Heurísticas para Resolução do Problema de Roteamento com Número Limitado de Caminhos

José Wilson Coura Pinto

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

Orientadores:
Simone de Lima Martins
Isabel Cristina Mello Rosseti

Niterói, Novembro de 2005.

Livros Grátis

http://www.livrosgratis.com.br

Milhares de livros grátis para download.

Heurísticas para Resolução do Problema de Roteamento com Número Limitado de Caminhos

José Wilson Coura Pinto

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

Aprovada por:

Profa. Simone de Lima Martins / IC-UFF (Presidente)

Prof. Celso da Cruz Carneiro Ribeiro / IC-UFF/PUC

Profa. Isabel Cristina Mello Rosseti / ONS

Prof. Luiz Satoru Ochi / IC-UFF

Prof. Mauricio Cardoso de Souza / UFMG

Niterói, Novembro de 2005.

À minha família e amigos, bens mais valiosos que tenho.

Agradecimentos

Inserir agradecimento...

Resumo da Tese apresentada à UFF como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação (M.Sc.)

Heurísticas para Resolução do Problema de Roteamento com Número Limitado de Caminhos

José Wilson Coura Pinto

Novembro/2005

Orientadora: Simone de Lima Martins e Isabel Cristina Mello Rosseti
Programa de Pós-Graduação em Computação

O problema de roteamento com número limitado de caminhos analisado neste trabalho consiste em rotear um conjunto de demandas, minimizando o congestionamento e/ou o atraso de propagação, onde cada demanda pode ser roteada por um número limitado de caminhos.

Devido à complexidade computacional apresentada por este problema, optou-se por desenvolver soluções utilizando metaheurísticas objetivando-se obter soluções ótimas ou próximas da ótima em tempo computacional viável.

Neste trabalho, são descritas heurísticas construtivas e procedimentos de busca local utilizados para implementar uma heurística GRASP para o problema proposto. Técnicas de reconexão de caminhos são empregadas com o objetivo de encontrar soluções de qualidade superior às geradas pelo GRASP. Experimentos computacionais utilizando instâncias encontradas na literatura são realizados. Apresenta-se uma comparação da qualidade das soluções obtidas entre o GRASP puro e as quatro variantes da heurística GRASP com reconexão de caminhos.

Abstract of Thesis presented to UFF as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

Heuristics for Resolution of the Resource Allocation Problem with Limited Number of Paths

José Wilson Coura Pinto

Octuber/2005

Advisors: Simone de Lima Martins e Isabel Cristina Mello Rosseti
Department: Computer Science

The Automatic Clustering Problem ...

Palavras-chave

1. Otimização Combinatória
2. Heurísticas
3. Metaheurísticas
4. Redes de Computadores
5. GRASP
6. Reconexão de Caminhos

Glossário

GRASP	:	Greedy Randomized Adaptative Search Procedures;
LRC	:	Lista Restrita de Candidatos;
RC	:	Reconexão de Caminhos;
QoS	:	Quality of Service (Qualidade de Serviço);

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Os problemas de fluxo em redes formam uma importante subclasse dos problemas de otimização e são utilizados para modelar problemas de várias áreas, como telecomunicações, logistica, transportes e etc. Os problemas de fluxo em redes com fluxo simples(Problemas de fluxo simples), onde existe apenas uma demanda na rede, consistem em encontrar o fluxo em cada arco, para enviar uma demanda de um conjunto S com uma ou mais origens, até um conjunto T com um ou mais destinos, otimizando algum critério estabelecido segundo as necessidades do problema, como por exemplo, minimizar os custos ou maximizar a quantidadade de fluxo enviada. Os problemas de fluxo em redes com mais de uma demanda (Problema de Multifluxo), que aparecem na literatura desde os anos 60 [21], são uma generalização dos problemas de fluxo simples, onde ao invés de uma única demanda, temos agora um conjunto com K demandas distintas. Dessa forma temos as K demandas compartilhando a capacidade dos arcos desta rede.

O problema de roteamento em redes, onde se deseja rotear demandas entre nós desta rede, minimizando o congestionamento e o atraso de propagação nos arcos da rede, pode ser caracterizado como uma problema de multifluxo. Cada demanda pode ser dividida em subdemandas que são roteadas utilizando um número limitado de caminhos não necessariamente disjuntos. Quando o número limite de caminhos é igual a 1 e deseja-se somente minimizar o congestionamento, este problema se reduz ao problema NP-Difícil de roteamento de multifluxo [4]. Quando o número máximo

de caminhos aumenta, se torna mais fácil a redução do custo das soluções, pois qualquer partição de demanda é permitida. Em redes de comunicação modernas, como as redes MPLS(Multi Protocol Label Switching), pode-se dividir as demandas e roteá-las por caminhos diversos(Label Switched Paths), não necessariamente disjuntos. Caso se utilize um maior número de caminhos, pode-se minimizar o congestionamento, mas à medida que este número aumenta há um deterioramento no desempenho dos protocolos. Portanto, deseja-se utilizar um número intermediário de caminhos, que implica em uma maior complexidade para resolução do problema.

Problemas classificados como NP-Difícil não podem ser tratados por algoritmos exatos, seja por indisponibilidade de tempo ou poder computacional. Nestes casos, métodos aproximados, chamados heurísticas, são freqüentemente utilizados com o objetivo de encontrar a solução ótima ou próxima da ótima para o problema. Heurísticas tentam encontrar a melhor solução possível para um problema, dentro das condições de esforço computacional oferecidos.

GRASP (*Greedy Randomized Adaptive Search Procedure*) é uma metaheurística desenvolvida por Feo e Resende [9], baseada em um algoritmo iterativo, onde cada iteração é constituída por duas fases: a primeira fase, chamada fase de construção e a segunda fase, uma fase de busca local. Várias propostas de melhoramento da metaheurística GRASP têm surgido na literatura, entre elas o GRASP reativo [26], a reconexão de caminhos [31] e filtro de soluções [24]. A Reconexão de caminhos utilizada hibridamente com o GRASP tem apresentado bons resultados em problemas com características semelhantes a este [31].

O objetivo desta dissertação é propor uma heurística GRASP com reconexão de caminhos para o problema de roteamento com número limitado de caminhos. No desenvolvimento deste procedimento, combinaram-se diversas propostas para as fases de construção e de busca local do GRASP, assim como para o procedimento de reconexão de caminhos. Avaliaram-se estas propostas através de testes realizados em instâncias utilizadas na literatura [7, 10, 30].

O restante desta dissertação está organizado nos seguintes capítulos:

- Capítulo 2 - *O Problema de Roteamento com Número Limitado de Caminhos*: Apresentam-se a descrição do problema e trabalhos relacionados.

- Capítulo 3 - *A Heurística GRASP com Reconexão de Caminhos*: Apresentam-se as especificações dos procedimentos desenvolvidos para fase de construção, fase de busca local, e a definição dos métodos de reconexão de caminhos.

- Capítulo 4 - *GRASP com Reconexão de Caminhos Proposto*: Apresentam-se o GRASP proposto por este trabalho e os procedimentos de reconexão de caminhos desenvolvidos.

- Capítulo 5 - *Resultados Computacionais*: Apresentam-se os resultados obtidos através de testes realizados, utilizando-se diferentes combinações de métodos e parâmetros para definir os procedimentos GRASP e de Reconexão de caminhos.

- Capítulo 6 - *Conclusões*: Discutem-se os resultados encontrados e sugerem-se direções para trabalhos futuros.

Capítulo 2

O Problema de Roteamento com Número Limitado de Caminhos

Neste capítulo, o problema de roteamento com número limitado de caminhos é apresentado, assim como sua descrição matemática. Na Seção 2.1, é apresentada a descrição do problema. Na Seção 2.2, são apresentados alguns trabalhos relacionados encontrados na literatura.

2.1 Descrição do Problema

Considere uma rede de computadores com todos os seus hospedeiros e enlaces, e um conjunto de demandas, onde cada demanda associada a um par de hospedeiros origem-destino deve ser roteada através desta rede, usando uma quantidade (R) limitada de caminhos. A principal característica que difere este problema de roteamento com número limitado de caminhos *(p-BNDP) p-bounded network design problem* [8] do tradicional problema de fluxo em redes é a possibilidade de dividir a carga que se deve transportar entre um par origem-destino, em até R caminhos distintos. Dessa forma, o problema consiste em encontrar caminhos possíveis para rotear simultaneamente a carga de todas as demandas, minimizando o congestionamento e o atraso de propagação.

Pode-se formular este problema da seguinte maneira. Considere G(V,A) um grafo direcionado que representa uma rede, onde V é o conjunto que contém n nós, e A um conjunto com m arestas, representando os hospedeiros e enlaces, respectivamente. Para cada arco $(i, j) \in A$, são associadas uma variável $b_{ij} \in \mathbb{R}^+$ que representa a capacidade ou banda desta aresta e uma variável $d_{ij} \in \mathbb{R}^+$ que representa o atraso relativo a esta aresta. O conjunto K contém p elementos que representam as demandas. Cada elemento $k \in K = (o_k, d_k, l_k)$, deverá ser roteado pelo grafo G, onde o_k representa o nó origem, d_k o nó destino e l_k a carga da demanda k, medida em bits/segundo.

O objetivo deste problema consiste em encontrar a melhor configuração para rotear todas as p demandas, minimizando o congestionamento e atraso de propagação em toda a rede. Cada demanda pode ser roteada através de no máximo R caminhos distintos, onde R é um parâmetro de entrada. C_k é o conjunto com $r(k) \in \{1, 2, ..., R\}$ caminhos, usados para rotear a carga l_k associada a demanda k. Cada caminho $c_k^h \in C_k$ é utilizado para rotear uma determinada quantidade de fluxo $f_k^h \in \mathbb{R}^+$ da carga associada a demanda k. A variável $x_{ij}^{kh} = 1$ se, e somente se, o caminho c_k^h utiliza a aresta (i, j) para rotear sua carga, caso contrário, $x_{ij}^{kh} = 0$. Sendo assim, define-se o fluxo total t_{ij}, que atravessa cada aresta (i, j), através da equação

$$t_{ij} = \sum_{k \in K} \sum_{h=1}^{R} x_{ij}^{kh} \; . \; f_k^h \tag{2.1}$$

Define-se o custo ϕ_{ij}, associado a cada aresta $(i, j) \in A$, através da combinação linear:

$$\phi_{ij} \; = \; (1 - \delta) \; . \; \phi_{ij}^a \; + \; \delta \; . \; \phi_{ij}^c \tag{2.2}$$

onde, ϕ_{ij}^a é a componente de atraso em função da carga que atravessa a aresta (i, j), ϕ_{ij}^c é a componente de congestionamento referente à aresta (i, j), e $\delta \in [0, 1]$ é um parâmetro de entrada, que indica qual o peso que cada uma das componentes possui. O coeficiente $(1 - \delta)$ corresponde ao peso da componente de atraso de propagação, e δ é o coeficiente correspondente a componente de congestionamento. Observa-se que, quanto mais próximo δ estiver de 1, maior será o peso da componente congestionamento, e que se $\delta = 1$, somente a componente congestionamento será

considerada. Entretanto, caso $\delta = 0$ apenas a componente de atraso de propagação será usada no calculo de ϕ_{ij}.

Atrasos em fila são causados principalmente pelo congestionamento da rede e geralmente são calculados em função do volume deste congestionamento. Entretanto, algumas redes podem apresentar grandes distâncias e cargas pequenas, onde neste caso o atraso de propagação precisa ser considerado. Para maiores informações sobre atrasos em redes [19]. A forma escolhida para se calcular o custo da componente atraso de propagação $\phi^a_{ij} \in \mathbb{R}^+$, para cada aresta $(i,j) \in A$, onde t_{ij} é o fluxo total de carga que passa pela aresta (i,j), e d_{ij} é o coeficiente de atraso de propagação associado a esta aresta, é definida por:

$$\phi^a_{ij} = t_{ij} \ . \ d_{ij} \tag{2.3}$$

Seja $u_{ij} = \frac{t_{ij}}{b_{ij}}$, $\forall\ (i,j) \in A$ a taxa de utilização da aresta (i,j) que representa a razão entre a quantidade de fluxo que atravessa esta aresta e sua capacidade. Define-se o custo da componente congestionamento ϕ^c_{ij} para cada aresta $(i,j) \in A$, segundo uma função convexa e linear por partes, proposta por Fortz e Thorup [10], apresentada na Figura 2.1. Esta função, atribui penalidades diferentes para cada faixa de congestionamento, sendo maiores as penalidades quando o fluxo se aproxima do limite de banda da aresta. O objetivo é distribuir a carga homogeneamente entre as arestas da rede, afim de minimizar o somatório dos custos ϕ^c_{ij}. Observa-se que os fluxos que excedem a capacidade de uma aresta, $\frac{t_{ij}}{b_{ij}} > 1$, são fortemente penalizados. Em contrapartida, se uma aresta (i,j) não possui fluxo, consequentemente $\phi^c_{ij} = 0$. Esta função definida por:

$$\phi^c_{ij} \ = \ b_{ij} \ . \ \begin{cases} u_{ij}, & se\ u_{ij} \in [0, 1/3), \\ 3.u_{ij} - 2/3, & se\ u_{ij} \in [1/3, 2/3), \\ 10.u_{ij} - 16/3, & se\ u_{ij} \in [2/3, 9/10), \\ 70.u_{ij} - 178/3, & se\ u_{ij} \in [9/10, 1), \\ 500.u_{ij} - 1468/3, & se\ u_{ij} \in [1, 11/10), \\ 5000.u_{ij} - 16318/3, & se\ u_{ij} \in [11/10, \infty). \end{cases} \tag{2.4}$$

A escolha desta função, em parte, se deve ao fato de termos a utilização de cada aresta u_{ij} separada em intervalos(classes), com cada classe sofrendo uma penalização distinta. Isto facilita a minimização do custo total da rede, uma vez que utilizando-se deste método, procura-se trocar a classe em que cada uma das arestas se encontra para uma classe que possui menor penalização. Esta função é utilizada também em outros trabalhos [1, 11, 12, 35], os autores justificam seu uso afirmando que, uma função de congestionamento baseada na teoria das filas seria exponencial. Surgiu então a ideia de aproximar esta exponencial por uma função linear por partes com o objetivo de possibilitar uma modelagem linear, e a utilização de ferramentas de programação linear para resolver problemas de multifluxo. Fortz e Thorup [12] testaram variações dos coeficientes desta função e não observaram alterações siguinificativas nos resultados. Em [35], algumas funções de avaliação são comparadas e a função Fortz-Thorup utilizada neste trabalho apresentou bons resultados, sobretudo quando o objetivo é minimizar a maxima utilização, ou seja, diminuir o congestionamento da aresta mais congestionada.

