underline{x}_{ij}^h = 0$ ($\overline{x}_{ji}^h = 0$) caso contrário.

Estabelecemos $\overline{x}_{ij}^h = 1$ para toda aresta $(i,j) \in E'$, supondo que o pacote $h \in \overline{K}$ é roteado através da aresta (i,j) de i para j. Observamos que o cálculo de $\Delta\phi_{ij}$ não seria

alterado se tivéssemos estabelecido $\overline{x}_{ji}^{h} = 1$, isto é, se tivéssemos suposto que o pacote h é roteado através da aresta (i, j) de j para i.

O custo associado com a aresta $(i, j) \in E'$ na solução atual é dado por:

$$\phi_{ij}(\underline{x}_{ij}^{1}, ..., \underline{x}_{ij}^{p}, \underline{x}_{ji}^{1}, ..., \underline{x}_{ji}^{p}).$$

Da mesma maneira, o custo associado com a aresta $(i, j) \in E'$ após o pacote l ter sido roteado será:

$$\phi_{ij}(\overline{x}_{ij}^{1}, ..., \overline{x}_{ij}^{p}, \overline{x}_{ji}^{1}, ..., \overline{x}_{ji}^{p}).$$

Com isto o incremento $\Delta\phi_{ij}$ associado ao roteamento do pacote $l \in K$ e à aresta $(i, j) \in E'$, é definido por:

$$\Delta\phi_{ij} = \phi_{ij}(\overline{x}_{ij}^{1}, ..., \overline{x}_{ij}^{p}, \overline{x}_{ji}^{1}, ..., \overline{x}_{ji}^{p}) - \phi_{ij}(\underline{x}_{ij}^{1}, ..., \underline{x}_{ij}^{p}, \underline{x}_{ji}^{1}, ..., \underline{x}_{ji}^{p}).$$

Tal incremento corresponderá então ao peso atribuido a cada arco $(i, j) \in E'$ a ser considerado pelo algoritmo de caminho mais curto na fase de construção e também na fase de busca local, como veremos adiante.

Por exemplo, seja $C = \{r_1, r_2, r_3, r_4, r_5\}$, onde $r_1 = 10$, $r_2 = 10$, $r_3 = 3$, $r_4 = 5$ e $r_5 = 15$. Suponha que a LCR será composta pelos três maiores pacotes, isto é, LCR $= \{r_1, r_2, r_5\}$. Um destes três pacotes será selecionado aleatoriamente da LCR. Suponhamos também que estamos iniciando o procedimento de construção da solução, isto é, nenhum pacote foi roteado até o momento, consequentemente, $\phi_{ij} = 0$. Além disto, r_2 tenha como origem o nó 1 e o destino o nó 6 e que este pacote tenha sido selecionado da LCR para ser roteado. Para que possamos definir o caminho mais curto para transmitir r_2 de sua origem a seu destino, precisamos calcular os pesos $(\Delta\phi_{ij})$ de cada aresta do grafo, como a solução corrente tem como $\phi_{ij} = 0$ então $\Delta\phi_{ij}$ será exatamente o valor da função objetivo (ϕ_{ij}).

Para o cálculo do peso $(\Delta\phi_{ij})$ suponhamos os seguintes dados de entrada o grafo [4.1] com suas arestas $(i < j)$ e seus respectivos dados de entrada os valores de b_{ij}, c_{ij} e d_{ij} conforme apresentado na tabela abaixo, 4.1.

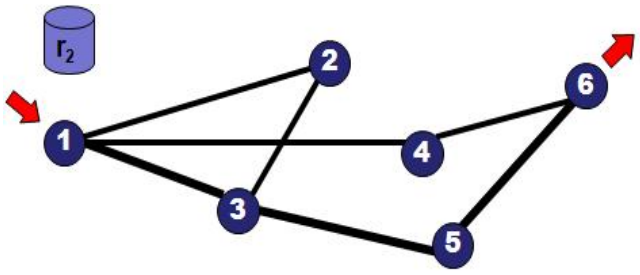

Figura 4.1: Exemplo: Grafo.

Tabela 4.1: Dados de entrada.

i	j	b_{ij}	c_{ij}	d_{ij}
1	2	20	3	1
1	3	10	1	1
1	4	5	2	1
2	3	15	3	1
3	5	10	4	1
4	6	10	3	1
5	6	10	1	1

Para cada aresta (i, j) do grafo [4.1] calcula-se ϕ_{ij} supondo que a aresta (i, j) poderia vir à fazer parte do caminho que levará o pacote r_2 de sua origem a seu destino. Para simplificar os cálculo do nosso exemplo, fixaremos $\delta = 1$, com isto estaremos considerando apenas o termo do congestionamento.

Tomemos a aresta $(1, 2)$ com ϕ_{12} inicial igual a zero e ϕ_{12} atual igual a b_{12} multiplicado pelo termo de penalização, isto é, para a aresta (1,2) calculamos $u_{12} = \frac{y_{12}}{b_{12}} = \frac{r_2(x_{12} + x_{21})}{b_{12}}$. Então temos $u_{12} = \frac{10}{20} = 0,5$ e com isto podemos avaliar a função de penalização. Nesta percebe-se que esta aresta estará sendo penalizada $3 * u_{12} - \frac{2}{3}$ pois $u_{12} \in [\frac{1}{3}, \frac{2}{3})$. Logo a ϕ_{12} atual será igual a $b_{12} = 20$ multiplicado pela penalização de $0,833$ e $\phi_{12} = 20 * 0,833 = 16,67$, consequentemente, $\Delta\phi_{12} = 16,67 - 0 = 16,67$. Ao final destes cálculos estariamos com todos os pesos $\Delta\phi_{ij}$ atribuidos a cada aresta do

grafo ($i < j$), tais pesos são enviados ao algoritmo de Dijkstral para que este encontre o menor caminho entre a aresta (1,6) levando em consideração que para cada aresta do grafo $\Delta\phi_{ij} = \Delta\phi_{ji}$.

Observe que para o grafo [4.1] exitem duas possibilidades para enviar o pacote r_2 do nó 1 para o nó 6:

- Caminho 1: $1 \rightarrow 4 \rightarrow 6$ onde $\phi_{14} + \phi_{46} = 22910,06$
- Caminho 2: $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6$ onde $\phi_{12} + \phi_{23} + \phi_{35} + \phi_{56} = 250,07$

O caminho que dará a menor contribuição na função objetivo será a escolhida, ou seja, o caminho escolhido para transmitir r_2 de sua origem ao seu destino neste exemplo, será $1 \rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow 6$ e estaremos fixando os valores das variaveis $x_{12} = 1$, $x_{23} = 1$, $x_{35} = 1$ e $x_{56} = 1$, todos os outros x_{ij} estarão fixos em zero. Logo a ϕ_{ij} inicial passa a ter o valor de $\phi_{ij} = 250,07$. Em seguida decrementa-se o valor dos c_{ij} utilizados para transmitir o pacote r_2. Estes procedimentos repetem-se até que todas os pacotes estejam associados a um caminho.

O procedimento de construção de uma solução, elemento a elemento, pode conduzir a pacotes não roteados, devido à impossibilidade de satisfazer a restrição $\sum_{k \in K}(x_{ij}^k + x_{ji}^k) \leq c_{ij}$. Em outras palavras, dada uma solução parcialmente construida por este procedimento, na qual um subconjunto dos pacotes já estejam roteadas, é possível que um dado pacote l, não tenha mais um caminho viável para ser roteado, devido à má utilização das arestas. Neste caso, o grafo G' é desconexo e não existe um caminho em G' que inicie na origem de l e termine em seu destino.

Ao verificamos que este problema ocorria em algumas instâncias de teste que utilizamos, propomos uma modificação na função objetivo que determina os elementos que entram na LCR na fase de construção.

Observamos que ao deparar com uma solução parcialmente construida que tornava inviável o roteamento das demais demandas, a fase de construção precisava ser reiniciada e a solução parcialmente construida era descartada levando a ineficiência do procedimento.

Desta forma, para minimizar este problema, adicionamos à função objetivo do problema um termo que mensura a utilização de cada aresta, através da razão $\frac{soma_{ij}}{c_{ij}}$,

onde $soma_{ij}$ representa a quantidade de vezes que a aresta (i, j) foi utilizada na solução parcialmente construida para transmitir os pacotes até então roteados e c_{ij} representa o máximo de pacotes que podem ser roteados através da aresta. Tal critério tem como objetivo privilegiar as arestas menos utilizadas durante o processo de construção da solução. Desta forma, a parcela ϕ_{ij} que compõe a função objetivo passa a ser definida na fase de construção, para ser utilizada no algoritmo de caminho mais curto, por:

$$\begin{aligned}\phi'_{ij}(x^1_{ij}, ..., x^p_{ij}, x^1_{ji}, ..., x^p_{ji}) &= \lambda_1 \phi^d_{ij}(x^1_{ij}, ..., x^p_{ij}, x^1_{ji}, ..., x^p_{ji}) \\ &+ \lambda_2 \phi^b_{ij}(x^1_{ij}, ..., x^p_{ij}, x^1_{ji}, ..., x^p_{ji}) \\ &+ \lambda_3 \frac{soma_{ij}}{c_{ij}}.\end{aligned}$$

Definimos λ_1, λ_2 e λ_3 de forma que $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

Finalmente, observamos que na fase da construção uma rota é determinadas a cada iteração. Ao final desta fase devemos ter então uma solução viável para o PCVP, com uma rota para cada pacote. Na próxima Seção descreveremos o procedimento de busca local que parte da solução construida, em busca por uma solução melhor.

4.1.2 Busca Local

Partindo de cada solução construida na fase de construção do GRASP, emprega-se um procedimento de pesquisa local na tentiva de melhorar a solução.

Cada solução construida na primeira fase pode ser vista como um conjunto de rotas, uma para cada pacote. O procedimento de busca local consiste em melhorar, quando possível, cada rota na solução corrente.

