Universidade Federal Fluminense

LUCAS DE OLIVEIRA BASTOS

Soluções Heurísticas para o Problema de Atribuição de Localidades a Anéis em Redes SONET

NITERÓI

2005

Universidade Federal Fluminense

LUCAS DE OLIVEIRA BASTOS

Soluções Heurísticas para o Problema de Atribuição de Localidades a Anéis em Redes SONET

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre. Área de concentração: Otimização Combinatória.

Orientador:
Luiz Satoru Ochi

Co-orientador:
Elder M. Macambira

NITERÓI

2005

Soluções Heurísticas para o Problema de Atribuição de Localidades a Anéis em Redes SONET

Lucas de Oliveira Bastos

Dissertação de Mestrado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Mestre.

Aprovada por:

Luiz Satoru Ochi / IC-UFF (Presidente)

Elder Magalhães Macambira / UFPB

Simone de Lima Martins / IC-UFF

Lúcia Maria de Assumpção Drummond / IC-UFF

Isabel Cristina Mello Rosseti / ONS

Niterói, 2 de Dezembro de 2005.

Ao meu Pai, o Rei.

Agradecimentos

Ao Rei que me presenteou com a oportunidade e com os recursos para chegar até aqui.

Aos meus pais e irmãos pela torcida e pelo carinho dispensados a mim todo esse tempo que estive longe deles.

Aos professores do Instituto de Computação pela dedicação no ensino, pela cobrança nos momentos certos e pela ajuda prestada sempre que necessário. Sem eles eu não teria conseguido.

Ao Luiz Satoru e ao Elder Macambira que estiveram comigo o tempo todo ajudando com idéias e sugestões.

Aos meus amigos estudantes da Pós-Graduação pelos momentos que passamos juntos entre estudos e diversão e pelas indispensáveis dicas sobre LaTeX e *Linux* durante todo o período do mestrado.

Aos funcionários da Pós-Graduação pelo excelente trabalho prestado.

E a todos os amigos que, de alguma forma, me apoiaram e ajudaram todo esse tempo, estendo meus sinceros agradecimentos.

Resumo

Neste trabalho, abordaremos um problema de otimização combinatória que surge no projeto de redes de telecomunicações. Este problema é conhecido como Problema de Atribuição de Localidades a Anéis SONET (*SONET ring assignment problem*) (SRAP). Neste problema, cada localidade cliente deve ser atribuída a exatamente um anel SONET e um anel especial, chamado de Anel Federal, interliga os anéis entre si. É imposta sobre cada anel uma restrição de capacidade. O objetivo do problema é encontrar uma atribuição de localidades clientes que minimize o número total de anéis utilizados. Descreveremos um procedimento GRASP, incluindo o conceito de reconexão de caminhos, para encontrar soluções de boa qualidade para o SRAP. Em adição às instâncias disponíveis na literatura, desenvolvemos instâncias maiores para testar nossos algoritmos. Experimentos computacionais sobre as instâncias disponíveis para teste são reportados, comparando o GRASP com reconexão de caminhos com o GRASP proposto previamente (sem reconexão de caminhos) e com outros algoritmos encontrados na literatura. Os resultados dos experimentos ilustram a eficiência do método proposto, sobre outros métodos, em obter soluções ótimas ou muito próximas do valor ótimo.

Abstract

In this work, we consider a combinatorial optimization problem that arises in telecommunications networks design. It is known as the SONET ring assignment problem (SRAP). In this problem, each client site has to be assigned to exactly one SONET ring and a special ring, called Federal Ring, interconnects the other rings together. A capacity constraint on each ring is also imposed. The problem is to find a feasible assignment of the client sites minimizing the total number of rings used. We describe a greedy randomized adaptive search procedure (GRASP), including the path-relinking concept, for finding good-quality solutions of the SRAP. In addition to instances found in literature, we have developed new larger instances to test our algorithms. Computational experiments on benchmark instances are reported, comparing the GRASP with path-relinking with previously proposed pure GRASP (without path-relinking) and with other algorithms found in the literature. Experimental results illustrate the effectiveness of the proposed method, over other methods, to obtain solutions that are either optimal or very close to it.

Palavras-chave

1. GRASP
2. Metaheurísticas
3. Otimização Combinatória
4. Particionamento em Grafos
5. Problema de Atribuição de Localidades a Anéis SONET
6. Projeto de Redes de Telecomunicações
7. Reconexão de Caminhos

Glossário

ADM	:	Add-Drop Multiplexer;
DCS	:	Digital Cross-Connect System;
GRASP	:	Greedy Radomized Adaptative Search Procedure;
GRASP+PR	:	GRASP com Reconexão de Caminhos;
HVR	:	Heurística de Vizinhança Relativa;
SDH	:	Synchronous Digital Hierarchy;
SONET	:	Synchronous Optical NETwork;
SRAP	:	SONET Ring Assignment Problem.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Capítulo 1

Introdução

Com a evolução dos serviços oferecidos aos usuários pelas indústrias de telecomunicações, surgem novos desafios no planejamento das suas redes e, conseqüentemente, a necessidade de resolver problemas de otimização durante essa fase do projeto.

Um dos aspectos relevantes ao se abordar problemas oriundos do planejamento de uma rede de telecomunicações é que podemos representá-los por meio de grafos. Assim, através dessa representação, pode-se encontrar um conjunto de vértices ou arestas que satisfaça as restrições do problema e que otimize uma função objetivo.

O interesse desse trabalho está relacionado com o planejamento de redes de telecomunicações, mais especificamente, à definição da topologia e da tecnologia a ser empregada em uma rede. Esse planejamento consiste na determinação das conexões entre um conjunto de localidades, a um baixo custo, atendendo a um conjunto de restrições. Dentre essas restrições, a capacidade de sobrevivência da rede é fundamental atualmente.

A capacidade de sobrevivência de uma rede é a capacidade dessa rede permanecer ativa caso ocorra falha numa localidade ou numa ligação. A topologia em anel aliada ao emprego da tecnologia SONET (*Synchronous Optical Network*) atende a esse requisito.

O problema de planejamento que estudaremos neste trabalho pode ser definido da seguinte forma:

> Dado um conjunto de localidades L, pertencente a uma rede de telecomunicações, uma matriz de demandas d_{uv} entre duas localidades u e v, com $u, v \in L$, um inteiro B representando a capacidade de tráfego num anel e uma função de custo c, deseja-se determinar uma atribuição de localidades a subconjuntos, que representam os anéis, tal que a capacidade desses subconjuntos seja satisfeita de acordo com o parâmetro B e todas as demandas entre as localidades

sejam satisfeitas com o menor custo possível.

Esse problema faz parte do projeto físico de uma rede e é denominado Problema de Atribuição de Localidades a Aneis SONET (*SONET Ring Assignment Problem*) (SRAP).

O principal objetivo desse trabalho consiste em propor heurísticas para a resolução do SRAP de forma satisfatória. Para isso, tentaremos resolver o problema através de um algoritmo GRASP puro e de um algoritmo GRASP com Reconexão de Caminhos. Cada fase dos algoritmos propostos é composta por heurísticas que serão estudadas em detalhes ao longo desse trabalho. Além disso, reportaremos os experimentos computacionais sobre as instâncias disponíveis para teste e compararemos os algoritmos propostos com outros algoritmos encontrados na literatura.

O restante desse trabalho está organizado da seguinte forma: no Capítulo 2 apresentaremos o Problema de Atribuição de Localidades a Aneis SONET. Introduziremos algumas características do padrão SONET e da topologia de uma rede SONET. Forneceremos a descrição detalhada do problema e comentaremos os trabalhos encontrados na literatura. No Capítulo 3, introduziremos a meta-heurística GRASP e o método de Reconexão de Caminhos. Em seguida, apresentaremos os algoritmos GRASP puro e GRASP com reconexão de caminhos propostos, bem como as heurísticas que compõem cada uma das fases desses algoritmos. Os resultados computacionais serão apresentados no Capítulo 4 bem como alguns detalhes sobre as implementações, parâmetros e equipamentos utilizados. Em seguida, no Capítulo 5, avaliaremos os algoritmos propostos através da análise dos resultados obtidos. A qualidade média dos resultados dos testes computacionais para o GRASP puro e para o GRASP com reconexão de caminhos será empregada na verificação do desempenho desses algoritmos com aqueles encontrados na literatura. Além disso, faremos uma avaliação probabilística empírica desses algoritmos. Finalmente, as conclusões sobre o trabalho e as sugestões para trabalhos futuros serão apresentadas no Capítulo 6.

Capítulo 2

Problema de Atribuição de Localidades a Anéis SONET

O planejamento (ou projeto) de uma rede SONET [1, 2] é uma tarefa bastante complexa, onde uma série de restrições tecnológicas devem ser levadas em consideração. Esse trabalho pode ser dividido em projeto físico (determinação dos subconjuntos de localidades que darão origem aos anéis) e projeto lógico (estabelecimento das conexões entre as localidades).

Neste capítulo, falaremos sobre o problema da determinação dos subconjuntos de localidades que comporão os anéis de uma rede SONET. Este problema é chamado de Problema de Atribuição de Localidades a Anéis SONET (*SONET Ring Assignment Problem*) (SRAP). O SRAP pode ser descrito formalmente como um problema de particionamento em grafos onde os vértices do grafo representam as localidades e os pesos associados às arestas representam as demandas de tráfego entre as localidades.

Na Seção 2.1 abordaremos alguns aspectos relevantes sobre o padrão SONET, mais especificamente, sobre seu sinal base de transporte e sobre a topologia em anel. Em seguida, na Seção 2.2, descreveremos o SRAP, definiremos as restrições consideradas neste trabalho e apresentaremos uma definição formal para o problema. Finalmente, na Seção 2.3, daremos uma visão geral dos trabalhos encontrados na literatura citando os algoritmos propostos e seus resultados.

2.1 O Padrão SONET

O *Synchronous Optical Network* (SONET) é um padrão de transporte ótico de telecomunicações formulado pela *Exchange Carriers Standard Association* (ECSA) para os Estados Unidos e outras indústrias. O objetivo fundamental do padrão SONET é prover a

infra-estrutura de transporte para telecomunicações globais até o ano de 2020 aproximadamente.

Em seguida ao desenvolvimento do padrão SONET, a colaboração entre a *American National Standards Institute* (ANSI) e a *Comité Consultatif International Téléphonique et Télégraphique* (CCITT) produziram um padrão internacional chamado *Synchronous Digital Hierarchy* (SDH). Esse padrão é responsável pela intercomunicação entre a hierarquia não síncrona da CCITT e o padrão SONET.

Na Seção 2.1.1 serão dados mais detalhes sobre os sinais e taxas de comunicação do padrão SONET e seus correspondentes no padrão SDH. Em seguida, na Seção 2.1.2, serão mostrados os detalhes mais relevantes acerca da topologia em anel de uma rede segundo o padrão SONET.

2.1.1 O Sinal SONET

O padrão SONET define uma tecnologia para o transporte de muitos sinais de diferentes capacidades através de uma hierarquia ótica, flexível e síncrona. Isto é feito por intermédio de um esquema de multiplexação chamado *Byte-Interleaving* que simplifica a multiplexação e oferece gerenciamento da rede ponto-a-ponto.

O sinal SONET base é um sinal de transporte síncrono nível 1 ou, simplesmente, STS-1 que opera a 51,84 Mb/s. Os sinais de nível mais alto são múltiplos inteiros do STS-1. O conjunto de todos esses sinais forma a família STS-N.

A Tabela 2.1 mostra um resumo da hierarquia SONET. A primeira coluna mostra o sinal SONET, a segunda coluna mostra a taxa de bits correspondente e a terceira coluna o sinal equivalente no padrão SDH.

Sinal	Taxa de bits (Mb/s)	Equivalente SDH
STS-1, OC-1	51,84	STM-0
STS-3, OC-3	155,52	STM-1
STS-12, OC-12	622,08	STM-4
STS-48, OC-48	2488,32	STM-16
STS-192, OC-192	9953,28	STM-64
STS-768, OC-768	39813,12	STM-256

Tabela 2.1: Hierarquia SONET.

2.1.2 Topologia de uma rede SONET

A topologia típica de uma rede SONET consiste em um conjunto de localidades conectadas através de um ou mais anéis. Por exemplo, cada localidade pertence a um único anel e envia, recebe e transmite sinais através de um dispositivo chamado *Add-Drop Multiplexer* (ADM). Múltiplos ADMs podem ser dispostos numa configuração anelar tanto para tráfego unidirecional quanto bidirecional. A principal vantagem dessa topologia é a sobrevivência da rede. Se um cabo de fibra ótica romper ou falhar, os multiplexadores enviam automaticamente os serviços afetados por um caminho alternativo.

A Figura 2.1 simula a ocorrência de falha num anel SONET. A princípio o fluxo de dados no anel se dá apenas no sentido horário mas, no momento do rompimento de um cabo de fibra ótica, o fluxo nos dois sentidos é ativado. Dessa forma, todas as localidades permanecem acessíveis sem que haja interrupção dos serviços de comunicação da rede.

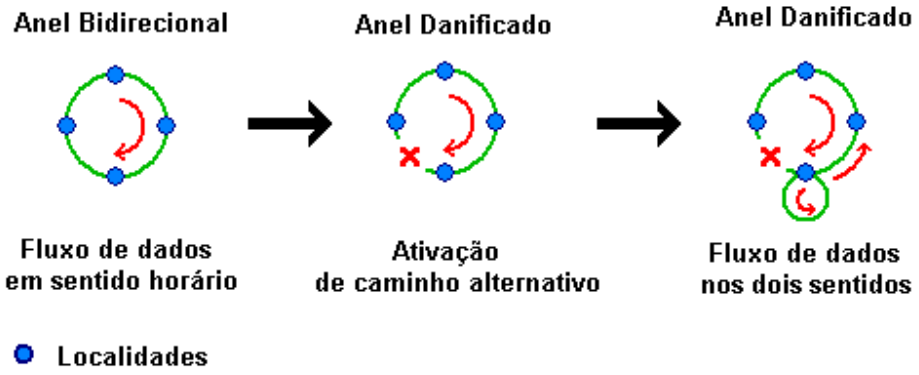

Figura 2.1: Sobrevivência de uma rede SONET.

Finalmente, os anéis podem ou não realizar uma comunicação entre si. Caso essa comunicação ocorra, ela será feita por meio de um equipamento bastante caro, chamado *Digital Cross Connect System* (DCS), que liga os anéis locais a um anel especial denominado Anel Federal (AF) ou backbone.

A quantidade máxima de tráfego num anel local é limitada pela capacidade do anel que corresponde à largura de banda dos ADMs utilizados. Essa capacidade deve acomodar a soma das demandas de todas as localidades pertencentes ao anel local. No caso do anel federal, a quantidade máxima de tráfego é limitada pela largura de banda dos DCSs utilizados e deve acomodar a soma das demandas dos anéis da rede. Os valores possíveis de largura de banda dos ADMs e DCSs correspondem às taxas de bits dos sinais SONET (veja a tabela 2.1).

A Figura 2.2 mostra uma rede segundo a topologia em anel utilizando a tecnologia

SONET. Cada ADM conecta uma localidade à rede através de um anel local. O tráfego entre os anéis locais é feito através dos DCSs que compõem o anel federal.

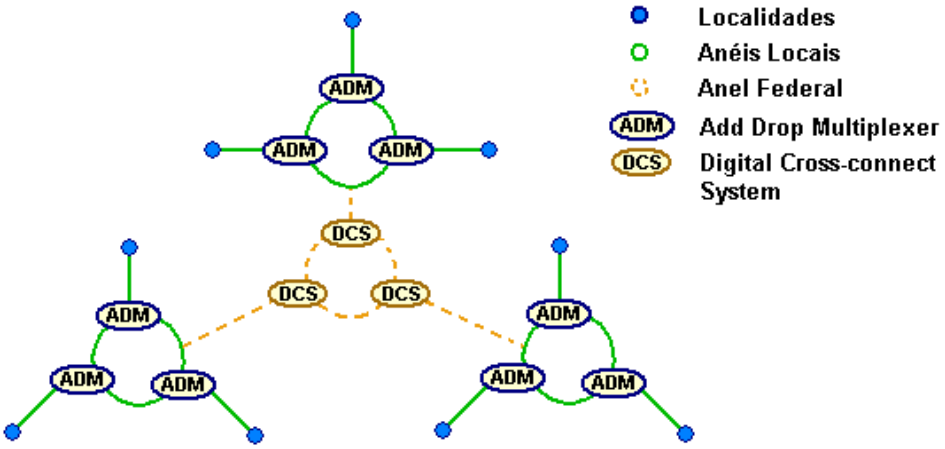

Figura 2.2: Topologia em anel de uma rede SONET.

2.2 Descrição do Problema

Nesta seção, descreveremos o problema de atribuição de localidades a anéis que surge no planejamento de uma rede SONET. Conforme visto na Seção 2.1.2, uma rede SONET é composta por um número n de localidades agrupadas em anéis onde uma série de restrições devem ser atendidas. Devido à complexidade do projeto físico de tal rede, é possível projetar algoritmos para a resolução do SRAP com diferentes conjuntos de restrições e considerações onde o objetivo é minimizar o custo total da rede. No entanto, semelhantemente aos trabalhos encontrados na literatura, consideraremos as seguintes restrições:

(i) cada localidade tem que ser atribuída a um único anel;

(ii) a capacidade máxima de cada anel, inclusive a do anel federal, é limitada por um valor comum, representado por B;

(iii) o número de anéis locais, que corresponde ao custo com os DCSs instalados, deve ser minimizado.

O custo principal envolvido no projeto de construção de uma rede SONET está diretamente relacionado ao custo dos componentes ADMs e DCSs. Custos com cabos de fibra ótica possuem grau bem menor de relevância. Como os ADMs são consideravelmente mais baratos que os DCSs hoje em dia, a quantidade de DCSs se torna determinante para o

custo total da rede. Dessa forma a topologia em anel mais atrativa é aquela que utiliza um número mínimo de anéis.

2.2.1 Definição e Complexidade

Seja $G = (V, E)$ um grafo completo não direcionado e um inteiro positivo B. Associado a cada aresta (u, v) em E há um inteiro não negativo d_{uv}. Uma solução viável do problema corresponde a uma partição do conjunto de vértices V em conjuntos disjuntos $V_1, V_2, \ldots, V_k$ tal que:

$$\sum_{(u,v)\in G[V_j]} d_{uv} \leq B, \quad \forall j \in \{1, \ldots, k\}, \tag{2.1}$$

$$\sum_{j=1}^{k} \sum_{(u,v)\in\delta(V_j)|u<v} d_{uv} \leq B, \tag{2.2}$$

onde $G[V_j]$ é o grafo induzido por V_j em G e $\delta(V_j)$ é o conjunto de arestas com apenas uma extremidade em V_j. O inteiro d_{uv} corresponde à demanda de tráfego entre as localidades u e v em ambos os sentidos. Se não houver demanda entre u e v o valor de d_{uv} é zero.

A restrição definida pela equação (2.1) é aquela que estabelece o limite de capacidade B aos anéis locais da rede. A equação (2.2), por sua vez, impõe um tráfego máximo no anel federal menor ou igual ao inteiro B que corresponde à capacidade dos equipamentos utilizados na construção da rede.

A decisão sobre a existência ou não de uma partição dos vértices de G em conjuntos disjuntos $V_1, V_2, \ldots, V_k$ caracteriza o problema de atribuição de localidades a anéis. Este problema é NP-Difícil (veja Goldschmidt *et. al.* [3]).

Consideremos agora uma rede SONET R com n localidades e a anéis e seja $c(x)$ a função que dá o custo do componente x da rede, queremos determinar o custo total da rede. Há um custo fixo e um custo variável associado a cada anel. A restrição (i), na Seção 2.2, levanta a necessidade de um ADM para cada localidade para prover a comunicação da mesma com o anel ao qual pertence e da restrição (ii) concluímos que todos os ADMs devem ter a mesma capacidade dada por B. Sendo assim, o custo dos ADMs é fixo.

O custo variável tem duas componentes. Como visto anteriormente na Seção 2.1.2, cada anel na rede necessita de um DCS para se conectar à rede. A outra parte do custo variável é devida aos cabos de fibra ótica nos anéis que é uma função linear do número de

localidades em cada anel e do tamanho dos anéis. Dessa forma, podemos chegar ao custo aproximado de uma rede SONET, desprezando outros itens de custo menor, dado por

$$c(\mathrm{R}) = n \times c(\mathrm{ADM}) + a \times c(\mathrm{DCS}) + c(\text{fibra ótica}), \tag{2.3}$$

Para definir uma função objetivo para o problema, devemos analisar individualmente os custos de cada componente da equação de custo total. Como vimos, o custo dos ADMs é constante e, além disso, consideravelmente menor do que o custo dos DCSs. Levando em conta a parte variável do custo, devemos atentar para o fato de que o custo dos DCSs são muito mais elevados do que os custos com cabos de fibra ótica. Podemos então, simplificar a equação (2.3) de custo da rede para

$$c(\mathrm{R}) = a \times c(\mathrm{DCS}) \tag{2.4}$$

Levando em conta que utilizaremos sempre o mesmo tipo de equipamento no projeto da rede, podemos considerar o custo dos DCSs constante e igual a 1. Logo, podemos definir nossa função objetivo como uma função do número de anéis da rede dada pela equação $f = a$, onde se deseja minimizar o valor de f e a é o número de anéis da rede.

Esta definição está de acordo com a restrição (iii) descrita na Seção 2.2 acima, onde o que se deseja minimizar é o número de DCSs, ou seja, o número de anéis da rede.

2.3 Trabalhos Preliminares

Goldschmidt, Laugier e Olinick estudaram o SRAP pela primeira vez em [3]. Os autores demonstraram que o SRAP é NP-Difícil, propuseram técnicas de programação inteira e desenvolveram três heurísticas gulosas para a resolução do problema: *edge-based*, *cut-based* e *node-based*.

Na heurística *edge-based* as arestas são dispostas em ordem crescente de peso. A cada iteração, dois anéis diferentes conectados pela aresta de maior peso são unidos se o anel resultante for viável, ou seja, se a demanda do anel resultante for menor ou igual a B. Na heurística *cut-based* os dois anéis que compartilham o maior tráfego entre si são unidos se o anel resultante for viável. Para essas duas heurísticas, os procedimentos são iniciados com a atribuição de cada nó a um anel diferente. Por sua vez, a heurística *node-based* recebe como entrada um número tentativa k de anéis e atribui a cada um dos k anéis

um nó escolhido aleatoriamente. A cada iteração a heurística seleciona o anel r com a capacidade máxima não utilizada e, então, adiciona a r a localidade l cuja demanda de tráfego seja a maior dentre as localidades que se comunicam com r. A heurística *node-based* é repetida dez vezes e, a cada solução viável encontrada, o valor de k é decrementado em uma unidade. Quando a melhor solução viável é maior do que o limite inferior dado por

$$k_{lb} = \frac{\sum_{(u,v)\in E|u<v} d_{uv}}{B}, \tag{2.5}$$

onde E é o conjunto de arestas do grafo do problema, d_{uv} é a demanda de tráfego entre os vértices u e v e B é o limite de capacidade dos anéis, é feita uma tentativa de provar a otimalidade da instância por meio do *software* CPLEX [4].

Estas heurísticas foram aplicadas sobre um conjunto de 160 instâncias com número de localidades variando entre 15 e 50 e densidade do grafo entre 5% e 72%. Algumas dessas instâncias foram geradas aleatoriamente e outras são instâncias reais obtidas de uma empresa de telecomunicações. As heurísticas *edge-based* e *cut-based* foram executadas primeiro e seus resultados utilizados como o valor de k para a heurística *node-based*. Os resultados da aplicação dos algoritmos heurísticos foram comparados com os resultados produzidos pelos métodos exatos. Os autores mostraram evidências empíricas que os algoritmos heurísticos apresentam um bom desempenho e que, muitas vezes, encontravam as soluções ótimas. De fato, 97,46% das instâncias investigadas foram solucionadas otimamente utilizando as três heurísticas.

Aringhieri, Dell'Amico e Grasselli [5] introduziram um algoritmo busca tabu para a resolução do SRAP em 2001. O algoritmo é composto por uma vizinhança de busca chamada *node improvement neighborhood* ou apenas N1, duas listas tabu (*node-list* e *node-from-list*) e quatro funções objetivo (z_1, z_2, z_3 e z_4). A vizinhança de busca N1 gera todas as soluções possíveis ao mover um nó do seu anel original para um outro anel existente ou para um anel novo e, portanto, vazio. A lista *node-list* contém os nós movidos e a lista *node-from-list*, por sua vez, contém os nós movidos e seus anéis originais. A idéia por trás dessas listas tabu é fixar um nó movido e proibir o seu retorno ao anel original por algumas iterações. Finalmente, as funções objetivo estudadas têm como meta reduzir o número k de anéis da rede e, simultaneamente, reduzir o gargalo na rede dado por

$$BN = \sum_{j=1}^{k} \sum_{(u,v)\in\delta(V_j)|u<v} d_{uv}, \tag{2.6}$$

onde $\delta(V_j)$ é o conjunto de arestas com uma extremidade no anel j e d_{uv} é a demanda de tráfego associada à aresta (u, v).

Os experimentos foram realizados sobre as mesmas instâncias introduzidas por Goldschmidt et. al. em [3]. O algoritmo tabu proposto encontrou a solução ótima em 100,0% dessas instâncias.

Recentemente, em 2005, Aringhieri e Dell'Amico [6] estudaram vários algoritmos para a solução do SRAP. Os autores modelaram o problema segundo a teoria dos grafos e propuseram procedimentos para gerar soluções iniciais, bem como, duas vizinhanças de busca e listas tabu. Eles consideraram além da vizinhança de busca N1 introduzida em [5], mais uma vizinhança N2 que analisa o espaço de busca definido ao se escolher dois anéis distintos r_1 e r_2 e, então, mover um nó qualquer de r_1 para r_2 e, em seguida, mover um nó qualquer de r_2 para r_1.

O primeiro algoritmo desenvolvido em [6] foi chamado de BTS (*Basic Tabu Search*) contendo apenas alguns elementos básicos de busca tabu. Em seguida, foram propostas várias técnicas de diversificação e intensificação tais como: Reconexão de Caminhos [7], *eXploring Tabu Search* (XTS) [8] e *Scatter Search* [9].

O algoritmo de reconexão de caminhos foi implementado em duas versões. Uma delas consiste em determinar o número mínimo de movimentos para transformar a solução atual na melhor solução e a outra se baseia na manutenção de um conjunto de soluções elite e na determinação dos menores caminhos que transformam a solução atual em cada uma das soluções elite.

Aringhieri e Dell'Amico, em seu trabalho, apresentaram ainda um novo algoritmo fundamentado em vizinhanças múltiplas chamado DMN (*Diversification by Multiple Neighborhhods*). O procedimento usa, na maior parte do tempo, uma vizinhaça de busca eficiente, dita N2, mas em algum momento assume uma outra vizinhança N3 cujo objetivo é produzir soluções bastante diferentes, mas não necessariamente boas ou viáveis.

Os resultados mostraram que o procedimento DMN encontrou a solução ótima em 100,0% das instâncias produzidas por Goldschmidt et al. em [3]. Para as novas instâncias propostas em [6] e denotadas por AD o melhor resultado foi produzido pelo método XTS com 98,7% de soluções ótimas encontradas.