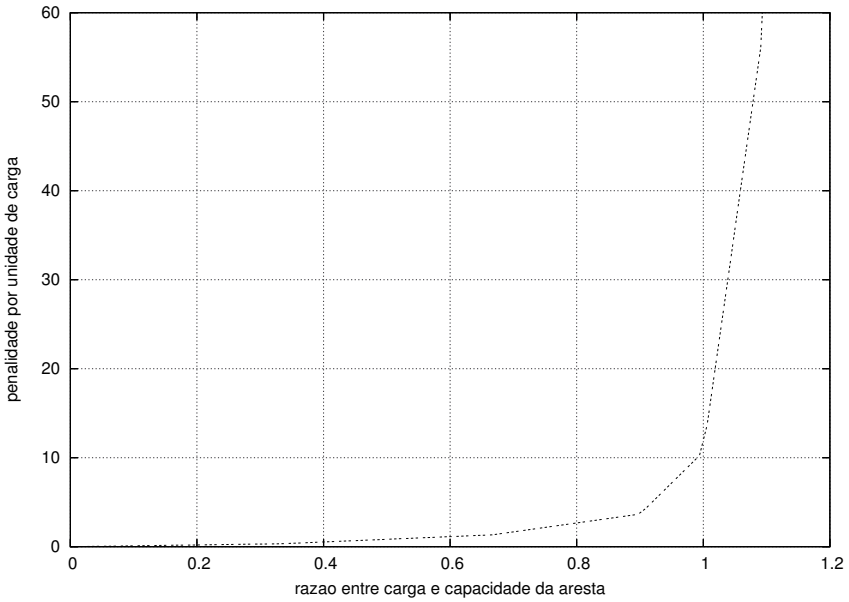

Figura 2.1: Função convexa linear por partes

Dessa forma, o problema de roteamento em redes com número limitado de caminhos consiste em rotear todas as p demandas, através do grafo $G(V, A)$, usando no máximo R caminhos. Com o objetivo de minimizar a função custo ϕ_{ij}.

2.2 Trabalhos Relacionados

Nesta seção são apresentados alguns trabalhos encontrados na literatura, relacionados com os temas estudados neste trabalho.

Em [8], Duhamel, Vatinlen, Mahey e Chauvet desenvolvem um método para definir um limite no número de caminhos utilizados no processo de minimização do congestionamento. Este artigo relata que a determinação de um número mínimo de caminhos para rotear determinada quantidade de fluxo é um problema NP-Difícil.

Outro problema semelhante é abordado em [36], onde dois objetivos são perseguidos, minimizar o congestionamento na rede e minimizar a quantidade de caminhos necessários para rotear um conjunto de demandas. O procedimento proposto é aplicado no problema de multifluxo usando vários caminhos. Foi desenvolvido um algoritmo que tenta gerar um conjunto de caminhos que minimiza o congestionamento usando o menor número de caminhos possíveis.

Guerreiro e Mancini [17] desenvolveram um estudo experimental para comparar três algoritmos para resolver o problema de fluxo máximo entre dois nós com K caminhos disjuntos, ou seja, sem nenhuma aresta comum. Os autores concluíram que para redes puramente randômicas o algoritmo de Kishimoto encontrou, em média, soluções de maior qualidade. Nas redes de grids aleatórios, o algoritmo de busca binária obteve melhor desempenho. Para redes de grids 3-D randômicas, novamente o melhor desempenho foi obtida pelo algoritmo de Kishimoto, seguido pelos algoritmos de fluxo com k-rotas e algoritmo de busca binária.

O problema do fluxo máximo concorrente apresentado por Shahrokhi e Matula [34] consiste em encontrar um fator máximo que as demandas possam ser multiplicadas, de tal modo que exista um fluxo viável satisfazendo as demandas resultantes, obedecendo as restrições de capacidade. Desenvolveu-se um algoritmo aproximativo

polinomial no caso de demandas variadas e capacidades uniformes. Neste trabalho mostra-se também que o problema de fluxo máximo concorrente está relacionado com o problema de congestionamento mínimo.

O problema de fluxo em redes usando apenas um caminho para cada demanda foi introduzido por Kleinberg [18], como uma generalização do problema de caminhos disjuntos. Kleinberg mostrou que este problema pode ser utilizado para modelar vários problemas, como por exemplo, empacotamento, particionamento, balanceamento de carga e roteamento de PVCs. Martens e Skutella [23] também abordam um problema de minimização de congestionamento, onde se permite a divisão do fluxo em k-caminhos com restrições adicionais impostas pelas capacidades das arestas.

Em [4], Baier, Köhler e Skutella estudam o problema de encontrar um fluxo máximo entre um único par origem-destino usando k-caminhos e respeitando as restrições de capacidade. Este problema é classificado como NP-Difícil. Também foram abordados os problemas de encontrar o fluxo máximo para múltiplas demandas usando exatamente k-caminhos e de encontrar o número mínimo de caminhos necessários para rotear um determinado fluxo, que também foram classificados como NP-Difícil.

Existem alguns trabalhos encontrados na literatura, onde os autores utilizam a metaheurística GRASP com reconexão de caminhos para abordar problemas similares ao desta dissertação.

Um problema bastante semelhante a este foi apresentado por Resende e Ribeiro [30], onde algumas variantes do GRASP com reconexão de caminhos foram propostas para o problema de roteamento de PVCs, onde se deseja minimizar a atraso e/ou o congestionamento da rede. Em [13], Ghamlouche, Crainic e Gendreau desenvolvem um procedimento de reconexão de caminhos para o problema de roteamento multifluxos, onde soluções elite são construídas através de um algoritmo de busca tabu usando uma vizinhança baseada em ciclos, e um procedimento de reconexão de caminhos considerando a vizinhança baseada em ciclos é realizado entre duas soluções elite pertencentes ao repositório. Os autores mostram que este método apresentou resultados muito satisfatórios.

Capítulo 3

A Heurística GRASP com Reconexão de Caminhos

A metaheurística GRASP (Greedy Randomized Adaptive Search Prodedure), desenvolvida por Feo e Resende [9], é baseada em um algoritmo iterativo onde cada iteração é constituída por duas fases: a primeira fase, chamada fase de construção, é responsável por gerar uma solução inicial, construída de forma gulosa-aleatória, e a segunda fase, uma fase de busca local, onde as soluções que pertencem a uma determinada vizinhança são pesquisadas até que um ótimo local seja encontrado, sendo que em algumas vezes este ótimo local é também o ótimo global.

O pseudocódigo de um algoritmo GRASP é apresentado na figura 3.1. Este procedimento busca minimizar a função $\Phi(s)$, que calcula o custo total da solução s, e tem $maxIT$ como parâmetro de entrada que informa a quantidade de iterações do algoritmo. Na linha 1, inicializa-se a variável Φ^* com um valor muito grande, utilizada para armazenar o custo da melhor solução, e s^* que armazena a melhor solução como vazia. Em cada uma das $maxIT$ iterações, uma solução inicial x é construída pela *fase de construção*, e em seguida, aplica-se uma *busca local* sobre esta solução inicial. Entre as linhas 5 e 8 atualiza-se a melhor solução encontrada s^* e seu custo Φ^*. Na linha 10 a melhor solução é retornada.

```
procedimento GRASP(maxIT);
1           Φ* ← ∞; s* ← ⊘;
2           para i = 1, ..., maxIT faça
3               x = ConstroiSolucaoRandomica();
4               y = BuscaLocal(x);
5               se Φ(y) < Φ* então
6                   s* ← y;
7                   Φ* ← Φ(y);
8               fim-se;
9           fim-para;
10          retornar (s*);
fim.
```

Figura 3.1: Pseudocódigo de um GRASP para um problema de minimização

3.1 Fase de Construção

Nesta fase, é construída uma solução válida através de um método iterativo, ou seja, inserindo-se um elemento de cada vez na solução que está sendo construída. A cada iteração um *elemento candidato* deverá ser inserido na solução, o qual será escolhido aleatóriamente dentre aqueles que pertencem a *Lista Restrita de Candidatos (LRC)*. A LRC é determinada a cada iteração da seguinte forma. Inicialmente todos os elementos $c \in C$, onde C contém todos os elementos que não fazem parte da solução inicial, são ordenados de forma decrescente de acordo com uma função gulosa $f : C \rightarrow \mathbb{R}$, onde a função f calcula para cada elemento c a contribuição que sua escolha determinará para minimizar a função objetivo. A heurística é dita adaptativa, uma vez que, a cada iteração, a função f é recalculada para todos os elementos, para que as mudanças ocorridas através da inserção do último elemento sejam consideradas. Em seguida, escolhem-se os candidatos que farão parte da LRC. A seguir serão descritos dois modos bastante utilizados na literatura para esta escolha [29]. O primeiro é baseado na cardinalidade, ou seja, escolhe-se um parâmetro $\alpha \in [1, K]$, onde K é o número de elementos que ainda não foram inseri-

dos, e a LRC é formada pelos α primeiros elementos. Dessa forma, quando $\alpha = 1$ o algoritmo semi-guloso transforma-se em um algoritmo puramente guloso, assim como, quando $\alpha = K$ a escolha do elemento se torna totalmente aleatória, uma vez que todos os elementos $c \in C$ pertencem também a LRC. A segunda estratégia, utiliza o *valor* da função f associada a cada candidato. Desta forma, utilizando-se o parâmetro $\alpha \in [0, 1]$ os elementos que pertencem a LRC são definidos por $\{c \in C \mid f_{min} \geq f(c) \geq (f_{min} + \alpha(f_{max} - f_{min}))\}$, onde f_{min} é o menor valor de f, e f_{max} é o maior valor. Observa-se que neste caso, caso α seja igual a 0 a escolha será puramente gulosa, assim como, quanto mais próximo de 1 fôr α, maior será a aleatoriedade da escolha, sendo totalmente aleatória quando $\alpha = 1$. Segundo [25], a maioria dos algoritmos GRASP encontrados na literatura define o tamanho da LRC baseando-se no valor da função gulosa. O valor de α pode permanecer fixo ao longo de todas as iterações ou variar ao longo delas, Resende et. al. [28] propuseram o uso de um α que fosse gerado aleatoriamente no intervalo $[0, 1]$ em cada iteração do GRASP.

Em cada iteração, um elemento deverá ser escolhido, dentre os que pertencem à LRC, para ser inserido na solução parcialmente construída. A escolha deste elemento pode ser realizada segundo algumas estratégias[29]. Em uma dessas estratégias, cada elemento que pertence a LRC possui a mesma probabilidade de ser escolhido. Outro modo utilizado, é atribuir probabilidades diferentes para cada elemento. Em [30], por exemplo, Ribeiro e Resende usam uma função $p : C \rightarrow [0, 1]$ que define a probabilidade que cada elemento possui de ser escolhido. No artigo acima citado é usada uma função $p(c) = \frac{fc}{\sum_{i \in LRC} fi}$ para calcular a probabilidade que determinado elemento pertencente a LRC possui de ser escolhido.

A figura 3.2 mostra a fase de construção de um GRASP, onde na primeira linha uma solução inicial x é inicializada como vazia. Na linha 2, a função *InicializaConjuntoCandidatos*() atribui ao conjunto C uma lista com todos os elementos candidatos ordenados em função de f. Enquanto existirem candidatos, executam-se os passos das linhas 4 a 7. Na linha 4, constrói-se a LRC utilizando o conjunto C de candidatos e o parâmetro α. Na linha 5, um elemento é selecionado de forma aleatória, na linha 6 o elemento escolhido c_e é inserido na solução x e na linha 7

```
procedimento ConstroiSolucaoAleatoria(α);
1            x ← ∅;
2            C ← InicializaConjuntoCandidatos();
3            enquanto C ≠ ∅ faça
4                    LRC ← ConstroiLRC(C, α);
5                    c_e ← SelecionaElementoLRC(LRC);
6                    x ← x ∪ {c_e};
7                    C ← C\{c_e};
8            fim-enquanto;
9            retornar (x);
fim.
```

Figura 3.2: Pseudocódigo do algoritmo guloso-aleatório usado na fase de construção de um GRASP

este elemento é retirado do conjunto C. A função *ConstroiLRC(*C, α*)* é responsável por recalcular, a cada iteração, a função f para todo $c \in C$, ordenar todos esses elementos e definir o tamanho da LRC de acordo com o parâmetro α.

3.2 Busca Local

Como na maior parte dos casos a solução ótima não é encontrada na fase de construção, aplica-se uma busca local com o objetivo de encontrar um ótimo local ou até mesmo o ótimo global, uma vez que a busca local tem como principal meta encontrar um ótimo local dentro da *bacia de atração* na qual se encontra a solução inicial. Caso a solução inicial esteja contida na bacia de atração onde está o ótimo global, este último será encontrado.