Dada uma solução inicial para o problema $\bar{x}$, define-se então a vizinhança $N_1(\bar{x})$ aonde será realizada a busca local, como o conjunto de soluções que diferem de $\bar{x}$ em exatamente uma rota.

Para obter-se os vizinhos de $\bar{x}$; para cada pacote $k \in K$, um de cada vez, remove-se as r_k unidades do fluxo de cada aresta em sua rota inicial. Em seguida, calcula-se o incremento $\Delta\phi_{ij}$ associado a cada aresta $(i, j) \in E$, considerando que ela será utilizada no roteamento deste pacote. Novamente consideramos apenas as arestas de E que ainda tenham capacidade de transmitir mais um pacote. Um novo trajeto é calculado como

uma tentativa de se achar uma solução viável melhor, para isto usa-se os incrementos $\Delta\phi_{ij}$ como medida de distância entre os nós i e j e aplica-se, como na fase de contrução, um algoritmo de caminho mais curto. Se o novo trajeto for diferente do inicial, a nova solução constitui um vizinho de $\bar{x}$ em $N_1(\bar{x})$.

Na fase de busca local, no cálculo do incremento $\Delta\phi_{ij}$, voltamos à função objetivo do problema ϕ_{ij}, como definida em (4.1).

Note que a restrição $\sum_{k\in K}(x_{ij}^k + x_{ji}^k) \leq c_{ij}$ já não pode conduzir a pacotes não roteados, pois caso não haja um caminho melhor que o inicial para rotear um dado pacote, a rota inicial será mantida.

Na tentativa de melhorar cada solução construida, avaliaremos todas as possibilidades de troca de caminhos, isto é, a possibilidade de transferir cada um dos pacotes para outro caminho.

Observamos que a vizinhança $N_1(\bar{x})$ tem no máximo p elementos, onde p é o número de pacotes a serem roteados. Toda vez que tentamos rotear um pacote por um caminho melhor, um novo vizinho é obtido, a menos que o algoritmo de caminho mais curto não consiga encontrar um caminho melhor que o inicial, caso em que a rota inicial é mantida. Notamos ainda que, desta forma, todos os vizinhos que obtemos para $\bar{x}$ têm custo menor que $\bar{x}$.

Consideramos a busca na vizinhança N_1 usando ambas as estratégia de *best-improving* e de *first-improving*. No caso da estratratégia *best-improving* todos os vizinhos são investigados e a solução inicial é substituida pela melhor solução encontrada. Na estratégia de *first-improving* troca-se a solução inicial pelo primeiro vizinho que tenha custo menor. Como todos os vizinhos que geramos pelo nosso procedimento tem custo menor que a solução de partida, na estratégia de *first-improving* paramos no primeiro vizinho obtido, atualizando a solução inicial.

Este procedimento é repetido até que nenhum trajeto melhor possa ser encontrado, quando finalmente temos um ótimo local relativo a vizinhança N_1 para o problema.

Na Figura 3.3, apresentada no Capítulo 3, podemos ver um pseudo-códico que representa a fase de busca local.

4.2 GRASP com VND aplicado ao PCVP

Nesta Seção propomos a substituição da fase de busca local da meta-heurística GRASP pelo VND, apresentadas no Capítulo 3, de forma a construir uma heurística híbrida para solucionar o PCVP, denotada por GRASP-VND-PCVP.

Propomos a substituição da busca local do GRASP-PCVP, descrita na Seção 4.1.2, por uma heurística baseada na estrutura do VND, onde duas diferentes estruturas de vizinhança são consideradas.

A primeira é a vizinhança N_1, já descrita na Seção 4.1.2. A segunda vizinhança será denotada por N_2. Dada uma solução $\bar{x}$, define-se a vizinhança $N_2(\bar{x})$, como o conjunto de soluções que diferem de $\bar{x}$ em exatamente duas rotas.

Para obter-se os vizinhos de $\bar{x}$ em $N_2(\bar{x})$; para cada par de pacotes $k_1, k_2 \in K$, um par de cada vez, removem-se r_{k_1} e r_{k_2} unidades do fluxo de cada aresta de suas rotas iniciais. Em seguida, calcula-se o incremento $\Delta\phi_{ij}$ associado a cada aresta $(i, j) \in E$, considerando que ela será utilizada no roteamento do pacote k_1. Uma nova rota, é calculada na tentativa de se achar uma solução viável melhor. Para isto usa-se os incrementos $\Delta\phi_{ij}$ como medida de distância entre os nós i e j e aplica-se, mais uma vez, um algoritmo de caminho mais curto. Considera-se então a nova rota para o pacote k_1 e incorpora-se a mesma, à solução. Em seguida, repete-se o procedimento para o pacote k_2. Se a soma dos custos das duas novas rotas, para k_1 e k_2, for menor que a soma dos custos de suas rotas na solução inicial, a nova solução constitui um vizinho de $\bar{x}$ em $N_2(\bar{x})$.

Consideramos a seguinte ordenação dos pacotes para a escolha das duplas de pacotes a serem novamente roteados. Primeiramente, condideramos $k_1, k_2, \ldots, k_p$, de forma que $r_{k_1} \geq r_{k_2} \geq \ldots \geq r_{k_p}$. Escolhemos as duplas de forma determinística, na seguinte ordem: $(k_1, k_2), (k_1, k_3), \ldots, (k_1, k_p), \ldots, (k_{p-1}, k_p)$.

Observamos que devido ao tamanho desta vizinhança ser maior que o tamanho de N_1, da ordem de p^2, a busca na vizinhança N_2 é interrompida depois que um número máximo de vizinhos foi analisado sem haver melhora na solução corrente.

Cada iteração do algoritmo GRASP-VND-PCVP inicia com a fase de construção do GRASP, conforme foi descrito na Seção 4.1.1, até que tenhamos uma solução viável para

o PCVP. Em seguida executa-se a heurística VND como um método de busca local, terminando após explorar todos os vizinhos em N_1 e um número máximo de vizinhos N_2, sem ter sucesso.

4.3 GRASP com ILS aplicado ao PCVP

Apresentamos nesta Seção uma nova proposta de heurística híbrida para resolver o PCVP. Nesta heurística combinamos a meta-heurística GRASP e ILS.

Repetimos abaixo a Figura já apresentada no Capítulo 3 com o pseudo código do algoritmo ILS, com o intuito de fazer a correspondência entre cada etapa do algoritmo geral para o ILS e o procedimento heurístico que propomos.

Procedimento `Iterated Local Search`

1 $s^0 \longleftarrow$ *Gera Solução Inicial ()*

2 $s^* \longleftarrow$ *Busca Local* (s^0);

3 **Enquanto** `Critério de Parada não satisfeito` **faça**;

4 s**'** $\longleftarrow$ *Perturbação (histórico,* s^**)* ;

5 s^***'** $\longleftarrow$ *Busca Local (*s**'***)*

6 $s^* \longleftarrow$ *Critério de Aceitação (histórico,* s^*,s^***'**);

7 **Fim Enquanto**;

8 **Retorna** (s^*);

Fim `Iterated Local Search`;

Figura 4.2: Pseudo código Iterated Local Search.

Propomos para resolver o PCVP a heurística GRASP-ILS-PCVP, cujos passos em cada iteração, são exatamente os apresentados no algoritmo da Figura 4.2. Na heurística proposta gera-se a solução inicial no passo 1 do algoritmo com a fase de construção do GRASP, descrita na Seção 4.1.1 e em seguida aplica-se à solução construida a busca local do GRASP, descrita na Seção 4.1.2. Desta forma, inicia-se cada iteração da heurística proposta com uma iteração do GRASP-PCVP.

Em seguida repete-se sucessivas iterações, nas quais perturba-se o ótimo local corrente

e realiza-se uma nova busca local na vizinhança da solução perturbada, isto é, retira-se aleatóriamente um r_k da solução corrente, onde na primeira vez será a solução construida pelo GRASP-PCVP, e tenta-se encontrar um novo caminho para se transmitir o mesmo pacote, sem se preocupar se este novo caminho irá contribuir mais ou menos do que o caminho inicial. Esta nova escolha do caminho será a mesma empregada na fase de construção do GRASP-PCVP.

Aplica-se uma busca local em N_1 na solução perturbada em N_1, caso encontre-se uma solução melhor, a solução inicial (melhor solução conhecida) é atualizada e uma nova perturbação é realizada em N_1, caso contrário passamos para uma nova vizinhança N_2 e repete-se o procedimento de perturbação em N_2. Realiza-se a busca local em N_2, caso haja melhora, atualiza-se a melhor solução conhecida e volta-se para N_1, caso contrário permanessemos em N_2.

Baseando-se na melhor solução conhecida decide-se se a próxima perturbação será aplicada na nova solução ou na melhor solução conhecida. Se por este critério decide-se pela melhor solução conhecida, o nível de perturbação é incrementado, caso contrário ele é reiniciado. Optamos por estabelecer o nível máximo de perturbação P igual a 2.

Desta forma, inicia-se cada iteração no passo 3 do algoritmo da Figura 4.2 com um ótimo local s^*, melhor solução conhecida, e perturbamos esta solução dentro de uma das vizinhanças, inicialmente em N_1, encontrando s'. Move-se para esse vizinho, sem o compromisso deste melhorar a solução. Executa-se então uma busca local na vizinhança $N_1(s')$. Se a solução encontrada na busca local for melhor que a melhor solução conhecida (s^* ou s'), atualizamos a solução, e voltamos para a fase de perturbação. Caso contrário, mantemos a melhor solução s^* ou s' e repetimos o procedimento utilizando a vizinhança N_2.