Macambira [10, 11] propôs um algoritmo *branch-and-price* para a resolução do SRAP de forma exata. Seu trabalho forneceu os valores das soluções ótimas para duas classes de instâncias encontradas na literatura e denotadas por C1 e C2. Além disso, mesmo sendo

um algoritmo exato onde, teoricamente, se espera um tempo computacional mais elevado em relação aos métodos heurísticos, seus resultados apresentaram tempos competitivos para as instâncias investigadas. Mais detalhes sobre essas instâncias e sobre o algoritmo *branch-and-price* podem ser encontradas no trabalho original do autor.

Outros problemas de otimização em redes como k-SRAP, Problema de Partição de Arestas SONET, Problema de Planejamento de Redes de Multi-Anéis, dentre outros, foram também estudados por diversos autores [3, 12, 13]. Resende apresentou em 2003 [14] quatro aplicações para problemas reais de telecomunicações onde a otimização combinatória é crucial. O primeiro problema está relacionado à localização dos PoP (*point-of-presence*), para um provedor de serviços de internet. O segundo problema consiste em rotear otimamente circuitos virtuais permanentes para um serviço *Frame Relay*. O terceiro problema é uma variação do SRAP onde um número pré-determinado de anéis candidatos é conhecido. O objetivo é escolher um subconjunto de anéis dentre os anéis candidatos que otimizem a rede SONET. O quarto problema está ligado ao planejamento de redes de telecomunicações globais.

Capítulo 3

Algoritmos Propostos

Neste capítulo serão apresentados os algoritmos propostos para a resolução do problema de atribuição de localidades a anéis SONET. Uma vez que o problema é NP-Difícil (veja [3]), não se conhece nenhum algoritmo com tempo de execução polinomial, no pior caso, que possa encontrar sempre uma solução ótima para o SRAP. No entanto, é possível desenvolver heurísticas capazes de encontrar soluções ou ótimas ou, pelo menos, soluções viáveis, com probabilidade elevada.

Neste trabalho, foi desenvolvido um algoritmo GRASP com Reconexão de Caminhos para a resolução do SRAP. O procedimento GRASP é um método iterativo multi-partida. Cada iteração é composta de uma fase de construção onde uma solução inicial é obtida e, em seguida, de uma fase de busca local onde a solução inicial pode ser refinada dando origem a uma outra solução de melhor qualidade. Na fase de construção, foram desenvolvidas quatro heurísticas construtivas: *Random Edge-Based, Random Cut-Based, Random Node-Based* e a Heurística de Vizinhanças Relativas (HVR). As três primeiras heurísticas são uma adaptação, incluindo aleatoriedade, das heurísticas construtivas introduzidas por Goldschmidt, Olivier e Olinick em [3]. A heurística HVR introduzida em [15] utiliza-se de um mecanismo de detecção de pares de nós considerados vizinhos relativos, segundo um critério guloso, para serem colocados nos mesmos anéis. O algoritmo GRASP desenvolvido aqui utiliza ainda um procedimento de reconexão de caminhos para introduzir uma maior intensificação à fase de busca.

Na Seção 3.1, serão introduzidos alguns aspectos relevantes sobre a metaheurística GRASP e sobre a técnica de reconexão de caminhos. Além disso, serão introduzidas notações e definições importantes para a implementação dos algoritmos desenvolvidos. Na seção 3.3.1, a fase de construção será detalhada bem como as heurísticas construtivas desenvolvidas. Na Seção 3.3.2, introduziremos a vizinhança de busca local utilizada.

Finalmente, na Seção 3.3.3, o algoritmo de reconexão de caminhos será detalhado.

3.1 Metaheurística GRASP

O GRASP (*Greedy Randomized Adaptive Search Procedure*) é uma metaheurística cujo propósito é encontrar soluções aproximadas para problemas de otimização combinatória. O GRASP foi introduzido em 1989 por Feo e Resende [16] num artigo descrevendo uma heurística probabilística para o Problema de Recobrimento (*Set Covering Problem*). Desde então, a metodologia GRASP experimentou constante desenvolvimento e vem sendo aplicada com sucesso na resolução de vários problemas de otimização combinatória [15, 17, 18, 19, 20].

O GRASP pode ser visto como um método iterativo que aplica repetidamente busca local sobre diferentes soluções iniciais geradas de forma independente e, por isso, é chamado de heurística multi-partida. Cada iteração do GRASP é composta por uma fase de construção onde uma solução inicial S_0 é construída, seguida por uma fase de busca local onde S_0 é utilizada como entrada. Ao final de cada iteração, uma nova solução pode ser encontrada. A melhor solução produzida dentre todas as iterações é retornada como a solução final do GRASP.

O Algoritmo 1 ilustra um procedimento GRASP genérico. A melhor solução S_{melhor} inicia vazia (passo 3). A cada iteração (passos 4 a 8), o algoritmo usa a fase de construção para gerar uma solução inicial S_0 (passo 5). Em seguida, uma busca local é aplicada sobre a solução inicial construída (passo 6) e uma nova solução pode ser obtida. Finalmente, se possível, a melhor solução global é atualizada. O algoritmo termina quando a condição de parada for satisfeita (passo 4). Comumente esta condição de parada é um número máximo de iterações.

Algoritmo 1 Algoritmo GRASP padrão

```
1: procedimento GRASP
2: lerDadosDeEntrada();
3: S_melhor ← ∅;
4: enquanto condicaoParada ≠ verdadeiro faça
5:    S_0 ←construirSolucao();
6:    S_bl ←aplicarBuscaLocal(S);
7:    S_melhor ←escolherMelhor(S_bl, S_melhor);
8: fim enquanto
9: retornar(S_melhor);
10: fim GRASP
```

A fase de construção é também iterativa onde uma solução inicial é construída elemento a elemento através de um procedimento guloso, adaptativo e randômico. Por exemplo, um particionamento de vértices é feito um vértice por vez assim como uma árvore é construída uma aresta por vez.

Seja C o conjunto de todos os elementos que podem ser adicionados a uma solução parcial S_p durante a fase de construção. O benefício de cada elemento ao ser adicionado a S_p é avaliado através de uma função gulosa g. Uma lista restrita de candidatos (LRC) é criada, então, com os elementos de C melhor avaliados por g. A escolha de um elemento pertencente à LRC para fazer parte de S_p é feita de forma aleatória. A heurística é adaptativa porque os benefícios associados a cada elemento do conjunto C são atualizados a cada iteração. A diversificação fica por conta da escolha aleatória dos elementos pertencentes à LRC para fazerem parte da solução.

Uma vez concluída a construção de uma solução inicial S_0, um procedimento de busca local deve ser aplicado a S_0 na tentativa de gerar uma outra solução S_{bl} de melhor qualidade na vizinhança de S_0. Dessa forma, S_0 é dita localmente ótima se não existir nenhuma solução $S_{bl} \in N(S_0)$ melhor do que S_0, onde $N(S_0)$ é a vizinhança de busca de S_0 em relação ao procedimento de busca local.

3.2 Reconexão de Caminhos

Reconexão de caminhos foi proposto pela primeira vez por Glover [21] como uma estratégia de intensificação na exploração de trajetórias entre soluções elite obtidas por busca tabu e por *scatter search*. O uso da reconexão de caminhos num procedimento GRASP, como uma estratégia de intensificação e diversificação, foi proposto inicialmente por Laguna e Martí [22]. A reconexão de caminhos é um aperfeiçoamento bastante significativo para o procedimento GRASP puro, resultando em melhorias na qualidade da solução e no tempo de computação. Várias melhorias, extensões e aperfeiçoamentos foram realizados desde então e aplicados com sucesso na resolução de vários problemas de otimização combinatória [7, 22, 23, 24, 25].

A reconexão de caminhos é sempre aplicada sobre um par de soluções denotado por (S_{base}, S_{guia}). A solução S_{base} é obtida da iteração atual do GRASP enquanto a solução S_{guia} é uma solução obtida do conjunto de soluções elite. O conjunto de solucões elite é denotado por $\mathcal{E}$ e o tamanho deste conjunto será no máximo igual ao valor do parâmetro `maxElite`.

O fundamento base da reconexão de caminhos é a possível existência de soluções de boa qualidade no espaço que contém todas as soluções com elementos comuns das duas soluções S_{base} e S_{guia}. Levando em consideração que o tamanho desse espaço é exponencialmente grande, a reconexão de caminhos explora as trajetórias entre as soluções S_{base} e S_{guia} através de um procedimento guloso onde a melhor solução encontrada no caminho é mantida. Em outras palavras, S_{base} é modificada passo a passo até chegar a S_{guia}. Todas as soluções intermediárias obtidas em cada passo formam o caminho de S_{base} até S_{guia}:

$$S_{base} = S_0,\ S_1,\ \ldots,\ S_n = S_{guia} \tag{3.1}$$

onde n corresponde ao número de soluções intermediárias entre a solução S_{base} e S_{guia}.

O Algoritmo 2 ilustra a implementação de um procedimento de reconexão de caminhos padrão. No passo 2, selecionamos aleatoriamente uma solução, suficientemente diferente de S_{base}, do conjunto elite através do procedimento `escolherSolucaoElite`. Nos passos 3 e 4, definimos a solução inicial S_0 e definimos a melhor solução do caminho de busca S_{melhor}. O laço entre os passos 5 e 18 avalia as soluções do caminho entre S_{base} e S_{guia}. A cada iteração uma solução S_k é obtida de forma gulosa, conforme ilustrado nos passos 7 a 13. O número de iterações dado por n corresponde ao número de trocas necessárias para transformar S_{base} em S_{guia}. O conjunto $\Delta(S_k, S_{guia})$ (passo 7) contém todos os movimentos de troca entre S_k e S_{guia}. Nos passos 8 a 12, as trocas possíveis são avaliadas pela função objetivo f e, então, a melhor troca é efetuada no passo 14. Se necessário, a melhor solução é atualizada no passo 16. Finalmente, a melhor solução encontrada durante o processo completo é retornada no passo 19.

Num procedimento GRASP, duas estratégias básicas têm sido utilizadas para a aplicação da reconexão de caminhos:

- o procedimento de reconexão de caminhos é aplicado a todos os pares de soluções elite. Isto pode ser feito periodicamente durante as iterações do GRASP ou após todas as iterações terem sido realizadas, como um passo de pós-otimização;
- o procedimento de reconexão de caminhos é aplicado como uma estratégia de intensificação à cada ótimo local obtido após a fase de busca local.

A aplicação do procedimento de reconexão de caminhos como uma estratégia de in-

Algoritmo 2 Procedimento Reconexão de Caminhos padrão

```
1: procedimento reconexaoCaminhos(S_base, E)
2: S_guia ← escolherSolucaoElite(E, S_base);
3: S_0 ← S_base;
4: S_melhor ← S_guia;
5: para k = 0, ..., n faça
6:    valorMin ← 0;
7:    para todo i ∈ Δ(S_k, S_guia) faça
8:       S ← trocar(S_k, i);
9:       se f(S) < valorMin então
10:         i_melhor ← i;
11:         valorMin ← f(S);
12:      fim se
13:   fim para
14:   S_{k+1} ← trocar(S_k, i_melhor);
15:   se f(S_{k+1}) < f(S_melhor) então
16:      S_melhor ← S_{k+1};
17:   fim se
18: fim para
19: retornar(S_melhor);
20: fim reconexaoCaminhos
```

tensificação durante as iterações do GRASP parece ser mais eficiente do que utilizá-lo apenas como um passo de pós-otimização. A primeira opção garante que cada ótimo local obtido na fase de busca local do GRASP seja investigado mais cuidadosamente. Adicionalmente, a combinação de intensificação com pós-otimização pode resultar numa estratégia promissora.

O Algoritmo 3 mostra um GRASP com Reconexão de Caminhos padrão onde são aplicadas as duas estratégias explicadas acima. No passo 8, o procedimento de reconexão de caminhos é aplicado sobre a solução localmente ótima S obtida pela fase de busca local. Este passo corresponde à aplicação da segunda estratégia. Nos passos 13 a 20, a primeira estratégia é aplicada onde todas as soluções elite são submetidas ao procedimento de reconexão de caminhos duas-a-duas.

3.3 GRASP Aplicado ao SRAP

Antes de iniciar a descrição detalhada do algorimo GRASP proposto para a resolução do SRAP precisamos introduzir algumas notações importantes para a compreensão das suas fases.

Seja a o índice de um anel qualquer numa solução S com k anéis. Os valores associados

Algoritmo 3 Algoritmo GRASP com Reconexão de Caminhos padrão

```
1: procedimento GRASP-RC
2: lerDadosDeEntrada();
3: ε ← ∅;
4: S_melhor ← ∅;
5: enquanto condicaoParada ≠ verdadeiro faça
6:    S ←construirSolucao();
7:    S ←aplicarBuscaLocal(S);
8:    S ←aplicarReconexaoCaminhos(S, S_melhor);
9:    S_melhor ←escolherMelhor(S, S_melhor);
10:   atualizarConjuntoElite(ε, S);
11: fim enquanto
12: {Fase de Pós-Otimização:}
13: para i = 1 to tamConjuntoElite −1 faça
14:    para j = i + 1 to tamConjuntoElite faça
15:       S_i ← ε[i];
16:       S_j ← ε[j];
17:       S ←aplicarReconexaoCaminhos(S_i, S_j);
18:       S_melhor ←escolherMelhor(S, S_melhor);
19:    fim para
20: fim para
21: retornar(S_melhor);
22: fim GRASP
```

aos tráfegos deste anel a na rede são:

$$demI_a = \sum_{u,v \in a \mid u<v} d_{uv}, \tag{3.2}$$

$$demE_a = \sum_{u \in a,\, v \notin a} d_{uv}, \tag{3.3}$$

$$demT_a = demI_a + demE_a, \tag{3.4}$$

onde $demI_a$, $demE_a$ e $demT_a$ representam as demandas interna, externa e total no anel a, respectivamente. O anel a será viável se $demT_a \leq B$.

Podemos ainda definir, com respeito à solução S:

$$demAF = \frac{\sum_{i=1}^{k} demE_i}{2}, \tag{3.5}$$

onde $demAF$ é a demanda total no anel federal. O anel federal será viável se $demAF$ for

menor ou igual a B. A solução S será viável se todos os seus anéis forem viáveis incluindo o anel federal.

Dadas as definições acima, devemos agora prover os algoritmos de busca com algum mecanismo de avaliação da qualidade das soluções encontradas. Em outras palavras, é necessário definir uma função objetivo para guiar a busca na direção das melhores soluções.

O objetivo principal do GRASP proposto é encontrar uma solução viável com o menor número possível de anéis. Obviamente, durante cada uma das fases do algoritmo, podemos nos deparar com soluções viáveis e também com soluções inviáveis, onde é desejável reduzir o número de anéis dessas soluções e, adicionalmente, se deseja viabilizar soluções inviáveis. Devemos ressaltar ainda que soluções viáveis são sempre melhores que soluções inviáveis independente do número de anéis dessas soluções. Sendo assim, é necessário penalizar soluções inviáveis impedindo que estas obtenham avaliação melhor que uma solução viável.

Seja S uma solução qualquer para o SRAP, a equação que define a função objetivo z é dada por:

$$z = \begin{cases} B \times k & \text{se } S \text{ for viável,} \\ f \times B \times k + demAF & \text{se } S \text{ for inviável,} \end{cases} \tag{3.6}$$

onde k é o número de anéis da solução S e f é um fator de penalidade.

Podemos verificar que a equação (3.6) avalia soluções viáveis de forma distinta das inviáveis conforme discutido anteriormente. No caso de soluções viáveis, a única meta seria reduzir o número de anéis e, por isso, a função de avaliação ignora outros parâmetros fazendo com que todas as soluções viáveis com o mesmo número de anéis possuam o mesmo valor de z.

Soluções inviáveis, por sua vez, são tratadas de forma mais cuidadosa em (3.6). O fator f penaliza uma solução inviável $S_{inviavel}$ para que esta possua avaliação pior do que soluções viáveis.

Conforme veremos nas seções seguintes, todas as heurísticas construtivas desenvolvidas geram soluções viáveis mínimas (soluções onde nenhum par dos seus anéis pode ser unido gerando um único anél viável) ou soluções inviáveis. Goldshmidt et al. [3] mostraram que se uma solução viável com k anéis é mínima então $k \leq 2 \times k^*$, onde k^* é o número de anéis da solução ótima. Por outro lado, nenhuma solução pode ter menos do que k_{lb} anéis. Sendo assim, seja S' uma solução inviável com k' anéis e k^* o número de anéis da solução ótima. O valor de f para garantir que toda solução viável possua no

mínimo $2 \times k^*$ anéis é dado por $f = (2 \times k^*)/k'$. Se $k' \geq k^*$ então $f = 2$ é suficiente. Entretanto, se $k' < k^*$ não podemos determinar matematicamente o valor máximo de f pra satisfazer essa condição pois esse valor dependeria de cada instância especificamente. Por isso, determinamos empiricamente o valor de $f = 4$ como suficiente para garantir que o valor da função objetivo de uma solução inviável seja pior do que os valores das funções objetivo das soluções viáveis para as instâncias teste disponíveis.

No caso de soluções inviáveis com o mesmo número de anéis, a diferença no valor de z é dada pela componente $demAF$. Estamos assumindo, nesse caso, que todos os anéis locais de uma solução são viáveis e, portanto, a inviabilidade dessas soluções é causada apenas pela demanda no anel federal. Mostraremos nas próximas seções que os algoritmos propostos garantem a veracidade dessa consideração. Na Seção 3.3.3, definiremos uma nova função objetivo para explorar soluções com anéis locais inviáveis.

Nas seções seguintes descreveremos em detalhes as fases de construção, de busca local e de reconexão de caminhos desenvolvidas.

3.3.1 Fase de Construção

A fase de construção num algoritmo GRASP é a fase onde uma solução é construída iterativamente através de um ou mais procedimentos gulosos. Esta etapa é representada pelo procedimento `construirSolucao`() visto no passo 5 do Algoritmo 1.

A cada iteração da fase construtiva, a escolha do próximo elemento a compor a solução inicial é determinada pela ordenação de todos os elementos numa lista de candidatos com relação a uma função gulosa. Esta função mede o benefício da seleção de cada elemento. O componente probabilístico é adicionado através da escolha aleatória de um dentre os melhores candidatos da lista, mas não necessariamente o melhor deles. A lista dos melhores candidatos é chamada de Lista Restrita de Candidatos (LRC). Esta técnica permite que diferentes soluções sejam obtidas a cada iteração do GRASP mas não necessariamente compromete a qualidade da componente gulosa do método.

O Algoritmo 4 exibe um pseudo código padrão para a fase de construção de um GRASP. A solução inicial começa vazia (passo 2). No laço que vai do passo 3 ao passo 8, a lista restrita de candidatos é construída (passo 4), um elemento da lista é selecionado aleatoriamente (passo 5) e colocado em S (passo 6). Finalmente a função gulosa é adaptada levando em consideração o elemento e adicionado à soluçao S no passo 7.

As heurísticas *cut-based*, *edge-based* e *node-based* propostas em [3] utilizam uma estra-

Algoritmo 4 Fase de construção padrão de um GRASP

```
1: procedimento construirSolucao()
2: S ← ∅;
3: enquanto S não tiver sido construída faça
4:    criarLRC(LRC);
5:    e ← selecionarElementoAleatoriamente();
6:    S ← S ∪ {e};
7:    adaptarFuncaoGulosa();
8: fim enquanto
9: retornar(S);
10: fim construirSolucao
```

tégia gulosa para encontrar soluções para o SRAP. As heurísticas *cut-based* e *edge-based*, da forma como foram definidas no trabalho original, não apresentam o componente probabilístico discutido acima. Por isso essas heurísticas produzem sempre a mesma solução. Desenvolvemos, então, novas implementações dessas heurísticas incluindo aleatoriedade em alguns passos para aumentar a diversificação e adequar os procedimentos para a fase de construção do GRASP. Desenvolvemos também, uma nova versão da heurística *node-based*. Dentre as diferenças do nosso algoritmo em relação à heurística proposta em [3], estão a viabilidade de todos os anéis locais gerados e a utilização da nossa versão da heurística *cut-based* como rotina de pós-processamento. Denominamos essas novas heurísticas de *random cut-based*, *random edge-based* e *random node-based*. Desenvolvemos ainda uma nova heurística baseada em vizinhanças relativas para a fase construtiva. Antes de detalhar as implementações desses algoritmos, nas próximas seções, forneceremos algumas definições importantes.

Seja k o número de anéis de uma solução heurística do SRAP, a operação de unir dois anéis i e j é definida como uma operação que atribui todos os nós do anel j para o anel i. Uma localidade u é dita adjacente ou adjunta a um anel i se ela não pertencer ao anel i mas existir uma aresta (u, v) tal que a localidade v perteça ao anel i. Uma solução é dita viável se todos os seus anéis, incluindo o federal, tiverem suas restrições de capacidade atendidas.

Dada uma solução viável mínima qualquer com k anéis podemos garantir que a solução ótima possui k^* anéis, com $k^* \in [k_{lb}, 2k]$.

Os algoritmos construtivos descritos nas próximas seções foram projetados na tentativa de encontrar soluções viáveis de boa qualidade para o SRAP.

3.3.1.1 Heurística *Random Edge-Based*

A heurística *random edge-based* analisa, passo a passo, as arestas do grafo do problema individualmente incluindo um fator aleatório para a escolha de elementos.

O algoritmo inicia atribuindo cada um dos n nós a um anel diferente e, assim, construindo uma solução com n anéis. Chamaremos essa solução inicial de solução trivial daqui em diante. Note que se o problema for viável, todos os anéis da solução trivial também o serão. Essa solução só será inviável se a demanda no anel federal for maior do que B.

O algoritmo *random edge-based* lista todas as arestas $(u, v) \in \delta(i) \cap \delta(j)$ em ordem decrescente de peso numa lista restrita de candidatos LRC. Cada aresta é analisada individualmente. A retirada de um elemento da LRC é feita aleatoriamente com maior probabilidade de escolha para as arestas de maior peso. O algoritmo une os anéis i e j caso a demanda do anel resultante não ultrapasse o valor de B. Note que a demanda no anel federal só poderá decrescer à medida que os anéis forem sendo unidos.

Esse algoritmo retorna uma solução mínima. Além disso, se os anéis locais inicialmente forem viáveis, estes nunca gerarão um anel inviável. Se uma solução gerada for viável, podemos garantir que ela utiliza no máximo duas vezes o número de anéis da solução ótima, conforme definido na Seção 3.3.1.

O pseudo código da heurística *random edge-based* implementado nesse trabalho é ilustrado pelo Algoritmo 5.

Algoritmo 5 Heurística *random edge-based*

```
1: procedimento randomEdgeBased()
2: iniciar();
3: S ← construirSolucaoTrivial();
4: LRC ← {(u, v) ∈ E|u < v};
5: ordenar(LRC);
6: enquanto LRC ≠ ∅ faça
7:    (u, v) ←retirarAresta(LRC);
8:    se anel(u) ∪ anel(v) for viável então
9:       unirAneis(anel(u), anel(v));
10:   fim se
11: fim enquanto
12: unirAneisDesconexos();
13: retornar(S);
14: fim randomEdgeBased
```

No passo 2, é executado o método `iniciar()` que prepara as estruturas de dados e

carrega a instância do problema. No passo 3, é criada a solução trivial e atribuída a S. Em seguida, no passo 4, a LRC recebe todas as arestas do grafo inicial e então, é ordenada no passo 5. Cada aresta $(u, v) \in LRC$ é obtida (passo 7) e analisada individualmente (passo 8). Aqui é introduzido um fator de aleatoriedade na escolha das arestas onde as primeiras têm maior probabilidade de serem escolhidas. Se a demanda resultante da união do anel `anel`(u) com o anel `anel`(v) for menor ou igual a B, os anéis são unidos (passo 9). Esse procedimento é repetido até que a lista de arestas candidatas LRC fique vazia. Finalmente, no passo 12, o procedimento `unirAneisDesconexos`() é executado na tentativa de encontrar anéis i, j cujo conjunto $\delta(i) \cap \delta(j)$ seja vazio, mas que possam ser unidos gerando um anel viável. Um efeito semelhante à execução desse método é obtido em [3] pela adição de arestas de peso zero ao grafo inicial para todos os pares de vértices u e v tais que não exista a aresta (u, v). Contudo, em termos de desempenho, esta última abordagem é menos conveniente pois é preciso analisar um número maior de arestas na fase inicial do algoritmo, sobretudo se o grafo original for esparso.

3.3.1.2 Heurística *Random Cut-Based*

Como o seu nome sugere a heurística *random cut-based* é baseada na análise passo a passo de cortes do grafo original do problema incluindo um fator aleatório para a escolha de elementos em cada passo do processo.

O algoritmo inicia atribuindo cada um dos n nós a um anel diferente, construindo, assim, a solução trivial do problema. Obviamente, as considerações sobre a viabilidade dos anéis discutidas para a heurística *Random Edge-Based* são válidas também aqui.

Seja $G[i]$ o grafo induzido pelo anel i e seja $\delta(i)$ o conjunto de todas as arestas com exatamente uma extremidade em i. Além disso, definimos uma aresta especial entre dois anéis i e j chamada de aresta virtual e representada por (i, j). Associado à cada aresta virtual existe um inteiro d_{ij} que representa a demanda comum entre os anéis i e j e corresponde à soma das demandas associadas a todas as arestas reais adjuntas aos anéis i e j. A cada iteração, uma lista restrita de candidatos LRC é criada com os pares de anéis i e j que podem ser unidos, isto é, $demT_{G[i] \cup G[j]}$ menor ou igual a B. Em seguida, um par de anéis é escolhido na LRC com probabilidade proporcional ao valor de d_{ij}, ou seja, os pares de anéis com maior demanda comum têm maior chance de serem escolhidos. Todos os nós do anel j são, então, atribuídos ao anel i. O processo continua até que não seja mais possível unir anéis.

Esta heurística também retorna uma solução mínima e, portanto, se esta for viável a

solução utiliza no máximo duas vezes o número de anéis de uma solução ótima.

O Algoritmo 6 mostra o pseudo código do procedimento *random cut-based*.

Algoritmo 6 Heurística *random cut-based*

```
1: procedimento randomCutBased()
2: iniciar();
3: S ← construirSolucaoTrivial();
4: enquanto for possível reunir anéis faça
5:    LRC ← {(i, j) ∈ E_virtuais | i < j};
6:    ordenar(LRC);
7:    enquanto LRC ≠ ∅ faça
8:       (i, j) ←retirarAresta(LRC);
9:       se i ∪ j for viável então
10:         unirAneis(i, j);
11:      fim se
12:   fim enquanto
13: fim enquanto
14: unirAneisDesconexos();
15: retornar(S);
16: fim randomCutBased
```

No passo 2 é executado o método `iniciar()` que prepara as estruturas de dados e carrega a instância do problema. Em seguida, no passo 3, S é iniciada com a construção da solução trivial. Nos passos 5 e 6, é criada a LRC com a inserção das arestas virtuais (i, j) dispostas em ordem decrescente do valor de d_{ij}. No passo 8, uma aresta é aleatoriamente retirada da LRC com probabilidade proporcional ao valor de d_{ij}. Caso a demanda total resultante da união do anel i com o anel j seja menor ou igual a B, os anéis são unidos no passo 10. O procedimento `unirAneisDesconexos()`, no passo 14, tenta encontrar anéis i, j cujo conjunto $\delta(i) \cap \delta(j)$ seja vazio, mas que possam ser unidos gerando um anel viável. Note que, esse procedimento não modifica a demanda no anel federal, portanto, se a solução encontrada for inviável esse procedimento não a viabilizará. No entanto, reduzirá o seu número de anéis tornando-a melhor avaliada pela função objetivo.