Existem básicamente duas estratégias para o uso da busca local, *best-improving* e *first-improving*. No caso da estratégia *best-improving*, toda a vizinhança é verificada e a melhor *solução vizinha* torna-se a solução corrente para próxima iteração, caso ela exista. No caso da estratégia *first-improving*, a primeira solução vizinha que

possua custo melhor que a solução corrente é transformada na solução corrente e outra iteração da busca local é iniciada. Ambos os métodos terminam quando, em uma iteração, toda a vizinhança for verificada, e nenhum vizinho apresente melhor custo que a solução corrente. Um procedimento de busca local *best-improving* para um problema de minimização é apresentado na figura 3.3. Seja x, a solução inicial gerada pela fase de construção, y a solução corrente para cada iteração da busca local, y^* a solução com melhor custo e y' uma solução que pertence a vizinhança V de y. Em cada iteração, todos os elementos $y' \in V$ são verificados e a *solução vizinha* de melhor custo é transformada na solução corrente para a próxima iteração. O procedimento termina quando, em uma iteração qualquer, toda a vizinhança V fôr explorada e não se obtenha nenhuma solução melhor que a solução corrente.

```
procedimento BuscaLocal best(x);
1           Φ(y*) ← ∞ ; y ← x;
2           enquanto Φ(y) < Φ(y*)
3                   y* ← y;
4                   V ← vizinhança(y);
5                   para cada y' ∈ V faça
6                           se Φ(y') < Φ(y) então
7                               y ← y';
8                           fim-se;
9                   fim-para;
10          fim-enquanto;
11          retornar (y*);
fim.
```

Figura 3.3: Pseudocódigo da busca local *best-improving*

A figura 3.4 apresenta um procedimento de busca local que usa a estratégia *first-improving*. A diferença para o método *best-improving* é que na linha 8 o laço iniciado na linha 5 é interrompido, quando é encontrada uma solução vizinha melhor que a solução corrente.

```
procedimento BuscaLocal_first(x);
1        Φ(y*) ← ∞ ; y ← x;
2        enquanto Φ(y) < Φ(y*)
3                y* ← y;
4                V ← vizinhança(y);
5                para cada y' ∈ V faça
6                        se Φ(y') < Φ(y) então
7                            y ← y';
8                            break-para;
9                        fim-se;
10               fim-para;
11       fim-enquanto;
12       retornar (y*);
fim.
```

Figura 3.4: Pseudocódigo da busca local *first-improving*

3.3 Reconexão de Caminhos

A *Reconexão de Caminhos* foi originalmente proposta por Glover [14] como estratégia de intensificação para explorar trajetórias que conectam soluções elite, previamente obtidas através de procedimentos de *busca tabu* e *scatter search* [15, 16]. Inicialmente, dadas duas soluções válidas para o problema, determina-se qual delas será a solução inicial s_i e qual será a solução guia s_g. Partindo da solução inicial, deve-se explorar a trajetória de soluções que segue em direção a solução guia. Esta trajetória é construída trocando, a cada passo, atributos de s_i por atributos de s_g, armazenando-se a melhor solução encontrada no passo, como a solução corrente para a realização do próximo passo. Desse modo, a vizinhança da solução inicial é mais explorada que a vizinhança da solução guia. O procedimento termina quando o único movimento de troca que resta transforma a solução corrente na própria solução guia. A solução final será a melhor solução pertencente a este caminho.

Na figura 3.5 é apresentado o pseudocódigo de um procedimento de reconexão

```
procedimento Reconexao(s_i, s_g);
1       Δ(s, s_g) ← CalculaDiferencaSimetrica(s_i, s_g);
2       Φ* ← CustoMin{Φ(s_i), Φ(s_g)};
3       s* ← SolucaoMin{Φ(s_i), Φ(s_g)};
4       s ← s_i;
5       enquanto Δ(s, s_g) ≠ ∅ faça
6           m* ← RetiraMovMin{Φ(s ⊕ m) : m ∈ Δ(s, s_g)};
7           s ← s ⊕ m*;
8           Δ(s, s_g) ← CalculaDiferencaSimetrica(s, s_g);
9           se Φ(s) < Φ* então
10              Φ* ← Φ(s);
11              s* ← s;
12          fim-se;
13      fim-enquanto;
14      retornar (s*);
fim.
```

Figura 3.5: Pseudocódigo do procedimento de reconexão de caminhos para um problema de minimização

de caminhos, onde s_i é a solução inicial e s_g a solução guia. Inicialmente é calculada a diferença simétrica $\Delta(s_i, s_g)$, ou seja, o conjunto de movimentos necessários para transformar a solução inicial na solução guia. Nas linhas 2 e 3 são armazenados o custo mínimo Φ^* e a melhor solução s^* corrente. Na linha 4, a solução corrente é inicializada com a solução inicial s_i. Enquanto existir algum movimento pertencente ao conjunto $\Delta(s, s_g)$, um novo passo é realizado na direção da solução guia. A cada passo, todos os movimentos $m \in \Delta(s, s_g)$ são aplicados na solução corrente s, e o movimento m^* que apresenta melhor custo é escolhido. Uma nova solução corrente s é produzida através da operação $s \oplus m^*$ na linha 7, e o conjunto de movimentos $\Delta(s, s_g)$ é recalculado na linha 8. Caso o custo da solução corrente $\Phi(s)$ seja menor que o custo mínimo Φ^*, o custo mínimo e a melhor solução são atualizados. O procedimento termina quando $\Delta(s, s_g) = \oslash$, e retorna a solução s^*.

3.4 Grasp com Reconexão de Caminhos

O uso de reconexão por caminhos (RC) como estratégia de intensificação aplicada ao GRASP foi originalmente proposto por Laguna e Martí [22]. Desde então, diversas propostas do uso da reconexão de caminhos sucederam este trabalho [3, 30, 32]. Recentemente pode-se destacar o trabalho de Resende e Ribeiro [31] que mostra os últimos avanços na aplicação do GRASP com reconexão de caminhos.

Na utilização da RC, cria-se um conjunto de *soluções elite*, denominado Repositório, que contém as soluções que apresentem boa qualidade. Para cada procedimento proposto, devem ser definidas algumas características a serem utilizadas no gerenciamento do Repositório, tais como, seu tamanho e sua política de inserção de elementos.

Encontram-se na literatura duas estratégias básicas para o uso da reconexão de caminhos:

- a reconexão de caminhos é aplicada em todos os pares de soluções elite que pertencem ao Repositório. Este tipo de reconexão pode ser realizado periodicamente durante a execução do GRASP ou após a execução de todas as iterações, como um processo de pós-otimização;
- a reconexão de caminhos aplicada como estratégia de intensificação da busca local, ou seja, a reconexão é aplicada entre a solução obtida pela busca local e uma solução elite, após cada iteração do GRASP ou com certa periodicidade.

Alguns trabalhos mostram que, quando a reconexão de caminhos é usada como método de intensificação da fase de busca local, se obtêm resultados melhores do que quando é usada apenas a pós-otimização [31]. Entretanto, quando os dois métodos são combinados, soluções ainda melhores são construídas [31]. Caso o método escolhido seja o de intensificação, a reconexão é realizada entre uma solução gerada pelo método de busca local e uma solução elite, aleatoriamente escolhida dentre as que pertencem ao Repositório. Inicialmente este Repositório está vazio. Uma solução gerada pela fase de busca local ou pela reconexão de caminhos se torna uma solução

candidata a fazer parte do Repositório, caso esta solução tenha boa avaliação de qualidade e seja suficientemente distinta das que pertencem ao Repositório. Uma vez que o Repositório ainda não se encontre cheio, basta inserir a solução candidata, entretanto quando não existem mais lugares livres no Repositório, deve-se usar uma estratégia de substituição de elementos do Repositório. Uma estratégia existente é retirar do Repositório a solução com pior qualidade. Outras opções sugerem retirar a solução que mais se parece com a solução que deverá ser inserida, mantendo assim a diversidade entre as soluções [31]. Neste mesmo trabalho, Resende e Ribeiro mostram que, recentemente, diversas alternativas estão sendo consideradas e combinadas em novas implementações para o procedimento de reconexão de caminhos [3, 33]. São elas:

- **reconexão forward:** onde a solução escolhida para ser a solução inicial é a de pior qualidade e a solução guia a de melhor qualidade;
- **reconexão backward:** onde a solução escolhida para ser a solução inicial é a de melhor qualidade e a solução guia a de pior qualidade;
- **reconexão periódica:** a reconexão de caminhos não é aplicada em todas as iterações do GRASP, mas somente com certa periodicidade;
- **reconexão back and forward:** duas diferentes trajetórias são exploradas. Na primeira utiliza-se a solução de melhor qualidade como inicial e a solução de pior qualidade como guia, e na segunda inverte-se o sentido da reconexão, usando a solução de pior qualidade como inicial e a solução de melhor qualidade como guia;
- **reconexão mista:** os dois sentidos da reconexão são explorados simultaneamente. Substituindo atributos da solução inicial por atributos da solução guia, são geradas diversas soluções intermediárias. A melhor destas soluções será usada no próximo passo como solução guia, e não como solução inicial, que agora será a antiga solução guia, ou seja, os papéis são invertidos, mantendo-se esta alternância até que as soluções inicial e guia sejam iguais;

- **reconexão aleatória:** ao invés de selecionar sempre o melhor movimento ainda não selecionado a cada passo, seleciona-se um elemento d forma aleatória para formar a solução corrente;
- **reconexão truncada:** onde toda a trajetória entre a solução inicial e a solução guia não é investigada, mas apenas parte dela.

Um procedimento GRASP, onde a reconexão de caminhos é usada como método de intensificação, é apresentado na figura 3.6. Este procedimento pode ser dividido em três etapas principais:

1. **fase de construção:** onde uma solução inicial é construída de forma gulosa-aleatória;
2. **busca local:** onde a vizinhança para a solução anteriormente construída é definida e explorada, até que um mínimo local seja encontrado;
3. **reconexão de caminhos:** onde, a partir da segunda iteração, será executada a reconexão de caminhos entre a solução encontrada na fase de busca local, e uma *solução elite* escolhida aleatoriamente no Repositório.

Após a segunda iteração, quando já existe pelo menos um elemento no Repositório de soluções elite, as três etapas acima descritas são executadas. Depois da construção de uma solução inicial e a obtenção de um ótimo local nas linhas 4 e 5, respectivamente, a reconexão é iniciada. Escolhe-se aleatoriamente no Repositório de soluções uma solução na linha 7, e, na linha 8, determina-se qual solução será inicio e qual solução será guia. A reconexão propriamente dita é realizada na linha 9. Na linha 13, a solução x^* é inserida no Repositório, caso obedeça aos critérios adotados para inserção no Repositório. Nas linhas 14 a 17, a melhor solução é armazenada.

Caso o método escolhido seja o de pós-otimização, ao finalizar a execução de todas as iterações do GRASP, a reconexão de caminhos será realizada entre todos os possíveis pares de soluções que pertençam ao Repositório. Este método é

```
procedimento GRASP+RC(maxIT);
1           P ← ∅;
2           Φ* ← ∞;
3           para i = 1, ..., maxIT faça
4               x ← ConstroiSolucaoAleatoria();
5               y ← BuscaLocal(x);
6               se i ≥ 2 então
7                   z ← EscolheRandomicamenteSolucaoElite(P);
8                   DeterminaSolucaoInicialeGuia(y, z);
9                   x* ← ReconexaoDeCaminhos(y, z);
10              senão
11                  x* ← y;
12              fim-se;
13              P ← InsereRepositorio(P, x*);
14              se Φ(x*) < Φ* então
15                  Φ* ← Φ(x*);
16                  s* ← x*;
17              fim-se;
18          fim-para;
19          retornar (s*);
fim.
```

Figura 3.6: Pseudocódigo do procedimento de reconexão de caminhos para um problema de minimização

```
procedimento GRASP+RC(maxIT);
1            P ← ⊘;
2            para i = 1, ..., maxIT faça
3                 x ← ConstroiSolucaoRamdomica();
4                 y ← BuscaLocal(x);
5                 P ← InsereRepositorio(P, y);
6            fim-para;
7            PosOtimizacao(P);
8            s* ← SolucaoMin(P);
9            retornar (s*);
fim.
```

Figura 3.7: Pseudocódigo de um GRASP com reconexão de caminhos, usando o método de pós-otimização, para um problema de minimização

ilustrado na figura 3.7. Este procedimento consiste em preencher o Repositório de soluções elite durante as $maxIT$ iterações do GRASP, onde, em cada iteração, a função *InsereRepositório(P, y)* é responsável por inserir o elemento y no Repositório P caso o elemento candidato y satisfaça as condições de qualidade de solução e diversidade. Terminadas todas as iterações, realiza-se a reconexão de caminhos entre todos os pares de soluções elite que pertençam ao Repositório P, como mostrado na linha 7. Na linha 8, o algoritmo encontra a melhor solução do conjunto elite, que, consequentemente, será a solução de melhor qualidade encontrada pela heurística.

Capítulo 4

O GRASP com Reconexão de Caminhos Proposto

Neste capítulo será apresentado o GRASP com reconexão de caminhos a partir de agora denominado GRASP+RC, proposto por esta dissertação. No desenvolvimento deste algoritmo várias estratégias foram desenvolvidas para a fase de construção, busca local e procedimento de reconexão.

Na seção 4.1 será definido o método para se calcular o caminho mínimo entre um par origem-destino nas fases de construção e busca local. Na seção 4.2 será apresentada a fase de construção usada no GRASP+RC, assim como os passos realizados no processo de seu desenvolvimento. Procedimentos desenvolvidos para a fase de busca local são descritos na seção 4.3. A seção 4.4 apresenta o procedimento de reconexão de caminhos utilizado.

4.1 Definição do Caminho Mínimo entre Origem e Destino

O algoritmo de Dijkstra [6] é utilizado para determinar o caminho de custo mínimo necessário para se rotear determinada quantidade de carga l, associada a uma

demanda k, entre um nó origem o_k e um nó destino d_k . Este algoritmo usa pesos, previamente atribuídos a cada aresta para determinar este menor caminho. Neste problema, determina-se o peso w_{ij} associado a cada aresta $(i,j) \in A$, calculando-se o impacto apresentado pela possível inserção da carga l, nesta aresta. Este peso é calculado da seguinte forma: Seja ϕ_{ij}, o custo associado a aresta (i,j), definido por

$$\phi_{ij} \;=\; (1-\delta)\;.\;\phi^a_{i,j} \;+\; \delta\;.\;\phi^c_{ij}, \tag{4.1}$$

onde $\phi^a_{i,j}$ é a componente de atraso de propagação, e ϕ^c_{ij} o congestionamento, relativos à aresta (i,j). O parâmetro de entrada $\delta \in [0,1]$, determina o peso que cada uma das componentes anteriores possui, no cálculo do custo ϕ_{ij}, como apresentado em detalhes no capítulo 2. Considere também,

$$\bar{\phi}_{ij} \;=\; (1-\delta)\;.\;\bar{\phi}^a_{i,j} \;+\; \delta\;.\;\bar{\phi}^c_{ij}, \tag{4.2}$$

como o possível novo custo da aresta (i,j), onde este novo custo $\bar{\phi}_{ij}$, é calculado após a adição da carga l na carga atual t_{ij}, que atravessa a aresta (i,j). Desta forma, o aumento da carga t_{ij} resulta no aumento das componentes de atraso e congestionamento relativos a aresta (i,j). Sendo assim, o peso associado a aresta (i,j) é definido por

$$w_{ij} = \bar{\phi}_{ij} \;-\; \phi_{ij}, \tag{4.3}$$

ou seja, a diferença entre o possível novo custo $\bar{\phi}_{ij}$, calculado após a inserção da carga l na aresta (i,j), e o custo atual desta aresta ϕ_{ij}.