Ao realizar a busca na vizinhança N_1 utilizamos novamente a estratégia *first improving*, descrita anteriormente. Na busca realizada na vizinhança N_2, novamente utilizamos um número máximo de iterações sem melhora como critério de parada da busca, quando concluímos que a solução corrente é um ótimo local. Esta busca é equivalente a que foi realizada no algoritmo GRASP-VND-PCVP com apenas uma alteração no que diz respeito a escolha da dupla de pacotes, que serão novamente roteados na busca pela vi-

zinhança N_2. Nesta heurística, esta escolha é aleatória, e não determinística. O objetivo desta estratégia é aumentar a diversidade das soluções.

Capítulo 5

Experimentos Computacionais

Neste Capítulo apresentamos os resultados computacionais obtidos com os testes realizados com as heurísticas proposta nesta tese. O objetivo dos testes é comparar as metodologias entre si de forma a avaliar se um maior esforço realizado na busca local dos algoritmos é compensado por soluções de melhor qualidade.

Todos experimentos desta etapa do trabalho foram realizados no laboratório de otimização do PESC/COPPE/UFRJ. A máquina utilizada no laboratório de otimização possui a seguinte configuração: processador Intel R Xeon R X5472 3.00GHz, 16 gigabytes de memória RAM, sistema operacional Linux. Para a compilação dos algoritmos foi utilizado o compilador gcc (Gnu Compiler Collection).

Para resolver o problema PCVP de forma exata e também os subproblemas gerados no método híbrido a ser descrito no Capítulo 6, foi utilizado o pacote Dash Optimization, modulos: Xpress-Optimizer e Xpress-BCL, [69].

Na Tabela 5.1 descrevemos as heurísticas implementadas: GRASP-PCVP, GRASP-VNP-PCVP e GRASP-ILS-PCVP.

5.1 O Conjunto de Instâncias-Teste

Nos experimentos computacionais realizados no decorrer desse trabalho foram utilizadas as instâncias descritas na Tabela 5.2.

Tabela 5.1: Metodologia empregada.

Heurística	Descrição
GRASP-PCVP	GRASP aplicado ao PCVP
GRASP-VND-PCVP	GRASP com busca local VND aplicado ao PCVP
GRASP-ILS-PCVP	GRASP com ILS, utilizando busca local do VND aplicado ao PCVP

As instâncias *hierarquia de 2 níveis* são formadas por uma rede criada artificialmente *synthetic network*, usando o gerador GT-ITM, que se baseia no modelo de Calvert et al. e Zegura et al., conforme pode ser visto em [54]. Neste conjunto de instâncias , as arestas são de dois tipos:

- Acessos locais
- Acessos para longas distâncias

Considera-se que arestas do mesmo tipo têm capacidades iguais. As arestas de acesso local têm capacidades menores que as arestas de longa distância.

As instâncias *Waxman*, são formadas por nós que são pontos uniformente distribuidos, onde a probabilidade de ter uma aresta entre dois nós u e v é dado por $\eta e^{(\frac{-\delta(u,v)}{2\theta})}$, onde η é o parâmetro usado para controlar a densidade do grafo, $\delta(u, v)$ é a distância Euclidiana de u e v e θ é a distância máxima entre todos os nós , [54, 13, 14].

Todas as aresta têm a mesma capacidade. As demandas são tais que diferentes nós tem diferentes níveis de atividades, modelando os pontos com mais atividade da rede. As demandas são relativamente maiores entre pares de nós mais próximos.

As instâncias *Frame-relay* foram criadas artificialmente por outro gerador como pode ser visto em [47], tais redes têm características mais similares às redes de *frame-relay* reais. De acordo com [54], essas instâncias são as maiores, até hoje, encontradas na literatura. Tais instâncias foram disponibilizadas em [68].

Na Tabela 5.2 mostramos, para cada instância, o nome, o tipo de rede, o número de nós na rede ($|V|$), o número de arestas ($|E|$) e o número de pacotes de dados que serão

roteados ($|K|$).

Tabela 5.2: Instâncias obtidas da literatura.

	Intância	Tipo de Rede	$\|V\|$	$\|E\|$	$\|K\|$
1	hier50a	hierarquia de 2 níveis	50	148	2450
2	fr250	Frame - relay	60	344	250
3	fr500	Frame - relay	60	453	500
4	fr750	Frame - relay	60	498	750
5	fr1000	Frame - relay	60	518	1000
6	fr1250	Frame - relay	60	535	1250
7	wax50a	Waxman	50	230	2220

Para todas as instâncias apresentadas na Tabela 5.2, definimos o que denotamos por soluções alvo. Para gerar a solução alvo de cada instância, aplicamos os procedimentos GRASP-PCVP, GRASP-VND-PCVP e GRASP-ILS-PCVP para resolvê-la, onde o critério de parada considerado foi de 100 iterações. Cada procedimento foi aplicado 20 vezes a cada instância. Definimos a melhor solução encontrada pelos procedimentos GRASP-PCVP, GRASP-VND-PCVP e GRASP-ILS-PCVP como s_1^*, s_2^*, s_3^*, respectivamente.

Considerando que o mais lento dos procedimentos é capaz de alcançar uma solução alvo em tempo computacional razoável, para cada instância definimos como solução alvo 10% da pior solução dentre $\{s_1^*, s_2^*, s_3^*\}$.

Além das instâncias obtidas da literatura, consideramos em nossos testes mais cinco instâncias de pequeno porte geradas a partir dos dados da instância $fr250$. Para gerar estas instâncias, consideramos uma solução viável para a instância $fr250$ e fixamos as rotas de 230 pacotes na rota estabelecida pela solução. As rotas dos outros 20 pacotes deverão ser definidas pela solução do problema. A escolha das 20 rotas que definem a instância gerada é realizada de forma aleatória. Observamos que as capacidades das arestas da rede foram recalculadas, levando-se em conta os pacotes que já passavam por

elas. Todas estas instâncias são definidas em uma rede com 60 nós, 344 arestas e 250 pacotes. O número de demandas que devem ser roteadas é igual a 20.

Obtivemos a solução ótima destas instâncias, utilizando o pacote Xpress com seus parâmetros *default*. Apresentamos na Tabela 5.3, o valor da solução ótima encontrada (z^*) e o tempo de processamento do XPRESS em segundos, para cada instância.

Tabela 5.3: Instâncias de pequeno porte.

Instância	z^*	Tempo
Inst 1	3.97364e+06	24
Inst 2	4.01563e+06	36
Inst 3	6,24477e+4	23
Inst 4	4.91663e+04	15
Inst 5	4.03692e+06	24

Todos os testes nesta tese foram realizados considerando o parâmetro $\delta = 1$, o qual está presente na função objetivo do problema PCVP. Ao estabelecer $\delta = 1$, estamos considerando apenas a minimização do congestionamento na rede. A componente da função objetivo relacionada ao atraso, neste caso, é excluída. Consideramos $\delta = 1$, com o intuito de acompanhar os testes realizados em [54].

A seguir descrevemos os resultados encontrados para cada experimento.

5.2 Testes com instâncias de pequeno porte

Os primeiros testes descritos neste trabalho têm a finalidade de validar nossas propostas. Nestes testes verificamos o desempenho das heurísticas propostas nas instâncias menores apresentadas na Tabela 5.3, comparando os resultados obtidos pelas heurísticas com a solução ótima dos problemas.

Executamos cada heurística 10 vezes considerando o critério de parada de 10 iterações em todas elas. Dado z^*, a solução ótima de uma dada instância e $\bar{z}$, uma solução encontrada por alguma das heurísticas, definimos:

$$\text{GAP}\% = \frac{\bar{z} - z^*}{\bar{z}} \times 100. \tag{5.1}$$

Na Tabela 5.4 apresentamos para cada heurística proposta e para cada instância da Tabela 5.3, o menor GAP%, obtido com a melhor solução encontrada nas 10 execuções da heurística , apresentamos a média dos valores de GAP% obtidos nas 10 execuções (MGAP%) e apresentamos o tempo médio de processamento dos algoritmos em segundos. Observamos que todas as heurísticas propostas chegam na solução ótima destes problemas em quase todas as execuções. O tempo de processamento é menor que 2 segundos para todas elas.

Tabela 5.4: Desempenho das heurísticas em instâncias pequenas.

	GRASP			GRASP-VND			GRASP-ILS-PCVP		
Instância	GAP%	MGAP%	Tempo	GAP%	MGAP%	Tempo	GAP%	MGAP%	Tempo
Inst 1	0	0,0008	0,4	0	0,0008	1	0	0,0008	0,6
Inst 2	0	0,02	0,4	0	0,02	1,2	0	0,02	0,5
Inst 3	0	0,0046	0,4	0	0,0046	1,2	0	0,0046	0,5
Inst 4	0	0,00001	0,4	0	0	1,2	0	0	0,6
Inst 5	0	0,19	0,4	0	0,15	1,2	0	0,15	0,5

5.3 Testes com instâncias da literatura

Reportamos nesta Seção os resultados obtidos quando aplicamos as heurísticas propostas às instâncias obtidas da literatura.

Nos testes realizados com estas instâncias, executamos cada heurística 20 vezes utilizando dois dferentes critérios de paradas.

Primeiramente executamos as heurísticas 20 vezes cada com o critério de parada definido por um número de iterações igual 100. Em seguida executamos novamente as heurísticas 20 vezes cada, considerando como critério de parada a obtenção da solução alvo de cada instância.

Observamos que o critério dado pelo número de iterações nos dá informação a qualidade das soluções sem uma preocupação maior com o tempo, enquanto que o segundo critério de parada relacionado a uma solução alvo nos dá mais informação sobre a taxa convergência dos métodos, nos permitindo verificar qual dos métodos converge mais rapidamente para uma boa solução para o problema.

Uma vez que não conhecemos a solução ótima das instâncias retiradas da literatura, optamos por substituir, na definição de GAP% em (5.1) o valor da solução ótima da instância z^* pelo valor da melhor solução obtida para a instância em todos os testes que realizamos, com as três heurísticas proposta.