3.3.1.3 Heurística *Random Node-Based*

Apesar das heurísticas *edge-based* e *cut-based* propostas por Goldschmidt em [3] não apresentarem um fator de diversificação satisfatório, a heurística *node-based* o faz, na fase inicial do algoritmo, na contrução dos k anéis iniciais, através da atribuição aleatória de uma localidade para cada anel criado. Contudo, a versão implementada aqui possui diferenças relevantes, conforme veremos, além de utilizar a heurística *random cut-based*,

como rotina de pós-processamento, e não puramente a versão *cut-based* implementada em [3]. Assim, a denominamos de heurística *random node-based*.

A heurística *random node-based* recebe como entrada um parâmetro k indicando o número de anéis desejado para a solução gerada e atribui uma localidade a cada um dos k anéis aleatoriamente. A escolha dessas localidades influencia diretamente na geração de duas listas restritas de candidatos: LRC_a e LRC_b. Portanto, escolhas diferentes tendem a gerar soluções diferentes e, consequentemente, de qualidades diferentes. Inicialmente, o algoritmo gera uma lista LRC_a com os anéis iniciais dispostos em ordem decrescente de capacidade. A cada iteração, um anel a é retirado aleatoriamente da LRC_a, com probabilidade proporcional à sua capacidade, e uma nova lista LRC_b de localidades adjuntas ao anel a é criada. As localidades em LRC_b são adicionadas aleatoriamente ao anel a se a demanda em a não ultrapassar o limite B. Nesse caso, a probabilidade de seleção das arestas em LRC_b é também diretamente proporcional à demanda de tráfego associada às mesmas. O processo é repetido até que todos os anéis em LRC_a sejam analisados. A restrição de capacidade imposta na atribuição de localidades a anéis garante a viabilidade de todos os anéis gerados. Contudo, o anel federal pode terminar inviável. Finalmente a heurística *random cut-based* é executada no grafo resultante na tentativa de unir anéis e encontrar uma solução com menos de k anéis.

O Algoritmo 7 mostra o pseudo código da heurística *random node-based*.

O algoritmo inicia no passo 2 com a execução do método `iniciar`() que prepara as estruturas de dados e carrega a instância do problema. No passo 3, k anéis são criados e uma localidade é atribuida aleatoriamente para cada anel. A lista de anéis LRC_a é criada e, em seguida, ordenada por capacidade de tráfego disponível (passos 4 e 5), ou seja, os anéis com maior capacidade de tráfego aparecem nas primeiras posições da lista. Nos passos seguintes (passos 6 a 16), os anéis em LRC_a são analisados na tentativa de englobar localidades adjuntas sem ultrapassar seus limites de capacidade. Para isso, o algoritmo escolhe um anel $a \in LRC_a$ através do método `retirarAnel`(LRC_a) (passo 7). Esse método escolhe um anel aleatoriamente em LRC_a com maior probabilidade de escolha para os anéis com maior capacidade disponível. Em seguida, o método `listarLocalidadesAdjuntas`(a) recebe o anel a como entrada e constrói a lista LRC_b composta por todas as localidades adjuntas ao anel a (passo 8). LRC_b é ordenada no passo 9. Nos passos 10 a 15, as localidades em LRC_b são retiradas uma a uma (passo 11) e atribuídas ao anel a (passo 13) se a demanda resultante for menor ou igual a B. Após esse processo, as localidades que permanecerem desalocadas são alocadas através do

Algoritmo 7 Heurística *random node-based*

```
1: procedimento randomNodeBased()
2: iniciar();
3: S ← construirAneisIniciais(k);
4: LRC_a ←listarAneis(S);
5: ordenar(LRC_a);
6: enquanto LRC_a ≠ ∅ faça
7:    a ←retirarAnel(LRC_a);
8:    LRC_b ← listarLocalidadesAdjuntas(a);
9:    ordenar(LRC_b);
10:   enquanto LRC_b ≠ ∅ faça
11:      l ← removerLocalidade(LRC_b);
12:      se demT_{a ∪ {l}} ≤ B então
13:         anexar(a, l);
14:      fim se
15:   fim enquanto
16: fim enquanto
17: S ←alocarLocalidadesDesconectadas(S);
18: S ←aplicarRandomCutBased(S);
19: retornar(S);
20: fim randomNodeBased
```

procedimento `alocarLocalidadesDesconectadas`() (passo 17). O método consiste em procurar, para cada localidade desconectada, um anel que a comporte mantendo a sua viabilidade. Caso não seja possível alocar todas as localidades nos anéis existentes, um novo anel é criado até que todas as localidades sejam alocadas. Finalmente, a heurística *random cut-based* é aplicada sobre a solução encontrada (passo 18) na tentativa de reduzir o seu número de anéis. A solução resultante é retornada no passo 19.

3.3.1.4 Heurística de Vizinhanças Relativas

Desenvolvemos um novo algoritmo para a fase de construção na tentativa de gerar melhores resultados para as instâncias onde as heurísticas *random edge-based*, *random cut-based* e *random node-based* não foram suficientemente eficientes. Denominamos esse novo algoritmo de Heurística de Vizinhanças Relativas (HVR).

A idéia fundamental da Heurística de Vizinhanças Relativas é estabelecer conjuntos de localidades vizinhas em relação a um certo critério para serem mantidas nos mesmos anéis. Além disso, queremos evitar que localidades compartilhando tráfego elevado sejam dispostas em anéis distintos sobrecarregando o tráfego no anel federal. O critério considerado aqui estabelece que duas localidades u e v são consideradas vizinhas relativas se compartilharem diretamente a maior demanda entre todas as suas demandas comuns.

Em outras palavras, u e v são vizinhas relativas se o valor d_{uv} associado à aresta (u, v) for tal que $d_{uv} = max\{d_{rs}|(r, s) \in \delta(u) \cup \delta(v)\}$. Após esse passo, as localidades vizinhas são colocadas no mesmo anel se a demanda total do anel resultante for menor do que B. Este procedimento gera no mínimo um anel, se todas as localidades forem consideradas vizinhas entre si, e no máximo $(\frac{n}{2})$ anéis, se todas as localidades forem consideradas vizinhas duas a duas, onde n é o número de localidades do problema. As k localidades restantes são adicionadas a k novos anéis criados, uma para cada anel. Depois disso, a HVR tenta unir anéis da mesma forma como a heurística *random node-based* 3.3.1.3 o faz.

Algoritmo 8 Heurística de Vizinhanças Relativas

```
1: procedimento HVR()
2: iniciar();
3: E_v ← ∅;
4: {Gerando informações sobre vizinhanças}
5: para todo (u, v) ∈ E faça
6:    D ← max{d_uv : (u, v) ∈ δ(u) ∪ δ(v)};
7:    se d_uv ≥ D então
8:       E_v = E_v ∪ (u, v);
9:    fim se
10: fim para
11: {Gerando anéis a partir das informações de vizinhança}
12: para todo (u, v) ∈ E_v faça
13:    reunirNoMesmoAnel(u, v);
14: fim para
15: para todo u ∈ V, tal que, u está desalocado faça
16:    alocarEmNovoAnel(u);
17: fim para
18: {Unindo anéis}
19: executarRandomNodeBasedParcial(S);
20: retornar(S);
21: fim HVR
```

Os passos do algoritmo construtivo HVR são descritos pelo Algoritmo 8. O algoritmo inicia no passo 2 com a execução do método `iniciar`() que prepara as estruturas de dados e carrega a instância do problema. No passo 3, a lista de arestas vizinhas inicia vazia e vai sendo construída passo-a-passo de acordo com o critério de vizinhança discutido acima (passos 5 a 10). Nos passos 12 a 17, as informações de vizinhança são utilizadas para formar os anéis iniciais. Durante esse processo, podem restar localidades desalocadas. Essas localidades são então, alocadas nos passos 15 a 17. Finalmente, o algoritmo `executarRandomNodeBasedParcial` é executado sobre a solução parcial S na tentativa de melhorá-la (passo 19). Os procedimentos `alocarEmNovoAnel` e `executarRandomNodeBasedParcial` são os responsáveis pela diversificação da solução

gerada, pois as informações de vizinhança por si só gerariam sempre as mesmas soluções. A solução resultante é retornada no passo 20.

3.3.1.5 Algoritmo Construtivo Consolidado

Uma vez descritas as heurísticas construtivas desenvolvidas neste trabalho, podemos detalhar o procedimento `CONSTRUIR_SOLUCAO` que será utilizado no GRASP. Este procedimento é o algoritmo de construção consolidado que se utiliza das heurísticas definidas nas seções acima.

Antes de determinar a forma como gerenciaríamos a construção das soluções iniciais, fizemos vários testes preliminares com o objetivo de determinar a eficiência de cada uma das heurísticas desenvolvidas e decidir se devemos utilizar uma determinada heurística proporcionalmente à sua eficiência. Um dos testes realizados consistiu em utilizar cada heurística separadamente sobre um conjunto de instâncias e verificar o tempo médio e a qualidade média dos resultados obtidos. Um outro teste foi feito utilizando um algoritmo de adaptação, onde cada heurística foi utilizada inicialmente um número x de vezes. Este procedimento faz uso de um tipo de memória que guarda o desempenho de cada construtivo durante o período de treinamento e, em seguida, utiliza-se dessa informação para executar os algoritmos proporcionalmente ao seu desempenho. Isso gera o que chamamos de GRASP reativo.

Algoritmo 9 Fase de Construção

```
1: procedimento CONSTRUIR_SOLUCAO()
2: iniciar();
3: avalie (itAtual % 4)
4:     caso 0 : S ←randomCutBased();
5:     caso 1 : S ←randomEdgeBased();
6:     caso 2 : S ←randomNodeBased();
7:     caso 3 : S ←HVR();
8: fim avalie
9: retornar(S);
10: fim CONSTRUIR_SOLUCAO
```

Antes de propor o algoritmo consolidado para a fase de construção é necessário avaliar o desempenho médio das heurísticas construtivas. Para isso, foram utilizadas algumas instâncias disponíveis na literatura e, então, realizados dois tipos de testes: o primeiro consistiu em executar cada heurística construtiva individualmente e isoladamente sobre cada instância e o segundo em aplicar cada heurística juntamente com o procedimento de busca local N3 (veja a Seção 3.3.2, Algoritmo 12).

No primeiro teste, o melhor desempenho foi alcançado pela heurística *Random Node-Based* com 90,4% de convergência, seguido pela heurística ACVR com 90,0% de convergência. As heurísticas *Random Cut-based* e *Random Edge-based* ficaram empatadas com 73,2% de convergência. Com o auxílio da busca local (segundo teste) o melhor desempenho foi o da heurística *Random Node-Based* com 99,7% de convergência seguida pela heurística ACVR com 99,5% de convergência. As heurísticas *Random Cut-based* e *Random Edge-based* obtiveram 98,6% e 98,7% de convergência, respectivamente.

Os resultados destes testes preliminares mostraram que, apesar de as heurísticas *Random Node-Based* e ACVR obtiveram os melhores desempenhos quando usadas isoladamente, as quatro heurísticas apresentam um desempenho médio muito próximo quando utilizadas em conjunto com a busca local. Durante os testes foi possível observar também que uma dada heurística obteve melhores resultados para algumas instâncias e, em seguida, piores resultados para outras instâncias.

Pontanto, não é tarefa trivial determinar em qual momento uma heurística é melhor que a outra no algoritmo construtivo consolidado. Além disso, um algoritmo de gerenciamento muito complexo demanda um maior tempo computacional e pode não compensar no resultado final. Portanto, foi decidido utilizar um procedimento muito simples de gerenciamento onde cada heurística é utilizada de acordo com um escalonamento FIFO em fila circular. Assim, todas as heurísticas são utilizadas independentemente, uma em cada iteração.

O Algoritmo 9 ilustra o procedimento `CONSTRUIR_SOLUCAO`. No passo 2, o método `iniciar` carrega as estruturas de dados e a instância do problema. Em seguida, nos passos 3 a 8, uma heurística construtiva é escolhida de acordo com o parâmetro `itAtual` que indica o número da iteração corrente. A solução inicial S é, então, gerada pela heurística selecionada e retornada no passo 9.

3.3.2 Fase de Busca Local

Dada uma solução S, uma vizinhança de busca N é um subconjunto de soluções $N(S)$. A solução S é dita localmente ótima se não houver nenhuma solução em $N(S)$ melhor do que S. Como não há garantia que a solução gerada pela fase construtiva do GRASP é localmente ótima com relação às definições de vizinhança consideradas, torna-se importante a aplicação de um procedimento de busca local na tentativa de aperfeiçoar as soluções geradas na etapa de construção do GRASP.

O sucesso de um algoritmo de busca local está diretamente associado à escolha adequada das vizinhanças de busca, à eficiencia das técnicas de busca associadas e à solução inicial. Durante o desenvolvimento do GRASP proposto, investigamos quatro estruturas de vizinhança: N_1, N_2, N_3 e N_4. Cada vizinhança possui características e objetivos distintos e, portanto, podem se mostrar mais eficiente em determinadas situações e menos eficiente em outras. Por isso, é necessário a escolha adequada do melhor momento para a aplicação de determinada vizinhança sobre uma solução S gerada na fase construtiva.

As soluções iniciais produzidas pelos algoritmos construtivos propostos na Seção 3.3.1 apresentam algumas características relevantes que devem ser consideradas na definição das vizinhanças de busca aqui investigadas. Uma solução inicial S_0 com k anéis pode ser viável ou inviável. No caso de S_0 ser viável, todos os seus anéis, incluindo o anel federal serão viáveis também. Além disso, k pode ser maior ou igual ao número de anéis da solução ótima k^*. Dessa forma, o objetivo de um procedimento de busca sobre uma solução viável consiste em reduzir o valor de k, caso este seja maior do que k^*, mantendo a solução viável. Por outro lado, se S_0 for inviável, todos os seus anéis locais serão viáveis e apenas o anel federal será inviável. Nesse caso, o valor de k será maior ou igual a k_{lb}. Sendo assim, o principal objetivo de um procedimento de busca é viabilizar S_0 através da viabilização do seu anel federal e, com isso, cair no caso anterior.

As duas primeiras vizinhanças investigadas N_1 e N_2 foram introduzidas por Macambira em [10] e denominadas de $Pr1$ e $Pr2$, respectivamente. Contudo, devido às características do nosso GRASP, as implementações de N_1 e N_2 não são exatamente iguais àquelas em [10] conforme mostraremos a seguir.

A vizinhança de busca N_1 consiste na procura por um vértice u pertencente a um anel r e realocá-lo para um outro anel t de forma a reduzir a demanda no anel federal sem inviabilizar nenhum anel local. Em outras palavras, considere dois anéis r e t da solução S, uma localidade $u \in r$ é movida para o anel t se $demT_{t\cup\{u\}}$ menor ou igual a B e $demAF$ for reduzida.

A vizinhança N_2, por sua vez, analisa a possibilidade de se escolher dois nós $u \in a_u$ e $v \in a_v$ para efetuar uma troca simultânea de anéis, tal que, após a troca, $u \in a_v$ e $v \in a_u$. Espera-se com isso, reduzir a demanda no anel federal e viabilizar a solução construtiva. Esta vizinhança foi introduzida em [10] com o nome de procedimento de perturbação $Pr2$.

Os procedimentos N_1 e N_2 só são aplicados sobre soluções cujos anéis locais são viáveis e o anel federal é inviável. Nesses procedimentos, uma troca só é efetuada se houver uma redução de demanda no anel federal dada por $demAF$ (veja a equação 3.5) e, além disso, os

anéis resultantes permanecerem viáveis. Obviamente, esse movimento melhora a avaliação da solução pela função objetivo, dada pela equação 3.6. Os procedimentos $Pr1$ e $Pr2$, por sua vez, trabalham com anéis locais viáveis e inviáveis. Caso um movimento de troca melhore a avaliação da solução, dada pela função objetivo, este é efetuado.

Algoritmo 10 Procedimento de busca local N_1

```
1: procedimento buscaLocal_N1(S)
2: u_m ← nulo;
3: r_m ← nulo;
4: t_m ← nulo;
5: dem_menor ← MAX_INT;
6: para cada anel r ∈ S faça
7:    para cada localidade u ∈ r faça
8:       para cada anel t ∈ S, tal que, r ≠ t faça
9:          se (demT_{t∪{u}} ≤ B) e (demAF_AposTroca < dem_menor) então
10:            dem_menor ← demAF_AposTroca ;
11:            u_m ← u;
12:            r_m ← r;
13:            t_m ← t;
14:         fim se
15:      fim para
16:   fim para
17: fim para
18: se dem_menor < MAX_INT então
19:   r_m ← r_m \ u_m;
20:   t_m ← t_m ∪ {u_m};
21: fim se
22: retornar(S);
23: fim buscaLocal_N1
```

Os procedimentos de busca local associados às vizinhanças N_1 e N_2 estão descritos nos algoritmos 10 e 11, respectivamente.

O procedimento de busca N_1 procura o vértice $u \in r$ cuja troca para um outro anel t produza a maior redução de demanda no anel federal sem inviabilizar os anéis r e t. Essa tarefa pode ser vista no algoritmo 10 entre os passos 2 e 17. Em seguida, o melhor vértice é escolhido e a troca efetuada (passos 18 a 21). A solução é retornada no passo 22. A complexidade de tempo da busca pelo melhor vértice é $O(n \times k^2)$, onde n é o número de localidades e k é o número de anéis. Dessa forma, o uso freqüente deste algoritmo pode elevar consideravelmente o tempo computacional do GRASP.

O procedimento de busca N_2 é muito similar ao procedimento de busca N_1. A diferença fundamental está no passo 10 onde aparece um quarto laço aninhado. A consequência disso é uma elevação na ordem da complexidade de tempo do algoritmo, dada por $O(n^2 \times k^2)$,

onde n é o número de localidades e k é o número de anéis. Assim, podemos dizer que o procedimento N_2 deve gastar n vezes mais tempo que o procedimento N_1. Obviamente, o cuidado na escolha do momento certo para usar este algoritmo é ainda mais importante do que no caso de N_1.

Algoritmo 11 Procedimento de busca local N_2

```
1: procedimento buscaLocal_N2(S)
2: u_m ← nulo;
3: N_m ← nulo;
4: r_m ← nulo;
5: t_m ← nulo;
6: dem_menor ← MAX_INT;
7: para cada anel r ∈ S faça
8:    para cada localidade u ∈ r faça
9:       para cada anel t ∈ S, tal que, r ≠ t faça
10:         para cada localidade v ∈ t faça
11:            se (demT_{t∪{u}} ≤ B) e (demT_{r∪{v}} ≤ B) e (demAF_AposTroca < dem_menor)
               então
12:               dem_menor ← demAF_AposTroca ;
13:               u_m ← u;
14:               N_m ← u;
15:               r_m ← r;
16:               t_m ← t;
17:            fim se
18:         fim para
19:      fim para
20:   fim para
21: fim para
22: se dem_menor < MAX_INT então
23:    r_m ← r_m \ u_m;
24:    t_m ← t_m ∪ {u_m};
25:    t_m ← t_m ∪ {v_m};
26:    r_m ← r_m ∪ v_m;
27: fim se
28: retornar(S);
29: fim buscaLocal_N2
```

A vizinhança N_3, introduzida em [25], considera a possibilidade de esvaziar o anel de menor demanda total em S distribuindo suas localidades entre os outros anéis da solução. Observamos que muitas soluções geradas na fase construtiva apresentam um ou mais anéis com pouco tráfego que podem ser esvaziados sem inviabilizar outros anéis. Para isso, a vizinhança N_3 procura o anel t mais promissor para conter o vértice $u \in r$ sem ultrapassar a capacidade de t dada por B. Este procedimento é repetido até que o anel r seja esvaziado ou nenhuma retirada adicional de vértices de r seja possível. Note

que, como os anéis locais são viáveis, este procedimento tende a reduzir o tráfego no anel federal. Além disso, o procedimento de busca associado à N_3 não inviabiliza soluções viáveis. No entanto, não há garantia que o anel de menor demanda poderá ser esvaziado através da busca local associada a N_3. Investigamos ainda uma variação de N_3 que tenta esvaziar uma lista de anéis de S do menor para o maior em termos de demanda. Esta variação, que poderíamos chamar de $N_3^{'}$, lista os anéis por ordem crescente de demanda e, a cada iteração, tenta esvaziar um anel da lista. Inicialmente o primeiro anel da lista é escolhido e, se o esvaziamento for bem sucedido, a lista é recriada e o procedimento repetido. Caso contrário, o próximo anel da lista é escolhido e processo continua até que a lista seja inteiramente percorrida. É fácil perceber que $N_3^{'} \supset N_3$ e, portanto, o número de soluções investigadas por $N_3^{'}$ é maior. No entanto, percebemos empiricamente que se não for possível esvaziar o menor anel, a probabilidade de esvaziamento de anéis maiores é muito baixa. Essa constatação, aliada ao fato de que o custo, em termos de tempo computacional, de $N_3^{'}$ é muito mais elevado do que o de N_3, decidimos não utilizar $N_3^{'}$ em nosso procedimento GRASP. No entanto, aplicamos N_3 seguidamente se o menor anel for esvaziado.

Algoritmo 12 Procedimento de busca local N_3

```
1: procedimento buscaLocal_N3(S)
2: k ← lerNumeroAneis(S);
3: a_menor ← selecionarMenorAnel(S);
4: repita
5:    para cada v ∈ a_menor faça
6:       moveu ← moverParaOAnelMaisPromissor(v, S);
7:    fim para
8: até (a_menor = ∅) ou moveu = falso
9: se lerNumeroAneis(S) < k então
10:   S ← buscaLocal_N3(S);
11: fim se
12: retornar(S);
13: fim buscaLocal_N3
```

O Algoritmo 12 ilustra o pseudo-código do procedimento de busca local N_3. Inicialmente o algoritmo armazena o número de anéis da solução S na variável k (passo 2). O anel de menor demanda é selecionado e atribuído a a_{menor} no passo 3. Em seguida, nos passos 5 e 6, cada vértice $v \in a_{menor}$ é analisado na tentativa de movê-lo para um anel t tal que $t \neq a_{menor}$, $demT_{t\cup\{v\}} \leq B$ e $demT_{t\cup\{v\}} = min(demT_{r\cup\{v\}}), \forall r \in A(S)$ onde $A(S)$ é o conjunto de todos os anéis da solução S. No passo 9, o algoritmo verifica se a_{menor} foi esvaziado através da leitura do número atual de anéis e comparação com o número anterior de anéis de S. Caso afirmativo, a busca local é repetida sobre a solução atual

(passo 10). Finalmente, a solução resultante é retornada no passo 12.

Finalmente, investigamos uma quarta vizinhança, denominada de N_4 cujo objetivo é aumentar o número de anéis numa solução através da redistribuição de tráfego dos anéis de tal forma que o tráfego no anel federal seja reduzido e a solução seja viabilizada. Isso é razoável, pois para algumas instâncias, as heurísticas construtivas geram, muito freqüentemente, soluções inviáveis com menos anéis do que a solução ótima. A vizinhança N_4, nesses casos, é fundamental para a descoberta de soluções viáveis pois as demais vizinhanças de busca, discutidas anteriormente, não contemplam a possibilidade de aumento do número de anéis numa solução. A aplicação do procedimento N_4 é feita se não existir nenhuma solução viável descoberta e se, após um tempo t de processamento, a freqüência de soluções geradas com um número k_{min} de anéis for elevada. Estas soluções com k_{min} anéis são submetidas à N_4.

Algoritmo 13 Procedimento de busca local N_4

```
1: procedimento buscaLocal_N4(S)
2: a ← criarAnel(S);
3: S_nova ← S ∪ a;
4: valorMin ← demAF;
5: para i = 1, ..., n faça
6:    para cada j = 1, ..., n|j ∉ a faça
7:       a_j ← lerAnel(j);
8:       moverLocalidade(j, a);
9:       se calcularDemAF() < valorMin então
10:         valorMin ← calcularDemAF();
11:         l ← j;
12:      fim se
13:      moverLocalidade(j, a_j);
14:   fim para
15:   moverLocalidade(l, a);
16: fim para
17: se S_nova for viável então
18:    retornar(S_nova);
19: senão
20:    retornar(S);
21: fim se
22: fim buscaLocal_N4
```

O procedimento N_4 está detalhado no Algoritmo 13. Inicialmente o algoritmo cria um novo anel a seguido de uma nova solução S_{nova} formada pela união de S com a (passos 2 e 3). O laço que vai do passo 6 até o passo 14 procura o movimento que mais reduza a demanda no anel federal. Nos passos 7 e 8, a localidade j é movida para o anel a. O movimento é avaliado no passo 9 e, se for melhor que o movimento anterior, as variáveis são

atualizadas para guardar o novo movimento. No passo 13, o movimento é desfeito e uma nova localidade j é escolhida em seguida pelo laço. No passo 15, o melhor movimento é realizado. O algoritmo continua até que todas as localidades sejam avaliadas. Finalmente, se a solução gerada S_{nova} for viável é retornada no passo 18.

Como temos mais de uma vizinhança, após a fase de construção, é necessário decidir quais vizinhanças descritas acima serão aplicadas à solução construída e em que ordem isso será feito. Para ajudar a determinar o comportamento de cada uma das heurísticas de busca descritas acima, fora feitos vários testes onde as vizinhanças de busca foram aplicadas. O primeiro teste consistiu em aplicar cada vizinhança individualmente sobre a solução inicial produzida pela fase de construção. Em seguida, as quatro vizinhanças foram aplicadas sequencialmente sobre a solução inicial. Dessa forma, N_1 é aplicada sobre S_0 gerando uma nova solução S_1 que é então submetida à vizinhança N_2 e assim sucessivamente. Finalmente, as quatro vizinhanças foram aplicadas num esquema semelhante ao do segundo teste com a diferença de que cada vizinhança é aplicada sucessivamente até que não seja possível melhorar a solução e, nesse momento, a próxima vizinhança é selecionada.

Os testes foram realizados sobre um conjunto restrito de instâncias disponíveis na literatura. Os resultados do primeiro teste mostraram que os procedimentos de busca N_1, N_2 e N_4 são muito pouco eficientes se comparados com o procedimento N_3. Além disso, esses procedimentos apresentaram tempos computacionais bem mais elevados do que N_3. Os esquemas implementados pelos segundo e terceiro testes aumentaram demasiadamente o tempo computacional mas não melhoraram os resultados. Dessa forma, foi decidido utilizar apenas o procedimento N_3 como busca local no GRASP proposto aqui.

3.3.3 Reconexão de Caminhos

O algoritmo de reconexão de caminhos, em nossa implementação, funciona através de um conjunto $\mathcal{E}$ de soluções elite distintas encontradas durante a execução. Esse conjunto é formado pelas soluções encontradas durante as iterações prévias do GRASP.