4.2 Fase de Construção

Na fase de construção, uma solução inicial é construída usando para cada demanda k um único caminho c^1_k que deve rotear toda sua carga l_k. Os caminhos são inseridos um a um na solução até que todas as K demandas sejam roteadas. A estratégia usada para escolher qual demanda será inserida na solução a cada iteração foi desenvolvida através de numerosos testes que levaram em consideração a qualidade e a diversidade das soluções construídas, assim como o tempo dispensado.

Inicialmente, constrói-se uma lista C com todas as K demandas ordenadas de forma decrescente ou crescente em função de sua carga l_k. A cada iteração, define-se a Lista Restrita de Candidatos, a partir de agora denominada apenas LRC e, escolhe-se uma demanda $k \in LRC$ a qual é retirada da lista C. O caminho mínimo c_k^1 para rotear esta demanda é calculado através do algoritmo de **Dijkstra** [6] e é inserido na solução inicial S_i.

Considerando que $\alpha \in [0, 1]$ é o parâmetro que define o tamanho da LRC, l_{max} é a maior carga encontrada na lista C e l_{min} a menor, foram desenvolvidas as seguintes estratégias para definir o tamanho da LRC:

1. **Crescente/Posição**, onde os $\alpha.|C|$ primeiros elementos compõem a LRC e a lista C é ordenada de forma *crescente* em relação a carga l_k.
2. **Decrescente/Posição**, onde os $\alpha.|C|$ primeiros elementos compõem a LRC e a lista C é ordenada de forma *decrescente* em relação a carga l_k.
3. **Crescente/Valor**, onde os elementos que compõem a LRC são aqueles cujo valor de l_k se encontra no intervalo $[\ l_{min}\ ,\ l_{min} + \alpha(l_{max} - l_{min})\ ]$ e a lista C é ordenada de forma *crescente* em relação a carga l_k.
4. **Decrescente/Valor**, onde os elementos que compõem a LRC são aqueles cujo valor de l_k se encontra no intervalo $[\ l_{max}\ ,\ l_{max} + \alpha(l_{min} - l_{max})\ ]$ e a lista C é ordenada de forma *decrescente* em relação a carga l_k.

A Figura 4.1 apresenta o procedimento da fase de construção do GRASP proposto. Nas linhas 1 e 2 são inicializadas a solução inicial S_i e a lista C com todas as K demandas ordenadas de acordo com a política adotada para definir a LRC. Enquanto ainda existirem demandas a serem roteadas o laço das linhas 3 a 9 será executado. Na linha 4 cria-se a LRC e na linha 5 é selecionado um elemento $k \in LRC$. Na linha 6, define-se um caminho c_k^1 para que seja roteada a carga l_k, pelo procedimento Dijkstra [6]. Nas duas últimas linhas deste laço são atualizadas a solução inicial, com a inclusão do caminho c_k^1, e a lista C de onde será retirada a demanda k. Ao final deste laço, a solução inicial, que estará completa com um caminho para cada demanda, poderá ser retornada.

```
procedimento CONSTRUTIVO_GULOSO_ALEATORIO(α);
1           S_i ← ∅;
2           C ← InicializaConjuntoCandidatos();
3           enquanto C ≠ ∅ faça
4                       LRC ← DefineLRC(C);
5                       k ← SelecionaElementoLRC(LRC);
6                       c_k^1 ← Dijkstra(k);
7                       S_i ← S_i ∪ {c_k^1};
8                       C ← C\{k};
9           fim-enquanto;
10          retornar (S_i);
fim.
```

Figura 4.1: Pseudocódigo da fase de construção do GRASP proposto

4.3 Busca Local

Para a fase de Busca Local foram desenvolvidas três estratégias, busca local **por aresta**, busca local **por demanda** e busca local **por caminho**. Todas partem de uma solução inicial S_i gerada pela fase de construção, onde existe apenas um caminho c_k^1 para rotear a carga de cada demanda k. Os procedimentos baseiam-se então em gerar novos caminhos para o roteamento destas cargas, visando equilibrar as cargas pelos caminhos utilizados de forma que o custo da solução seja o menor possível. A firgura 4.2 ilustra a estrutura da solução em cada um das duas fases do procedimento GRASP. Na fase de construção observa-se a solução inicial, onde cada demanda k tem toda sua carga roteada apenas por um único caminho c_k^1. Enquanto em uma solução parcial, gerada pela fase de busca local, pode-se observar que a carga associada a cada demanda pode ser dividida em até R caminhos.

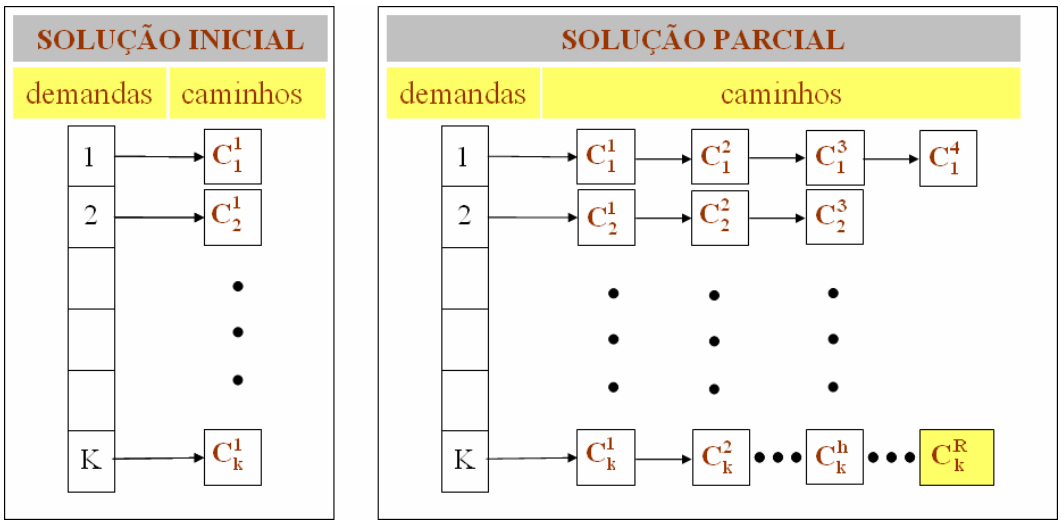

Figura 4.2: Esquema da solução inicial gerada pela fase de construção e Solução parcial durante a execução fase de Busca Local

4.3.1 Busca Local *por aresta*

Esta primeira estratégia utiliza o custo de cada aresta como fator de escolha de quais caminhos serão criados para rotear a carga de cada demanda. A cada iteração da busca local, gera-se uma lista L com todas as m arestas, ordenadas de forma decrescente em função de seu custo ϕ_{ij}, e define-se uma Lista Restrita de Candidatos LRC_{bl}. A LRC_{bl} contém os $\beta.|L|$ primeiros elementos de L, onde $\beta \in [0,1]$ é um parâmetro de entrada. A cada aresta $(i,j) \in LRC_{bl}$ selecionada, retira-se 50% da carga f_k^h de cada caminho c_k^h que utiliza esta aresta, cuja demanda k associada não tenha esgotado o número máximo de caminhos R permitidos para roteá-la. Em seguida, calcula-se para cada carga $f_k'^h$ retirada, um novo caminho mínimo $c_k'^h$, que poderá usar ou não a aresta (i,j). Estes caminhos são inseridos na solução, caso determinem uma melhora em seu custo. Caso contrário, são descartados, e suas cargas são devolvidas para os caminhos originais. Na hipótese de se encontrar um caminho mínimo calculado $c_k'^h$ igual a um caminho qualquer $c \in C_k$, a carga a ser roteada $f_k'^h$ é apenas adicionada ao caminho c já existente, e não é criado um novo caminho. O procedimento termina quando numa iteração qualquer, todas as arestas $(i,j) \in L$ forem verificadas e o custo da solução não apresente melhora.

As Figuras 4.3(a), 4.3(b) e 4.3(c) ilustram uma iteração da fase de busca local por aresta. Em cada figura é apresentado um modelo da rede e a solução parcial referente. Na figura 4.3(a) observa-se na solução parcial duas colunas, a primeira caracterizando as demandas, onde se encontram, número da demanda, o par de nós origem-destino e a quantidade de carga. Na segunda coluna encontram-se os caminhos utilizados para rotear a totalidade da carga associada a cada demanda. Cada caminho é representado o gráfico por setas coloridas, os caminhos c_1^2 e c_2^2 são representados por setas pontilhadas. Após ser escolhida aleatoriamente uma aresta, como já foi explicado anteriormente, é retirado de cada caminho que utiliza esta aresta 50% de sua carga, como mostra a figura 4.3(b). No terceiro passo, ilustrado pela figura 4.3(c), calcula-se um novo caminho mínimo para cada carga retirada da aresta. Nesta ilustração, usada como exemplo, encontrou-se um novo caminho mínimo (c_1^2) para a demanda 1, para a demanda 2 verificou-se que o caminho mínimo calculado já existe, dessa forma adiciona-se a este caminho as 50 unidades de carga retiradas anteriormente, e para a demanda 3 não se encontrou um caminho de custo menor que o caminho inicial de onde a carga foi originalmente retirada, sendo assim a carga foi apenas devolvida.

Na figura 4.4, encontra-se o procedimento de busca local por aresta. Neste procedimento, a primeira linha inicializa a solução corrente S com a solução inicial S_i, e na segunda, atribui-se infinito a Φ^* que representa o melhor custo encontrado. O laço da linha 3 a linha 15 se repete enquanto se verificar uma melhora no custo da solução corrente. O melhor custo Φ^* é atualizado na linha 4. Uma lista L com m arestas ordenadas de forma decrescente em função do seu custo ϕ_{ij} para a solução corrente S é inicializada na linha 5. O laço interno, da linha 6 a linha 14, define uma iteração. A cada iteração, uma LRC_{bl} é gerada a partir de L, na linha 7. Em seguida, uma aresta $(i, j) \in LRC_{bl}$ é selecionada aleatoriamente e retirada da lista L. Na linha 10, uma nova solução S_{aux} é gerada pela função $AtualizaSolucao(S, (i, j))$, através da criação de novos caminhos utilizados para rotear as cargas retiradas dos caminhos que utilizam a aresta (i, j), como descrito anteriormente. Nas linhas 11 e 12, a solução corrente S é atualizada, caso o custo de S_{aux} seja menor do que o custo de S. No final do procedimento, retorna-se a melhor solução encontrada S.

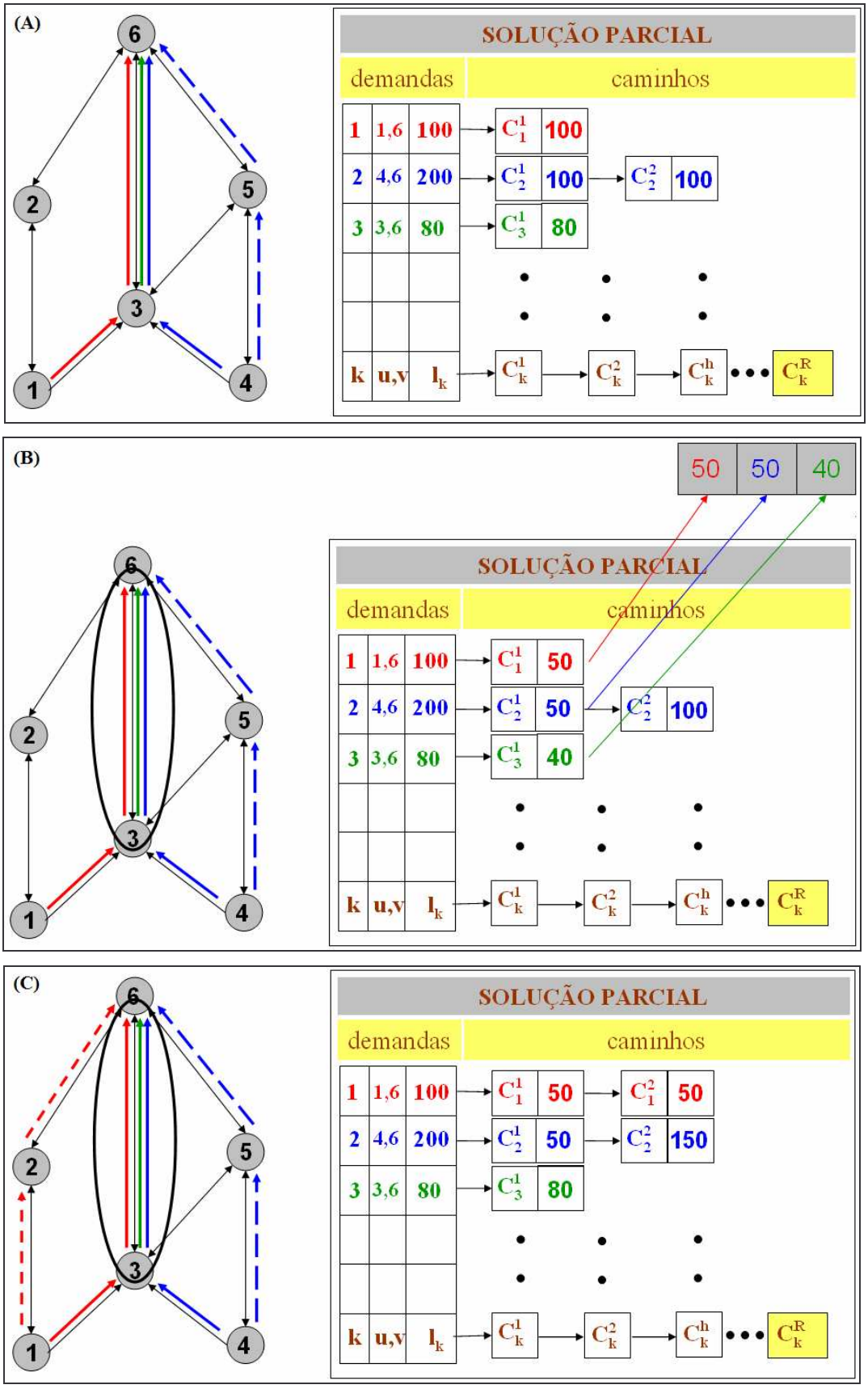

Figura 4.3: Estratégia de Busca Local por Aresta

```
procedimento BUSCA_LOCAL_por_aresta(S_i, β);
1       S ← S_i;
2       Φ* ← ∞;
3       enquanto Φ(S) < Φ* faça
4           Φ* ← Φ(S);
5           L ← InicializaConjuntoCandidatos();
6           enquanto L ≠ ∅ faça
7               LRC_bl ← GeraLRC_bl(L, β);
8               (i, j) ← SelecionaArestaLRC_bl();
9               L ← L\{(i, j)};
10              S_aux ← AtualizaSolucao(S, (i, j));
11              se Φ(S_aux) < Φ(S) então
12                  S ← S_aux;
13              fim-se;
14          fim-enquanto;
15      fim-enquanto;
16      retornar (S);
fim.
```