Na Tabela 5.5 apresentamos resultados análogos aos que foram apresentados na Tabela 5.4 para instâncias pequenas, o GAP% associado à melhor solução obtida nas primeiras 20 execuções de cada heurística, que considera como critério de parada o número de iterações, a média dos 20 valores obtidos para GAP% (MGAP%), o tempo médio de processamento das heurísticas considerando o número fixo de iterações (tempo médio 100 iterações) e o tempo médio de processamento das heurísticas para atingir a solução alvo (tempo médio solução alvo). Todos os tempos são apresentados em segundos.

Observa-se dos resultados na Tabela 5.5 que as três heurísticas propostas chegaram em 100 iterações a menos de 2% da melhor solução conhecida em cinco das sete instâncias consideradas, no entanto foi a heurística baseada na meta-heurística ILS a responsável pela obtenção da melhor solução conhecida em todas as instâncias consideradas. Considerando a média dos valores de GAP%, verificamos que GRASP-ILS-PCVP é também a heurística mais robusta, tendo apresentado os menores valores para a medida.

Considerando-se a qualidade das soluções obtidas, portanto, a heurística GRASP-ILS-PCVP gerou as melhores soluções, seguida pelas outras duas, que apresentaram comportamentos bastante parecidos. Verificamos apenas uma pequena melhoria na qualidade das soluções apresentadas pela heurística GRASP-VND-PCVP em relação ao GRASP-PCVP. Deve-se mencionar que a pequena melhoria na qualidade das soluções obtidas já pelo GRASP-VND-PCVP em relação à heurística mais pura GRASP-PCVP é devida à busca local mais intensa que considera duas vizinhanças, a vizinhança N_1, considerada

também pelo GRASP-PCVP e a vizinhança mais abrangente N_2. Apesar desta busca local mais intensa já melhorar ligeiramente a qualidade dos resultados obtidos a maior melhoria é, de fato, observada no GRASP-ILS-PCVP. Acreditamos, portanto, que as perturbações do ILS contribuíram de forma decisiva para que a busca não fique presa em ótimos locais de baixa qualidade.

Por outro lado, a busca local mais intensa e o procedimento de perturbação levam a um aumento considerável no esforço computacional, que pode ser observado através dos altos tempos médios de processamento da heurística GRASP-ILS-PCVP quando comparados aos tempos das outras duas. Este aumento no tempo do GRASP-ILS-PCVP já era esperado, uma vez que optamos por um número fixo de iterações no primeiro experimento. Ao observarmos a última coluna da tabela, podemos analisar a velocidade de convergência das heurísticas para uma solução sub-ótima e concluímos, neste caso, que heurística foram bem competitivas.

5.3.1 Análise de distribuição de probabilidade empírica

Neste experimento, procurou-se verificar a velocidade de convergência de cada método para soluções com uma medida mínima de qualidade. O experimento considera a distribuição empírica de probabilidade de se atingir um determinado valor alvo da função objetivo, ou seja, considera-se o tempo necessário para produção de soluções com custo menor ou igual ao valor alvo.

Conforme mencionado no início deste capítulo, os valores alvo considerados nos experimentos deste trabalho foram definidos de modo a permitir que mesmo o mais lento dos algoritmos termine em um tempo computacional razoável. Nesse experimento foram avaliadas as metodologias GRASP-PCVP, GRASP-PCVP-VND e GRASP-ILS-PCVP. Para a análise, foi computado o tempo necessário para realizar 100 execuções de cada heurística.

Para cada heurística, os 100 resultados que representam o tempo de processamento da heurística para atingir a solução alvo nas 100 execuções são postos em ordem crescente.

Associa-se então o i-ésimo menor tempo de execução t_i a probabilidade

$$p_i = \frac{(i - 1/2)}{100},$$

para todo $i = 1, \ldots, 100$.

Gerando os pontos

$$v_i = (t_i, p_i), 1 \leq i \leq 100,$$

que são então plotados em um gráfico de distribuição de probabilidade empírica para a heurística. Este gráfico nos fornece empiricamente, para cada tempo, a probabilidade da heurística atingir a solução alvo naquele tempo.

Esta metodologia de análise foi proposta em [4] e tem sido bastante usada na literatura.

Podemos observar no gráfico da Figura 5.1, a comparação entre os resultados encontrados pelas heurísticas GRASP-PCVP (G), GRASP-VND-PCVP (GVND) e GRASP-ILS-PCVP (GILS), para a instância $fr250$, a qual utilizamos para exemplificação. Observa-se que por meio do gráfico, que para esta instância, a metodologia ILS acrescentou uma maior contribuição para a metodologia GRASP, pois esta se destaca com as maiores probabilidades de se encontrar a solução alvo em tempos computacionais menores. Além disto, pode-se perceber também que há pouca diferença entre as metodologias GRASP-PCVP e GRASP-VND-PCVP para este exemplo.

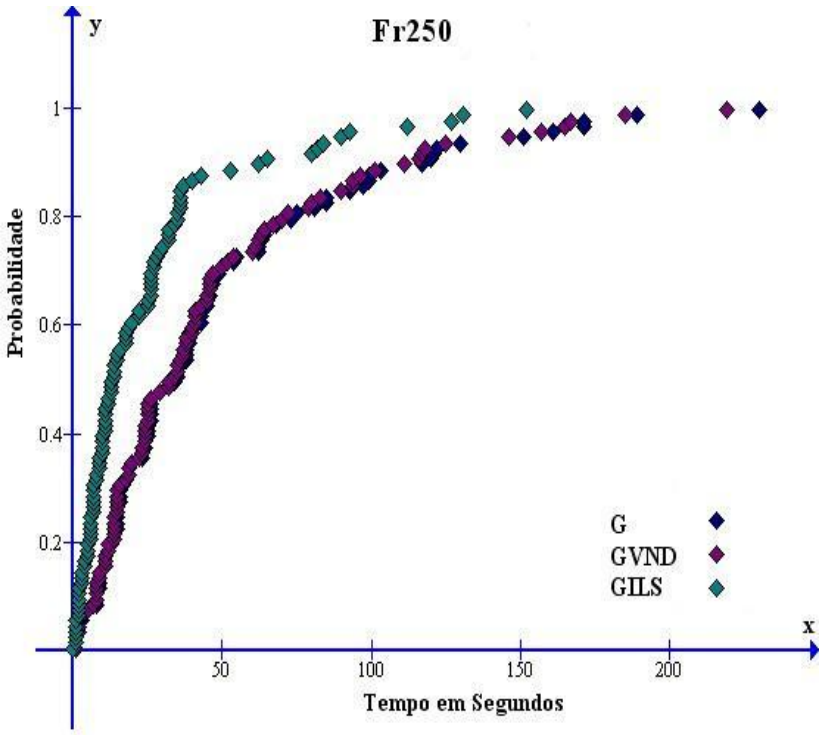

Figura 5.1: Distribuição empírica de probabilidade para a instância $fr250$.

Tabela 5.5: Desempenho das heurísticas em instâncias da literatura.

Intância	Heurística	GAP(%)	MGAP(%)	Tempo médio 100 iterações	Tempo médio solução alvo
	GRASP-PCVP	0,53	9,80	56,55	33,75
fr250	GRASP-VND-PCVP	0,53	8,76	89,05	54,70
	GRASP-ILS-PCVP	0	8,72	595,05	189,70
	GRASP-PCVP	0,63	16,26	223,85	149,80
fr500	GRASP-VND-PCVP	0,67	14,68	413,10	285,40
	GRASP-ILS-PCVP	0	11,00	1013,45	612,70
	GRASP-PCVP	0,42	1,42	420,5	255,05
fr750	GRASP-VND-PCVP	0,35	1,42	535,35	326,25
	GRASP-ILS-PCVP	0	0,47	472,40	287,65
	GRASP-PCVP	0,57	0,92	1418,80	516,55
hier50a	GRASP-VND-PCVP	0,33	0,78	1422,85	517,50
	GRASP-ILS-PCVP	0	0,43	1445,10	525,85
	GRASP-PCVP	1,82	7,00	718,15	395,50
fr1000	GRASP-VND-PCVP	2,00	6,94	744,50	408,05
	GRASP-ILS-PCVP	0	5,69	757,10	414,95
	GRASP-PCVP	21,30	25,83	696,05	399,65
fr1250	GRASP-VND-PCVP	21,29	25,84	1061,35	603,85
	GRASP-ILS-PCVP	0	5,32	1063,80	605,20
	GRASP-PCVP	8,23	9,08	1086,10	647,85
wax50a	GRASP-VND-PCVP	7,49	8,91	1115,20	659,00
	GRASP-ILS-PCVP	0	6,32	1176,95	671,35

Capítulo 6

Método Híbrido Meta-Heurística/Exato

6.1 Introdução

Neste trabalho chamamos de métodos exatos os algoritmos que resolvem os problemas modelados como problemas de programação inteira de modo exato, ou seja, que fornecem certificados de otimalidade para as soluções obtidas, como um limite superior para o *gap* de dualidade dos problemas. Do ponto de vista teórico, este limite pode ser tão pequeno quanto se queira e converge para zero na solução ótima do problema. Na classe de métodos exatos encontram-se, por exemplo, os métodos *branch-and-bound* e *branch-and-cut*, implementados em quase todos os solvers comerciais que se propõem a resolver estes problemas, como o XPRESS e o CPLEX.

Em [54], o PCVP foi formulado como um problema de programação linear inteira, para o qual a função objetivo é modelada por partes, para representar o aumento não linear da penalidade imposta à violação da capacidade máxima de cada aresta da rede no que diz respeito à capacidade da largura de banda por ele transmitida.

Neste Capítulo reapresentamos o modelo matemático já apresentado no Capítulo 2 e remodelamos o problema como um problema de programação linear inteira com função objetivo linear, utilizando a forma clássica que transforma um modelo linear por partes em um modelo linear contínuo.

Observamos que ao tentar resolver a menor instância contida no nosso conjunto de problemas teste retirados da literatura com a aplicação do solver XPRESS, no ambiente computacional descrito na Seção de resultados numéricos do Capítulo anterior, após quatro dias de processamento, o solver não chegou à solução ótima do problema.