Definimos uma solução S para o SRAP como um vetor de cardinalidade n. Cada localidade cliente i, com $i = 1, \ldots, n$, pertence a um anel indicado em $S(i)$. Cada anel a_i é numerado de forma que $i = min\{l | l \in a_i\}$. Desta forma, soluções com o mesmo número de anéis não podem ser numeradas de forma diferente.

Sejam S_1 e S_2 duas soluções, onde S_1 é a solução localmente ótima obtida após a

aplicação da busca local e S_2 é uma das soluções elite em $\mathcal{E}$. O procedimento de reconexão de caminhos inicia com uma das soluções (digamos, S_1) e gradualmente a transforma na outra (S_2) fazendo $S_1(i) = S_2(i)$ onde i é escolhido entre todas as localidades l onde $S_1(l) \neq S_2(l)$.

A escolha de qual movimento fazer em cada passo é gulosa: nós realizamos o movimento mais promissor, ou seja, menos custoso. A saída do procedimento é a melhor solução encontrada no caminho de S_1 para S_2.

Um aspecto importante da nossa heurística é a escolha das soluções S_1 e S_2. Empiricamente, observamos que a aplicação da reconexão de caminhos sobre um par de soluções com diferente número de anéis é menos provável de ser bem sucedida do que sobre um par de soluções com o mesmo número de anéis.

Por outro lado, quanto mais longo o caminho entre as soluções, maior a probabilidade de encontrar um ótimo local inteiramente novo. Além disso, é razoável levar em consideração não apenas a qualidade da solução, mas também a diversidade ao se gerenciar as soluções do conjunto de soluções elite.

Dessa forma, implementamos uma estratégia para a seleção das soluções S_1 e S_2. Seja $na(S)$ o número de anéis em S, temos as seguintes situações às quais aplicamos o procedimento de reconexão de caminhos:

(i) $na(S_1) = na(S_2)$ e, S_1 e S_2 são inviáveis;

(ii) $na(S_1) < na(S_2)$ e S_1 é inviável;

(iii) $na(S_1) > na(S_2)$ e S_2 é inviável.

Se a condição (ii) for satisfeita reduzimos o número de anéis em S_2 através do esvaziamento do seu menor anel r_l movendo seus nós para os anéis mais promissores excluindo o próprio r_l. Repetimos esse processo até que $na(S_1) = na(S_2)$. Se a condição (iii) for satisfeita aplicamos a mesma estratégia para reduzir o número de anéis em S_1. Note que, ao fazer isso, e, exceto se S_1 se tornar viável ou S_2 se tornar viável, as situações (ii) e (iii) deixam de existir e a situação (i) se torna satisfeita.

Nos casos (iv), (v) e (vi) abaixo, não aplicamos o algoritmo de reconexão de caminhos porque a existência de uma solução viável o impede de encontrar uma solução viável com menor número de anéis:

(iv) $na(S_1) = na(S_2)$ e, S_1 é viável ou S_2 é viável;

(v) $na(S_1) < na(S_2)$ e S_1 é viável;

(vi) $na(S_1) > na(S_2)$ e S_2 é viável.

Após esta seleção, geramos dois caminhos, um de S_1 para S_2 e o outro de S_2 para S_1. Isto é feito porque, muitas vezes, estes caminhos visitam soluções intermediárias diferentes.

Um outro ponto de vista importante da heurística é o gerenciamento do conjunto de soluções elite. Em [22], os autores atualizam $\mathcal{E}$ através da manutenção das três soluções de melhor qualidade. Nós utilizamos uma outra abordagem para a obtenção das soluções elite no GRASP. Esta abordagem é explicada abaixo.

Vamos assumir que NR indica o número meta de anéis para as soluções candidatas ao procedimento de reconexão de caminhos. O valor de NR é definido no intervalo $[z_{lb}, na(S_{melhor})]$ com $z_{lb} = \left(\sum_{u \in V} \sum_{v \in V | u<v} d_{uv}\right) / B$ e $na(S_{melhor})$ é igual ao número de anéis da melhor solução encontrada S_{melhor}.

Algoritmo 14 Procedimento de Reconexão de Caminhos

```
1: procedimento PATHRL(n, S, T)
2: S_aux ← S;
3: S_mrc ← (z(T) < z(S) ? T : S);
4: para i = 1, ..., n faça
5:    para j = 1, ..., n faça
6:       se (S(i) ≠ T(i)) e (S(j) ≠ T(j)) então
7:          S(j) ← T(j);
8:          se (z(S) < z(S_aux)) então
9:             k ← j;
10:         fim se
11:         S(j) ← S_aux(j);
12:      fim se
13:   fim para
14:   S_aux(k) ← T(k);
15:   S(k) ← T(k);
16:   se z(S_aux) < z(S_mrc) então
17:      S_mrc ← S_aux;
18:   fim se
19: fim para
20: retornar(S_mrc);
21: fim PATHRL
```

Dessa forma, uma solução S é adicionada a $\mathcal{E}$ se ela satisfizer a condição: $na(S) = NR$. Uma vez aceita para inserção em P, a solução S substitui a última solução elite em P, que é retirada de P. Todas as soluções candidatas com mais que NR anéis precisam ter

seu número de anéis reduzido para serem submetidas à reconexão de caminhos, de acordo com as regras (i) a (iii) explicadas acima.

O Algoritmo 14 mostra os passos do procedimento de reconexão de caminhos para o SRAP descrito acima. Seja S a solução atual obtida pelo GRASP puro e T a solução guia. O procedimento fará no máximo n movimentos de troca de localidades clientes (passos 4 a 19) até que a solução atual se torne igual à solução guia T. A cada passo, a melhor troca é encontrada (passos 5 a 13) e aplicada a S (passos 14 e 15). O algoritmo também verifica se a solução gerada é melhor que a melhor solução conhecida S_{mrc} e, assim, a atualiza (passos 16 e 17). Finalmente, a melhor solução encontrada é retornada no passo 20.

A seguir, na Seção 3.3.4, mostraremos como cada um dos procedimentos e fases descritas até aqui são combinadas em nossa implementação.

3.3.4 GRASP com Reconexão de Caminhos

O Algoritmo 15 mostra o procedimento GRASP com reconexão de caminhos proposto. Inicialmente, o conjunto de soluções elite está vazio e a melhor solução S_{melhor} é obtida através do procedimento `construirSolucaoInicial`() que retorna a melhor solução puramente heurística e determinística (passos 2 e 3, respectivamente). Em outras palavras, este procedimento executa as heurísticas *edge-based*, *cut-based*, *node-based* e HVR sem aleatoriedade, ou seja, de forma puramente gulosa e aplica busca local sobre cada solução encontrada. A melhor solução gerada através desse processo é retornada.

A estrutura de laço no passo 4 controla a execução do algoritmo por no máximo `maxItr` iterações. A cada iteração, uma solução S é construída utilizando o algoritmo `CONSTRUIR_SOLUCAO` mostrado na Seção 3.3.1.5 (passo 5). Em seguida, S é submetida à busca local mostrada na Seção 3.3.2 (passo 5) para transformá-la na solução localmente ótima em sua vizinhança. No passo 7, S é comparada com a melhor solução global S_{melhor} e, se necessário, S_{melhor} é atualizada através do procedimento `atualizarMelhor`.

A condição $i >$ `nItrPR` (passo 8) determina o momento do início da execução do procedimento de reconexão de caminhos, bem como, da alimentação do conjunto de soluções elite. No passo 9, uma solução com o mesmo número de anéis de S é retirada de $\mathcal{E}$ caso houver. A condição do passo 10, verifica se T foi encontrada em $\mathcal{E}$. Caso afirmativo, e S seja inviável o procedimento é executado no sentido de S para T (passo 12). No passo 13, $\mathcal{E}$ é atualizado com a solução resultante r. Em seguida, no passo 14, o procedimento

Algoritmo 15 GRASP com reconexão de caminhos proposto

```
1: procedimento GRASP-PR(n)
2: P = ∅;
3: S_melhor ← construirSolucaoInicial();
4: para i = 1 to maxItr faça
5:    S ← CONSTRUIR_SOLUCAO();
6:    S ← aplicarBuscaLocal(S);
7:    atualizarMelhor(S, S_melhor);
8:    se i > nItrPR então
9:       T ← P(i) tal que na(P(i)) = na(S);
10:      se T ≠ ∅ então
11:         se não eViavel(S) então
12:            r ← PATHRL(n, S, T);
13:            atualizarConjuntoElite(r, P);
14:            atualizarMelhor(r, S_melhor);
15:            r ← PATHRL(n, T, S);
16:            atualizarConjuntoElite(r, P);
17:            atualizarMelhor(r, S_melhor);
18:         fim se
19:      senão se na(S) ∈ [z_lb, na(S_melhor) − 1] então
20:         atualizarConjuntoElite(S, P);
21:      fim se
22:   fim se
23: fim para
24: retornar S_melhor;
25: fim GRASP-PR.
```

`atualizarMelhor` atualiza a melhor solução S_{melhor} se necessário. Os passos 15 a 17, desempenham as mesmas tarefas da reconexão de caminhos, atualização de $\mathcal{E}$ e atualização da melhor solução, com a diferença que a reconexão de caminhos é realizada no sentido inverso, ou seja, de T para S.

Caso, não exista uma solução em $\mathcal{E}$ com o mesmo número de anéis de S e $na(S)$ pertencer ao intervalo de soluções meta, S é submetida ao procedimento `atualizarConjuntoElite` (passo 20). A melhor solução é retornada no passo 25.

Capítulo 4

Implementação e Resultados Computacionais

Neste capítulo, mostraremos os resultados computacionais obtidos em cada experimento realizado durante o nosso trabalho. Além disso, analisaremos esses resultados com relação aos resultados esperados teoricamente.

Os algoritmos descritos no Capítulo 3 foram implementados em linguagem C++. Foi desenvolvido um *framework*, composto basicamente de classes de listas, matrizes e procedimentos otimizados para tratar o SRAP, como base para o desenvolvimento dos algoritmos descritos neste trabalho. Os testes foram executados num computador AMD Athlon XP 2800+ (2083 MHz) com 512 MB de memória RAM.

As instâncias utilizadas nos testes foram classificadas em três classes. As propriedades e os detalhes da geração dessas instâncias estão descritos na seção 4.1. Os resultados computacionais, por sua vez, são apresentados na seção 4.2.

4.1 Instâncias do Problema

Executamos nossos testes sobre três classes de instâncias. As duas primeiras definidas originalmente por Goldschmidt *et al.* [3] e por Aringhieri e Dell'Amico [5]. A terceira classe de instâncias introduzimos aqui. A partir desse ponto, chamaremos as instâncias produzidas por Goldschmidt *et al.* de classe C1, as produzidas por Aringhieri e Dell'Amico de classe C2 e as produzidas neste trabalho de classe C3.

4.1.1 Classes C1 e C2

As classes de instâncias C1 e C2 são compostas por dois grupos de instâncias: geométricas e aleatórias. As instâncias geométricas apresentam agrupamentos naturais, ou seja, uma

localidade se comunica com mais freqüência com as localidades mais próximas a ela do que com as localidades mais distantes. As instâncias aleatórias, por sua vez, foram geradas pelos autores a partir de grafos completos com a retirada de arestas com probabilidade $p \in (0, 1)$. Cada grupo é composto por instâncias de capacidades diferentes para os anéis: $B = 155MB/s$ e $B = 622MB/s$. A nomenclatura dessas instâncias está relacionada à classe e às características das mesmas. Para a classe C1, os nomes das instâncias geométricas com $B = 155MB/s$ começam com GS (*Geometric Short*). Já as instâncias geométricas com $B = 622MB/s$ têm nomes começando por GL (*Geometric Large*). As instâncias aleatórias da classe C1 possuem nomes começando por RS (*Random Short*) e RL (*Random Large*) correspondendo às limitações de tráfego dadas por $B = 155MB/s$ e $B = 622MB/s$, respectivamente. Por sua vez, as instâncias da classe C2, possuem nomes começando com new.GL (*Geometric Low*), new.GH (*Geometric High*), new.RL (*Random Low*) e new.RH (*Random High*). Esses prefixos correspondem, respectivamente, às instâncias geométricas com $B = 155MB/s$ e $B = 622MB/s$ e às instâncias aleatórias com $B = 155MB/s$ e $B = 622MB/s$.

A classe C1 é composta por 118 instâncias viáveis e a classe C2 por 230 instâncias viáveis. Macambira, em [10], provou a otimalidade e forneceu os valores das soluções ótimas dessas instâncias.

4.1.2 Classe C3

A geração de instâncias com número elevado de localidades é uma tarefa bastante complexa devido à limitação de tráfego no anel federal dada por B. Ou seja, dado um algoritmo que distribui aleatoriamente demandas de tráfego entre as arestas de um grafo $G = (V, E)$ e dado um valor de B, quanto maior o número de localidades ($\|V\|$), maior a demanda total do grafo. Por outro lado, quanto maior a demanda total, maior o número mínimo de anéis necessários para gerar uma solução viável. Um número elevado de anéis implica numa quantidade de tráfego muito grande passando pelo anel federal e possivelmente numa instância inviável. Podemos tentar lidar com esse problema aumentando o valor de B ou reduzindo a cardinalidade de E.

Para gerar as instâncias da classe C3 foram propostos dois algoritmos. O primeiro deles, chamado de RTD (*Random by Traffic Density*), consiste em gerar um grafo aleatório dado um percentual de densidade de tráfego D_t desejado. Dessa forma, uma densidade de 100% gera um grafo 100% completo e uma densidade de 0% gera um grafo 100% desconexo. Dados os parâmetros n e B, quanto menor a densidade de tráfego, maior a

probabilidade de gerar uma instância viável do problema. Contudo, uma densidade de tráfego excessivamente reduzida pode gerar uma instância muito fácil de ser solucionada. O algoritmo RTD não garante que uma instância gerada seja viável e, além disso, exige a determinação dos parâmetros n, B e D_t num processo de tentativa e erro manual, demorado e pouco eficiente.

Na tentativa de solucionar as principais deficiências do algoritmo RTD foi proposto um novo algoritmo denominado RTNR (*Random with Target Number of Rings*), cujo propósito é gerar instâncias viáveis e, além disso, fornecer uma solução viável para cada instância gerada. O algoritmo RTNR é suprido com um número alvo de anéis para gerar um vetor solução inicial (solução base). Em seguida, o algoritmo RTNR tentará produzir uma instância cuja solução base pertença ao seu conjunto de soluções e, com isso, garantir que a mesma seja viável. A construção da instância é feita passo-a-passo através da distribuição aleatória das demandas de tráfego entre as arestas do grafo. Além do número alvo de anéis para a solução base, o algoritmo recebe como entrada a densidade do grafo desejada. De uma forma geral, uma maior densidade no grafo produz uma instância mais difícil de ser solucionada. Além disso, quanto maior o número de localidades mais difícil é a instância gerada.

A classe C3 foi produzida através do algoritmo RTNR com a geração de 99 instâncias viáveis com soluções conhecidas. A nomenclatura dessas instâncias não está relacionada com a capacidade dos anéis dada por B, mas pelo número de localidades da instância. Dessa forma, temos na classe C3 soluções nomeadas pelos prefixos RTNR_100, RTNR_150, RTNR_200, RTNR_250 e RTNR_300 seguidos pelo número da instância. Os valores de B utilizados na geração dessas instâncias variaram entre 51,84 MBps (STS-1) e 39813,12 MBps (STS-768) conforme a Tabela 2.1.

Por serem maiores e mais densas do que aquelas das classes C1 e C2, as instâncias da classe C3 são também mais difíceis. Para essas instâncias os valores das soluções ótimas ainda não são conhecidos.

4.2 Resultados Computacionais

Nas seções 4.2.1, 4.2.2 e 4.2.3 serão apresentados os resultados dos testes realizados neste trabalho. Serão fornecidos dados para comparação do desempenho do GRASP sem ativar a reconexão de caminhos com o desempenho do GRASP ativando a reconexão de caminhos. Desse ponto em diante, o GRASP sem reconexão de caminhos será chamdo simples-

mente de GRASP e o GRASP com reconexão de caminhos será chamado de GRASP+PR.

Resultados anteriores para as classes C1, C2 e C3 deram origem aos trabalhos [15, 25, 26].

4.2.1 Resultados da Classe C1

Para investigar o desempenho do algoritmo proposto sobre as instâncias da classe C1, desenvolvemos uma série de testes. Primeiramente, executamos o GRASP dez vezes sobre cada instância do problema e, em seguida, executamos o GRASP+PR dez vezes sobre essas mesmas instâncias. Em cada execução, utilizamos os mesmos parâmetros para ambos os algoritmos. O critério de parada utilizado foi encontrar uma solução viável com um número meta de anéis. Fixamos esse número meta igual ao número de anéis da solução ótima de cada instância. Além disso, caso a solução alvo não seja encontrada num máximo de 5000 iterações o algoritmo é parado. No caso do GRASP+PR a reconexão de caminhos foi ativada após 500 iterações para as instâncias das classes C1 e C2 e após 50 iterações para as instâncias da Classe C3. Os resultados desses experimentos, são apresentados nas Tabelas 4.1 a 4.4 em nove colunas. As primeiras três colunas trazem informações sobre a instância do problema que são: nome da instância, o número de anéis da solução ótima e o limite inferior k_{lb}. As três colunas seguintes trazem o número de anéis da melhor solução viável obtida pelo GRASP, o tempo computacional médio, em segundos, obtido nas dez execuções do algoritmo e a taxa de convergência ao alvo por execução do algoritmo. As três últimas colunas trazem essas mesmas informações para o GRASP+PR. A última linha de cada tabela, exibe as médias globais de tempo e as taxas de convergência globais para as instâncias da tabela. Para as instâncias da classe C1 a taxa de convergência corresponde ao número percentual de vezes qua a solução ótima foi encontrada.

As Tabelas 4.1 e 4.2 apresentam os resultados das instâncias geométricas dessa classe enquanto que as Tabelas 4.3 e 4.4 trazem os resultados das instâncias aleatórias.

A Tabela 4.1 exibe os resultados computacionais para as instâncias geométricas da classe C1 com $B = 155MB/s$. Podemos observar que o percentual de soluções ótimas encontradas nas dez execuções tanto do GRASP quanto do GRASP+PR foi de 100.0%. Os tempos computacionais, por sua vez, sofreram pequenas variações entre os algoritmos. Ora o GRASP apresentou menor tempo médio e ora o GRASP+PR o fez. Além disso, esses tempos foram de no máximo $0,011$ segundos para ambos os algoritmos e correspondem a pouquíssimas iterações. Verificamos, também, que as instâncias foram rapidamente

Tabela 4.1: Resultados computacionais para as instâncias geométricas da classe C1 com $B = 155MB/s$.

Instância			GRASP			GRASP+PR		
Nome	Opt	k_{lb}	# Anéis	Tempo (s)	TC(%)	# Anéis	Tempo (s)	TC(%)
GS_15_1	3	3	3	0,003	100,0	3	0,001	100,0
GS_15_4	3	2	3	0,002	100,0	3	0,000	100,0
GS_15_7	3	3	3	0,002	100,0	3	0,001	100,0
GS_15_9	3	2	3	0,000	100,0	3	0,006	100,0
GS_25_1	4	3	4	0,003	100,0	4	0,000	100,0
GS_25_3	3	3	3	0,002	100,0	3	0,002	100,0
GS_25_4	4	3	4	0,003	100,0	4	0,003	100,0
GS_25_7	3	3	3	0,000	100,0	3	0,002	100,0
GS_25_8	4	3	4	0,001	100,0	4	0,000	100,0
GS_30_1	4	3	4	0,003	100,0	4	0,003	100,0
GS_30_2	4	3	4	0,002	100,0	4	0,008	100,0
GS_30_3	4	3	4	0,000	100,0	4	0,000	100,0
GS_30_4	4	3	4	0,001	100,0	4	0,001	100,0
GS_30_5	3	3	3	0,005	100,0	3	0,005	100,0
GS_30_6	4	3	4	0,002	100,0	4	0,000	100,0
GS_30_7	4	3	4	0,002	100,0	4	0,002	100,0
GS_30_8	4	3	4	0,002	100,0	4	0,002	100,0
GS_30_9	4	3	4	0,003	100,0	4	0,002	100,0
GS_50_1	5	4	5	0,006	100,0	5	0,008	100,0
GS_50_4	6	5	6	0,011	100,0	6	0,009	100,0
GS_50_7	5	4	5	0,005	100,0	5	0,005	100,0
GS_50_8	5	5	5	0,011	100,0	5	0,011	100,0
GS_50_9	4	4	4	0,005	100,0	4	0,003	100,0
Médias			GRASP	0,003	100,0	GRASP+PR	0,003	100,0

resolvidas antes que a ativação da reconexão de caminhos ocorresse no GRASP+PR. Apesar da utilização dos mesmos parâmetros, da execução de cada algoritmo com o mínimo possível de processos concorrentes e da solução das instâncias antes da ativação da reconexão de caminhos, pequenas diferenças de tempo são esperadas. Essas diferenças podem ser devidas à diferentes demandas de CPU por outros processos no momento da execução dos algoritmos. No entanto, o tempo médio global da classe foi a mesma para os dois algoritmos ficando em $0,003$ segundos.

A Tabela 4.2 exibe os resultados computacionais para as instâncias geométricas da classe C1 com $B = 622MB/s$. As observações que fazemos dos resultados dessa tabela são similares àquelas comentadas sobre os resultados da Tabela 4.1. Novamente, a taxa de convergência tanto do GRASP quanto do GRASP+PR foi de $100,0\%$. Os tempos computacionais, por sua vez, variaram entre $0,000$ e $0,041$ segundo para o GRASP e entre $0,000$ e $0,042$ segundo para o GRASP+PR. Além disso, essas instâncias também foram resol-

Tabela 4.2: Resultados computacionais para as instâncias geométricas da classe C1 com $B = 622MB/s$.

Instância			GRASP			GRASP+PR		
Nome	Opt	k_{lb}	# Anéis	Tempo (s)	TC(%)	# Anéis	Tempo (s)	TC(%)
GL_15_1	3	2	3	0,000	100,0	3	0,000	100,0
GL_15_2	3	2	3	0,000	100,0	3	0,002	100,0
GL_15_3	2	2	2	0,002	100,0	2	0,000	100,0
GL_15_6	2	2	2	0,000	100,0	2	0,000	100,0
GL_15_7	2	2	2	0,002	100,0	2	0,000	100,0
GL_15_8	3	2	3	0,002	100,0	3	0,000	100,0
GL_15_9	3	2	3	0,002	100,0	3	0,000	100,0
GL_15_10	2	2	2	0,002	100,0	2	0,000	100,0
GL_25_1	3	2	3	0,001	100,0	3	0,002	100,0
GL_25_2	2	2	2	0,005	100,0	2	0,003	100,0
GL_25_3	2	2	2	0,003	100,0	2	0,003	100,0
GL_25_4	3	3	3	0,003	100,0	3	0,008	100,0
GL_25_7	4	3	4	0,009	100,0	4	0,011	100,0
GL_25_8	3	3	3	0,001	100,0	3	0,000	100,0
GL_25_9	3	3	3	0,000	100,0	3	0,000	100,0
GL_30_1	3	3	3	0,005	100,0	3	0,002	100,0
GL_30_2	4	3	4	0,002	100,0	4	0,002	100,0
GL_30_3	4	3	4	0,005	100,0	4	0,002	100,0
GL_30_4	3	3	3	0,002	100,0	3	0,000	100,0
GL_30_5	4	3	4	0,003	100,0	4	0,006	100,0
GL_30_9	4	3	4	0,002	100,0	4	0,002	100,0
GL_30_10	3	2	3	0,005	100,0	3	0,003	100,0
GL_50_1	5	4	5	0,005	100,0	5	0,011	100,0
GL_50_4	4	4	4	0,041	100,0	4	0,042	100,0
GL_50_5	5	4	5	0,011	100,0	5	0,012	100,0
GL_50_6	5	4	5	0,006	100,0	5	0,011	100,0
GL_50_7	5	4	5	0,016	100,0	5	0,011	100,0
Médias			GRASP	0,005	100,0	GRASP+PR	0,005	100,0

vidas antes que a ativação da reconexão de caminhos ocorresse na versão GRASP+PR. Semelhantemente às instâncias da Tabela 4.1, pequenas diferenças nos tempos médios são esperadas. Novamente, as médias de tempo dos dois algoritmos ficaram empatadas em $0,005$ segundos.

A Tabela 4.3 mostra os resultados computacionais para as instâncias aleatórias da classe C1 com $B = 155MB/s$. Mesmo com a taxa de convergência de 100,0% tanto do GRASP quanto do GRASP+PR, a instância RS_30_6 parece ligeiramente mais difícil do que as instâncias geométricas da mesma classe levando os algoritmos a gastar mais tempo, na média, para chegar à solução ótima. Os tempos computacionais variaram entre $0,000$ e $0,169$ segundo para o GRASP e entre $0,000$ e $0,176$ segundo para o GRASP+PR.

Tabela 4.3: Resultados computacionais para as instâncias aleatórias da classe C1 com $B = 155MB/s$.

Instância			GRASP			GRASP+PR		
Nome	Opt	k_{lb}	# Anéis	Tempo (s)	TC(%)	# Anéis	Tempo (s)	TC(%)
RS_15_1	3	2	3	0,000	100,0	3	0,002	100,0
RS_15_3	2	2	2	0,001	100,0	2	0,000	100,0
RS_15_4	3	2	3	0,000	100,0	3	0,000	100,0
RS_15_6	3	2	3	0,002	100,0	3	0,002	100,0
RS_15_8	3	2	3	0,000	100,0	3	0,000	100,0
RS_15_9	3	2	3	0,000	100,0	3	0,000	100,0
RS_15_10	3	2	3	0,003	100,0	3	0,005	100,0
RS_25_1	4	3	4	0,148	100,0	4	0,120	100,0
RS_25_2	3	3	3	0,000	100,0	3	0,000	100,0
RS_25_4	4	3	4	0,028	100,0	4	0,033	100,0
RS_25_5	4	3	4	0,000	100,0	4	0,005	100,0
RS_25_6	4	3	4	0,005	100,0	4	0,006	100,0
RS_25_7	4	3	4	0,008	100,0	4	0,013	100,0
RS_25_8	3	2	3	0,003	100,0	3	0,000	100,0
RS_25_9	4	3	4	0,003	100,0	4	0,002	100,0
RS_25_10	4	3	4	0,008	100,0	4	0,005	100,0
RS_30_1	3	3	3	0,002	100,0	3	0,005	100,0
RS_30_3	4	3	4	0,055	100,0	4	0,067	100,0
RS_30_4	4	3	4	0,008	100,0	4	0,005	100,0
RS_30_5	4	3	4	0,005	100,0	4	0,005	100,0
RS_30_6	4	3	4	0,169	100,0	4	0,176	100,0
RS_30_7	3	3	3	0,006	100,0	3	0,006	100,0
RS_30_8	4	3	4	0,005	100,0	4	0,003	100,0
RS_30_10	4	3	4	0,005	100,0	4	0,006	100,0
RS_50_1	5	4	5	0,091	100,0	5	0,095	100,0
RS_50_3	4	4	4	0,014	100,0	4	0,020	100,0
RS_50_4	4	4	4	0,020	100,0	4	0,017	100,0
RS_50_5	4	4	4	0,002	100,0	4	0,011	100,0
RS_50_6	4	3	4	0,012	100,0	4	0,008	100,0
RS_50_7	5	4	5	0,020	100,0	5	0,022	100,0
RS_50_10	4	3	4	0,005	100,0	4	0,008	100,0
Médias			GRASP	0,020	100,0	GRASP+PR	0,021	100,0

Semelhante às instâncias anteriores dessa classe, essas últimas também foram resolvidas antes que a ativação da reconexão de caminhos ocorresse na versão GRASP+PR do algoritmo.