Figura 4.4: Pseudocódigo da busca local por **aresta** do GRASP proposto

4.3.2 Busca Local *por demanda*

A busca local por demanda distribui de forma uniforme a carga de qualquer demanda k entre seus caminhos. A escolha de qual será o novo caminho criado é baseada nas demandas e não mais nas arestas, como vimos no procedimento anterior. Nesta estratégia, inicialmente cria-se uma lista L ordenada de forma decrescente em função de $q(k) = \frac{l_k}{r(k)}$, onde l_k é a carga da demanda k e $r(k)$ é o número de caminhos utilizados pela demanda k. A cada iteração, define-se uma LRC_{bl}, de onde serão selecionadas, uma por vez, todas as demandas. A LRC_{bl} é constituída pelas $\beta.|L|$ primeiras demandas da lista L, onde $\beta \in [0, 1]$ é um parâmetro de entrada. Selecionada uma demanda k aleatoriamente, caso seu número de caminhos

utilizados $r(k)$ não tenha alcançado o número máximo de caminhos permitido R, cria-se um novo caminho $c_k^{r(k)+1}$. Divide-se então, a carga l_k da demanda k escolhida, uniformemente entre todos os $r(k)$ caminhos, incluindo o novo. Desta forma, a carga de cada caminho $c_k^h \in C_k$ será igual a $q(k) = \frac{l_k}{r(k)}$. Este novo caminho $c_k^{r(k)+1}$ será definido, calculando-se o caminho mínimo para rotear a nova quantidade de carga $q(k)$. Caso a criação deste novo caminho provoque uma melhora no custo da solução corrente, este caminho será inserido nesta solução, caso contrário, nenhuma alteração é realizada. O procedimento chega ao seu fim, quando uma iteração qualquer termina e o custo da solução corrente não apresenta melhora.

Um exemplo de uma iteração da fase de busca local por demanda é apresentado na figura 4.5, onde inicialmente observa-se três demandas sendo roteadas através de seus caminhos associados(figura 4.5.a). Seleciona-se aleatoriamente uma demanda, neste exemplo a demanda 1(figura 4.5.b). Dividi-se então a carga da demanda 1 entre todos os caminhos associados a esta demanda, inclusive o novo caminho que esta sendo criado, neste caso, como existe apenas um caminho, a carga da demanda é dividida entre dois caminhos, como mostra a figura 4.5.c. Calculado este caminho mínimo, caso o custo total de rede diminua o novo caminho é inserido na solução, caso contrário, nenhuma alteração é realizada.

A Figura 4.6 apresenta o procedimento de busca local por demanda. Na primeira linha, inicializa-se a solução S com a solução inicial S_i, e atribui-se infinito a Φ^*, que representa o melhor custo encontrado em cada iteração. O laço localizado entre a linha 2 e a linha 17 será repetido enquanto uma melhora no custo da solução for obtida. Na linha 4, a lista L é inicializada com todas as demandas k, ordenadas de forma decrescente em função de suas cargas l_k. O laço interno da linha 5 a linha 16, será executado $|K|$ vezes em cada iteração, uma para cada demanda. Na linha 6, define-se a LRC_{bl} que possui os $(\beta.|L|)$ primeiros elementos da lista L. Em seguida, uma demanda $k \in LRC_{bl}$ é selecionada aleatoriamente, e na linha seguinte esta demanda é retirada da lista L. Verifica-se se a demanda k já esgotou seu limite máximo de caminhos R na linha 9. Caso esse limite ainda não tenha sido alcançado, cria-se um novo caminho $c_k^{r(k)+1}$ através da função $CriaNovoCaminho()$. Nesta função, inicialmente calcula-se a carga $q(k)$ de cada caminho através da razão entre a carga

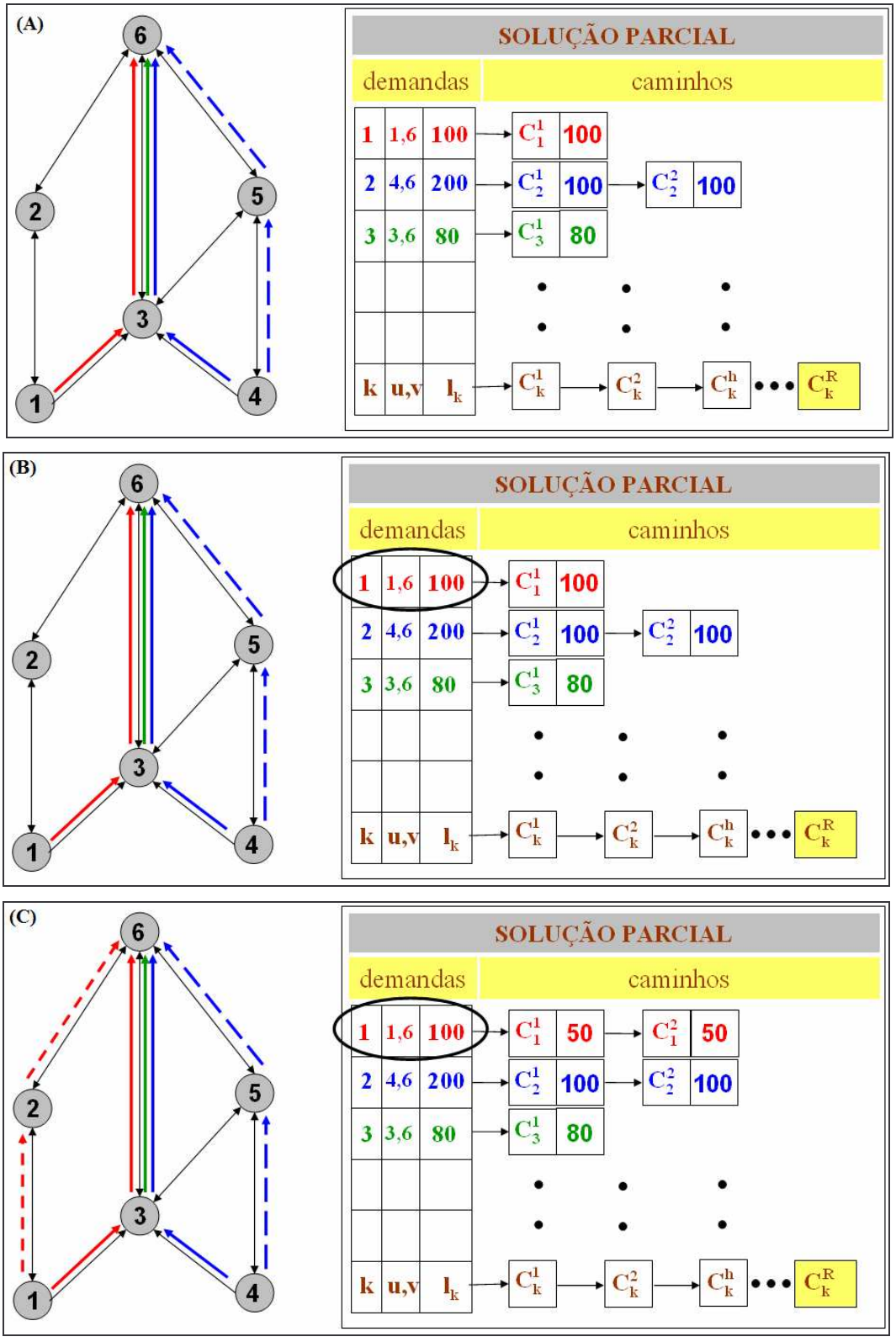

Figura 4.5: Estratégia de Busca Local por Demanda

da demanda l_k e o número atual de caminhos $r(k)$, incluindo o caminho que esta sendo criado. Desta forma, todos os caminhos de uma mesma demanda possuirão a mesma carga. Na segunda etapa desta função, calcula-se o caminho mínimo para rotear a carga do novo caminho $c_k^{r(k)+1}$. Na linha 11, a função *AtualizaSolucao*() gera uma solução S_{aux} com os efeitos das mudanças realizadas. Caso esta solução apresente melhor custo que a solução corrente, esta última será atualizada na linha 13. No final deste procedimento, retorna-se a solução corrente, contendo a solução encontrada de melhor custo.

```
procedimento BUSCA_LOCAL_por_demanda(S_i, β);
1            S ← S_i; Φ* ← ∞;
2            enquanto Φ(S) < Φ* faça
3                Φ* ← Φ(S);
4                L ← InicializaConjuntoCandidatos();
5                enquanto L ≠ ∅ faça
6                    LRC_bl ← GeraLRC_bl(L, β);
7                    k ← SelecionaElemento(LRC_bl);
8                    L ← L\{k};
9                    se r(k) < R então
10                       c_k^{r(k)+1} ← CriaNovoCaminho(k);
11                       S_aux ← AtualizaSolucao(S, k, c_k^{r(k)+1});
12                       se Φ(S_aux) < Φ(S) então
13                           S ← S_aux;
14                       fim-se;
15                   fim-se;
16               fim-enquanto;
17           fim-enquanto;
18           retornar (S);
fim.
```

Figura 4.6: Pseudocódigo da busca local por **demanda** do GRASP proposto

4.3.3 Busca Local *por caminho*

Assim como nas estratégias de busca local anteriores, este procedimento é aplicado a partir de uma solução inicial S_i, que possui apenas um caminho para cada demanda k. Basicamente, serão gerados novos caminhos para cada demanda, com o objetivo de dividir entre estes caminhos o fluxo que inicialmente era roteado por apenas um caminho.

Inicialmente, cria-se uma lista L que contém os caminhos c_k^h da solução corrente, ordenados de forma decrescente, em função de suas cargas f_k^h. Na primeira iteração, esta lista contém um número $|K|$ de caminhos igual ao número de demandas, pois existe somente um caminho por demanda na solução inicial, gerada pela fase de construção. A cada iteração é gerada uma LRC_{bl}, com os $(\beta.|L|)$ primeiros caminhos da lista L, onde $\beta \in [0,1]$ é um parâmetro de entrada. Seleciona-se então um caminho $c_k^h \in LRC_{bl}$, que significa o *h-ésimo* caminho da demanda k. Caso o número máximo de caminhos permitido para cada demanda ainda não esteja esgotado para a demanda k, um novo caminho será proposto. Retira-se então 50% do fluxo do caminho escolhido c_k^h, e calcula-se o caminho mínimo origem-destino para o fluxo retirado. Existem então três possibilidades:

- *o caminho mínimo calculado é igual ao caminho origem* c_k^h, sendo assim, o fluxo retirado é devolvido e o novo caminho não é criado;
- *o caminho mínimo calculado é igual a algum caminho já existente para a demanda* k, dessa forma, o fluxo retirado do caminho c_k^h é adicionado ao fluxo do caminho já existente;
- *o caminho mínimo calculado ainda não existe*, então, cria-se um novo caminho, atualiza-se a solução S e este caminho será posteriormente inserido na lista L.

A figura 4.7 apresenta uma iteração da fase de busca local por caminho. Inicialmente temos a solução parcial onde pode-se observar três demandas sendo roteadas pelos seus respectivos caminhos(figura 4.7.a). Um caminho é então selecionado,

neste exemplo o caminho c_3^1 (figura 4.7.b), metade da carga deste caminho é retirada e calcula-se o caminho mínimo para rotear esta carga. Caso o custo total da rede diminua um novo caminho é então criado, neste caso c_3^2, caso contrário, nenhuma alteração é realizada (figura 4.7.c).

Após todos os caminhos c_k^h que pertenciam à lista L no início desta iteração serem verificados, atualiza-se a lista L, inserindo os caminhos que foram criados nesta iteração, e uma nova iteração é realizada. Este procedimento termina quando em uma iteração todos os caminhos $c_k^h \in L$ são verificados e não se encontra melhora no custo da solução.