Propomos então a utilização do solver e dos métodos exatos (*branch-and-bound*) não para resolver as instâncias, mas para melhorar as soluções viáveis obtidas pelas meta-heurísticas, através da solução de subproblemas gerados a partir do modelo matemático proposto para o PCVP e destas soluções viáveis.

Os métodos desenvolvidos neste Capítulo são denominados métodos híbridos meta-heurística/exato, uma vez que utilizam conjuntamente métodos heurísticos e exatos (*branch-and-bound*). Estes métodos tiveram como motivação inicial a idéia apresentada no procedimento denominado *local branching*, proposto por Fishetti e Lodi [27], no qual eles incorporam a idéia de busca local, utilizada pelas meta-heurísticas, aos métodos exatos.

Na próxima Seção apresentaremos a reformulação do modelo matemático para o PCVP e na Seção seguinte descreveremos o procedimento *local branching*. Em seguida apresentaremos a proposta de um método híbrido meta-heurística/exato para resolver o PCVP e ao final do Capítulo apresentaremos os resultados computacionais obtidos com a sua implementação.

6.2 Modelo matemático para o PCVP

O seguinte modelo matemático foi apresentado no Capítulo 2 para o PCVP:

$$\text{Minimizar } \phi(x) = \sum_{(i,j)\in E, i<j} \phi_{ij}\ (x_{ij}^1, ..., x_{ij}^p, x_{ji}^1, ..., x_{ji}^p)$$

sujeito a:

$$\begin{aligned}
&\sum_{k\in K} r_k(x_{ij}^k + x_{ji}^k) \leq b_{ij}, \forall (i,j) \in E, i < j \\
&\sum_{k\in K} (x_{ij}^k + x_{ji}^k) \leq c_{ij}, \forall (i,j) \in E, i < j \\
&\sum_{(i,j)\in E} x_{ij}^k - \sum_{(i,j)\in E} x_{ji}^k = a_i^k, \forall i \in V, \forall k \in K \\
&x_{ij}^k \in \{0,1\}, \forall (i,j) \in E, \forall k \in K,
\end{aligned} \tag{6.1}$$

onde a parcela da função objetivo ϕ_{ij} associada a cada aresta $(i,j) \in E$ com $i < j$, é definida como uma combinação linear das componentes do atraso e do congestionamento da aresta, isto é:

$$\phi_{ij}(x_{ij}^1, ..., x_{ij}^p, x_{ji}^1, ..., x_{ji}^p) = (1-\delta).\phi_{ij}^d(x_{ij}^1, ..., x_{ij}^p, x_{ji}^1, ..., x_{ji}^p) + \delta.\phi_{ij}^b(x_{ij}^1, ..., x_{ij}^p, x_{ji}^1, ..., x_{ji}^p) \tag{6.2}$$

Foi mostrado no Capítulo 2 que a componente de congestionamento, ϕ_{ij}^b, depende da utilização das taxas u_{ij}, que foram definidas como $u_{ij} = \dfrac{y_{ij}}{b_{ij}}$ para cada aresta $(i,j) \in E$ com $i < j$. Lembremos que $y_{ij} = \sum_{k\in K} r_k(x_{ij}^k + x_{ji}^k)$ é o total do fluxo que atravessa a aresta (i,j). Conforme descrito anteriormente, a componente ϕ_{ij}^b é representada pela função linear por partes proposta por Fortz e Trorup (2000) [28]. Ela penaliza cada vez mais os fluxos que se aproximam ou que violam os limites de capacidade:

$$\phi_{ij}^b(x_{ij}^1, ..., x_{ij}^p, x_{ji}^1, ..., x_{ji}^p) = b_{ij} * \begin{cases} u_{ij} & u_{ij} \in [0, \frac{1}{3}) \\ 3.u_{ij} - \frac{2}{3} & u_{ij} \in [\frac{1}{3}, \frac{2}{3}) \\ 10.u_{ij} - \frac{16}{3} & u_{ij} \in [\frac{2}{3}, \frac{9}{10}) \\ 70.u_{ij} - \frac{178}{3} & u_{ij} \in [\frac{9}{10}, 1) \\ 500.u_{ij} - \frac{1468}{3} & u_{ij} \in [1, \frac{11}{10}) \\ 5000.u_{ij} - \frac{16318}{3} & u_{ij} \in [\frac{11}{10}, \infty) \end{cases} \tag{6.3}$$

A seguir apresentaremos uma reformulação do modelo (6.1), de forma a representar o PCVP como um problema de programação linear inteira, no qual novas restrições são adicionadas ao problema com o intuito de definir os valores adequados de custo da função objetivo linear por partes em função da carga na aresta. A reformulação utiliza a forma clássica de transformar um modelo linear por partes em um modelo linear contínuo.

$$\text{Minimizar } \phi(x) = \sum_{(i,j)\in E, i<j} \phi_{ij}\ (x_{ij}^1, ..., x_{ij}^p, x_{ji}^1, ..., x_{ji}^p)$$

sujeito a:

$$\begin{aligned}
&\sum_{k\in K}(x_{ij}^k + x_{ji}^k) \leq c_{ij}, \forall (i,j) \in E, i<j \\
&\sum_{(i,j)\in E} x_{ij}^k - \sum_{(i,j)\in E} x_{ji}^k = a_i^k, \forall i \in V, \forall k \in K \\
&\phi_{ij}^b \geq\ u_{ij}, \forall (i,j) \in E, i<j \\
&\phi_{ij}^b \geq 3\ u_{ij} - 2/3, \forall (i,j) \in E, i<j \\
&\phi_{ij}^b \geq 10\ u_{ij} - 16/3, \forall (i,j) \in E, i<j \\
&\phi_{ij}^b \geq 70\ u_{ij} - 178/3, \forall (i,j) \in E, i<j \\
&\phi_{ij}^b \geq 500\ u_{ij} - 1468/3, \forall (i,j) \in E, i<j \\
&\phi_{ij}^b \geq 5000\ u_{ij} - 16318/3, \forall (i,j) \in E, i<j \\
&x_{ij}^k \in \{0,1\}, \forall (i,j) \in E, \forall k \in K.
\end{aligned} \tag{6.4}$$

A função objetivo em (6.4) continua definida como em (6.2).

Pode-se observar que a restrição $\sum_{k\in K} r_k(x_{ij}^k + x_{ji}^k) \leq b_{ij}$ do modelo inicial, que define o limite de fluxo permitido em cada aresta, foi suprimida, isto é, foi permitida a relaxação desta restrição. Neste caso, sua violação é penalizada através das restrições incluídas no novo modelo, que garantem que o valor da componente de congestionamento seja corretamente estabelecido, de acordo com (6.3).

Já a restrição $\sum_{k\in K}(x_{ij}^k + x_{ji}^k) \leq c_{ij}$ continuará sendo imposta. Esta limita o número de pacotes que atravessam cada aresta.

Finalmente a restrição de conservação do fluxo $\sum_{(i,j)\in E} x_{ij}^k - \sum_{(i,j)\in E} x_{ji}^k = a_i^k$ é

também mantida e juntamente com as restrições $x_{ij}^k \in \{0,1\}$ determina que o fluxo não pode ser dividido, ou seja, o pacote deverá ser enviado integralmente do nó origem até o nó destino. Lembremos que a_i^k é dado, sendo igual a 1 se o nó i for a origem do pacote k, igual a -1 se o nó i for o destino do pacote k e igual a zero se o nó i não for origem nem destino do pacote k.

6.3 Local Branching

O procedimento *local branching* foi primeiramente proposto por Fishetti e Lodi [27] com o objetivo de acelerar a convergência e aumentar a eficiência de algoritmos exatos como *branch-and-bound*. O procedimento consiste em reduzir o espaço de busca de um dado problema através da inclusão em sua formulação matemática, de restrições denominadas cortes de *local branching*.

Considere o problema de programação inteira com variáveis binárias

$$\begin{array}{lll} (IP) & \text{Min} & c'x \\ Sujeito\ a: & & Ax = b \\ & & x \in \{0,1\}^n, \end{array} \tag{6.5}$$

Seja x^* uma solução viável para (IP), tal que $x_j^* = 1$, $j \in S$ e $x_j^* = 0$, $j \in N \backslash S$, onde $S \subset N = \{1, 2, ..., n\}$, e seja k um parâmetro inteiro e não negativo. Dada a solução x^*, o espaço de busca do problema é reduzido através da inclusão da desigualdade

$$d(x, x^*) = \sum_{j \in S}(1 - x_j^*) + \sum_{j \in N \backslash S} x_j^* \leq k \qquad (1)$$

A idéia básica do procedimento *local branching* é a de, uma vez conhecida uma solução viável x^* para o problema, otimizar sobre uma vizinhança de x^* definida por (1) e parametrizada por k, numa tentativa de acelerar a obtenção de melhores soluções. Esta idéia é similar à idéia utilizada na busca local da maioria das meta-heurísticas, uma vez que a desigualdade (1) indica que a solução corrente x^* pode ser alterada em no máximo k elementos. O parâmetro k define, portanto, o tamanho da vizinhança em

torno de x^* na qual otimiza-se. Como nas meta-heurísticas, k deve ser cautelosamente escolhido, uma vez que deve ser pequeno o suficiente para tornar o problema de fácil solução, mas grande o suficiente para que a vizinhança definida contenha soluções melhores que x^*. Fishetti e Lodi sugerem o uso de k entre 10 e 20, argumentando que levam aos melhores resultados na maioria das aplicações testadas. Eles indicam ainda a possibilidade de alterar o parâmetro a medida que o algoritmo é executado.