Finalmente, a Tabela 4.4 exibe os resultados computacionais para as instâncias aleatórias da classe C1 com $B = 622MB/s$. A instância RL_50_3 se mostrou bem mais difícil do que as outras instâncias da mesma classe levando os algoritmos a gastar mais tempo, em média, para chegar à solução ótima. O GRASP obteve um tempo médio de

Tabela 4.4: Resultados computacionais para as instâncias aleatórias da classe C1 com $B = 622MB/s$.

Instância			GRASP			GRASP+PR		
Nome	Opt	k_{lb}	# Anéis	Tempo (s)	TC(%)	# Anéis	Tempo (s)	TC(%)
RL_15_1	3	2	3	0,000	100,0	3	0,000	100,0
RL_15_3	3	2	3	0,003	100,0	3	0,001	100,0
RL_15_4	2	2	2	0,002	100,0	2	0,002	100,0
RL_15_5	3	2	3	0,000	100,0	3	0,001	100,0
RL_15_6	3	2	3	0,003	100,0	3	0,002	100,0
RL_15_7	2	2	2	0,002	100,0	2	0,002	100,0
RL_15_8	2	2	2	0,005	100,0	2	0,001	100,0
RL_15_9	3	2	3	0,000	100,0	3	0,000	100,0
RL_25_1	3	3	3	0,031	100,0	3	0,031	100,0
RL_25_2	3	2	3	0,003	100,0	3	0,002	100,0
RL_25_3	3	2	3	0,002	100,0	3	0,003	100,0
RL_25_4	3	3	3	0,139	100,0	3	0,113	100,0
RL_25_5	4	3	4	0,998	30,0	4	0,877	100,0
RL_25_6	4	3	4	0,005	100,0	4	0,000	100,0
RL_25_7	3	2	3	0,000	100,0	3	0,002	100,0
RL_25_8	4	3	4	0,028	100,0	4	0,028	100,0
RL_25_9	2	2	2	0,001	100,0	2	0,002	100,0
RL_25_10	3	2	3	0,003	100,0	3	0,000	100,0
RL_30_1	3	2	3	0,002	100,0	3	0,000	100,0
RL_30_2	4	3	4	0,005	100,0	4	0,008	100,0
RL_30_3	3	2	3	0,002	100,0	3	0,002	100,0
RL_30_4	3	2	3	0,003	100,0	3	0,002	100,0
RL_30_6	4	3	4	0,227	100,0	4	0,125	100,0
RL_30_7	4	3	4	0,005	100,0	4	0,006	100,0
RL_30_8	3	3	3	0,006	100,0	3	0,009	100,0
RL_30_9	3	3	3	0,005	100,0	3	0,003	100,0
RL_30_10	3	3	3	0,363	100,0	3	0,247	100,0
RL_50_1	4	3	4	0,012	100,0	4	0,013	100,0
RL_50_2	3	3	3	0,016	100,0	3	0,013	100,0
RL_50_3	4	4	-	5,480	0,0	4	73,120	50,0
RL_50_4	4	3	4	2,289	100,0	4	2,764	100,0
RL_50_5	3	3	3	0,016	100,0	3	0,005	100,0
RL_50_6	3	3	3	0,013	100,0	3	0,012	100,0
RL_50_7	5	4	5	5,380	10,0	5	1,274	100,0
RL_50_8	4	3	4	0,009	100,0	4	0,009	100,0
RL_50_9	4	3	4	0,011	100,0	4	0,011	100,0
RL_50_10	4	4	4	0,123	100,0	4	0,122	100,0
Médias			GRASP	0,411	93,0	GRASP+PR	2,130	98,6

5,480 segundos para essa instância e não encontrou o alvo. O GRASP+PR, por sua vez, apresentou 50,0% de convergência num tempo médio de 73,120 segundos. A instância RL_50_3 foi a única dessa classe onde o procedimento de reconexão de caminhos foi

necessário para garantir a otimalidade do resultado.

A maioria das instâncias com média acima de 0,1 segundos para o GRASP+PR tiveram sua média de tempo reduzidas por conta do procedimento de Reconexão de Caminhos. Os tempos computacionais médios para essas instâncias foram de 0,411 segundo para o GRASP e de 2,130 segundos para o GRASP+PR. As taxas de convergência médias ficaram em 93,0% para o GRASP e em 98,6% para o GRASP+PR.

Vale ressaltar que a média global de tempo do algoritmo GRASP+PR foi prejudicada por conta da instância RL_50_3 com o conjunto de parâmetros utilizados. Através do ajuste de parâmetros, como adiamento da ativação da reconexão de caminhos (nItrPR), tamanho do conjunto de soluções elite, freqüência de aplicação da reconexão de caminhos e mudança das sementes aleatórias, conseguimos reduzir a média de tempo computacional dessa instância para 7,318 segundos com 100% de convergência para a solução ótima. No entanto, não seria apropriado esse tipo de ajuste fino por instância para a análise global dos algoritmos e portanto o fizemos apenas a título de ilustração e, portanto, não incluímos esses resultados na Tabela 4.4.

4.2.2 Resultados da Classe C2

Realizamos os mesmos experimentos, descritos na Seção 4.2.1, sobre as instâncias da classe C2. Os resultados desses experimentos, são apresentados nas Tabelas 4.5 a 4.11. Essas tabelas seguem a mesma estrutura de colunas explicada na seção 4.2.1.

As Tabelas 4.5 a 4.8 apresentam os resultados das instâncias geométricas dessa classe enquanto que as Tabelas 4.9 a 4.11 trazem os resultados das instâncias aleatórias.

As Tabelas 4.5 e 4.6 trazem os resultados dos testes para as instâncias geométricas da classe C2 com $B = 155MB/s$. Para 6 dessas instâncias o GRASP não obteve taxa de convergência de 100%. O GRASP+PR, por sua vez, convergiu em 100% das execuções exceto para a instância new.GL_50_10.4 com 70% de convergência contra os 50% obtidos pelo GRASP para a mesma instância.

Para a maioria das instâncias onde o GRASP não conseguiu encontrar a solução alvo em todas as execuções, o GRASP+PR obteve um tempo computacional médio menor que o GRASP. Isso indica que, apesar de mais custoso em termos de tempo, o procedimento de reconexão de caminhos acelerou a convergência do algoritmo para a solução alvo nesses casos e, conseqüentemente, reduziu o número médio de iterações executadas. A exceção ficou por conta da instância new.GL_50_10.1 para a qual o tempo compu-

Tabela 4.5: Resultados computacionais para as instâncias geométricas da classe C2 com $B = 155MB/s$.

Instância			GRASP			GRASP+PR		
Nome	Opt	k_{lb}	# Anéis	Tempo (s)	TC(%)	# Anéis	Tempo (s)	TC(%)
new.GL_15_3.1	4	3	4	0,000	100,0	4	0,005	100,0
new.GL_15_3.2	4	3	4	0,005	100,0	4	0,005	100,0
new.GL_15_3.3	4	3	4	0,006	100,0	4	0,006	100,0
new.GL_15_3.4	4	3	4	0,003	100,0	4	0,003	100,0
new.GL_15_3.5	4	3	4	0,017	100,0	4	0,013	100,0
new.GL_15_3.6	4	3	4	0,005	100,0	4	0,006	100,0
new.GL_15_3.7	4	3	4	0,003	100,0	4	0,002	100,0
new.GL_15_3.8	4	3	4	0,005	100,0	4	0,005	100,0
new.GL_15_3.9	4	3	4	0,017	100,0	4	0,014	100,0
new.GL_15_3.10	4	3	4	0,005	100,0	4	0,003	100,0
new.GL_15_6.1	3	2	3	0,000	100,0	3	0,003	100,0
new.GL_15_6.2	3	2	3	0,013	100,0	3	0,011	100,0
new.GL_15_6.3	3	2	3	0,005	100,0	3	0,002	100,0
new.GL_15_6.4	3	2	3	0,006	100,0	3	0,003	100,0
new.GL_15_6.5	3	2	3	0,005	100,0	3	0,006	100,0
new.GL_15_6.6	3	2	3	0,000	100,0	3	0,001	100,0
new.GL_15_6.7	3	2	3	0,002	100,0	3	0,002	100,0
new.GL_15_6.8	3	2	3	0,014	100,0	3	0,025	100,0
new.GL_15_6.9	3	2	3	0,002	100,0	3	0,000	100,0
new.GL_15_6.10	3	2	3	0,000	100,0	3	0,005	100,0
new.GL_25_9.1	4	3	4	0,255	100,0	4	0,147	100,0
new.GL_25_9.2	4	3	4	0,027	100,0	4	0,028	100,0
new.GL_25_9.3	4	3	4	0,008	100,0	4	0,005	100,0
new.GL_25_9.4	4	3	4	0,077	100,0	4	0,062	100,0
new.GL_25_9.5	4	3	4	0,276	90,0	4	0,153	100,0
new.GL_25_9.6	4	3	4	0,008	100,0	4	0,009	100,0
new.GL_25_9.7	4	3	4	0,242	100,0	4	0,086	100,0
new.GL_25_9.8	4	3	4	0,022	100,0	4	0,020	100,0
new.GL_25_9.9	4	3	4	0,006	100,0	4	0,005	100,0
new.GL_25_9.10	4	3	4	0,006	100,0	4	0,005	100,0
new.GL_25_10.1	5	4	5	0,025	100,0	5	0,028	100,0
new.GL_25_10.2	5	4	5	0,016	100,0	5	0,019	100,0
new.GL_25_10.3	5	4	5	0,008	100,0	5	0,009	100,0
new.GL_25_10.4	5	4	5	0,022	100,0	5	0,019	100,0
new.GL_25_10.5	5	4	5	0,016	100,0	5	0,016	100,0
new.GL_25_10.6	5	4	5	0,009	100,0	5	0,009	100,0
new.GL_25_10.7	5	4	5	0,400	100,0	5	0,188	100,0
new.GL_25_10.8	5	4	5	0,012	100,0	5	0,012	100,0
new.GL_25_10.9	5	4	5	0,005	100,0	5	0,005	100,0
new.GL_25_10.10	5	4	5	0,009	100,0	5	0,010	100,0

tacional médio do GRASP+PR foi de 44,269 segundos, portanto, bem mais elevado que

Tabela 4.6: Resultados computacionais para as instâncias geométricas da classe C2 com $B = 155MB/s$ (Continuação).

Instância			GRASP			GRASP+PR		
Nome	Opt	k_{lb}	# Anéis	Tempo (s)	TC(%)	# Anéis	Tempo (s)	TC(%)
new.GL_30_10.1	5	4	5	0,036	100,0	5	0,036	100,0
new.GL_30_10.2	5	4	5	0,009	100,0	5	0,009	100,0
new.GL_30_10.3	5	4	5	0,044	100,0	5	0,050	100,0
new.GL_30_10.4	5	4	5	0,027	100,0	5	0,030	100,0
new.GL_30_10.5	5	4	5	0,059	100,0	5	0,058	100,0
new.GL_30_10.6	5	4	5	0,170	100,0	5	0,119	100,0
new.GL_30_10.7	5	4	5	0,014	100,0	5	0,013	100,0
new.GL_30_10.8	5	4	5	0,011	100,0	5	0,008	100,0
new.GL_30_10.9	5	4	5	0,042	100,0	5	0,039	100,0
new.GL_30_10.10	5	4	5	0,016	100,0	5	0,016	100,0
new.GL_50_6.1	6	5	6	3,830	10,0	6	1,908	100,0
new.GL_50_6.2	6	5	6	0,944	90,0	6	0,939	100,0
new.GL_50_6.3	6	5	6	0,817	100,0	6	0,972	100,0
new.GL_50_6.4	6	5	6	0,406	100,0	6	0,772	100,0
new.GL_50_6.5	6	5	-	4,358	0,0	6	2,073	100,0
new.GL_50_6.6	6	5	6	0,122	100,0	6	0,119	100,0
new.GL_50_6.7	6	5	-	4,364	0,0	6	1,519	100,0
new.GL_50_6.8	6	5	6	0,101	100,0	6	0,106	100,0
new.GL_50_6.9	6	5	6	0,087	100,0	6	0,094	100,0
new.GL_50_6.10	6	5	6	0,317	100,0	6	0,322	100,0
new.GL_50_10.1	5	4	5	2,613	50,0	5	44,269	70,0
new.GL_50_10.2	5	4	5	0,423	100,0	5	3,558	100,0
new.GL_50_10.3	6	4	6	0,083	100,0	6	0,084	100,0
new.GL_50_10.4	5	4	5	0,134	100,0	5	0,145	100,0
new.GL_50_10.5	5	4	5	0,378	100,0	5	1,364	100,0
new.GL_50_10.6	5	4	5	1,880	100,0	5	5,652	100,0
new.GL_50_10.7	6	4	6	0,072	100,0	6	0,075	100,0
new.GL_50_10.8	5	4	5	0,359	100,0	5	2,013	100,0
new.GL_50_10.9	6	4	6	0,014	100,0	6	0,022	100,0
new.GL_50_10.10	6	4	6	0,219	100,0	6	0,244	100,0
Médias			GRASP	0,336	94,9	GRASP+PR	0,965	99,6

os 2,613 segundos do GRASP. Apesar disso, esse aumento de tempo foi compensado pelo aumento da quantidade média de soluções ótimas encontradas para essa instância de 50% do GRASP para 70% do GRASP+PR. Além disso, o tempo do GRASP+PR pode ser reduzido drasticamente através da reconfiguração dos parâmetros de execução. Um teste feito foi retardar a ativação do procedimento de reconexão de caminhos. Com isso, o algoritmo executa as fases de construção e busca local por um número maior de iterações do total e, em caso de insucesso, utiliza as últimas iterações com o procedimento de reconexão de caminhos. Fazendo isso, foi possível reduzir o tempo médio dessa instância para

5,894 segundos mantendo a taxa de soluções ótimas em 70% das execuções. No entanto, mantivemos na Tabela 4.6 o valor obtido com os parâmetros do experimento, pois o ajuste específico de parâmetros por instância não permitiria avaliar a qualidade dos algoritmos como um todo.

Na média, o GRASP encontrou a solução ótima das instâncias geométricas da classe C1 com $B = 155MB/s$ em 94,9% das execuções do algoritmo e a média de tempo ficou em 0,336 segundo. O GRASP+PR, por sua vez, convergiu para o ótimo 99,6% das vezes e 0,965 segundo de tempo de execução. Apesar do GRASP+PR ter encontrado a solução ótima um número maior de vezes, devido ao tempo computacional médio mais elevado para algumas instâncias, o seu tempo médio global sobre as instâncias testadas ficou acima do tempo médio do GRASP.

As Tabelas 4.7 e 4.8 trazem os resultados dos testes para as instâncias geométricas da classe C2 com $B = 622MB/s$. Podemos verificar que, para 13 dessas instâncias, o GRASP não achou o ótimo em todas as execuções, e em 3 delas o GRASP não foi capaz de achar a solução ótima nenhuma vez. Os tempos de execução médios do GRASP para essas 13 instâncias variaram entre $0,506$ e $4,792$ segundos. O GRASP+PR, por sua vez, apresentou desempenho significantemente superior em termos de qualidade da solução. O algoritmo não foi capaz de encontrar a solução ótima apenas para a instância new.GH_50_3.4. Em termos de tempo computacional, o GRASP+PR apresentou tempos de execução médios mais elevados para aquelas instâncias onde o GRASP não teve um bom desempenho em termos de qualidade. A instância new.GH_50_10.7 foi a mais custosa, em termos de tempo computacional, com uma média de $111,491$ segundos. Tempos computacionais mais elevados para o GRASP+PR indicam que o procedimento de reconexão de caminhos precisou ser executado um maior número de vezes para que a solução ótima fosse encontrada. Contudo, o melhor desempenho em termos de qualidade do resultado justifica a elevação do tempo médio de execução do algoritmo GRASP+PR em comparação com o GRASP.

Para as demais instâncias geométricas da classe C2 com $B = 622MB/s$, tanto o GRASP quanto o GRASP+PR encontraram a solução ótima 100% das vezes e obtiveram médias de tempo similares. As médias globais para as instâncias geométricas da classe C2 com $B = 622MB/s$ foram de $0,869$ segundo de tempo computacional e $89,6\%$ de soluções ótimas encontradas para o GRASP e de $4,430$ segundos de tempo computacional e $97,0\%$ de soluções ótimas encontradas para o GRASP+PR.

As Tabelas 4.9 e 4.10 trazem os resultados dos testes para as instâncias aleatórias da

Tabela 4.7: Resultados computacionais para as instâncias geométricas da classe C2 com $B = 622MB/s$.

Instância			GRASP			GRASP+PR		
Nome	Opt	k_{lb}	# Anéis	Tempo (s)	TC(%)	# Anéis	Tempo (s)	TC(%)
new.GH_25_5.1	4	3	4	0,047	100,0	4	0,044	100,0
new.GH_25_5.2	4	3	4	0,008	100,0	4	0,009	100,0
new.GH_25_5.3	4	3	4	0,122	100,0	4	0,161	100,0
new.GH_25_5.4	4	3	4	0,052	100,0	4	0,053	100,0
new.GH_25_5.5	4	3	4	0,067	100,0	4	0,073	100,0
new.GH_25_5.6	4	3	4	0,083	100,0	4	0,071	100,0
new.GH_25_5.7	4	3	4	0,506	80,0	4	0,209	100,0
new.GH_25_5.8	4	3	4	0,048	100,0	4	0,073	100,0
new.GH_25_5.9	4	3	4	0,106	100,0	4	0,117	100,0
new.GH_25_5.10	4	3	4	0,863	60,0	4	0,176	100,0
new.GH_30_8.1	4	3	4	0,066	100,0	4	0,067	100,0
new.GH_30_8.2	4	3	4	0,488	100,0	4	0,227	100,0
new.GH_30_8.3	4	3	4	0,084	100,0	4	0,075	100,0
new.GH_30_8.4	4	3	4	0,514	100,0	4	0,330	100,0
new.GH_30_8.5	4	3	4	0,097	100,0	4	0,081	100,0
new.GH_30_8.6	4	3	4	0,089	100,0	4	0,077	100,0
new.GH_30_8.7	4	3	4	0,031	100,0	4	0,036	100,0
new.GH_30_8.8	4	3	4	0,286	100,0	4	0,179	100,0
new.GH_30_8.9	4	3	4	0,028	100,0	4	0,025	100,0
new.GH_30_8.10	4	3	4	0,011	100,0	4	0,008	100,0
new.GH_50_2.1	6	5	6	0,717	100,0	6	0,635	100,0
new.GH_50_2.2	6	5	6	0,964	100,0	6	1,872	100,0
new.GH_50_2.3	6	5	6	0,837	100,0	6	1,057	100,0
new.GH_50_2.4	6	5	6	1,036	100,0	6	0,647	100,0
new.GH_50_2.5	6	5	6	0,591	100,0	6	0,789	100,0
new.GH_50_2.6	6	5	6	2,731	50,0	6	1,075	100,0
new.GH_50_2.7	6	5	6	0,427	100,0	6	0,488	100,0
new.GH_50_2.8	6	5	6	0,181	100,0	6	0,192	100,0
new.GH_50_2.9	6	5	6	0,558	100,0	6	0,433	100,0
new.GH_50_2.10	6	5	6	1,691	90,0	6	0,781	100,0
new.GH_50_3.1	5	4	5	0,797	100,0	5	2,094	100,0
new.GH_50_3.2	5	4	5	1,756	100,0	5	1,974	100,0
new.GH_50_3.3	5	4	5	0,079	100,0	5	0,081	100,0
new.GH_50_3.4	5	4	6	4,790	0,0	6	111,491	0,0
new.GH_50_3.5	5	4	5	4,420	10,0	5	3,274	100,0
new.GH_50_3.6	5	4	5	0,980	100,0	5	1,455	100,0
new.GH_50_3.7	5	4	5	0,485	100,0	5	3,641	100,0
new.GH_50_3.8	5	4	5	0,222	100,0	5	0,306	100,0
new.GH_50_3.9	5	4	5	0,144	100,0	5	0,139	100,0
new.GH_50_3.10	6	4	6	0,885	100,0	6	0,551	100,0

classe C2 com $B = 155MB/s$. Podemos ver nas tabelas que, para 26 dessas instâncias, o

Tabela 4.8: Resultados computacionais para as instâncias geométricas da classe C2 com $B = 622MB/s$ (Continuação).

Instância			GRASP			GRASP+PR		
Nome	Opt	k_{lb}	# Anéis	Tempo (s)	TC(%)	# Anéis	Tempo (s)	TC(%)
new.GH_50_8.1	6	5	6	0,656	100,0	6	0,564	100,0
new.GH_50_8.2	6	5	6	0,581	100,0	6	0,555	100,0
new.GH_50_8.3	6	5	6	0,366	100,0	6	0,411	100,0
new.GH_50_8.4	6	5	6	0,131	100,0	6	0,178	100,0
new.GH_50_8.5	6	5	6	0,463	100,0	6	1,457	100,0
new.GH_50_8.6	6	5	6	0,348	100,0	6	0,325	100,0
new.GH_50_8.7	6	5	6	2,461	90,0	6	0,937	100,0
new.GH_50_8.8	6	5	6	0,528	100,0	6	0,292	100,0
new.GH_50_8.9	6	5	6	1,496	100,0	6	0,980	100,0
new.GH_50_8.10	6	5	6	0,416	100,0	6	0,494	100,0
new.GH_50_9.1	5	4	5	0,302	100,0	5	0,273	100,0
new.GH_50_9.2	5	4	5	0,049	100,0	5	0,050	100,0
new.GH_50_9.3	5	4	5	0,331	100,0	5	0,333	100,0
new.GH_50_9.4	5	4	6	4,792	0,0	5	51,352	60,0
new.GH_50_9.5	5	4	5	0,197	100,0	5	0,242	100,0
new.GH_50_9.6	5	4	5	4,602	20,0	5	2,368	100,0
new.GH_50_9.7	5	4	5	4,645	10,0	5	1,972	100,0
new.GH_50_9.8	5	4	5	0,134	100,0	5	0,138	100,0
new.GH_50_9.9	5	4	5	0,084	100,0	5	0,078	100,0
new.GH_50_9.10	5	4	5	0,160	100,0	5	0,178	100,0
new.GH_50_10.1	5	4	5	0,222	100,0	5	0,287	100,0
new.GH_50_10.2	5	4	5	0,361	100,0	5	0,387	100,0
new.GH_50_10.3	5	4	5	0,286	100,0	5	0,294	100,0
new.GH_50_10.4	5	4	5	0,153	100,0	5	0,250	100,0
new.GH_50_10.5	5	4	5	2,563	70,0	5	17,132	100,0
new.GH_50_10.6	5	4	5	0,153	100,0	5	0,145	100,0
new.GH_50_10.7	5	4	-	4,383	0,0	5	92,014	30,0
new.GH_50_10.8	5	4	5	1,874	90,0	5	1,137	100,0
new.GH_50_10.9	5	4	5	0,044	100,0	5	0,049	100,0
new.GH_50_10.10	5	4	5	0,148	100,0	5	0,164	100,0
Médias			GRASP	0,869	89,6	GRASP+PR	4,430	97,0

GRASP não achou o ótimo em todas as execuções e em 4 delas o GRASP não foi capaz de achar a solução ótima nenhuma vez. Os tempos de execução médios do GRASP para essas 26 instâncias variou entre $0,220$ e $5,280$ segundos. Mais uma vez, o GRASP+PR apresentou desempenho significantemente superior, em termos de qualidade da solução, em relação ao GRASP. O algoritmo encontrou a solução ótima $98,7\%$ das execuções para todas as instâncias aleatórias da classe C2 com $B = 155MB/s$ num tempo médio de $3,639$. O GRASP, por sua vez, encontrou a solução ótima em $80,7\%$ das execuções num tempo médio de $1,177$ segundos.

Tabela 4.9: Resultados computacionais para as instâncias aleatórias da classe C2 com $B = 155MB/s$.

Instância			GRASP			GRASP+PR		
Nome	Opt	k_{lb}	# Anéis	Tempo (s)	TC(%)	# Anéis	Tempo (s)	TC(%)
new.RL_15_2.1	3	2	3	0,011	100,0	3	0,011	100,0
new.RL_15_2.2	4	2	4	0,005	100,0	4	0,000	100,0
new.RL_15_2.3	3	2	3	0,005	100,0	3	0,006	100,0
new.RL_15_2.4	3	2	3	0,000	100,0	3	0,000	100,0
new.RL_15_2.5	3	2	3	0,014	100,0	3	0,017	100,0
new.RL_15_2.6	3	2	3	0,077	100,0	3	0,066	100,0
new.RL_15_2.7	4	2	4	0,003	100,0	4	0,003	100,0
new.RL_15_2.8	3	2	3	0,000	100,0	3	0,002	100,0
new.RL_15_2.9	4	2	4	0,120	100,0	4	0,063	100,0
new.RL_15_2.10	3	2	3	0,005	100,0	3	0,006	100,0
new.RL_15_5.1	3	2	3	0,005	100,0	3	0,006	100,0
new.RL_15_5.2	3	2	3	0,003	100,0	3	0,002	100,0
new.RL_15_5.3	3	2	3	0,002	100,0	3	0,005	100,0
new.RL_15_5.4	3	2	3	0,005	100,0	3	0,006	100,0
new.RL_15_5.5	3	2	3	0,220	70,0	3	0,194	100,0
new.RL_15_5.6	3	2	3	0,005	100,0	3	0,005	100,0
new.RL_15_5.7	3	2	3	0,002	100,0	3	0,003	100,0
new.RL_15_5.8	3	2	3	0,003	100,0	3	0,003	100,0
new.RL_15_5.9	3	2	3	0,002	100,0	3	0,002	100,0
new.RL_15_5.10	3	2	3	0,006	100,0	3	0,005	100,0
new.RL_25_3.1	4	3	4	0,044	100,0	4	0,047	100,0
new.RL_25_3.2	4	3	4	0,034	100,0	4	0,040	100,0
new.RL_25_3.3	4	3	4	0,455	90,0	4	0,179	100,0
new.RL_25_3.4	4	3	4	0,098	100,0	4	0,072	100,0
new.RL_25_3.5	4	3	4	0,047	100,0	4	0,039	100,0
new.RL_25_3.6	4	3	4	0,227	90,0	4	0,583	100,0
new.RL_25_3.7	4	3	4	0,160	100,0	4	0,076	100,0
new.RL_25_3.8	4	3	4	0,170	100,0	4	0,136	100,0
new.RL_25_3.9	4	3	4	0,133	100,0	4	0,122	100,0
new.RL_25_3.10	4	3	4	0,017	100,0	4	0,022	100,0
new.RL_30_2.1	4	3	4	0,095	100,0	4	0,102	100,0
new.RL_30_2.2	4	3	4	0,592	80,0	4	1,416	100,0
new.RL_30_2.3	4	3	4	1,255	50,0	4	14,119	70,0
new.RL_30_2.4	4	3	4	0,478	90,0	4	1,131	100,0
new.RL_30_2.5	4	3	4	0,155	100,0	4	0,212	100,0
new.RL_30_2.6	4	3	4	0,019	100,0	4	0,019	100,0
new.RL_30_2.7	4	3	4	0,631	100,0	4	0,955	100,0
new.RL_30_2.8	4	3	4	0,469	90,0	4	0,831	100,0
new.RL_30_2.9	4	3	4	0,137	100,0	4	0,151	100,0
new.RL_30_2.10	4	3	4	0,030	100,0	4	0,033	100,0

Finalmente a Tabela 4.11 mostra os resultados dos testes para as instâncias aleatórias

Tabela 4.10: Resultados computacionais para as instâncias aleatórias da classe C2 com $B = 155MB/s$ (continuacao).