A Figura 4.8 apresenta o procedimento de busca local por caminho. A primeira linha inicializa a solução S com a solução inicial S_i, e na segunda atribui-se infinito a Φ^* que representa o melhor custo encontrado em cada iteração. Nas linhas 3 e 4 inicializam-se as listas que serão utilizadas por este procedimento. Uma lista auxiliar Q, que servirá para armazenar os novos caminhos gerados em uma determinada iteração, será inicializada como vazia, e uma lista L é inicializada com uma entrada para cada caminho, pois a S_i possui apenas um caminho para cada demanda. O laço externo que se inicia na linha 5 e termina na linha 22 será executado enquanto cada iteração da busca local produzir melhorias no custo da solução. Cada execução deste laço será considerada uma iteração da busca local, desta forma, a cada iteração o melhor custo é atualizado na linha 6. Na linha 7 atualiza-se a lista L com os novos caminhos que foram criados na iteração anterior. Estes caminhos ficam temporariamente armazenados na lista auxiliar Q, que é esvaziada na linha 8. Na linha 9, é inicializado o laço interno, que será executado uma vez para cada caminho que pertence a L. Neste laço, inicialmente é criada a LRC_{bl} na linha 10, com os $(\beta.|L|)$ primeiros elementos da lista L. Na linha seguinte, é selecionado aleatoriamente dentre os elementos da LRC_{bl}, um caminho c_k^h, que representa o *h-ésimo* caminho da demanda k e na linha 12, retira-se este caminho da LRC_{bl}. Caso não tenham sido esgotados os caminhos para a demanda k, um novo caminho $c_k^{r(k)+1}$ é criado pela função $CriaCaminho()$. Esta função retira 50% da carga do caminho c_k^h e cria um novo caminho $c_k^{r(k)+1}$ calculando o caminho mínimo origem-destino para carga retirada. Na linha 15, uma *solução temporária* S_{aux} é criada, inserindo-se este

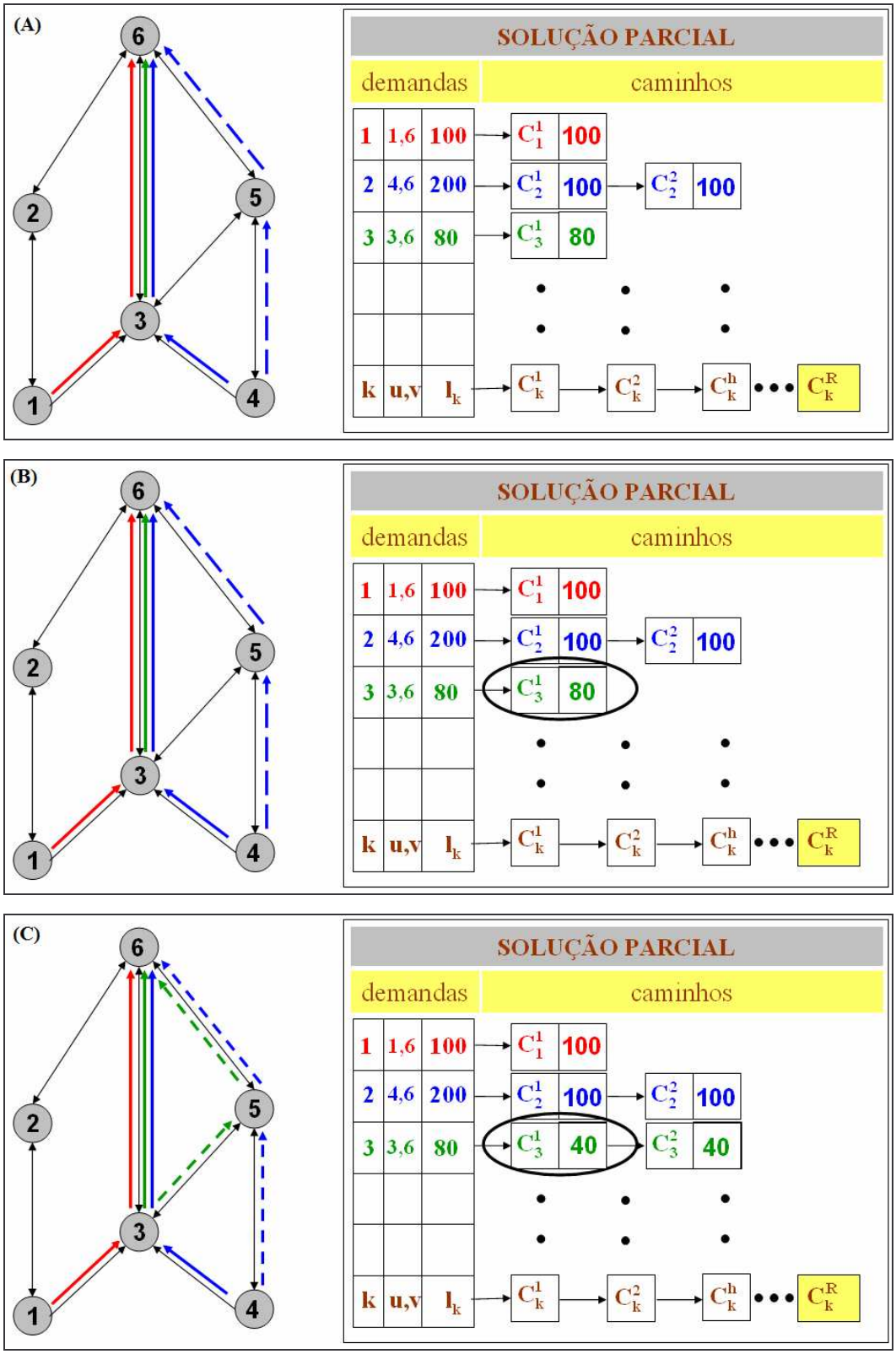

Figura 4.7: Estratégia de Busca Local por Caminho

```
procedimento BUSCA_LOCAL_por_caminho(S_i, β);
1   S ← S_i;
2   Φ* ← ∞;
3   Q ← ∅;
4   L ← InicializaConjuntoCandidatos();
5   enquanto Φ(S) < Φ* faça
6       Φ* ← Φ(S);
7       L ← AtualizaConjuntoCandidatos(L, Q);
8       Q ← ∅;
9       para ∀ c_k^h ∈ L faça
10          LRC_bl ← GeraLRC_bl(L, β);
11          c_k^h ← SelecionaElemento(LRC_bl);
12          L ← L\{c_k^h};
13          se r(k) < R então
14              c_k^{r(k)+1} ← CriaCaminho(c_k^h);
15              S_aux ← AtualizaSolucao(S, c_k^{r(k)+1});
16              se Φ(S_aux) < Φ(S) então
17                  S ← S_aux;
18                  Q ← InsereNovoCaminho(Q, c_k^{r(k)+1});
19              fim-se;
20          fim-se;
21      fim-para;
22  fim-enquanto;
23  retornar (S);
fim.
```

Figura 4.8: Pseudocódigo da busca local por **caminho** do GRASP proposto

novo caminho na solução corrente S. Na linha 16, é verificado se o custo da solução temporária é menor que o custo da solução corrente. Caso esta condição seja verdadeira, a solução S recebe a solução temporária S_{aux} na linha 17, e insere-se o novo caminho $c_k^{r(k)+1}$ na lista auxiliar Q. No fim do procedimento retorna-se a solução S, que representa a melhor solução encontrada em todas as iterações da busca local.

4.4 Reconexão por Caminhos

A reconexão de caminhos proposta por este trabalho é realizada como método de intensificação da busca local em todas as iterações do GRASP. Este procedimento é utilizado a partir da segunda iteração, uma vez que é necessário existir ao menos uma solução no repositório, para que a reconexão possa ser realizada. A reconexão de caminhos é aplicada em cada iteração entre uma solução gerada pelo GRASP(solução corrente) e uma solução selecionada no repositório. O sentido em que esta reconexão é realizada depende do método utilizado. Neste trabalho quatro estratégias foram propostas. A estratégia backward(*RCback*) que realiza a RC usando a solução de maior qualidade como *inicial* e a solução de menor qualidade como *guia*. Duas variantes desta estratégia também foram propostas, a *RCt50* e a *RCt20*, que realizam a reconexão no sentido backward, porém, percorrem apenas 50% e 20% da trajetória de reconexão, respectivamente. Outra estratégia proposta foi a reconexão de caminhos forward(*RCfor*), onde a solução de menor qualidade é utilizada como *inicial* e a solução de maior qualidade como *guia*.

Na Figura 4.9 pode-se observar o esquema de reconexão de caminhos. Como a dinâmica da reconexão de caminhos baseia-se em trocar atributos da solução inicial por atributos da solução guia, é de relevante importância a identificação destes elementos. Como ilustra a figura 4.9.a, uma solução é formada por K atributos, onde cada atributo esta associado a uma demanda e é formado pelo conjunto de caminhos usados para rotear a carga desta demanda. Duas soluções (S_1 e S_2) são consideradas iguais quando todos os atributos da solução S_1 são iguais a todos os atributos correspondentes da solução S_2. Assim como, dois atributos corresponden-

tes são considerados iguais quando o conjunto de caminhos e a quantidade de carga que percorre cada caminho são iguais. Para realizar-se o procedimento de reconexão de caminhos, inicialmente obtém-se a solução gerada pelo GRASP e seleciona-se aleatoriamente outra solução no repositório. Em seguida, de acordo com a estratégia utilizada, determina-se qual solução será a "Solução Inicial" e qual será a "Solução Guia". Na figura 4.9.b, os atributos 1 e 4 são iguais, ou seja, a cargas das demandas 1 e 4 são roteadas de forma exatamente igual tanto na solução inicial, quanto na solução guia. Cada um dos atributos da solução inicial, que seja diferente de seu correspondente na solução guia será substituído por este atributo correspondente. Os atributos devem ser substituídos um a um, e à cada substituição o custo da solução corrente deverá ser calculado. Ao final de cada passo, a substituição, denominada movimento, que resultou na solução de menor custo, torna-se a solução corrente para o próximo passo. No exemplo apresentado observa-se na figura 4.9.c que todos os atributos da solução inicial são iguais aos atributos da solução guia, alcançando o critério de parada do procedimento. Será então admitida como solução final, a solução de menor custo encontrada dentre as soluções verificadas durante o procedimento.

4.4.1 Gerenciando o Repositório de Soluções Elite

Para que se possa realizar a reconexão de caminhos, devem ser definidas as políticas de gerenciamento para o conjunto de soluções elite. Este conjunto, denominado **Repositório**, contém uma quantidade limitada de elementos, pré-determinada por um parâmetro $\boldsymbol{T}$ de entrada. Neste conjunto são realizadas basicamente dois tipos de ações: inclusão e seleção de elementos. A seleção de elementos, que consiste em escolher um elemento para que se realize a reconexão deste com uma solução corrente, é realizada de forma totalmente aleatória. Para gerenciar a inclusão de uma determinada solução no Repositório, definiram-se duas estratégias:

- **a solução corrente entra no lugar da pior solução do Repositório**: esta estratégia consiste em, dado que o custo de uma solução corrente é melhor que o custo da pior solução do Repositório, a solução corrente é inserida no lugar

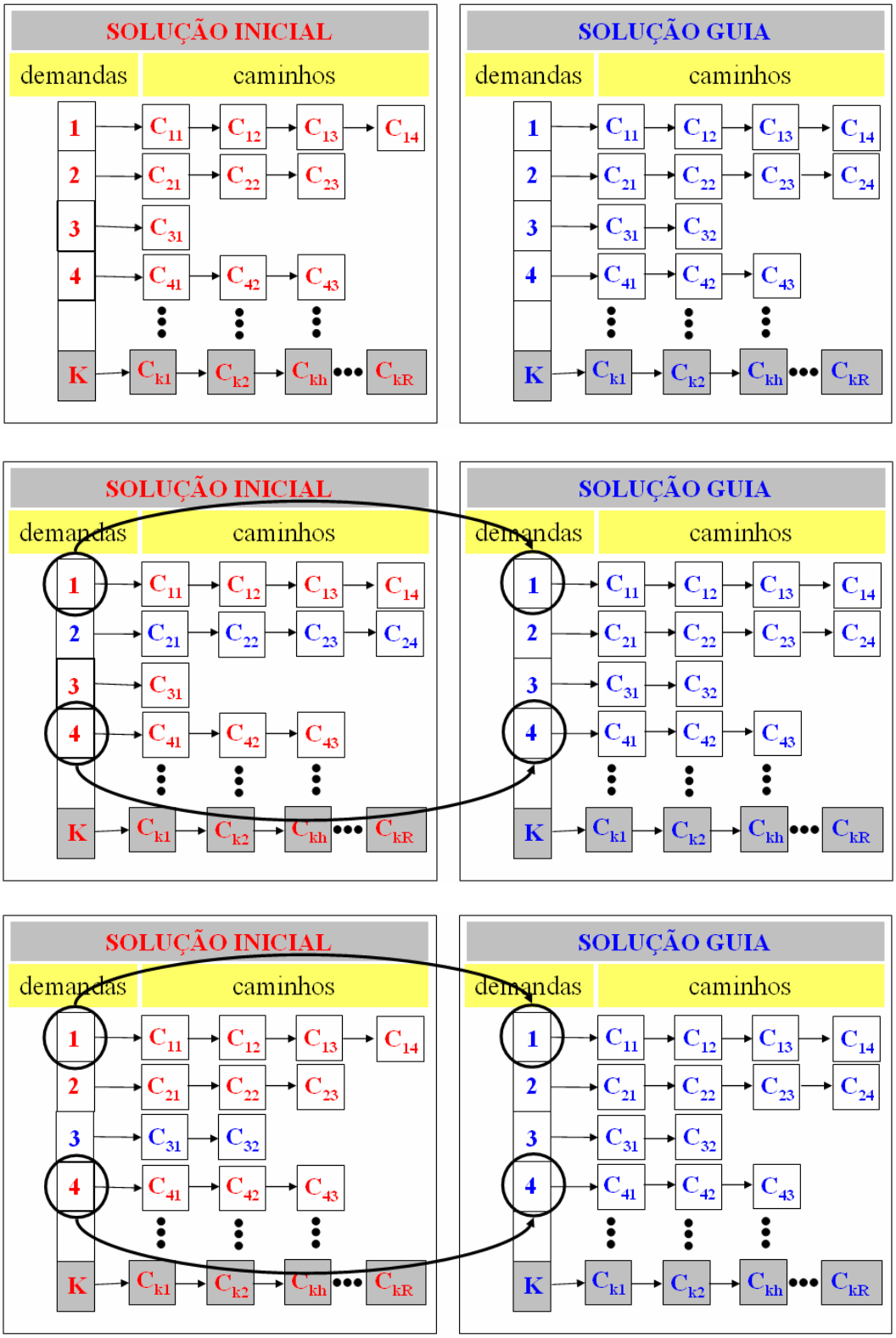

Figura 4.9: Exemplo do esquema de reconexão de caminhos utilizado

da pior solução;

- **a solução corrente entra no lugar da solução mais semelhante a ela**: caso a solução corrente apresente um custo menor que o custo da pior solução do Repositório, a solução corrente será inserida no lugar da solução que apresentar menor diversidade a ela, ou seja, a solução elite que for constituída pelo maior número de elementos iguais. Cumprindo assim o objetivo de manter a diversidade entre as soluções do Repositório.