A Figura 6.1 esquematiza o funcionamento básico do algoritmo através de uma árvore de enumeração de subproblemas. Nesta árvore de busca por melhores soluções, apenas os subproblemas representados pelos nós de índice par são resolvidos. O algoritmo é inicializado com uma solução viável x_1 e adiciona-se ao modelo o corte de local branching $d(x, x_1) \leq k$, que corresponde ao nó 2 na árvore. Resolve-se o subproblema representado por este nó e obtém-se a solução ótima x_2. Supondo que x_2 é melhor que x_1^*, eliminamos do modelo a desigualdade $d(x, x_1) \leq k$ e adicionamos ao mesmo tempo as desigualdades $d(x, x_1) \geq k + 1$ e $d(x, x_2) \leq k$, que nos leva ao nó 4 da árvore. Continuamos o procedimento até que algum critério de parada seja satisfeito. Como ilustrado na Figura 6.1, neste instante excluímos o último corte de local branching utilizado, $d(x, x_3) \leq k$, e adicionamos $d(x, x_3) \geq k+1$, abandonando o procedimento de local branching e passando à solução do problema representado pelo nó 7 na árvore.

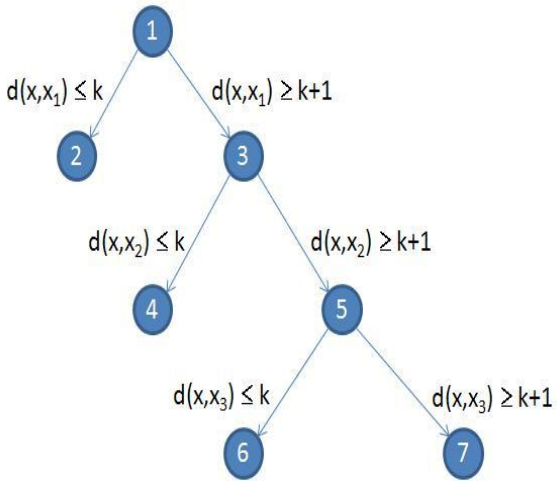

Figura 6.1: Local Branching.

Concluindo, o procedimento local branching leva para os métodos exatos o conceito de busca por melhores soluções numa vizinhança restrita em torno de uma dada solução viável. Ao contrário do que é realizado nas meta-heurísticas, no entanto, neste procedimento esta vizinhança é explorada através da aplicação de modelos de programação matemática e algoritmos exatos. Atualmente, o procedimento local branching é implementado em quase todos os solvers comerciais. No CPLEX, por exemplo, ele é identificado pelo nome polishing algorithm.

6.4 Um método híbrido meta-herística/exato

Neste trabalho propomos a incorporação da idéia introduzida pelo procedimento de local branching, de explorar a vizinhança de uma dada solução viável com a aplicação de modelos de programação matemática e algoritmos exatos, às meta-heurísticas apresentadas no Capítulo 4, constituindo assim, o que denominamos de métodos híbridos meta-herística/exato. Diante da dificuldade de explorar a região viável do PCVP através dos algoritmos exatos, propomos a aplicação dos algoritmos exatos, concentrando a sua busca numa parte reduzida da região viável do problema, definida como uma vizinhança das soluções viáveis obtidas para o mesmo.

Dada uma solução $\bar{x}$, definimos a vizinhança de $\bar{x}$ como o conjunto de soluções nas quais no máximo k rotas correspondentes a k diferentes pacotes podem ser alteradas. Observe que estas foram também as vizinhanças N_1 e N_2 consideradas anteriormente quando descrevemos as meta-heurísticas propostas nesta tese, ao tomar-se $k = 1$ e $k = 2$. Considerando-se valores maiores para k, no entanto, o procedimento descrito no Capítulo 4, tornou-se ineficiente, o que nos levou à proposta do método híbrido meta-herística/exato.

No algoritmo proposto, ao invés de incorporar restrições ao problema (6.4) de forma a impor que no máximo k rotas possam ser alteradas, conforme é realizado no procedimento *local branching*, optamos por escolher aleatoriamente estas rotas e em seguida re-otimizar as mesmas com a aplicação de um método exato. Desta forma, procuramos diversificar mais as soluções obtidas pelo algoritmo exato evitando a convergência prematura para

ótimos locais.

Assim sendo, dada uma solução $\bar{x}$, obtida por uma das meta-heurísticas apresentadas no Capítulo 4, propomos realizar uma nova busca local numa vizinhança de $\bar{x}$. Esta vizinhança é composta pelas soluções nas quais no máximo k pacotes são roteados por um outro caminho. Escolhemos aleatoriamente estes k pacotes e definimos como K o conjunto dos índices correspondentes aos pacotes escolhidos. Para que todos os outros pacotes permaneçam roteados pelo caminho estabelecido pela solução $\bar{x}$, fixamos todas as variáveis no modelo (6.4), com excessão das variáveis que definem as rotas dos pacotes escolhidos, da seguinte forma:

$$x_{ij}^k \leftarrow \bar{x}_{ij}^k, \;\; k \notin K, \;\; (i,j) \in E.$$

Resolvemos então o problema (6.4) com estas variáveis fixas, utilizando um algoritmo exato obtendo o melhor vizinho de $\bar{x}$ no espaço de busca estabelecido. Se este vizinho for uma solução para o problema, melhor que $\bar{x}$, atualizamos $\bar{x}$ e repetimos o procedimento. Caso não tenha ocorrido melhora, selecionamos novamente de forma aleatória outros k pacotes a serem re-otimizados e repetimos o procedimento. A busca é finalizada quando um critério de parada é satisfeito. Optamos por definir um tempo máximo de processamento como critério de parada para a busca local, conforme detalhado na Seção de resultados numéricos a seguir.

O algoritmo que descreve detalhadamente o método híbrido meta-heurística/exato é apresentado na Figura (6.2).

6.4.1 Heurística para escolha dos pacotes roteados pelo método exato

No procedimento apresentado na Seção anterior, k pacotes são escolhidos para serem roteados pelo método exato, enquanto as rotas dos outros pacotes permanecem fixas naquelas determinadas pela solução $\bar{x}$.

Propomos neste trabalho, que os pacotes sejam escolhidos de forma aleatória, de forma a diversificar o conjunto de soluções analisadas.

Foram implementadas duas estratégias de escolha dos pacotes. Na primeira a mesma probabilidade de escolha é associada a cada um dos pacotes do problema. Numa segunda estratégia, procuramos aumentar as chances de melhoria na solução com a aplicação do método exato, tornando livres as variáveis associadas aos pacotes correspondentes aos roteamentos mais caros na solução $\bar{x}$. Desta forma ordenamos os pacotes pelo custo de suas rotas na solução dada e escolhemos aleatoriamente k pacotes dentre os $3k$ pacotes mais caros. Caso k seja maior que um terço no número total de pacotes, as estratégias se igualam.

6.5 Resultados computacionais

Para ressaltar a contribuição do método híbrido meta-heurística/exato para melhorar as soluções encontradas pelas heurísticas, rodamos nos testes deste Capítulo apenas uma única iteração do algoritmo híbrido ($l = 1$) partindo da melhor solução obtida por duas das três heurísticas proposta no Capítulo 4: GRASP-PCVP e GRASP-VND-PCVP.

O parâmetro k que define o tamanho da vizinhança foi fixado em 20. O critério de parada utilizado para as buscas realizadas pelo algoritmo exato foi o tempo de execução de 30 minutos.

Rodamos o algoritmo híbrido 10 vezes para cada uma das duas soluções iniciais e para cada estratégia de escolha dos pacotes a serem roteados (como descrito na Seção 6.4.1). Os resultados obtidos são reportados na Tabela 6.1. A primeira coluna da tabela contém a sigla da método considerado, sendo que a sigla E1 antes do nome da heurística utilizada para gerar a solução inicial, representa que o método híbrido meta-heurística/exato foi implementado com a escolha totalmente aleatória dos pacotes, e a sigla E2 representa a escolha na qual uma maior probabilidade de escolha é dada para os pacotes de rotas mais caras. As colunas seguintes na tabela contêm o maior decréscimo percentual (Dif%) e o decréscimo percentual médio (MDif%) no valor da função objetivo do problema, obtidos com as 10 rodadas. Mais especificamente se o método híbrido meta-heurística/exato parte de uma solução $\bar{x}$ e termina com uma solução $\bar{\bar{x}}$, definimos

$$\text{Dif\%} = \frac{\phi(\bar{x}) - \phi(\bar{\bar{x}})}{\phi(\bar{x})} \times 100.$$

A instância utilizada nos testes deste Capítulo é a instância $fr250$, na qual 250 pacotes devem ser roteados numa rede com 60 nós e 338 arestas.

Tabela 6.1: Desempenho do metodo híbrido na instância $fr250$.

Método	Dif %	MDif%
E1-GRASP-PCVP	2,79	2,45
E1-GRASP-VND-PCVP	6,01	5,15
E1-GRASP-ILS-PCVP	3,46	2,64
E2-GRASP-PCVP	0,32	0,32
E2-GRASP-VND-PCVP	0,22	0,22
E2-GRASP-ILS-PCVP	0,96	0,95

Observamos nos resultados preliminares da Tabela 6.1 que o método híbrido meta-herística/exato obteve sucesso na melhoria das soluções obtidas pelas heurísticas GRASP-PCVP e GRASP-VND-PCVP para a instância fr250, tendo sido possível obter um decréscimo de até 6% no custo da solução obtida pela segunda heurística. Notamos também que a escolha dos pacotes, na qual é dada uma maior probabilidade de escolha aos pacotes de rota mais cara levou a resultados bem piores do que a escolha totalmente aleatória, mostrando-nos mais uma vez a importância de considerar-se uma grande diversidade de soluções na vizinhança analisada. Finalmente, observamos que a melhoria na solução é acompanhada de um aumento de custo computacional, tendo sido empregados 30 minutos de processamento na busca local do método híbrido meta-herística/exato para cada heurística.

Procedimento Método Hibrido meta-heristica/exato

1 **Dado** um parâmetro k inteiro e não negativo.