Instância			GRASP			GRASP+PR		
Nome	Opt	k_{lb}	# Anéis	Tempo (s)	TC(%)	# Anéis	Tempo (s)	TC(%)
new.RL_30_9.1	4	3	4	0,055	100,0	4	0,058	100,0
new.RL_30_9.2	4	3	4	0,103	100,0	4	0,098	100,0
new.RL_30_9.3	4	3	4	0,044	100,0	4	0,038	100,0
new.RL_30_9.4	4	3	4	0,895	80,0	4	3,425	100,0
new.RL_30_9.5	4	3	4	0,045	100,0	4	0,044	100,0
new.RL_30_9.6	4	3	4	0,466	100,0	4	0,944	100,0
new.RL_30_9.7	4	3	4	0,220	100,0	4	0,176	100,0
new.RL_30_9.8	4	3	4	0,125	100,0	4	0,247	100,0
new.RL_30_9.9	4	3	4	0,325	100,0	4	0,330	100,0
new.RL_30_9.10	4	3	4	1,244	30,0	4	7,250	80,0
new.RL_50_2.1	5	4	5	0,325	100,0	5	0,574	100,0
new.RL_50_2.2	5	4	5	0,952	100,0	5	1,436	100,0
new.RL_50_2.3	5	4	5	1,892	80,0	5	6,048	100,0
new.RL_50_2.4	5	4	5	5,027	0,0	5	7,244	100,0
new.RL_50_2.5	5	4	5	3,770	50,0	5	6,099	100,0
new.RL_50_2.6	5	4	5	1,927	90,0	5	1,764	100,0
new.RL_50_2.7	5	4	5	4,295	30,0	5	4,899	100,0
new.RL_50_2.8	5	4	5	4,917	0,0	5	32,245	90,0
new.RL_50_2.9	5	4	5	2,717	100,0	5	9,425	100,0
new.RL_50_2.10	5	4	5	5,063	10,0	5	3,258	100,0
new.RL_50_8.1	5	4	5	4,927	10,0	5	41,077	70,0
new.RL_50_8.2	5	4	5	3,753	50,0	5	6,132	100,0
new.RL_50_8.3	5	4	5	4,338	40,0	5	12,481	100,0
new.RL_50_8.4	5	4	5	4,647	30,0	5	7,505	100,0
new.RL_50_8.5	6	4	6	5,280	0,0	6	2,481	100,0
new.RL_50_8.6	5	4	5	1,158	90,0	5	2,560	100,0
new.RL_50_8.7	5	4	5	5,267	0,0	5	16,128	100,0
new.RL_50_8.8	5	4	5	3,134	70,0	5	15,108	100,0
new.RL_50_8.9	5	4	5	4,753	20,0	5	31,607	100,0
new.RL_50_8.10	5	4	5	5,011	10,0	5	13,360	100,0
Médias			GRASP	1,177	80,7	GRASP+PR	3,639	98,7

da classe C2 com $B = 622MB/s$. Para 8 dessas 20 instâncias o GRASP não achou o ótimo em todas as execuções e para 2 delas o GRASP não foi capaz de achar a solução ótima nenhuma vez. Isto corresponde a 40% e 10%, respectivamente. Os tempos de execução médios dessas 8 instâncias variaram entre $0,570$ e $1,820$ segundos para o GRASP. Mais uma vez, o GRASP+PR apresentou desempenho significantemente superior, em termos de qualidade da solução, em relação ao GRASP. O algoritmo encontrou a solução ótima $96,5\%$ das execuções para todas as instâncias aleatórias da classe C2 com $B = 622MB/s$ num tempo médio de $1,679$ segundos. O GRASP, por sua vez, encontrou a solução ótima

em $78,0\%$ das execuções num tempo médio de $0,522$ segundos.

Tabela 4.11: Resultados computacionais para as instâncias aleatórias da classe C2 com $B = 622MB/s$.

Instância			GRASP			GRASP+PR		
Nome	Opt	k_{lb}	# Anéis	Tempo (s)	TC(%)	# Anéis	Tempo (s)	TC(%)
new.RH_15_10.1	3	2	3	0,008	100,0	3	0,008	100,0
new.RH_15_10.2	3	2	3	0,003	100,0	3	0,005	100,0
new.RH_15_10.3	3	2	3	0,005	100,0	3	0,002	100,0
new.RH_15_10.4	3	2	3	0,008	100,0	3	0,003	100,0
new.RH_15_10.5	3	2	3	0,009	100,0	3	0,003	100,0
new.RH_15_10.6	3	2	3	0,003	100,0	3	0,009	100,0
new.RH_15_10.7	3	2	3	0,011	100,0	3	0,005	100,0
new.RH_15_10.8	3	2	3	0,005	100,0	3	0,012	100,0
new.RH_15_10.9	3	2	3	0,041	100,0	3	0,008	100,0
new.RH_15_10.10	3	2	3	0,003	100,0	3	0,077	100,0
new.RH_30_5.1	4	3	4	1,152	70,0	4	0,267	100,0
new.RH_30_5.2	4	3	4	0,034	100,0	4	0,030	100,0
new.RH_30_5.3	4	3	4	0,056	100,0	4	0,059	100,0
new.RH_30_5.4	4	3	4	1,092	80,0	4	0,245	100,0
new.RH_30_5.5	4	3	4	1,781	0,0	4	23,694	30,0
new.RH_30_5.6	4	3	4	0,570	90,0	4	0,225	100,0
new.RH_30_5.7	4	3	4	1,261	40,0	4	0,620	100,0
new.RH_30_5.8	4	3	4	1,152	50,0	4	1,402	100,0
new.RH_30_5.9	4	3	4	1,820	0,0	4	5,555	100,0
new.RH_30_5.10	4	3	4	1,417	30,0	4	1,350	100,0
Médias			GRASP	0,522	78,0	GRASP+PR	1,679	96,5

Durante estes experimentos, conseguimos encontrar a solução ótima de 219 das 230 instâncias da classe C2 com o GRASP e de 229 das 230 instâncias da classe C2 com o GRASP+PR. Para o GRASP, o maior tempo computacional médio foi de $5,280$ segundos e para o GRASP+PR foi de $111,491$ segundos. Pudemos observar que para as instâncias mais fáceis, onde o desempenho qualitativo do GRASP se igualou ao do GRASP+PR, os tempos computacionais foram bastante similares, ou seja, algumas vezes o GRASP vencia e outras vezes o GRASP+PR. Verificamos ainda que, nesses casos, o procedimento de reconexão de caminhos não foi ativado ou executou muito poucas vezes no algoritmo GRASP+PR e, portanto, os dois algoritmos tiveram aproximadamente o mesmo comportamento sobre essas instâncias.

Para as instâncias mais difíceis, onde o GRASP não obteve bom desempenho, ficou evidente a eficiência do procedimento de reconexão de caminhos do algoritmo GRASP+PR proposto. Excetuando a instância new.GH_50_3.4, onde não foi possível encontrar a solução ótima por nenhum dos algoritmos propostos, o GRASP+PR encontrou o ótimo

de todas as outras instâncias da classe C2 e em quase todas as execuções do algoritmo. Embora o GRASP+PR tenha obtido um melhor desempenho em termos de tempo para várias instâncias, para algumas delas, o algoritmo exigiu um tempo computacional mais elevado para encontrar a solução ótima do problema. Isso elevou o tempo computacional médio do algoritmo. Contudo, o GRASP não obteve o mesmo sucesso sobre essas mesmas instâncias. No próximo capítulo, analisaremos mais profundamente os dois algoritmos tanto em relação à qualidade dos resultados quanto em relação ao tempo de convergência. Esperamos, assim, obter conclusões definitivas sobre o desempenho global dos mesmos e responder a esse tipo de questão.

4.2.3 Resultados da Classe C3

Realizamos os mesmos experimentos, descritos nas seções 4.2.1 e 4.2.2 sobre as instâncias da classe C3. Os resultados desses experimentos, são apresentados nas Tabelas 4.12 a 4.17 em nove colunas. As primeiras três colunas trazem informações sobre a instância do problema que são: nome da instância, o número de anéis da solução viável base (veja a Seção 4.1.2) e o limite inferior k_{lb}. As três colunas seguintes trazem o número de anéis da melhor solução viável obtida pelo GRASP, o tempo computacional médio, em segundos, obtido nas dez execuções do algoritmo e o percentual de soluções viáveis encontradas nas 10 execuções do algoritmo para cada instância. As três últimas colunas trazem essas mesmas informações para o GRASP+PR. Diferentemente do número máximo de 5000 iterações para as instâncias das classes C1 e C2, fixamos um número máximo de 1000 iterações para as instâncias da classe C3 devido ao tempo computacional bem mais elevado dessas instâncias.

As Tabelas 4.12 e 4.13 trazem os resultados dos testes para as instâncias da classe C3 com 100 localidades. Podemos ver nas tabelas que, para 32 dessas 46 instâncias, o GRASP não achou soluções viáveis em todas as execuções e, em 19 das 46 instâncias testadas, o GRASP não foi capaz de achar solução viável nenhuma vez. Em contrapartida, o GRASP+PR apresentou desempenho significantemente superior, em termos de qualidade da solução, em relação ao GRASP. O algoritmo encontrou soluções viáveis para todas as 46 instâncias em 96,3% das execuções e num tempo médio de 46,63 segundos. O GRASP, por sua vez, encontrou soluções viáveis para 14 instâncias em 36,7% das execuções num tempo médio de 5,19 segundos. Além disso, para algumas instâncias onde as duas versões do algoritmo encontraram soluções viáveis, o GRASP+PR encontrou soluções melhores. Mas como, nos testes com essas instâncias, o alvo é encontrar uma solução viável qualquer não

Tabela 4.12: Resultados computacionais para as instâncias da classe C3 com 100 localidades.

Instância			GRASP			GRASP+PR		
Nome	S_{base}	k_{lb}	# Anéis	Tempo (s)	% V	# Anéis	Tempo (s)	% V
RTNR_100_01	6	5	-	7,22	0,0	6	59,46	100,0
RTNR_100_02	6	5	-	7,27	0,0	6	111,57	100,0
RTNR_100_03	6	5	6	6,39	10,0	6	31,91	100,0
RTNR_100_04	6	5	6	7,40	10,0	6	31,62	100,0
RTNR_100_05	6	5	6	7,38	10,0	6	40,93	100,0
RTNR_100_06	6	5	6	6,58	10,0	6	34,53	100,0
RTNR_100_07	6	5	-	7,12	0,0	6	43,28	100,0
RTNR_100_08	6	5	-	8,24	0,0	6	38,51	100,0
RTNR_100_09	6	5	6	7,29	10,0	6	26,31	100,0
RTNR_100_10	6	5	6	6,31	30,0	6	46,03	100,0
RTNR_100_11	4	3	4	0,12	100,0	4	0,13	100,0
RTNR_100_12	5	4	-	7,64	0,0	5	169,00	100,0
RTNR_100_13	5	4	-	7,53	0,0	5	71,53	100,0
RTNR_100_14	5	4	-	7,55	0,0	5	22,87	100,0
RTNR_100_15	5	4	-	7,55	0,0	5	63,31	100,0
RTNR_100_16	5	4	-	7,64	0,0	5	34,99	100,0
RTNR_100_17	4	3	4	0,11	100,0	4	0,12	100,0
RTNR_100_18	4	3	4	0,19	100,0	4	0,21	100,0
RTNR_100_19	4	3	4	0,19	100,0	4	0,19	100,0
RTNR_100_20	4	3	4	0,17	100,0	4	0,18	100,0
RTNR_100_21	5	4	5	0,15	100,0	5	0,15	100,0
RTNR_100_22	5	4	5	0,15	100,0	5	0,16	100,0
RTNR_100_23	6	5	6	0,32	100,0	6	0,78	100,0
RTNR_100_24	6	5	6	0,17	100,0	6	0,17	100,0

se pode afirmar que o GRASP não encontraria essas mesmas soluções ou outras melhores.

Os resultados dos testes para as instâncias de 150 localidades da classe C3 se encontram na Tabela 4.14. O GRASP se mostrou ineficiente na solução dessas instâncias. O algoritmo não encontrou solução viável para nenhuma delas. O tempo médio de execução do algoritmo ficou em 21,67 segundos. O GRASP+PR, pelo contrário, encontrou soluções viáveis para todas as 10 instâncias dessa categoria em 98% das execuções do algoritmo. O tempo médio de execução ficou abaixo de 254 segundos.

Os resultados dos testes para as instâncias de 200 localidades da classe C3 são exibidos na Tabela 4.15. O GRASP encontrou solução viável para 14 delas, o que corresponde a 45%. O tempo médio de execução do algoritmo ficou em 21,67 segundos e a taxa de soluções viáveis encontradas por execução ficou em 42,3%. O GRASP+PR, por sua vez, encontrou soluções viáveis para todas as 31 instâncias dessa categoria em 100% das execuções do algoritmo. O tempo médio de execução do GRASP+PR ficou abaixo de 256

Tabela 4.13: Resultados computacionais para as instâncias da classe C3 com 100 localidades.

Instância			GRASP			GRASP+PR		
Nome	S_{base}	k_{lb}	# Anéis	Tempo (s)	% V	# Anéis	Tempo (s)	% V
RTNR_100_25	7	5	6	0,16	100,0	4	0,16	100,0
RTNR_100_26	7	5	6	0,18	100,0	4	0,18	100,0
RTNR_100_27	4	3	-	9,00	0,0	4	190,09	60,0
RTNR_100_28	4	3	4	0,15	100,0	4	0,15	100,0
RTNR_100_29	4	3	4	3,36	80,0	4	2,66	100,0
RTNR_100_30	4	3	4	0,18	100,0	4	0,20	100,0
RTNR_100_31	4	3	4	0,16	100,0	4	0,16	100,0
RTNR_100_32	4	3	5	7,91	10,0	4	24,86	100,0
RTNR_100_33	4	3	-	7,44	0,0	4	53,27	100,0
RTNR_100_34	4	3	5	6,91	10,0	4	58,91	100,0
RTNR_100_35	5	5	-	6,91	0,0	4	19,63	100,0
RTNR_100_36	5	4	-	6,84	0,0	4	145,12	70,0
RTNR_100_37	5	4	-	10,69	0,0	4	88,63	100,0
RTNR_100_38	5	4	-	10,89	0,0	4	363,43	20,0
RTNR_100_39	5	4	5	4,13	50,0	4	21,10	100,0
RTNR_100_40	5	4	-	7,67	0,0	4	17,75	100,0
RTNR_100_41	5	4	5	8,11	10,0	4	15,16	100,0
RTNR_100_42	5	4	-	9,10	0,0	4	25,63	100,0
RTNR_100_43	5	4	-	8,31	0,0	4	111,96	100,0
RTNR_100_44	5	4	5	5,61	30,0	4	40,98	100,0
RTNR_100_45	5	4	5	6,33	20,0	4	40,74	100,0
RTNR_100_46	5	4	-	7,96	0,0	4	96,42	90,0
Médias			GRASP	5,19	36,7	GRASP+PR	46,63	96,3

Tabela 4.14: Resultados computacionais para as instâncias da classe C3 com 150 localidades.

Instância			GRASP			GRASP+PR		
Nome	S_{base}	k_{lb}	# Anéis	Tempo (s)	% V	# Anéis	Tempo (s)	% V
RTNR_150_01	6	5	-	20,12	0,0	6	169,68	100,0
RTNR_150_02	6	5	-	21,96	0,0	6	193,28	100,0
RTNR_150_03	6	5	-	22,33	0,0	6	197,03	100,0
RTNR_150_04	6	5	-	20,90	0,0	6	287,63	90,0
RTNR_150_05	6	5	-	22,30	0,0	7	223,50	100,0
RTNR_150_06	7	6	-	21,40	0,0	7	345,36	90,0
RTNR_150_07	7	6	-	19,64	0,0	7	154,96	100,0
RTNR_150_08	7	6	-	19,31	0,0	7	178,56	100,0
RTNR_150_09	7	6	-	21,66	0,0	7	155,75	100,0
RTNR_150_10	7	6	-	27,09	0,0	7	253,18	100,0
Médias			GRASP	21,67	0,0	GRASP+PR	215,89	98,0

segundos.

Para 3 instâncias de 200 localidades o GRASP obteve melhores soluções em termos

de qualidade. Contudo, como o alvo é encontrar uma solução viável qualquer, não se pode afirmar que o GRASP+PR não poderia obter essas mesmas soluções ou outras melhores.

Tabela 4.15: Resultados computacionais para as instâncias da classe C3 com 200 localidades.

Instância			GRASP			GRASP+PR		
Nome	S_{base}	k_{lb}	# Anéis	Tempo (s)	% V	# Anéis	Tempo (s)	% V
RTNR_200_01	6	5	-	65,46	0,0	6	470,09	100,0
RTNR_200_02	7	6	-	53,30	0,0	7	384,07	100,0
RTNR_200_03	7	6	-	53,02	0,0	7	483,42	100,0
RTNR_200_04	7	6	-	63,31	0,0	7	246,42	100,0
RTNR_200_05	6	5	-	56,55	0,0	6	336,89	100,0
RTNR_200_06	5	4	-	58,31	0,0	5	1251,00	100,0
RTNR_200_07	7	6	-	55,99	0,0	7	474,81	100,0
RTNR_200_08	8	7	-	55,43	0,0	8	452,89	100,0
RTNR_200_09	8	7	-	53,47	0,0	7	523,35	100,0
RTNR_200_10	7	6	-	53,45	0,0	6	357,41	100,0
RTNR_200_11	5	4	4	0,07	100,0	4	0,05	100,0
RTNR_200_12	5	4	4	0,20	100,0	5	0,05	100,0
RTNR_200_13	6	5	5	0,24	100,0	6	0,06	100,0
RTNR_200_14	6	5	5	0,33	100,0	6	0,06	100,0
RTNR_200_15	7	5	6	0,67	100,0	6	0,07	100,0
RTNR_200_16	7	5	6	0,63	100,0	6	0,06	100,0
RTNR_200_17	5	4	5	0,58	100,0	5	0,06	100,0
RTNR_200_18	5	4	4	0,05	100,0	4	0,05	100,0
RTNR_200_19	5	4	5	0,76	100,0	5	0,07	100,0
RTNR_200_20	5	4	5	0,58	100,0	5	0,06	100,0
RTNR_200_21	5	4	5	0,59	100,0	5	0,06	100,0
RTNR_200_22	5	4	-	49,78	0,0	5	588,16	100,0
RTNR_200_23	5	4	-	47,39	0,0	5	814,02	100,0
RTNR_200_24	5	4	5	44,78	10,0	5	38,55	100,0
RTNR_200_25	5	4	5	1,34	100,0	5	0,28	100,0
RTNR_200_26	5	4	5	1,16	100,0	5	0,09	100,0
RTNR_200_27	7	6	-	63,51	0,0	7	308,22	100,0
RTNR_200_28	7	6	-	50,95	0,0	7	263,24	100,0
RTNR_200_29	7	6	-	49,03	0,0	7	528,32	100,0
RTNR_200_30	7	6	-	46,56	0,0	7	256,55	100,0
RTNR_200_31	7	6	-	46,85	0,0	7	205,61	100,0
Médias			GRASP	31,43	42,3	GRASP+PR	257,55	100,0

A Tabela 4.16 apresenta os resultados dos testes para as instâncias de 250 localidades da classe C3. O GRASP encontrou solução viável para 3 das 6 instâncias, o que corresponde a 50%. O tempo médio de execução do algoritmo ficou em 63,97 segundos e a taxa de soluções viáveis encontradas por execução ficou em 40,0%. O GRASP+PR, por sua vez, encontrou soluções viáveis para todas as instâncias dessa categoria em 100% das

execuções do algoritmo. O tempo médio de execução ficou em 98,2 segundos.

Tabela 4.16: Resultados computacionais para as instâncias da classe C3 com 250 localidades.

Instância			GRASP			GRASP+PR		
Nome	S_{base}	k_{lb}	# Anéis	Tempo (s)	% V	# Anéis	Tempo (s)	% V
RTNR_250_01	5	4	5	1,64	100,0	5	1,75	100,0
RTNR_250_02	5	4	5	2,46	100,0	4	13,72	100,0
RTNR_250_03	5	4	5	66,20	40,0	5	79,91	100,0
RTNR_250_04	6	5	-	113,91	0,0	6	188,34	100,0
RTNR_250_05	5	4	-	90,77	0,0	5	97,73	100,0
RTNR_250_06	5	4	-	108,81	0,0	5	207,76	100,0
Médias			GRASP	63,97	40,0	GRASP+PR	98,20	100,0

Finalmente, a Tabela 4.17 exibe os resultados dos testes para as instâncias de 300 localidades da classe C3. O GRASP encontrou solução viável para 2 das 6 instâncias, o que corresponde a $33,3\%$. O tempo médio de execução do algoritmo ficou em 157,63 segundos e a taxa de soluções viáveis encontradas por execução ficou em 3,3%. O GRASP+PR, por sua vez, encontrou soluções viáveis para todas as instâncias dessa categoria em $73,3\%$ das execuções do algoritmo. O tempo médio de execução ficou em 2392,28 segundos.

Tabela 4.17: Resultados computacionais para as instâncias da classe C3 com 300 localidades.

Instância			GRASP			GRASP+PR		
Nome	S_{base}	k_{lb}	# Anéis	Tempo (s)	% V	# Anéis	Tempo (s)	% V
RTNR_300_01	5	4	-	166,87	0,0	5	4269,20	40,0
RTNR_300_02	5	4	-	168,63	0,0	5	4959,38	20,0
RTNR_300_03	6	5	6	143,65	10,0	6	1728,96	100,0
RTNR_300_04	6	5	6	144,59	10,0	6	2381,30	80,0
RTNR_300_05	7	5	-	160,76	0,0	7	658,27	100,0
RTNR_300_06	7	5	-	161,24	0,0	7	356,56	100,0
Médias			GRASP	157,63	3,3	GRASP+PR	2392,28	73,3

Capítulo 5

Análise dos Resultados

No capítulo anterior mostramos os resultados computacionais para cada instância das classes C1, C2 e C3 individualmente para cada algoritmo. Para cada grupo de instâncias, comentamos sucintamente os resultados. Neste capítulo avaliaremos o desempenho dos algoritmos propostos mais profundamente. Levaremos em consideração tanto a qualidade dos resultados quanto o tempo de convergência para a solução alvo.

Na Seção 5.1 avaliaremos os algoritmos quanto à qualidade média dos resultados obtidos e na Seção 5.2 faremos a análise probabilística desses resultados. Adicionalmente, na Seção 5.1.1, compararemos os resultados obtidos para as instâncias C1 e C2 com aqueles encontrados na literatura.

5.1 Avaliação dos Resultados Quanto à Qualidade Média das Soluções

Para avaliar o desempenho global dos algoritmos propostos, em relação à qualidade média das soluções apresentadas, compilamos novas tabelas, obtidas dos resultados das seções 4.2.1 a 4.2.3. Estas tabelas apresentam as médias dos resultados por grupos de instâncias. Nas Tabelas 5.1 e 5.2, mostramos os resultados obtidos para as classes C1 e C2, respectivamente, para cada algoritmo. Cada tabela lista os grupos de instâncias (tipo e número de instâncias viáveis), o percentual de instâncias onde a solução ótima foi encontrado pelo menos uma vez, a taxa de convergência ao alvo por execução e o tempo computacional médio (em segundos) para encontrar uma solução correspondente ao alvo (solução ótima) ou até completar um máximo de 5000 iterações. A Tabela 5.3 mostra os resultados para as instâncias da classe C3 geradas pelo algoritmo RTNR. Esta tabela lista os grupos de instâncias (tamanho e número de instâncias viáveis), o percentual de instâncias onde uma solução viável foi encontrada pelo menos uma vez, a taxa de convergência ao alvo (solução

viável) por execução e o tempo computacional médio correspondente a encontrar o alvo ou executar um máximo de 1000 iterações.

Tabela 5.1: Resultados médios para as instâncias da classe C1.

Instância		GRASP			GRASP+PR		
Tipo	# IV	Opt(%)	TC(%)	Tempo Médio (s)	Opt(%)	TC(%)	Tempo Médio (s)
GS	23	100,0	100,0	0,003	100,0	100,0	0,003
GL	27	100,0	100,0	0,005	100,0	100,0	0,005
RS	31	100,0	100,0	0,020	100,0	100,0	0,021
RL	37	97,3	93,0	0,411	100,0	98,6	2,130
Total	118	99,2	97,8	0,136	100,0	99,6	0,675

Podemos verificar na Tabela 5.1, que o GRASP+PR chegou à solução ótima de todas as instâncias da classe C1 enquanto o GRASP obteve as soluções ótimas de 117 das 118 (99,2%) instâncias dessa classe. Além disso, o procedimento de reconexão de caminhos possibilitou ao algoritmo GRASP+PR encontrar a solução ótima em 99,6% das 1180 execuções contra 97,8% do GRASP. O tempo computacional médio do GRASP+PR foi aproximadamente 5 vezes maior do que o tempo computacional médio do GRASP puro. Porém, vale ressaltar que, nesse caso, o tempo elevado de uma única instância prejudicou a média de tempo do GRASP+PR. Sem essa instância o tempo médio do GRASP+PR seria 2,6 vezes menor do que o tempo médio do GRASP. De qualquer forma, o ganho na qualidade média dos resultados do GRASP+PR justifica um aumento no tempo médio do algoritmo.

Tabela 5.2: Resultados computacionais para as instâncias da classe C2.

Instância		GRASP			GRASP+PR		
Tipo	# IV	Opt(%)	TC(%)	Tempo Médio (s)	Opt(%)	TC(%)	Tempo Médio (s)
GL	70	97,1	94,9	0,336	100,0	99,6	0,965
GH	70	95,7	89,6	0,869	98,6	97,0	4,430
RL	70	94,3	80,7	1,177	100,0	98,7	3,639
RH	20	90,0	78,0	0,522	100,0	96,5	1,679
Total	230	95,2	87,5	0,770	99,6	98,3	2,895

A Tabela 5.2 ilustra os efeitos da reconexão de caminhos sobre as instâncias da classe C2. Podemos perceber que, em termos de qualidade dos resultados, os algoritmos GRASP+PR obteve um melhor desempenho para a classe C2 pois encontrou a solução ótima de 99,6% das instâncias enquanto que o GRASP conseguiu o ótimo de apenas 95,2% delas. Além disso, o GRASP+PR apresentou taxa de convergência de 98,3% contra 87,5% do GRASP. Os percentuais exibidos na Tabela 5.2 indicam também que apenas para uma instância do tipo GH o GRASP+PR não encontrou a solução ótima enquanto

que o GRASP falhou em encontrar as soluções ótimas de 2 instâncias do tipo GL, 3 do tipo GH, 4 do tipo RL e 2 do tipo RH somando um total de 11 instâncias.