4.4.2 Procedimento de Reconexão

No procedimento de Reconexão apresentado na Figura 4.10, inicialmente escolhe-se de forma aleatória uma solução $S_{elite} \in Repositorio$. Nas linhas 2 e 3, são definidas a solução inicial S_i, e a solução guia S_g, de acordo com a estratégia utilizada no algoritmo. Nas linhas seguintes inicializa-se S^* com a melhor solução e Φ^* com o custo relativo à melhor solução.

Na linha 6, a função $CalculaDiferencaSimetrica()$ gera o conjunto $\Delta(S_i, S_g)$ da seguinte forma. Considere cada solução $S = \{C_1, C_2, ..., C_k\}$ formada por $|K|$ atributos, cada um associado a uma demanda k. Cada atributo $C_k = \{c_k^1, c_k^2, ..., c_k^{r(k)}\}$, é o conjunto dos caminhos usados para rotear a carga l_k da demanda k. Consideradas a solução inicial $S_i = \{\overline{C_1}, \overline{C_2}, ..., \overline{C_k}\}$ e a solução guia $S_g = \{\underline{C_1}, \underline{C_2}, ..., \underline{C_k}\}$, um atributo $\overline{C_k} \in S_i$ é considerado igual a um atributo $\underline{C_k} \in S_g$, se, e somente se, todos os caminhos $c_k^h \in \overline{C_k}$ usados para rotear a demanda k, são iguais aos caminhos $c_k^h \in \underline{C_k}$. Um caminho $c_k^h \in \overline{C_k}$ é considerado igual ao caminho $c_k^h \in \underline{C_k}$, se ambos roteiam a mesma fração da carga l_k da demanda k, usando as mesmas arestas para formar sua trajetória origem-destino. Aparentemente, pode-se supor que as soluções inicial e guia terão poucos ou nenhum atributos iguais, devido ao grande números de variáveis que compõem um atributo, porém isso não se comprovou nos testes realizados, onde em média 50% dos atributos eram iguais. O conjunto $\Delta(S_i, S_g)$, representa a diferença simétrica entre as duas soluções, ou seja, contém o conjunto de movimentos m_k necessários para transformar a solução inicial S_i na solução guia

S_g. Um movimento m_k significa a ação de substituir o atributo $\overline{C_k} \in S_i$ pelo atributo $\underline{C_k} \in S_g$, na solução inicial. Dessa forma, após se aplicar todos os movimentos $m_k \in \Delta(S_i, S_g)$ na solução inicial, esta se transforma na solução guia.

Enquanto a solução inicial for diferente da solução guia, o laço entre as linhas 7 e 15 será executado. Na linha 8, m_k^* recebe o movimento $m_k \in \Delta(S_i, S_g)$ que resulta na melhor solução construída através da aplicação do movimento m_k na solução S_i. Na linha seguinte, a solução inicial é atualizada aplicando-se o movimento m_k^*, ou seja, o atributo $\overline{C_k} \in S_i$ é substituído, na solução inicial, pelo atributo $\underline{C_k} \in S_g$. O conjunto $\Delta(S_i, S_g)$ é atualizado na linha 10. A melhor solução do procedimento é atualizada nas linhas 12 e 13. O procedimento retorna a melhor solução construída no caminho entre a solução inicial e guia.

procedimento RECONEXÃO_DE_CAMINHOS($S, Repositorio$);
1 $S_{elite} \leftarrow SelecionaSolucaoElite(Repositorio)$;
2 $S_i \leftarrow DefineSolucaoInicial(S, S_{elite})$;
3 $S_g \leftarrow DefineSolucaoGuia(S, S_{elite})$;
4 $S^* \leftarrow SelecionaMelhorSolucao(S, S_{elite})$;
5 $\Phi^* \leftarrow \Phi(S^*)$;
6 $\Delta(S_i, S_g) \leftarrow CalculaDiferencaSimetrica(S_i, S_g)$;
7 **enquanto** $S_i \neq S_g$ **faça**
8 $m_k^* \leftarrow SelecionaMelhorMov(\Delta(S_i, S_g))$;
9 $S_i \leftarrow AtualizaSolucao(S_i, S_g, m_k^*)$;
10 $\Delta(S_i, S_g) \leftarrow \Delta(S_i, S_g) \backslash \{m_k^*\}$;
11 **se** $\Phi(S_i) < \Phi^*$ **então**
12 $\Phi^* \leftarrow \Phi(S_i)$;
13 $S^* \leftarrow S_i$;
14 **fim-se**;
15 **fim-enquanto**;
16 **retornar** (S^*);
fim.

Figura 4.10: Pseudocódigo do procedimento de Reconexão de caminhos do GRASP proposto

Capítulo 5

Resultados Computacionais

Neste capítulo são apresentados os resultados computacionais obtidos através da utilização da heurística GRASP na resolução do problema de roteamento com número limitado de caminhos, assim como os resultados obtidos através da utilização do procedimento GRASP associado com diferentes métodos de reconexão de caminhos desenvolvidos. Vale ressaltar, que por se tratar de um problema novo, não foram encontrados na literatura dados que nos permitissem realizar uma comparação com os resultados obtidos neste trabalho.

Os testes dividiram-se em três conjuntos. No primeiro conjunto, definem-se os métodos e os parâmetros usados na implementação do procedimento GRASP, definidos a seguir:

- Métodos para definição do tamanho e da ordenação da LRC utilizada na fase de construção.
- Métodos de busca local.
- O parâmetro β, que determina o tamanho da lista de caminhos candidatos na fase de busca local.

No segundo conjunto de testes é realizado um estudo do comportamento do algoritmo em função da variação de alguns parâmetros:

- O estudo do número máximo de caminhos utilizado R.
- O estudo do parâmetro δ, que define os pesos do atraso e do congestionamento, na combinação linear $\phi_{ij} = (1 - \delta) \cdot \phi_{ij}^a + \delta \cdot \phi_{ij}^c$.

Na terceira bateria de testes, realizaram-se testes comparativos entre o GRASP sem reconexão de caminhos e quatro variantes do GRASP com reconexão de caminhos.

5.1 Ambiente computacional

O ambiente computacional utilizado para a execução destes testes, divide-se em dois conjuntos de máquinas. O primeiro, é composto por máquinas do tipo "A", que possuem processador Pentium IV - clock 2.80GHz - cache 512KB, com 488MB de memória RAM, sistema operacional (Linux)Fedora Core 2, com Kernel Linux 2.6.8-1.521. As máquinas do segundo tipo, denominado "B", possuem processador Athlon - clock 1.4GHz - cache 64KB, com 250MB de memória RAM, sistema operacional Mandrake 10.0, com Kernel Linux 2.6.3-15. Os procedimentos foram implementados em linguagem C e compilados com o compilador gcc versão 3.3.3.

5.2 Instâncias Utilizadas

As instâncias utilizadas neste trabalho dividem-se em três grupos. O primeiro grupo de instâncias é formado por três classes de problemas usados por Fortz e Thorup [10] e Resende e Ribeiro [30].

- *AT&T Worldnet Backbone* ***(att)***: Esta classe representa uma rede real, com 90 nós, 274 arestas e 272 demandas.
- *2-level hierarchical graphs* ***(hier50a, hier100a)***: As instâncias são artificialmente geradas através de gerador GT-ITM [38], baseado no modelo de Calvert et al. [5] e Zegura et al. [39] . Nestas instâncias são utilizadas apenas dois tipos de arestas: arestas de acesso local, com capacidades menores, e arestas

de longa distância, com maiores capacidades. A capacidade das arestas de mesmo tipo são iguais.

- *Waxman* ***(wax50a)***: Esta instância é gerada artificialmente, onde os nós são uniformemente distribuídos em um quadrado de 1 unidade quadrada. A probabilidade de existir uma aresta entre os nós u e v é dado por $\eta e^{-\delta(u,v)/2\theta}$, onde η é o parâmetro usado para controlar a densidade do grafo, $\delta(u,v)$ é a distância euclidiana entre u e v, e θ é a distância máxima entre dois nós [37]. Todas as arestas possuem a mesma capacidade. A carga das demandas são diferentes para nós com diferentes níveis de atividade.

Para o segundo grupo, foram geradas quatro instâncias [27], com características muito similares às redes *frame-relay*. O último grupo é formado por oito instâncias geradas randomicamente por Duhamel e Mahey [7], baseadas em quatro grafos planares com 10, 20, 30 e 40 nós.

A tabela 5.1 apresenta as instâncias utilizadas neste trabalho. Para cada instância são apresentados seu nome, seu tipo de rede, número de nós($|V|$), número de arestas($|\mathrm{A}|$) e o número de demandas($|K|$).

5.3 Construção

Nesta primeira seção, apresentam-se os testes realizados com as quatro propostas para a fase de construção do GRASP e a calibragem do parâmetro α. Para este conjunto de testes foram utilizadas seis instâncias e o parâmetro δ é definido como $\delta = 1$, ou seja, só foi considerado a componente congestionamento na função custo. Para cada uma delas, cada um dos métodos propostos executa 1000 iterações do GRASP executando somente a fase de construção, utilizando diferentes valores para $\alpha = 0.1, 0.25, 0.50, 0.75$. Para cada instância, quatro gráficos são construídos, cada um apresentando o custo obtido para cada uma das 1000 soluções geradas por cada método. Em cada gráfico, todos os métodos utilizam o mesmo valor para α. Nas figuras 5.1, 5.2, 5.3, 5.4, 5.5 e 5.6, os dois gráficos superiores foram gerados

Instância	Tipo	$\|V\|$	$\|A\|$	$\|K\|$
ATT	AT&T Worldnet Backbone	90	274	272
hier50a	2-level hierarchical graphs	50	148	2450
hier100a	2-level hierarchical graphs	100	360	9900
wax50a	Waxman	50	230	2220
fr250	Frame-relay	60	688	250
fr500	Frame-relay	60	906	500
fr750	Frame-relay	60	996	750
fr1000	Frame-relay	60	1036	1000
T_10_20	Randômica	10	42	20
T_10_45	Randômica	10	42	45
T_20_40	Randômica	20	102	40
T_20_80	Randômica	20	98	80
T_30_60	Randômica	30	160	60
T_30_120	Randômica	30	154	120
T_40_80	Randômica	40	214	80
T_40_160	Randômica	40	212	160

Tabela 5.1: Instâncias utilizadas

com $\alpha = 0.1$ e $\alpha = 0.25$, e os dois inferiores com $\alpha = 0.50$ e $\alpha = 0.75$. Pode-se observar que o método de construção que produziu soluções de melhor qualidade foi o método *decrescente/valor*. Neste caso, os elementos que compõem a LRC da fase de construção são aqueles que pertencem a lista de demandas C, ordenada de forma decrescente em função da carga associada a cada demanda, cujo valor da carga se encontra no intervalo $[\ l_{max}\ ,\ l_{max} + \alpha(l_{min} - l_{max})\ ]$. Em alguns casos, como as instâncias att, fr250 e fr500, fica evidente que este método apresenta os menores custos. Pode-se considerar que em média este método apresenta soluções de melhor qualidade. Com o objetivo de fazer uma melhor visualização destes resultados, apresenta-se na figura 5.7 um gráfico, onde se mostra o valor médio do custo obtido nas 1000 soluções geradas por um método de construção, utilizando-se um determinado valor de α . Dessa forma, podemos confirmar o desempenho superior do método decrescente/valor, além de identificar que o valor de α que apresentou os menores custos foi 0.1, sendo este valor utilizado nos demais testes deste trabalho.