2 **Repetir** l vezes

3 $\bar{x} \leftarrow$ solução da Meta-Heurística.

4 **Enquanto** *criterio de parada* não é satisfeito

5 Escolha k pacotes do problema a serem novamente roteados pelo algoritmo exato.

6 Seja K o conjunto dos índices dos pacotes escolhidos.

7 Fixe todas as variáveis no problema (6.4) com exceção das variáveis que correspondem ao roteamento dos k pacotes escolhidos, da seguinte forma:

$$x_{ij}^k \leftarrow \bar{x}_{ij}^k, \;\; k \notin K, \;\; (i,j) \in E.$$

8 Resolva o problema (6.4) usando um método exato.

9 $\bar{\bar{x}} \leftarrow$ solução ótima de (6.4).

10 **Se** $\bar{\bar{x}}$ é melhor que $\bar{x}$

11 **Então** $\bar{x} \leftarrow \bar{\bar{x}}$

12 **Fim-Se**

13 **Fim-Enquanto**

14 **Fim-Repetir**

Fim Método Hibrido meta-heristica/exato

Figura 6.2: Pseudo código do método híbrido.

Capítulo 7

Conclusões e Trabalhos Futuros

O Problema de Circuito Virtual Privado (PCVP) é um problema de grande importância para a otimização das redes de telecomunicações, no roteamento de demandas divididas em pacotes de dados.

Nesta tese consideramos a formulação matemática para este problema apresentada em [54], que consiste numa generalização do modelo de multifluxo inteiro, e propomos alguns métodos computacionais para resolvê-lo.

Inicialmente apresentamos uma heurística para o PCVP, desenvolvida por Resende e Ribeiro em [54] e baseada na meta-heurística GRASP. Propomos uma melhoria na fase de construção do algoritmo, gerando nossa primeira proposta de algoritmo, o qual denominamos GRASP-PCVP. Em seguida propomos mais duas heurísticas híbridas para o problema, utilizando abordagens encontradas em diferentes meta-heurísticas apresentadas na literatura. Mais especificamente, substituímos na nossa segunda proposta a busca local do GRASP por um algoritmo baseado na meta-heurística VND, dando origem à heurística GRASP-VND-PCVP. Finalmente, utilizamos uma heurística baseada na meta-heurística ILS para melhorar as soluções obtidas pelo GRASP, o que constituiu a heurística GRASP-ILS-PCVP.

Aplicamos as heurísticas propostas em instâncias de grande porte encontradas na literatura e em instâncias de pequeno porte geradas de forma aleatória a partir dos dados de uma das instâncias da literatura. Comparamos os resultados obtidos primeiramente com a solução ótima do problema para as instâncias pequenas, verificando que todas as

heurísticas chegavam na solução ótima destas instâncias. Em seguida, comparamos as heurísticas entre si, com o intuito de verificar se a hibridização promovida pelas abordagens das diferentes meta-heurísticas levaria à melhoria das soluções obtidas pelo GRASP puro. Concluímos que a perturbação utilizada na meta-heurística ILS foi decisiva para a melhoria dos resultados obtidos, levando o algoritmo a escapar de ótimos locais de pior qualidade. Esta melhoria, no entanto, teve um custo computacional, levando o algoritmo a utilizar um tempo consideravelmente maior para executar um número fixo de iterações. Com relação ao estudo de convergência dos algoritmos, que foi realizado com a distribuição de probabilidade empírica, e aplicado a uma instância da literatura, verificamos que a heurística GRASP-ILS-PCVP mostrou-se mais vantajosa, convergindo mais rapidamente para uma solução sub-ótima, solução esta definida como 10% mais cara que a pior dentre as melhores soluções encontradas pelas três heurísticas propostas.

Observamos que esta melhoria nos resultados, obtida com a utilização do procedimento de perturbação utilizado na meta-heurística ILS, também pode ser verificada em trabalhos da literatura para outros problemas de otimização combinatória. Por exemplo, na dissertação de mestrado de Gonçalves [32] e na tese de doutorado de Motta [43], comparações similares foram realizadas respectivamente para os problemas combinatórios de programação de tripulações e de recobrimento por rotas e as mesmas conclusões foram feitas sobre a utilização do ILS, o que mais uma vez confirma a importância do procedimento de perturbação para escapar de ótimos locais para estes problemas.

Propomos ainda nesta tese um método híbrido meta-herística / exato que utiliza tanto os procedimentos heurísticos para obter uma solução para o PCVP, quanto algoritmos exatos para melhorar a solução obtida através de buscas locais mais refinadas, que foram baseadas no procedimento *local branching*, introduzido por por Fishetti e Lodi [27] com o objetivo de acelerar a convergência de algoritmos exatos na solução de problemas de otimização combinatória com variáveis binárias. A idéia do procedimento é utilizar a programação matemática e os algoritmos exatos para realizar uma busca local numa vizinhança de uma dada solução, vizinhança esta definida por soluções nas quais apenas um subconjunto das variáveis diferem em valor da solução original. Esta idéia foi utilizada em nosso método híbrido meta-herística / exato, no qual consideramos uma reformulação

do modelo proposto em [54] com a qual substituímos a função objetivo definida por partes por uma função contínua. Concluímos mais uma vez que o procedimento leva à melhoria das soluções obtidas. Com sua implementação chegamos a melhorar a solução obtida pelas heurísticas propostas em até 6%. Conforme esperado, no entanto, os procedimentos exatos demandam um tempo computacional maior e podem ser empregados apenas, quando o fator tempo não é tão relevante, como nas situações em que se deseja determinar o roteamento a longo ou médio prazo.

Como trabalhos futuros para dar continuidade a esta pesquisa propomos os seguintes estudos:

- Uma vez que os tempos computacionais são altos para o GRASP-ILS-PCVP e uma vez que acreditamos que as perturbações introduzidas pelo ILS foram decisivas para a melhoria na qualidade das soluções geradas por esta heurística, mencionamos como trabalho futuro a utilização de vizinhanças mais simples, como a vizinhança N_1 na busca local do GRASP-ILS-PCVP.

- Com base no trabalho de Resende e Ribeiro [54], propomos a implementação da estratégia de religamento de caminhos na tentativa de melhorar as soluções obtidas.

- No que diz respeito ao método híbrido meta-herística / exato, propomos na última iteração do método a incorporação de restrições ao modelo matemático do PCVP, que definam a vizinhança a ser explorada, como é realizado no procedimento *local branching*. Desta forma, o otimizador decidiria que pacotes deveriam ser novamente roteados para melhorar a solução corrente. Observamos que em nossa proposta a escolha dos pacotes é realizada a priore e de forma aleatória para diversificar a solução obtida. O objetivo da inclusão desta última iteração seria a de intensificar ainda mais a busca na vizinhança da solução corrente.

- Finalmente, propomos a incorporação do procedimento de perturbação na busca local realizada no método híbrido meta-herística / exato.

Referências Bibliográficas

[1] Ahuja, A., Magnanti, T. and Orlin, J., *Network Flows : Theory, Algorithms and Applications*, Prentice-Hall, Inc. New Jersey, 1993.

[2] Ahuja, R., Orlin, J., Tiwari, A., *A Greedy Genetic Algorithm for the Quadratic Assignment Problem*, Computer and Operations Research, 27: 917-934, 2000.

[3] Aiex, R. M., *Uma investigação Experimental da Distribuição de Probabilidade do Tempo de Solução em Heurísticas GRASP e sua Aplicação na Análise de Implementações Paralelas*, Tese de D.Sc., PUC, Rio de Janeiro, RJ, Brasil, 2002.

[4] Aiex, R. M., Resende, M., Ribeiro, C. C., *Probability distribution of solution time in GRASP: An experimental investigation*, Journal of Heuristics, vol. 8, pp 343-373, Dec, 2002.

[5] Alvelos, F. P. P. C., *Branch-and-Price and Multicommodity Flows*, Tese de D.Sc., Universidade de Minho, Portugal, 2005.

[6] Amiri, A., Rolland, E. and Barkhi, R., *Bandwidth packing with queueing delay costs: Bounding and heuristic solution procedures*, European Journal of Operational Research, vol. 112, 635-645, 1999.

[7] Andrade, D. V., Buriol, L. S., Resende, M. G. C., Thorup, M., *Survivable Composite-Link IP Network Design With OSPF Routing*, October 2006, Wiley Interscience (www.interscience. wiley.com).

[8] Armony, M., Klincewicz, J., Luss, H, Rosenwein, M., *Design or stacked self-healing rings using a genetic algorithm*, Journal of Heuristics, vol. 6, 85-105, 2000.

[9] Barr, R. S., Kingsleyy, M. S. and Pattersonz, R. A., *Grooming Telecommunications Networks: Optimization Models and Methods*, Technical Report 05-emis-03 June 22, 2005.

[10] C. Barnhart, C.A. Hane, and P.H. Vance, *Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems*, Operations Research 48, 318-326, 2000.

[11] Binato, S., Hery, W. J., Loewnstern, D. M., Resende, M. G. C,. *A Greedy Randomized Adaptive Search Procedure for Job Shop Scheduling*, Essays and surveys on metaheuristics, 59-79, Kluwer Academic Publishers, 2001.

[12] Bresina, J. L., *Heuristic-biased stochastic sampling*, In: Proceedings of the thirteenth national conference on artificial intelligence (AAAI-96), p. 271-278. American Association for Artificial Intelligence, 1996.

[13] Buriol, L. S., França, P. M., Thorup, M.; Resende, M., *Network design for OSPF routing*, Proceedings of Mathematical Programming in Rio, p. 40-44, Búzios, Rio de Janeiro, Brazil, 2003.

[14] Buriol, L. S., Resende, M. G. C., Ribeiro, C.C. and Thorup, M., *A hybrid genetic algorithm for the weight setting problem in OPSF/IS-IS routing*, Technical Report, AT&T Labs. Research, 180 Park Avenue, Florham Park, NJ, 07932, USA, Wiley InterScience, 2003.