A Tabela 5.3, por sua vez, ilustra os efeitos da reconexão de caminhos sobre as instâncias da classe C3 em sete colunas. As duas primeiras colunas listam o tamanho das instâncias em número de localidades e número de instâncias viáveis por tamanho. As próximas três colunas exibem o número percentual de instâncias, por tipo, cujas soluções viáveis foram encontradas ao menos uma vez, a taxa de convergência à solução alvo (solução viável) por execução e o tempo computacional médio para o GRASP. As colunas seguintes mostram essas mesmas informações para o GRASP+PR.

Como vimos anteriormente, os resultados dos algoritmos propostos mostraram uma ligeira vantagem do algoritmo GRASP+PR sobre o algoritmo GRASP para as instâncias da classe C1 e C2. No entanto, para as instâncias da classe C3, podemos verificar uma superioridade muito mais expressiva da versão GRASP+PR. Os melhores resultados obtidos pelo GRASP foram sobre as instâncias de 100 localidades com $58,7\%$ de soluções viáveis encontradas, mas com taxa de convergência de apenas 36,7%. Em contrapartida, o algoritmo GRASP+PR encontrou 100% de soluções viáveis para essas mesmas instâncias com 93,6% de convergência. Vale ressaltar também que para as instâncias de 150 localidades o GRASP não foi capaz de gerar soluções viáveis em nenhuma das execuções. No total, o procedimento GRASP encontrou soluções viáveis para $46,5\%$ das instâncias com taxa de convergência de 32,9%. O GRASP+PR, por sua vez, encontrou soluções viáveis para 100% das 99 instâncias da classe C3 com taxa de convergência de 95,2%.

Tabela 5.3: Resultados computacionais para as instâncias da classe C3.

Instância		GRASP			GRASP+PR		
Tamanho	# IV	SV(%)	TC(%)	Tempo Médio (s)	SV(%)	TC(%)	Tempo Médio (s)
100	46	58,7	36,7	5,19	100,0	93,6	46,63
150	10	0,0	0,0	21,67	100,0	98,0	215,89
200	31	45,2	42,3	31,43	100,0	100,0	257,55
250	6	50,0	40,0	63,97	100,0	100,0	98,20
300	6	33,3	3,3	157,63	100,0	73,3	2392,28
Total	99	46,5	32,9	27,87	100,0	95,2	155,73

O tempo computacional médio do GRASP+PR foi aproximadamente $5,6$ vezes maior do que o tempo computacional médio do GRASP. Isso evidencia que, para essas instâncias, o procedimento de reconexão de caminhos foi executado mais vezes para que a solução alvo fosse encontrada. De fato, esses resultados mostram que as instâncias da classe C3 são consideravelmente mais difíceis do que as instâncias das classes C1 e C2 não só pelo tempo mais elevado, mas também pelo desempenho comparativo dos algoritmos em

relação às instâncias das classes C1 e C2.

A Tabela 5.4 lista as instâncias das classes C1 e C2 para as quais o GRASP não encontrou solução ótima. Além do nome da instância e do valor da solução ótima, a tabela mostra os resultados encontrados, bem como, o *gap* de otimalidade dado por $(100 \times [z_{GRASP} - z_{exato}]/z_{GRASP})$. Devemos notar que, exceto para as 8 instâncias para as quais o GRASP não encontrou solução viável, o *gap* é pequeno, ou seja, apesar do GRASP ter encontrando soluções com um *gap* entre $14,28\%$ e $16,67\%$ do valor da solução ótima, estes *gap*s correspondem a apenas um anel de diferença da solução ótima.

Tabela 5.4: Resultados computacionais para as instâncias onde o GRASP não encontrou a solução ótima.

Instância	z_{exato}	GRASP	$gap(\%)$
RL_50_3	4	–	–
new.GL_50_6.5	6	7	14,28
new.GL_50_6.7	6	7	14,28
new.GH_50_3.4	5	6	16,67
new.GH_50_9.4	5	6	16,67
new.GH_50_10.7	5	–	–
new.RL_50_2.4	5	–	–
new.RL_50_2.8	5	–	–
new.RL_50_8.5	6	–	–
new.RL_50_8.7	5	–	–
new.RH_30_5.5	4	–	–
new.RH_30_5.9	4	–	–

Para uma instância da categoria C2, o GRASP+PR não encontrou a solução ótima. A Tabela 5.5 apresenta esta instância. A heurística obteve uma solução com *gap* de $16,67\%$ da solução ótima. Este *gap* corresponde a apenas um anel de distância do número de anéis da solução ótima.

Tabela 5.5: Resultados computationais para a instância onde o GRASP+PR não encontrou a solução ótima.

Instância	z_{exato}	GRASP+PR	$gap(\%)$
new.GH_50_3.4	5	6	16,67

Como dissemos anteriormente, não conhecemos as soluções ótimas das 99 instâncias da classe C3 sendo a única informação disponível os limites inferior, dado por k_{lb}, e superior, dado pelo número de anéis da melhor solução viável conhecida. Por isso, não podemos analisar o desempenho dos algoritmos em relação ao critério da otimalidade. Contudo, os testes mostraram que o procedimento de reconexão de caminhos melhorou significativamente os resultados para essas instâncias mais difíceis.

Vale ressaltar que, para nenhuma das 447 instâncias testadas, o GRASP obteve melhor desempenho que o GRASP+PR em termos de qualidade média das soluções obtidas. Em contrapartida, para 125 dessas instâncias, ou seja 27, 96%, o GRASP+PR obteve melhores resultados. Dessas 125 instâncias, onde o GRASP+PR obteve melhores resultados, apenas 3 pertencem à classe C1, 52 estão na classe C2 e 70 na classe C3. Percentualmente, isso equivale a 2, 5%, 22, 6% e 70, 7%, respectivamente, em relação ao total de instâncias de cada classe. Esses percentuais estão relacionados diretamente ao grau de dificuldade das instâncias testadas e inversamente ao desempenho dos algoritmos sobre essas instâncias. Em outras palavras, o desempenho do algoritmo GRASP puro cai mais rapidamente à medida que a dificuldade das instâncias aumenta. Esse comportamento é ilustrado pela Figura 5.1. De fato, podemos observar que, para as instâncias mais fáceis (classe C1) os algoritmos obtiveram melhor desempenho e a vantagem do GRASP+PR sobre o GRASP foi menor. Para as instâncias de maior dificuldade (classe C3) os algoritmos obtiveram um desempenho global menor mas a vantagem do GRASP+PR sobre o GRASP foi mais acentuada pois este último sofreu uma queda consideravelmente maior de desempenho em relação às classes mais fáceis enquanto o GRASP+PR sofreu queda de desempenho abaixo de 5% em relação à classe C1. Para as instâncias da classe C2, de dificuldade moderada, os algoritmos obtiveram desempenhos globais e comparativos intermediários entre aqueles obtidos para as classes C1 e C3.

O gráfico da Figura 5.2 ilustra o desempenho dos dois algoritmos em relação ao percentual de instâncias para as quais os mesmos atingiram o alvo em, pelo menos, uma das 10 execuções. Percebemos que, à medida que as instâncias se tornam mais difíceis, o GRASP sofre uma queda de desempenho em relação a esse critério. Por outro lado, o GRASP+PR se mantém próximo aos 100% para as três classes de instâncias. O algoritmo atinge o alvo para 100% das instâncias da classe C1, em seguida, sofre uma pequena redução para 99,6% das instâncias da classe C2 e, em seguida, retorna aos 100% para a classe C3.

5.1.1 Comparação com Outras Heurísticas da Literatura

Uma vez analisado o comportamento dos algoritmos propostos e evidenciada a superioridade do GRASP+PR sobre o GRASP para as instâncias testadas, é desejável comparar os resultados obtidos pelo GRASP+PR com aqueles existentes na literatura. Os resultados dessa comparação, são mostrados nas Tabelas 5.6, 5.7 e 5.8 a seguir.

A Tabela 5.6 compara os resultados do GRASP+PR em termos de tempo com os re-

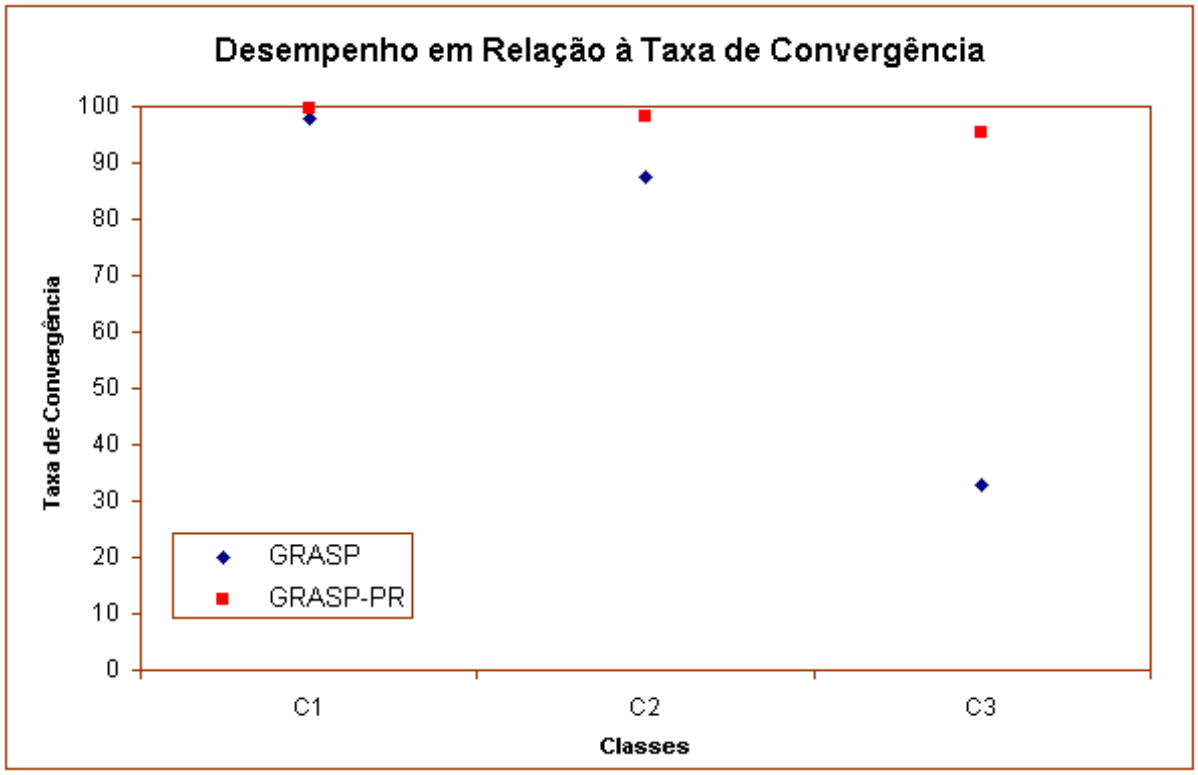

Figura 5.1: Desempenho em relação à taxa de convergêcia.

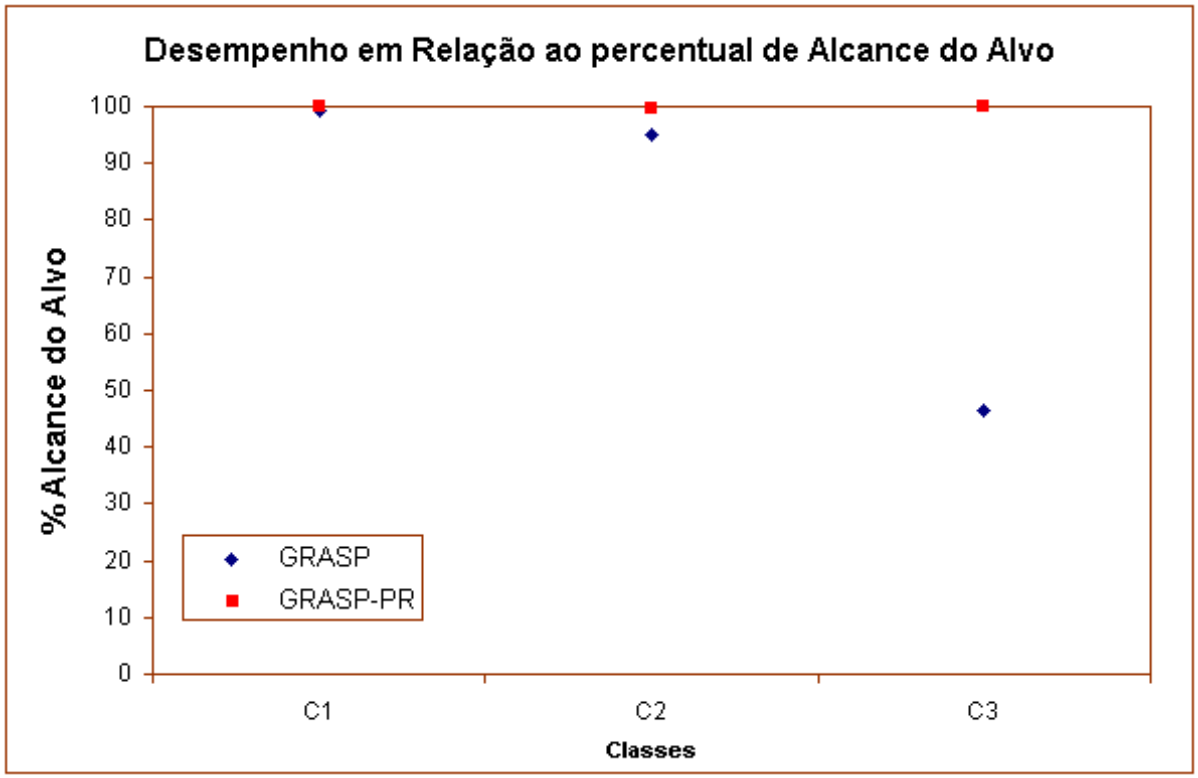

Figura 5.2: Desempenho em relação ao percentual de alcance do alvo.

sultados obtidos pelas heurísticas *edge-based*, *cut-based* e *node-based* propostas em [3]. As primeiras duas colunas mostram a classe de instâncias e o número de instâncias viáveis, as duas colunas seguintes apresentam o número percentual de soluções ótimas encontradas e o tempo computacional médio do GRASP+PR e, as colunas seguintes, exibem essas mesmas informações para as heurísticas *edge-based*, *cut-based* e *node-based*. O equipamento computacional utilizado para a realização dos experimentos em [3] foi uma estação de trabalho SPARCstation 20 Modelo 151 com um processador hyperSPARC de 150 Mhz e 288 megabytes de memória RAM. As heurísticas foram implementadas em MATLAB [27]. Para efeito de comparação, foram tomados os tempos computacionais das três heurísticas para cada uma das 118 instâncias viáveis da classe C1. Em seguida, a média global de tempo foi calculada. Em termos de qualidade, foi tomada a melhor solução dentre as três heurísticas e calculado o percentual de soluções ótimas encontradas.

Tabela 5.6: Comparação dos resultados obtidos pelo GRASP+PR com as heurísticas *edge-based*, *cut-based* e *node-based*.

Instâncias		GRASP+PR		EB, CB, NB	
Classe	# SV	# SO(%)	T(s)	# SO(%)	T(s)
C1	118	100,0	0,68	91,5	9,96

Os dados mostram que, em termos de qualidade, o GRASP+PR foi muito superior às heurísticas EB, CB e NB ao encontrar 100% de soluções ótimas contra apenas 91,5% dessas últimas. Em termos de tempo, as heurísticas EB, CB e NB apresentaram um tempo médio acima de 14 vezes superior ao tempo médio do GRASP+PR. Vale ressaltar que o equipamento computacional utilizado em [3] é menos poderoso que o equipamento utilizado em nossos testes. Dessa forma, uma vantagem de tempo do GRASP+PR já era esperada apesar de não ser possível quantificá-la com precisão.

A Tabela 5.7 compara o desempenho do GRASP+PR com o desempenho do procedimento XTS proposto em [6]. A estrutura de colunas dessa tabela é a mesma da tabela 5.6.

Os resultados mostram que o GRASP+PR conseguiu resultados melhores em termos da qualidade das soluções obtidas. Para a classe C1, foram obtidas pelo GRASP+PR 100% de soluções ótimas contra as 89,8% obtidas pelo procedimento XTS. Para a classe C2, o GRASP+PR obteve o melhor desempenho encontrando 99,6% de soluções ótimas contra 98,7% obtidas pelo procedimento XTS.

Em termos de tempo computacional, podemos perceber que, mesmo utilizando um equipamento computacional menos poderoso (Pentium III/1 Ghz com 256 MB RAM com

Tabela 5.7: Comparação dos resultados obtidos pelo GRASP+PR com o procedimento XTS.

Instâncias		GRASP+PR		XTS	
Classe	# SV	# SO(%)	T(s)	# SO(%)	T(s)
C1	118	100,0	0,68	89,8	0,34
C2	230	99,6	2,89	98,7	0,40

sistema operacional Linux e linguagem de programação C ANSI), o tempo médio obtido para o procedimento XTS foi menor do que o tempo médio obtido para o GRASP+PR. No entanto, precisa ser dito que muito poucas instâncias exigiram um tempo computacional elevado para que o GRASP+PR chegasse à solução ótima. Mesmo assim, isso foi suficiente para elevar o tempo médio do algoritmo. Em contrapartida, o GRASP+PR obteve um melhor desempenho em termos de qualidade das soluções sobretudo para a classe C1. De qualquer forma, ainda pode ser possível otimizar o algoritmo GRASP+PR reduzindo o tempo médio sem comprometer a qualidade dos resultados.

A Tabela 5.8 compara os resultados do GRASP+PR com o procedimentos DMN [6]. A estrutura de colunas dessa tabela é a mesma da tabela 5.6.

Tabela 5.8: Comparação dos resultados obtidos pelo GRASP+PR com o procedimento DMN.

Instâncias		GRASP+PR		DMN	
Classe	# SV	# SO(%)	T(s)	# SO(%)	T(s)
C1	118	100,0	0,68	100,0	0,07
C2	230	99,6	2,89	98,7	0,38

Os resultados mostram que o GRASP+PR conseguiu resultados melhores em termos da qualidade das soluções obtidas para a classe C2 com 99,6% de soluções ótimas contra 98,7% obtidas pelo procedimento DMN. Para a classe C1, ambos os algoritmos obtiveram 100% de soluções ótimas.

Em termos de tempo computacional, podemos perceber que, mesmo utilizando um equipamento computacional menos poderoso (Pentium III/1 Ghz com 256 MB RAM com sistema operacional Linux e linguagem de programação C ANSI), o tempo computacional médio requerido pelo procedimento DMN foi menor do que o tempo médio do GRASP+PR. Sobretudo, no que se refere às instâncias da classe C1, o tempo do procedimento DMN foi aproximadamente 10 vezes menor do que o tempo gasto pelo GRASP+PR. Como os dois algoritmos obtiveram o mesmo desempenho em termos de qualidade da solução, precisamos admitir que para essa classe de instâncias o procedimento DMN foi mais eficiente com um tempo bem menor. No caso da classe C2, o tempo do procedimento DMN

foi cerca de 7 vezes menor do que o tempo gasto pelo GRASP+PR. Contudo, nesse caso, o GRASP+PR obteve melhor desempenho em termos de qualidade. Além disso, como dito anteriormente, algumas instâncias como a instância new.GH_50_10.7 por exemplo, exigiram um tempo médio muito elevado para atingir a solução ótima prejudicando a média do algoritmo. Contudo, podemos perceber que ainda há margem para otimizar o algoritmo GRASP+PR e reduzir seu tempo médio sem prejudicar o desempenho do mesmo.

Finalmente, a Tabela 5.9 compara os resultados do GRASP+PR com os resultados obtidos pelo algoritmo exato *Branch & Price* proposto em [10]. A estrutura de colunas dessa tabela é semelhante à da tabela 5.6. A única diferença é que as colunas que traziam o número percentual de soluções ótimas encontradas foram retiradas.

Tabela 5.9: Comparação dos resultados obtidos pelo GRASP+PR com os obtidos pelo algoritmo *Branch & Price*.

Instâncias		GRASP+PR	*Branch & Price*
Classe	# SV	T(s)	T(s)
C1	111 (*)	0,02	12,01
C2	230	2,89	26,70

Como o *Branch & Price* é um algoritmo exato, este encontrou todas as soluções ótimas das instâncias testadas e, portanto, o interesse aqui está em comparar os tempos médios dos dois algoritmos. Vale ressaltar que, apesar de a classe C1 possuir 118 instâncias viáveis, Macambira forneceu os resultados de apenas 111 dessas instâncias em [10]. Por isso, o tempo médio do GRASP proposto aqui foi recalculado levando em conta apenas essas 111 instâncias. Esse fato está enfatizado na Tabela 5.9 pela presença do símbolo (*) ao lado do número de instâncias viáveis da classe C1 na segunda coluna.

Os testes em [10] foram realizados num computador com processador AMD K6 de 450 MHz e com 256 MB de memória RAM. Em termos de tempo computacional, a média do algoritmo *Branch & Price* foi aproximadamente 632 vezes maior do que a média do GRASP+PR para a classe C1 e cerca de 9 vezes maior do que a do GRASP+PR para a classe C2. Deve-se levar em conta que, se os experimentos fossem realizados no mesmo equipamento essas diferenças de tempo seriam menores pois o equipamento utilizado por Macambira é mais lento. Contudo, principalmente para as instâncias da classe C1, o tempo médio do GRASP+PR ainda seria menor. Além disso, as instâncias das classes C1 e C2 são relativamente pequenas. Ao executar os dois algoritmos sobre instâncias maiores como as da classe C3, por exemplo, os tempos do algoritmo *Branch & Price* devem sofrer um maior impacto do que o GRASP+PR.

5.2 Avaliação Probabilística Empírica dos Algoritmos Propostos

Na seção anterior, avaliamos o desempenho dos algoritmos considerando a qualidade média dos resultados obtidos, num limite de iterações previamente estabelecido. Uma outra metodologia, introduzida em [28], consiste am analisar empiricamente os resultados individuais de cada algoritmo sobre determinadas instâncias. O método consiste em estabelecer um valor alvo para a solução do problema e coletar o tempo gasto pelo algoritmo para atingir aquele alvo. Adicionalmente, podem ser escolhidos alvos mais fáceis e alvos mais difíceis para analisar como cada algoritmo se comporta frente a esses alvos.

Para estudar os efeitos da reconexão de caminhos sobre o GRASP, comparamos seus resultados com os resultados do GRASP+PR para 5 instâncias. Escolhemos a instância RL_25_5 da classe C1, a instância new.GH_50_9.7 da classe C2 e as instâncias RTNR_100_09, RTNR_200_24 e RTNR_250_03 da classe C3. O motivo pelo qual nos restrigiremos apenas a essas instâncias consiste em praticidade, uma vez que, nesta avaliação cada algoritmo foi executado 100 vezes para cada instância considerada. No entanto, acreditamos que a análise do desempenho dos algoritmos para as outras instâncias não fica prejudicada, já que distorções no padrão de comportamento dos algoritmos não foram observadas nos testes anteriores.

Para as instâncias das classes C1 e C2 estabelecemos como alvo a solução ótima do problema e, para as instâncias da classe C3, estabelecemos como alvo uma solução viável qualquer. Consideramos os dois alvos difíceis pois, mesmo com um alvo mais flexível, as instâncias da classe C3 são mais difíceis. Não estabelecemos alvos mais fáceis para as classes C1 e C2 pois estes não permitiriam evidenciar os efeitos da reconexão de caminhos devido à relativa facilidade que ambos possuem em solucionar essas instâncias. Para a classe C3 não estabelecemos alvos mais fáceis pois seríamos obrigados a utilizar soluções inviáveis como alvo. Além do alvo, para cada instância, estabelecemos um tempo limite de execução para evitar perdas de tempo excessivas com execuções sem sucesso.

Cada algoritmo foi executado 100 vezes para cada instância usando os mesmos parâmetros. Os tempos foram tomados e dispostos em ordem crescente numa lista L ordenada. A cada tempo de CPU t_i obtido, foi associada a probabilidade $p_i = (i - 1/2)/100$, onde i é a ordem que t_i aparece na lista ordenada L e corresponde ao número de vezes que um tempo menor ou igual a t_i aparece na lista. A plotagem foi feita então tomando cada ponto (t_i, p_i).

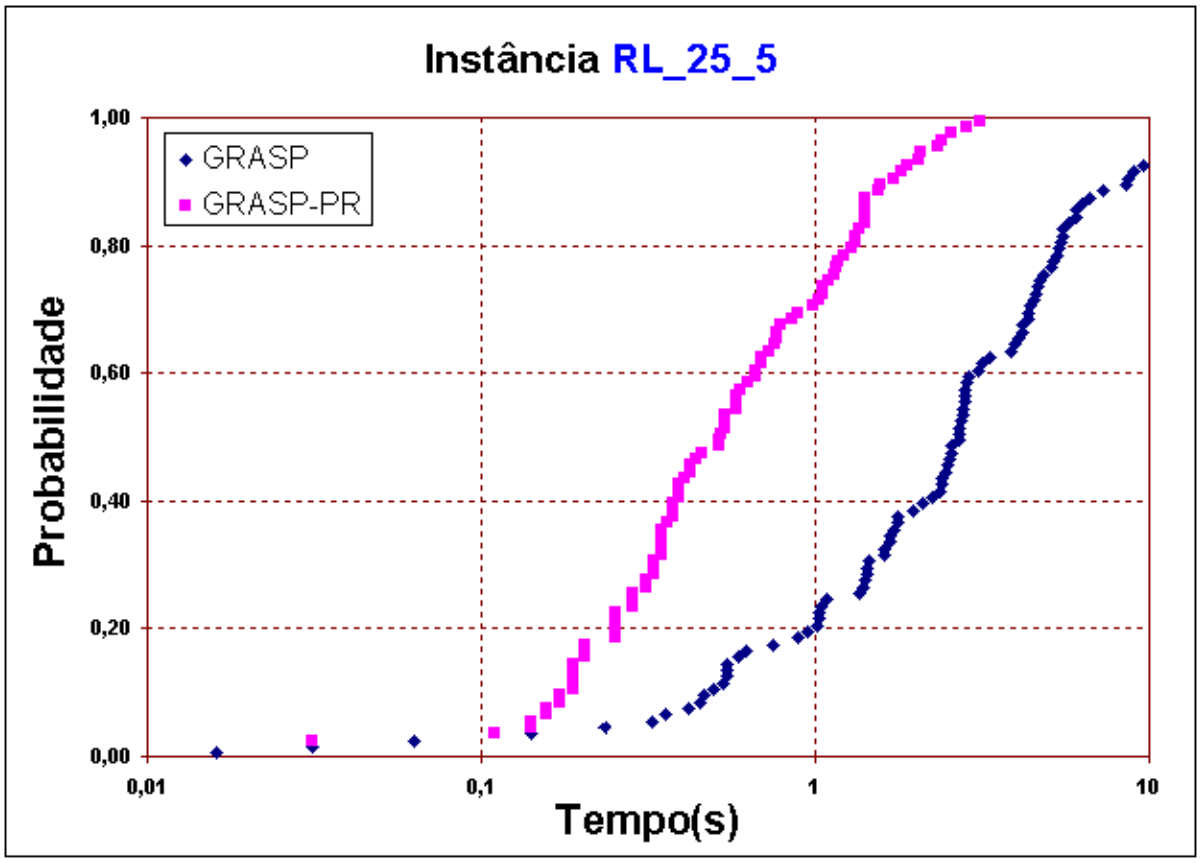

Figura 5.3: Resultados da instância RL_25_5.