[15] Calvert, K., Doar, M. and Zegura, E. W., *Modeling Internet topology*, IEEE Communications Magazine, vol. 35 , 160-163, 1997.

[16] Colomé, R,, Serra, D., *Consumer Choice in Competitive Location Models: Formulations and Heuristics*, Papers in Reginal Science, vol. 80, 439-464, 2001.

[17] Cook, W. J., Cunningham, W. H., Pulleyblank, W. R. and Schrijver, A., *Combinatorial optimization*, Wiley-Interscience Series in Discrete Mathematics and Optimization. New York: John Wiley & Sons Inc. A Wiley-Interscience Publication, 1998.

[18] Cormen, T, Leiserson, C., Rivest, R., Stein, C. *Introduction to algorithms*, 2nd ed., The MIT Press, 2003.

[19] Dahl, G., Martin, A. and Stoer, M., *Routing through virtual paths in layered telecommunication networks*, Operations Research, vol. 47 , 693-702, 1999.

[20] Dias, T. C. S., *Algoritmos Heurísticos e Metaheurísticas Híbridas Aplicadas ao Planejamento de Uma Rede de Telecomunicações com Topologia Anel-Estrela*, Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2006.

[21] Ericsson, M., Resende, M. G. C. and Pardalos, P.M., *A genetic algorithm for the weight setting problem in OSPF routing*, Journal of Combinatorial Optimization, vol. 6, 299-333, 2002.

[22] Even, S., *Graph Algorithms*, Computer Science Press, Rockville, 1980.

[23] Feo, T., Resende, M. G. C., *A probabilistic heuristic for a computationally dificult set covering problem* Operations Research Letters, vol. 8, 67-71, 1989.

[24] Feo, T., Resende, M. G. C., *Greedy Randomized Adaptive Search Procedure* Journal of Global Optimization, vol. 6, 109-133, 1995.

[25] Festa, P., Resende, M. G. C., *GRASP : An Annotated bibliography*, AT&T Labs Research Technical Report, Jan, 2001.

[26] Festa, P., Resende, M. G. C., Pardalos, P., Ribeiro, C.C., *GRASP and vns for max cut*, In: Extended Abstracts of the Fourth Metaheuristics International Conference, 371-376, Julho, 2001.

[27] Fischetti, M., A. Lodi, A., *Local branching*, Mathemathical Programming, vol. 98(1-3), 23-47, 2003.

[28] Fortz, B., Thorup. M. *Internet Traffic Engineering by Optimizing OSPF weights*, Proceedings of the IEEE INFOCOM. The Conference on Computer Communications, IEEE Press, Tel-Aviv, 519.528, 2000.

[29] Glover, F., Kochenberger, G. A., *Handbook of Metaheuristics*, Kluwer Academic Publishers, 2003.

[30] Glover, F., Hanafi, S., *Tabu Search and Finite Convergence*, Discrete Applied Mathematics, vol. 119 (1-2), 3-36, 2002.

[31] Glover, F, Laguna, M., *Tabu Search*, Kluwer Academic Publisher, 1997.

[32] Gonçalves, T. L., *Meta-heurísticas para o problema de programação de tripulações*, Dissertação de M.Sc., COPPE/UFRJ, Rio de Janeiro, RJ, Brasil, 2009.

[33] Hart, J. P. and Shogan., A. W., *Semi-greedy heuristics: An empirical study*, Operations Research Letters, vol. 6, 107-114, 1987.

[34] Laguna, M., Martí, R., *Grasp and Path Relinking for 2-Layer Straight Line Crossing Minimization*, Informs Journal on Computing, vol. 11, 44 - 52, 1999.

[35] Laguna, M., González-Velardes, J., *A search heuristic for just-in-time scheduling in parallel machines*, Jornal of intelligent Manufacturing, vol. 2, 253-260, 1991.

[36] Laguna M. and Glover, F., *Bandwidth packing: A tabu search approach*, Management Science, vol. 39 , 492-500, 1993.

[37] LeBlanc, L. J., Chifflet, J. and Mahey, P., *Packet routing in telecommunication networks with path and flow restrictions*, INFORMS Journal on Computing vol. 11, 188-197, 1999.

[38] Li, Y., Pardalos, P., Resende, M. G. C., *A greedy randomized adaptive search procedure for the quadratic assignment problem*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, 237-261, American Mathematical Society, 1994.

[39] Lin, S. and Kernighan, B. W., *An Effective Heuristic Algorithm for the Traveling-Salesman Problem*, Operations Res., vol. 21, 498-516, 1973.

[40] Lourenço, H. R., Paixão, J., Portugal, R., *Metaheuristics for The Bus-Driver Scheduling Problem*, Technical report Departament of Economics and Management, Universitat Pompeu Fabra, 25-27, Barcelona, Spain, 1998.

[41] Martins, S., Resende, M. G. C., Ribeiro, C, Pardalos, P., *A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy*, Journal of Global Optimization, vol. 17, 267-283, 2000.

[42] Mladenovic, N., Hansen, P., *Variable neighborhood search*, Computers and Operations Research, vol. 24, 1097-1100, 1997.

[43] Motta, L. C. S., *O problema de recobrimento por rotas: algoritmos e regras de redução*, Tese de D.Sc., Universidade Federal Fulminense, RJ, Brasil, 2010.

[44] Parker, M. and Ryan, J., *A column generation algorithm for bandwidth packing, Telecommunication Systems*, Telecommunication Systems, vol 2 (1), 185-195, 1993.

[45] Pitsoulis, L. S., Resende, M. G. C., *Greedy Randomized Adaptive Search Procedure*, Handbook of Applied Optimization, P.M. Pardalos and M.G.C. Resende, Eds., Oxford University Press, 168-183, 2002.

[46] Prais, M; Ribeiro, C; *Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment.*, Informs Journal on Computing, 12: 164-176, 2000;

[47] Resende, L. I. P. and Resende, M. G. C., *A GRASP for frame relay permanent virtual circuit routing*, in P. Hansen and C.C. Ribeiro, editors, Extended Abstracts of MIC'99 - The III Metaheuristics International Conference, Angra dos Reis, 397-401, 1999.

[48] Resende, M. G.C., *Greedy Randomized Adaptive Search Procedure (GRASP)*, AT &T Labs Research Technical Report, Dez, 1998.

[49] Resende, M. G. C., *A bibliography of GRASP*, AT &T Labs Research Technical Report, Jan, 2001.

[50] Resende, M. G. C., Ribeiro, C. C., *Greedy Randomized Adaptive Search Procedure* , AT&T Labs Research Technical Report, Set, 2001.

[51] Resende, M. G. C. and Ribeiro, C. C., *Greedy randomized adaptive search procedures: Advances and applications*, Handbook of Metaheuristics, 2nd Edition, M. Gendreau and J.-Y. Potvin (Eds.), Springer,July 7, 2008.

[52] Resende, M. G. C., Ribeiro, C. C., *Greedy Randomized Adaptive Search Procedure* , Handbook in Metaheuristics, 219-249, Aug, 2002.

[53] Resende, M. G. C., *Combinatorial Optimization in Telecomunications*, AT&T Labs Research Technical Report, Jul, 2001.

[54] Resende, M. G. C., Ribeiro, C. C., *A GRASP with Path-Relinking for Private Virtual Circuit Routing*, Networks, vol. 41 (2), 104-114, 2003.

[55] Reeves, C. R., *Modern Heuristic Techniques for Combinatorial Problems*, Jonh Wiley & Sons inc, New York, 1993.

[56] Ribeiro, C., Souza, M. *Variable neighborhood search for the degree constrained minimum spanning tree problem*, Discrete Applied Mathematics, vol. 118, 43-54, 2001.

[57] Ribeiro, C. C., Uchoa, E., Werneck, R., *A hybrid GRASP with perturbations for the Steiner problem in graphs*, INFORMS, Journal on Computing, vol. 14, 228-246, 2002.

[58] Szwarcfiter, Jayme Luiz; *Grafos e Algoritmos Computacionais*, Editora Campus,1984.

[59] Shyur, C., Wen, U. E., *Optimizing the system of virtual paths by tabu search*, European Journal of Operational Research, vol. 129, 650-662, 2001.

[60] Sung, C. S., Park, S. K., *An algorithm for configuring embedded networks in reconfigurable telecommunication networks*, Telecommunication Systems, vol. 4 , 241-271, 1995.

[61] Viana, G. V. R, *Meta-Heurísticas e Programação paralela em Otimização Combinatória*, UFC Edições, Fortaleza, 1998.

[62] Vieira, C. E. C., *Metaheurística Aplicada ao Problema de Alocação de Canal em Sistemas de Telecomunicações Móveis*, Dissertação de M.Sc., Instituto Militar de Engenharia, Rio de Janeiro, RJ, Brasil, 2001.

[63] Walkowiak, K., *Ant Algorithm for Flow Assignment in connection-oriented networks*, Int. J. Appl. Math. Comput. Sci., Vol. 15, No. 2, 205-220, 2005.

[64] Yee, J. R. and Lin, F. Y. S., *A routing algorithm for virtual circuit data networks with multiple sessions per O-D pair*, Networks, vol. 22, 185-208, 1992.

[65] Zegura, E. W., Calvert, K. L., Bhattacharjee, S., *How to model an internetwork*, in Proceedings of 15th IEEE Conference on Computer Communications (INFOCOM), IEEE Press, San Francisco, 594-602, 1996.

[66] Zegura, E. W., *GT-ITM: Georgia Tech internetwork topology models (software)*, Technical report, Georgia Institute of Technology, 1996 (online document at http://www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz.

[67] *www.idcresearch.com*

[68] *http://www.research.att.com/.mgcr/data/pvc-routing.tar.gz*

[69] *www.fico.com/xpress, Fico Xpress Optimization Suite 7.0.2.*

[70] *patricialopes1@gmail.com*