Para a instância RL_25_5, o tempo máximo de execução foi fixado em 10 segundos. Os tempos de execução dos dois algoritmos e a probabilidade de cada um atingir o alvo são apresentados na Figura 5.3. O gráfico mostra a eficiência do GRASP+PR em encontrar a solução ótima do problema. Num tempo de aproximadamente 3,2 segundos o algoritmo alcança probabilidade de 100% em atingir o alvo enquanto que o GRASP apresenta probabilidade de 62% nesse mesmo tempo. Após 10 segundos, o GRASP atinge uma probabilidade de 93% de encontrar a solução ótima. Esses resultados mostram que, mesmo sendo um procedimento mais custoso em termos de tempo computacional exigido por iteração, o GRASP+PR converge para a solução ótima muito mais rápido que o GRASP.

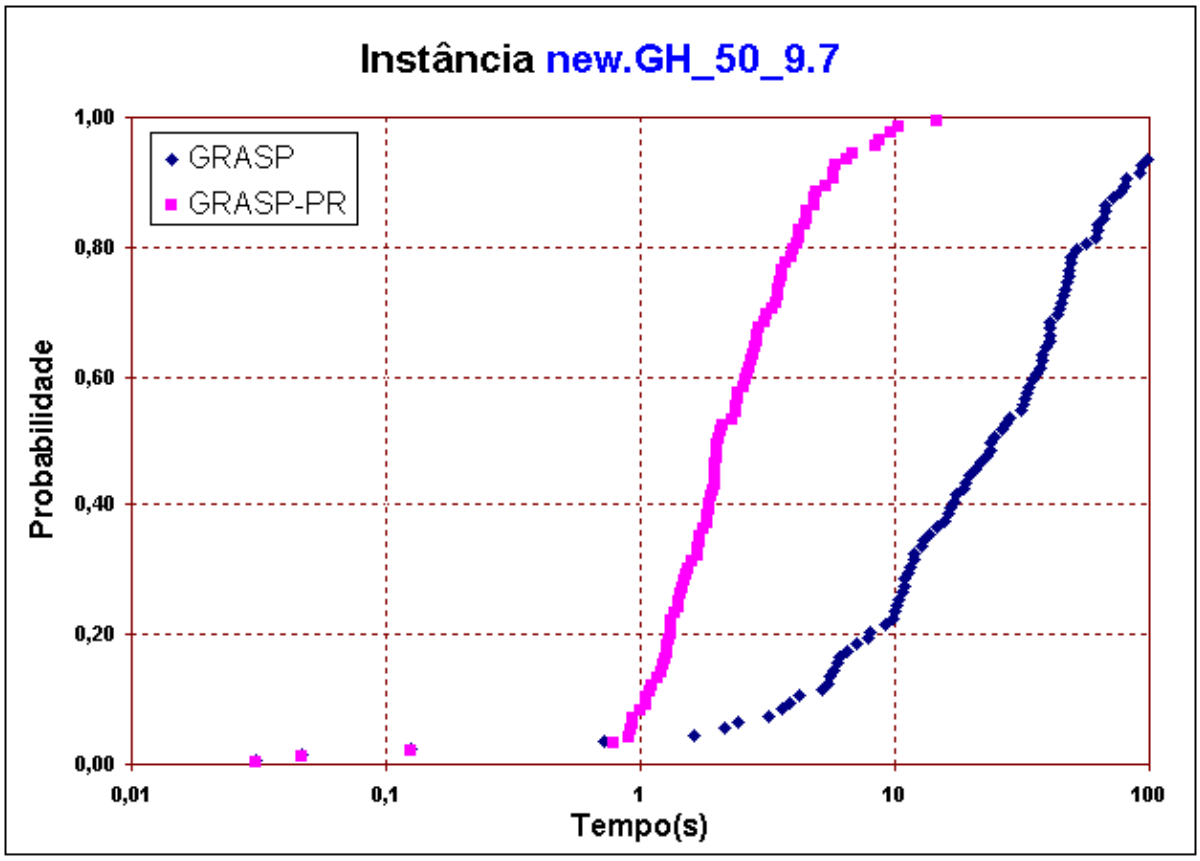

Figura 5.4: Resultados da instância new.GH_50_9.7.

Para a instância new.GH_50_9.7, o tempo máximo de execução foi fixado em 100 segundos. Os resultados das 100 execuções dos algoritmos são apresentados no gráfico da Figura 5.4. Podemos observar que até o tempo de aproximadamente 0,8 segundo os dois algoritmos apresentam o mesmo desempenho. A partir desse momento, o procedimento de reconexão de caminhos começa a atingir o alvo com maior probabilidade. Para esta instância, aproximadamente 15 segundos foram suficientes para que o GRASP+PR atin-

gisse 100% de convergência contra 36% do GRASP. Após 100 segundos, o GRASP atinge uma probabilidade de 94% de encontrar a solução ótima.

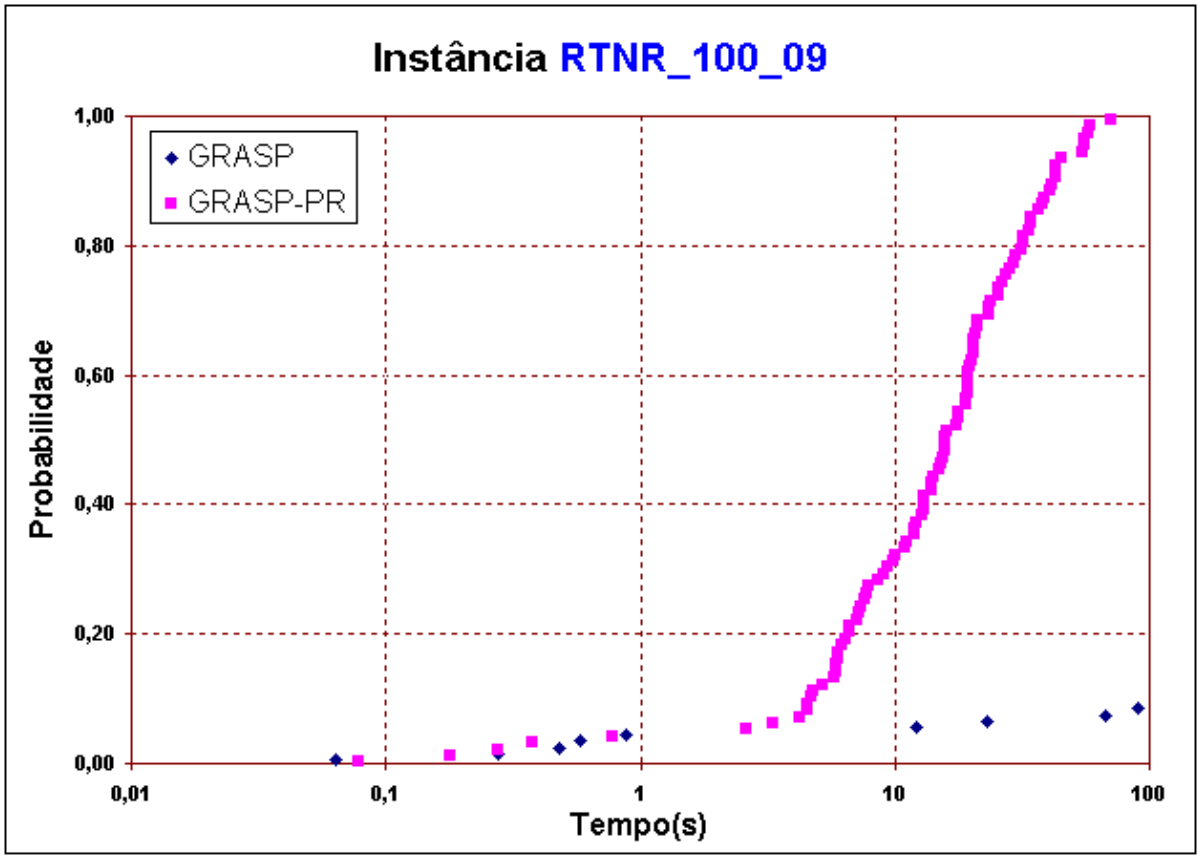

Figura 5.5: Resultados da instância RTNR_100.

No caso da instância RTNR_100, o tempo máximo de execução foi fixado em 100 segundos. Os resultados das 100 execuções dos algoritmos são apresentados no gráfico da Figura 5.5. Para essa instância o desempenho do algoritmo GRASP foi bem inferior ao do GRASP+PR. Até o tempo de aproximadamente 2 segundos os dois algoritmos apresentam o mesmo desempenho com 5% de convergência para o alvo. Após cerca de 10 segundos o GRASP continua com 5% de convergência enquanto que o GRASP+PR já obtém a marca de 37% de convergência para o alvo. Em torno de 71 segundos o GRASP+PR atinge 100% de convergência contra apenas 8% de convergência do GRASP. No tempo limite de 100 segundos a probabilidade do GRASP em achar o alvo é de 9%. Admitindo, de forma bastante otimista, uma curva de convergência aproximadamente linear, a convergência do GRASP a 100% só se dará acima de 1100 segundos o que equivale a mais de 15 vezes o tempo de convergência do GRASP+PR.

A Figura 5.6 ilustra os resultados para a instância RTNR_200. Para esta instância, utilizamos o mesmo tempo máximo de execução da instância RTNR_100, fixado em 100

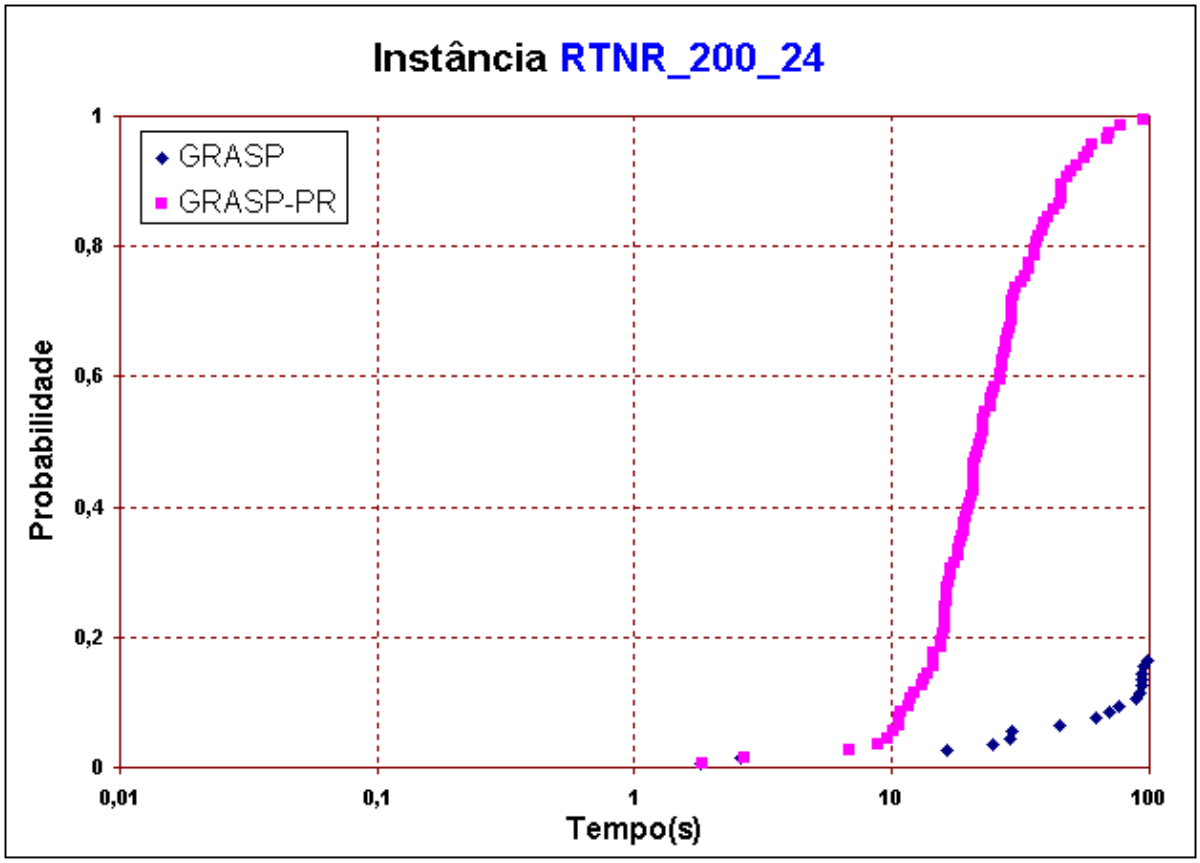

Figura 5.6: Resultados da instância RTNR_200.

segundos. Os resultados das 100 execuções dos algoritmos mostram, mais uma vez, um desempenho inferior do algoritmo GRASP em relação ao GRASP+PR. Até o tempo de aproximadamente 6 segundos os dois algoritmos apresentam o mesmo desempenho com 2% de convergência para o alvo. Após 30 segundos o GRASP atinge apenas 6% de convergência enquanto o GRASP+PR já obtém a marca de 74% de convergência para o alvo. Em cerca de 94 segundos o GRASP+PR atinge 100% de convergência contra apenas 15% de convergência do GRASP. No tempo limite de 100 segundos a probabilidade do GRASP em achar o alvo é de 17%. Se tomarmos o tempo que cada algoritmo atinge essa taxa de convergência teremos 15 segundos para o GRASP+PR e 99 segundos para o GRASP o que equivale a 6,6 vezes mais tempo.

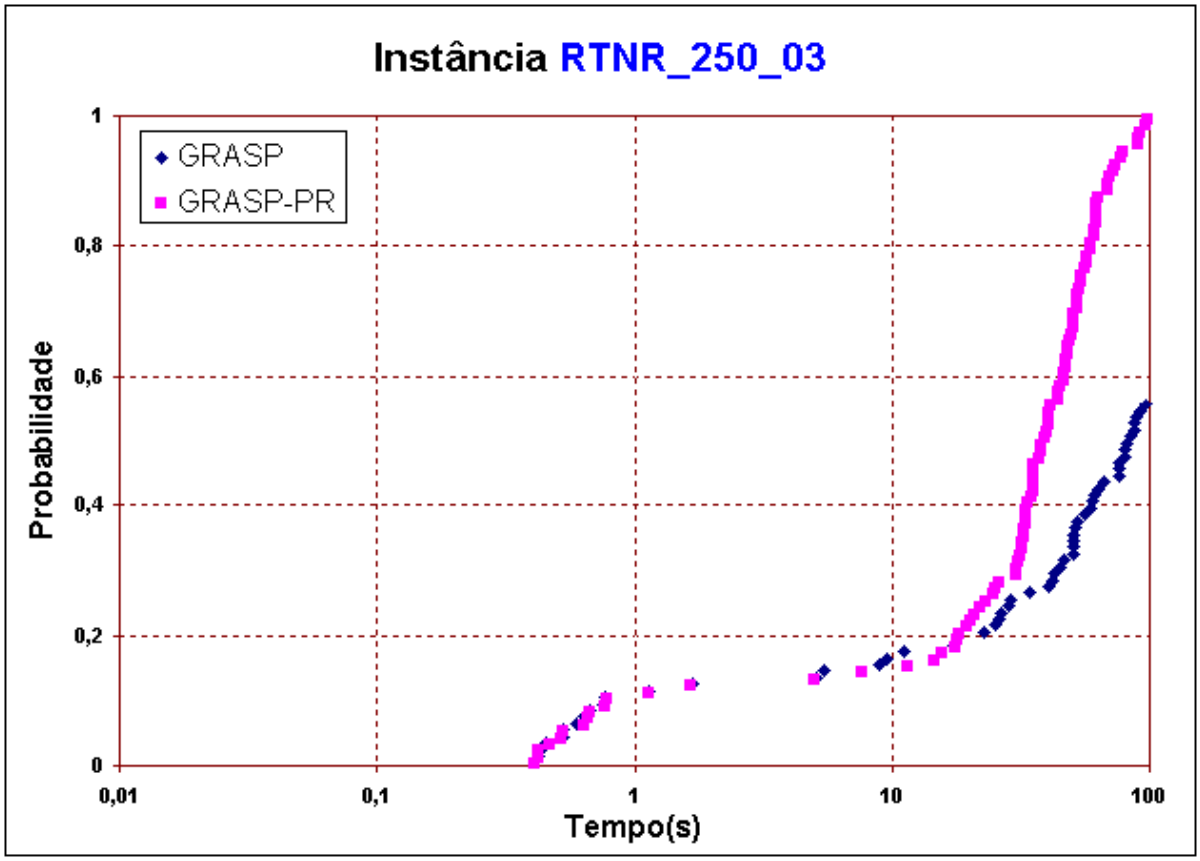

Figura 5.7: Resultados da instância RTNR_250.

A Figura 5.7 trás os resultados para a instância RTNR_250. Para esta instância, também utilizamos, como tempo máximo de execução, 100 segundos. Os resultados das 100 execuções dos algoritmos mostram, mais uma vez, que os mesmos obtiveram um padrão de comportamento semelhante e apresentaram melhor desempenho para o GRASP+PR. No entanto, o gráfico nos mostra uma pequena vantagem do GRASP no intervalo de tempo que vai de 6 segundos a 18 segundos. Esse comportamento já era esperado, visto que no

início da utilização do procedimento de reconexão de caminhos é gasto um tempo extra para obter um número mínimo de soluções no conjunto de soluções elite. Quanto maior a instância, um tempo maior é necessário para que o conjunto elite atinja o número ideal de soluções que permita o procedimento de reconexão de caminhos obter um bom desempenho. Como as instâncias analisadas anteriormente são menores, o tempo gasto com a preparação do conjunto elite não afetou tanto assim o desempenho do GRASP+PR. Além disso, o desempenho do GRASP para essa instância se mostrou muito bom em comparação às outras instâncias analisadas.

Após cerca de 18 segundos o algoritmo GRASP+PR se torna mais eficiente do que o GRASP. Aos 50 segundos as taxas de convergência atingem 32% para o GRASP e 66% para o GRASP+PR. Em 98 segundos o GRASP+PR atinge 100% de convergência contra 56% de convergência para o GRASP. No tempo limite de 100 segundos a probabilidade do GRASP em achar o alvo é ainda de 56%. Se tomarmos o tempo que cada algoritmo atinge essa taxa de convergência teremos 42 segundos para o GRASP+PR o que equivale a 2,3 vezes menos tempo.

Capítulo 6

Conclusões e Sugestões para Trabalhos Futuros

Neste trabalho, apresentamos um GRASP para o Problema de Atribuição de Localidades a Anéis SONET. A fase construtiva do algoritmo se utilizou de quatro heurísticas gulosas randomizadas: *Random Edge-Based, Random Cut-Based, Random Node-Based* e a Heurística de Vizinhanças Relativas (HVR). Para a fase de busca local, propusemos quatro vizinhanças e seus respectivos procedimentos de busca local. Desenvolvemos, então, duas versões do algoritmo, uma sem a utilização da reconexão de caminhos, a qual denominamos simplesmente GRASP, e a outra com o uso da técnica de reconexão de caminhos, aqui denominada GRASP+PR. Dessa forma, mostramos como a técnica da reconexão de caminhos foi usada para melhorar o desempenho da fase de busca local.

Realizamos extensivos experimentos computacionais sobre as instâncias disponíveis na literatura e também sobre novas instâncias maiores e mais difíceis as quais geramos durante a realização deste trabalho. Os algoritmos foram executados 10 vezes sobre cada uma das 447 instâncias disponíveis utilizando os mesmos parâmetros. Os resultados computacionais foram coletados e avaliados sob o critério da qualidade média e comparados com aqueles disponíveis na literatura. Além disso, avaliamos empiricamente a probabilidade de convergência dos algoritmos propostos em função do tempo computacional.

O GRASP+PR se mostrou bem mais robusto melhorando o desempenho do GRASP puro tanto no aspecto de encontrar soluções melhores quanto no aspecto de apresentar um menor tempo de convergência para a solução alvo. De fato, mostramos que para nenhuma das 447 instâncias testadas, o GRASP obteve melhor desempenho do que o GRASP+PR em termos de qualidade média das soluções obtidas e, em contrapartida, para 125 dessas instâncias o GRASP+PR obteve melhores resultados. Mostramos também que, o desempenho do algoritmo GRASP puro cai mais rapidamente à medida que a dificuldade

das instâncias aumenta.

Os resultados mostraram também que, frente aos algoritmos encontrados na literatura, o GRASP+PR conseguiu resultados melhores em termos da qualidade das soluções obtidas. Para a classe C1, os resultados obtidos pelo GRASP+PR e pela heurística DMN [6] foram ambos de 100% de soluções ótimas contra 97,5% das heurísticas desenvolvidas em [3] e 89,8% obtidos pelo procedimento XTS [6]. Para a classe C2, o GRASP+PR obteve o melhor desempenho encontrando 99,6% de soluções ótimas contra os 98,3% conseguidos pelo procedimento DMN e contra os 98,7% obtidos pelo procedimento XTS.

Como propostas para trabalhos futuros podemos citar as seguintes possibilidades:

- A utilização de outras meta-heurísticas como busca tabu, VNS e algoritmos genéticos juntamente com o GRASP de forma híbrida e utilizando o procedimento de reconexão de caminhos proposto aqui;
- Desenvolver versões paralelas dessas metaheurísticas na tentativa de reduzir os tempos computacionais para instâncias ainda maiores. Além disso, com a comunicação entre processadores, o que tornará estes algoritmos cooperativos e mais adaptativos, poderemos tentar obter soluções de melhor qualidade do que as obtidas pelas suas versões seqüenciais.

Referências

[1] OMIDYAR, C. G.; ALDRIDGE, A. Introduction to SDH/SONET. *Communications Magazine*, p. 30–33, September 1993.

[2] WASEM, O. J.; WU, T. H.; CARDWELL, R. H. Survivable SONET networks - design methodologies. *IEEE Journal on Selected Areas in Communications*, v. 12, p. 205–212, 1994.

[3] GOLDSCHMIDT, O.; LAUGIER, A.; OLINICK, E. V. SONET/SDH ring assignment with capacity constraints. *Discrete Applied Mathematics*, v. 129, p. 99–128, 2003.

[4] CPLEX OPTIMIZATION INC. *CPLEX linear optimizer 4.0.8*. Incline Village, NV, USA, 1995.

[5] ARINGHIERI, R.; DELL'AMICO, M.; GRASSELLI, L. Solution of the SONET ring assignment problem with capacity constraints. Technical Report 12, DISMI - University of Modena and Reggio Emilia, 2001.

[6] ARINGHIERI, R.; DELL'AMICO, M. Comparing metaheuristic algorithms for the SONET network design problems. *Journal of Heuristics*, v. 11, p. 35–57, 2005.

[7] RESENDE, M. G. C.; RIBEIRO, C. C. GRASP with path-relinking: recent advances and applications. In: IBARAKI, K. N. T.; YAGIURA, e. M. (Ed.). *Metaheuristics: progress as real problem solvers*. : Springer, 2005. p. 29–63.

[8] DELL'AMICO, M.; TRUBIAN, M. Solution of large weighted equicut problems. *European Journal of Operational Research*, v. 106, p. 500–521, 1998.

[9] GLOVER, F. Scatter search and path relinking. In: CORNE, M. D. I. D.; GLOVER, e. F. (Ed.). *New Ideas in Optimization*. : McGraw Hill, 1999. p. 297–316.

[10] MACAMBIRA, E. M. *Modelos e Algoritmos de Programação Inteira no Projeto de Redes de Telecomunicações*. Tese (Doutorado) — COPPE, Programa de Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro, Brasil, 2003.

[11] MACAMBIRA, E. M.; MACULAN, N.; SOUZA, C. C. de. A column generation approach for SONET ring assignment. *Networks*, Novembro 2005.

[12] GOLDSCHMIDT, O. et al. The SONET edge-partition problem. *Networks*, v. 41, p. 3–23, 2003.

[13] MORLEY, G. D.; GROVER, W. D. Tabu search optimization of optical ring transport networks. In: *Proceedings IEEE Globecom 2001*. San Antonio, Texas, USA: , 2001.

[14] RESENDE, M. G. C. Combinatorial optimization in telecommunications. In: PARDALOS, P.; KOROTKICH, e. V. (Ed.). *Optimization and Industry: New Frontiers*. : Kluwer Academic Publishers, 2003. p. 59–112.

[15] BASTOS, L. de O.; OCHI, L. S.; MACAMBIRA, E. M. A relative neighborhood GRASP for the SONET ring assignment problem. In: *International Network Optimization Conference - INOC*. Lisbon, Portugal: , 2005. p. 833–838.

[16] FEO, T.; RESENDE, M. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, v. 8, p. 67–71, 1989.

[17] FEO, T. A.; RESENDE, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, v. 6, p. 109–133, 1995.

[18] MARTINS, S. et al. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, v. 17, p. 267–283, 2000.

[19] RESENDE, M.; RIBEIRO, C. Greedy randomized adaptive search procedures. *Handbook of Metaheuristics*, Kluwer Academic Publishers, p. 219–249, 2003.

[20] FESTA, P.; RESENDE, M. GRASP: An annotated bibliography. *Essays and Surveys on Metaheuristics*, Kluwer Academic Publishers, p. 325–367, 2002.

[21] GLOVER, F. Tabu search and adaptive memory programing: advances, applications and challenges. In: BARR, R. H. R.; KENNINGTON, e. J. (Ed.). *Interfaces in Computer Science and Operations Research*. : Kluwer, 1996. p. 1–75.

[22] LAGUNA, M.; MARTí, R. GRASP and path-relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, v. 11, p. 44–52, 1999.

[23] RESENDE, M. G. C.; RIBEIRO, C. C. Greedy randomized adaptive search procedures. In: GLOVER, F.; KOCHENBERGER, e. G. (Ed.). *Handbook of Metaheuristics*. : Kluwer Academic Publishers, 2003. p. 219–249.

[24] FESTA, P. et al. GRASP with path-relinking for the weighted maximum satisfiability problem. In: *IV Workshop on Efficient and Experimental Algorithms - WEA2005*. Santorini, Greece: , 2005. p. 367–379.

[25] BASTOS, L. de O.; OCHI, L. S.; MACAMBIRA, E. M. GRASP with path-relinking for the SONET ring assignment problem. In: NEDJAH, N. et al. (Ed.). *Hybrid Intelligent Systems - HIS*. Rio de Janeiro, RJ, Brazil: , 2005. p. 239–244.

[26] MACAMBIRA, E. M. et al. Algoritmos eficientes para o projeto de uma rede de telecomunicações com topologia em anel. In: IBARAKI, K. N. T.; YAGIURA, e. M. (Ed.). *V Encontro Regional de Matemática Aplicada e Computacional ERMAC*. Natal, RN, Brasil: , 2005.

[27] THE MATHWORKS INC. *MATLAB 4.2*. 3 Apple Hill Drive Natick, MA, USA, 1994.

[28] AIEX, R.; BINATO, S.; RESENDE, M. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, v. 29, p. 393–430, 2003.