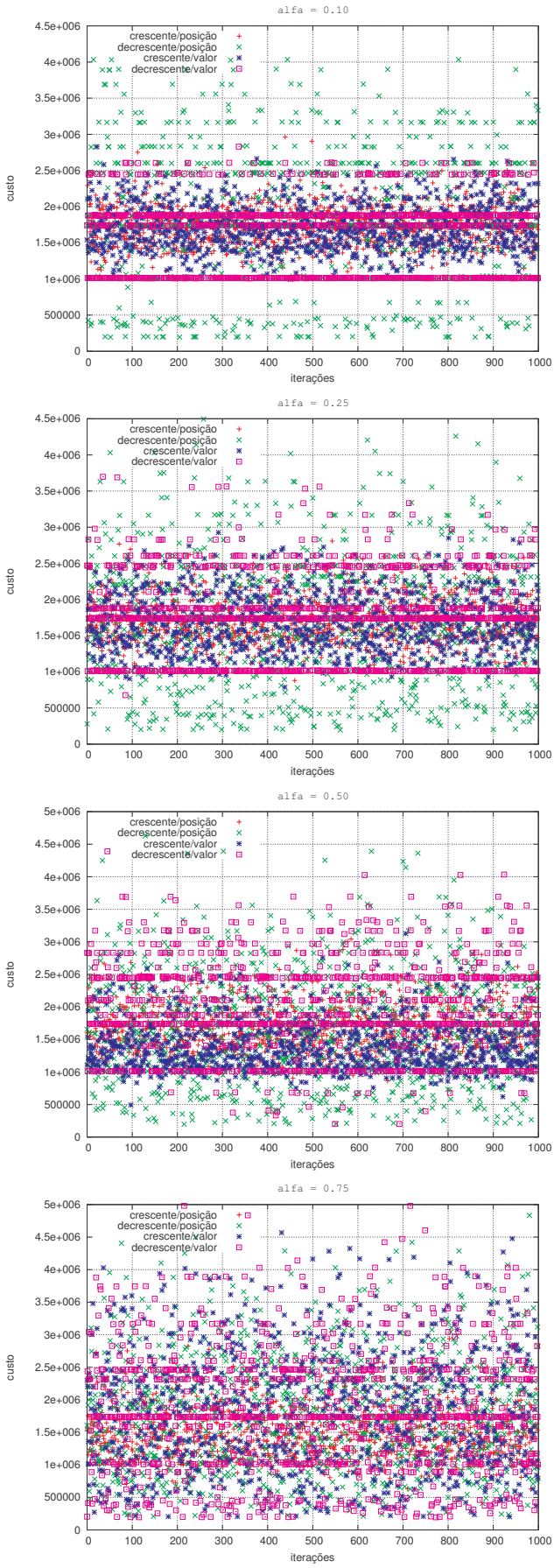

Figura 5.1: Métodos propostos para a fase de construção. Instância: att

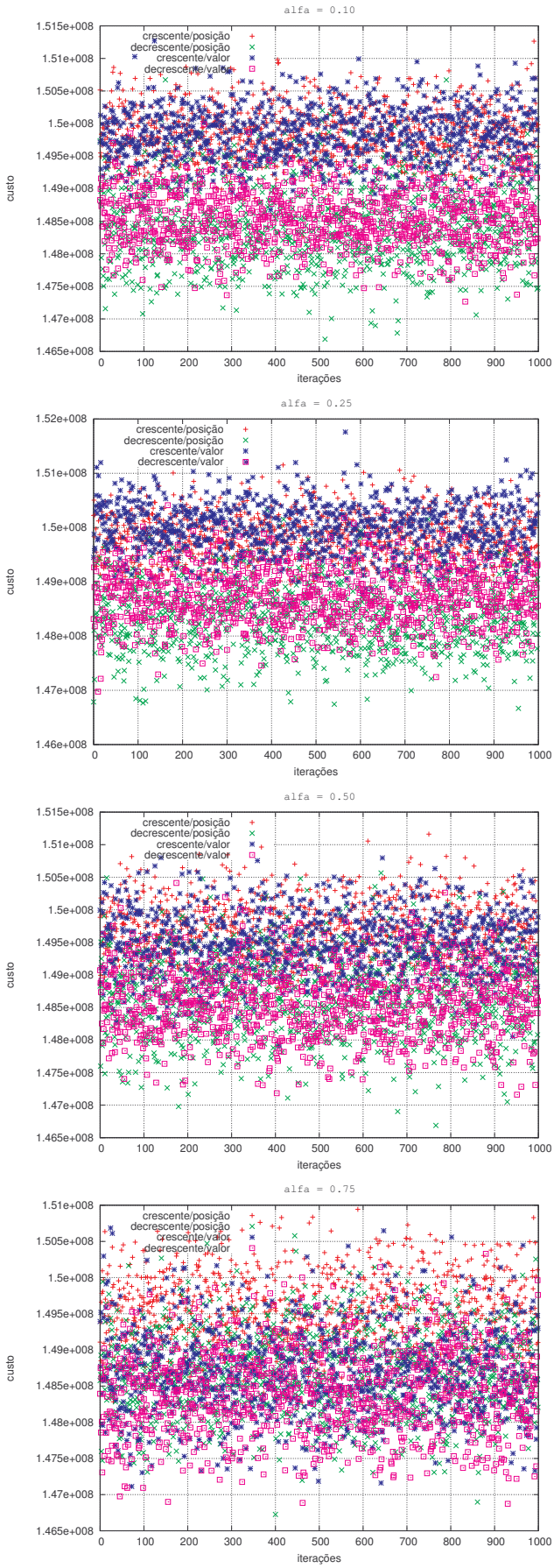

Figura 5.2: Métodos propostos para a fase de construção. Instância: hier50a

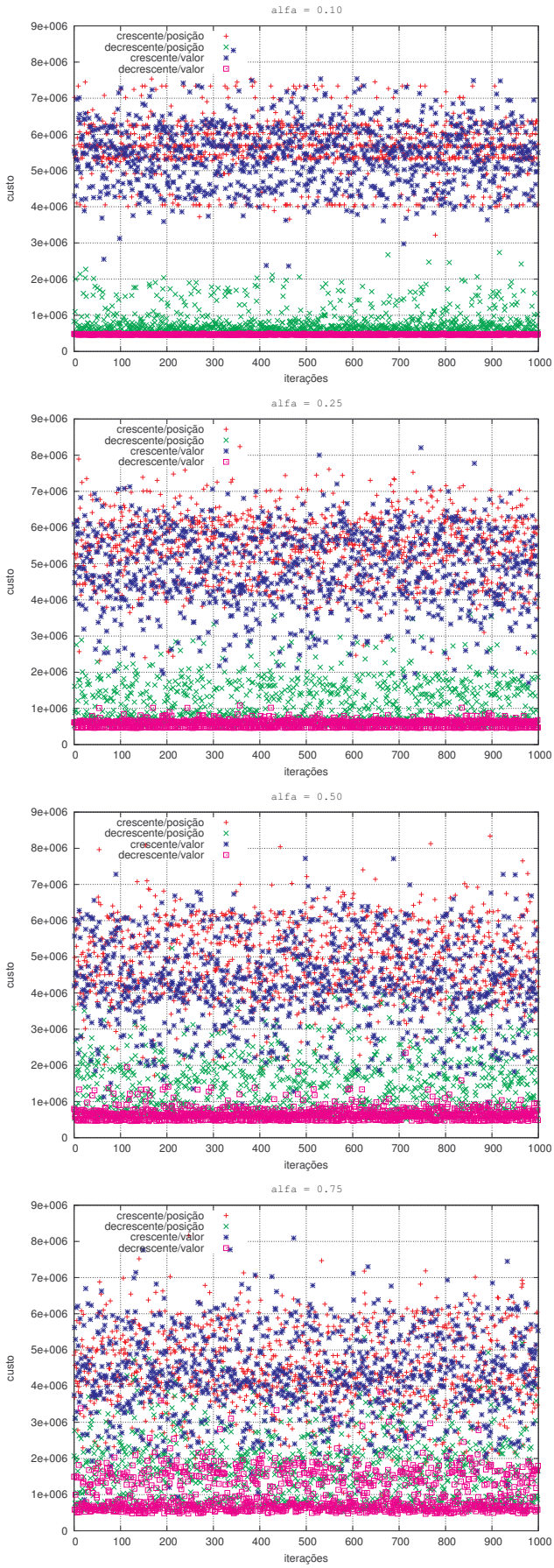

Figura 5.3: Métodos propostos para a fase de construção. Instância: fr250

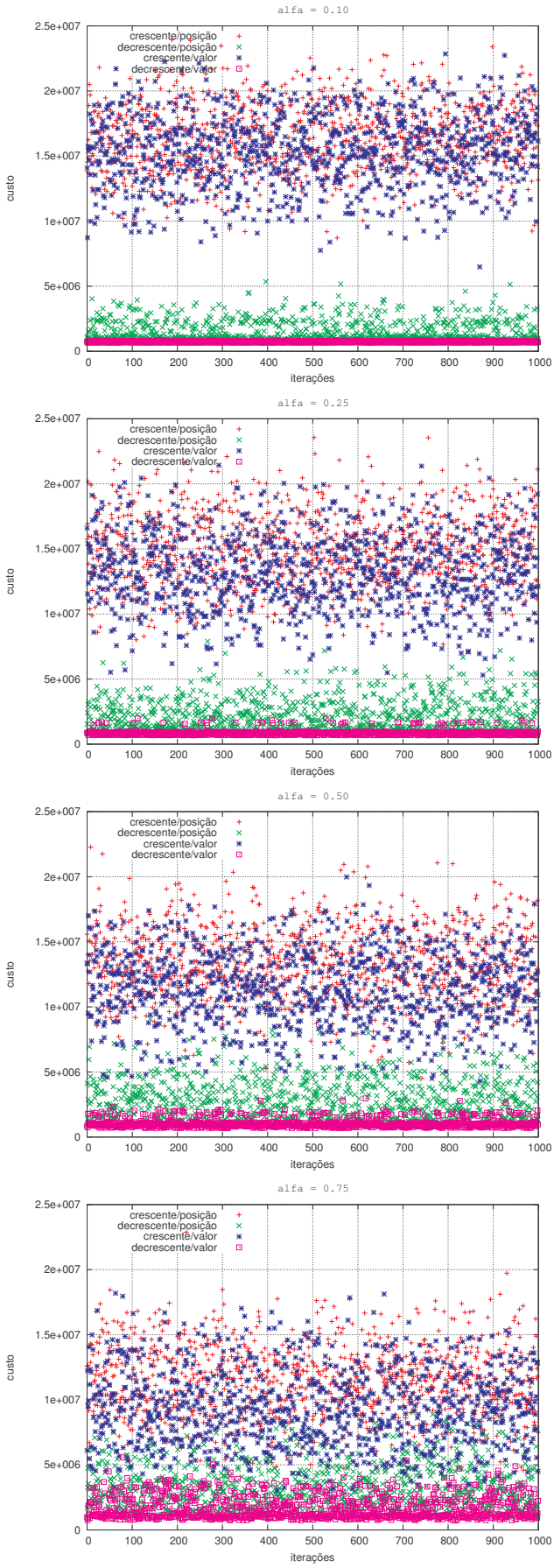

Figura 5.4: Métodos propostos para a fase de construção. Instância: fr500

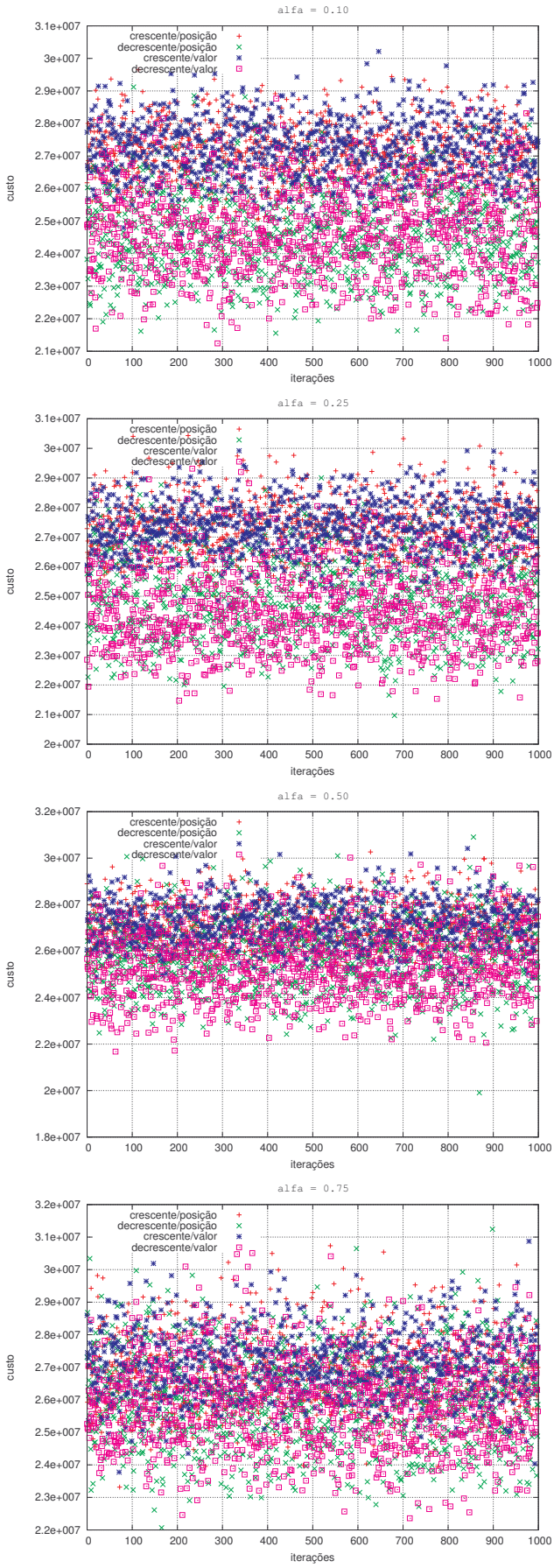

Figura 5.5: Métodos propostos para a fase de construção. Instância: T_30_60

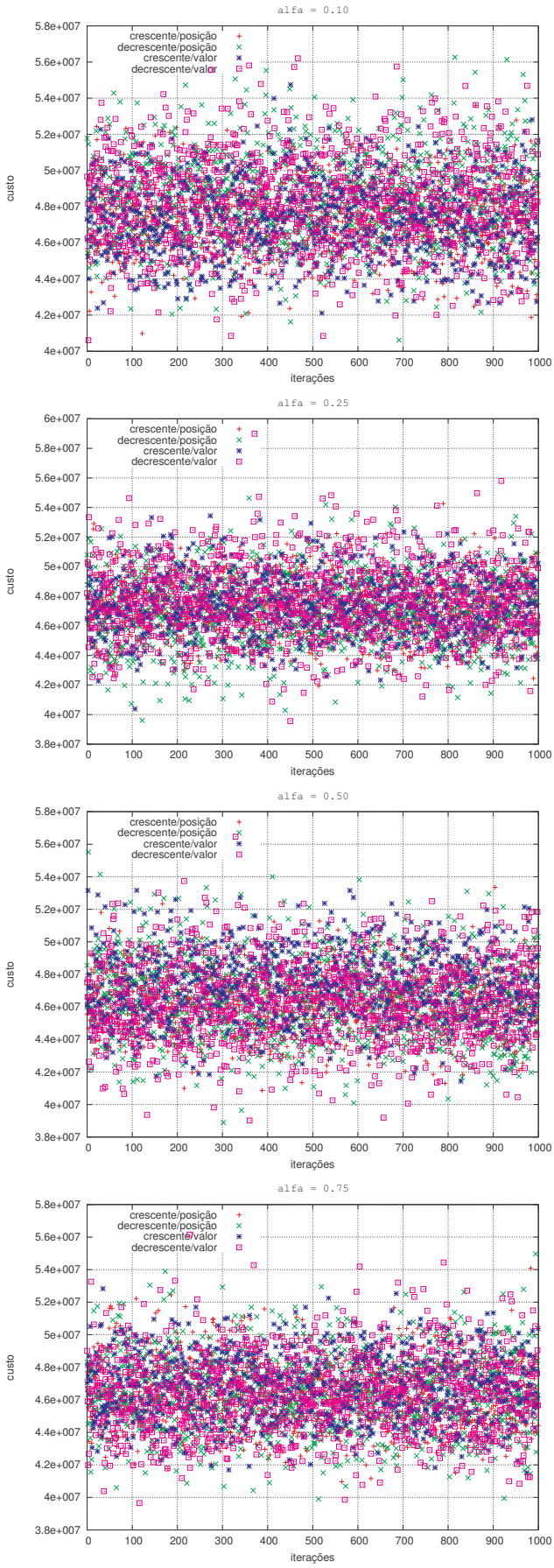

Figura 5.6: Métodos propostos para a fase de construção. Instância: T_40_80

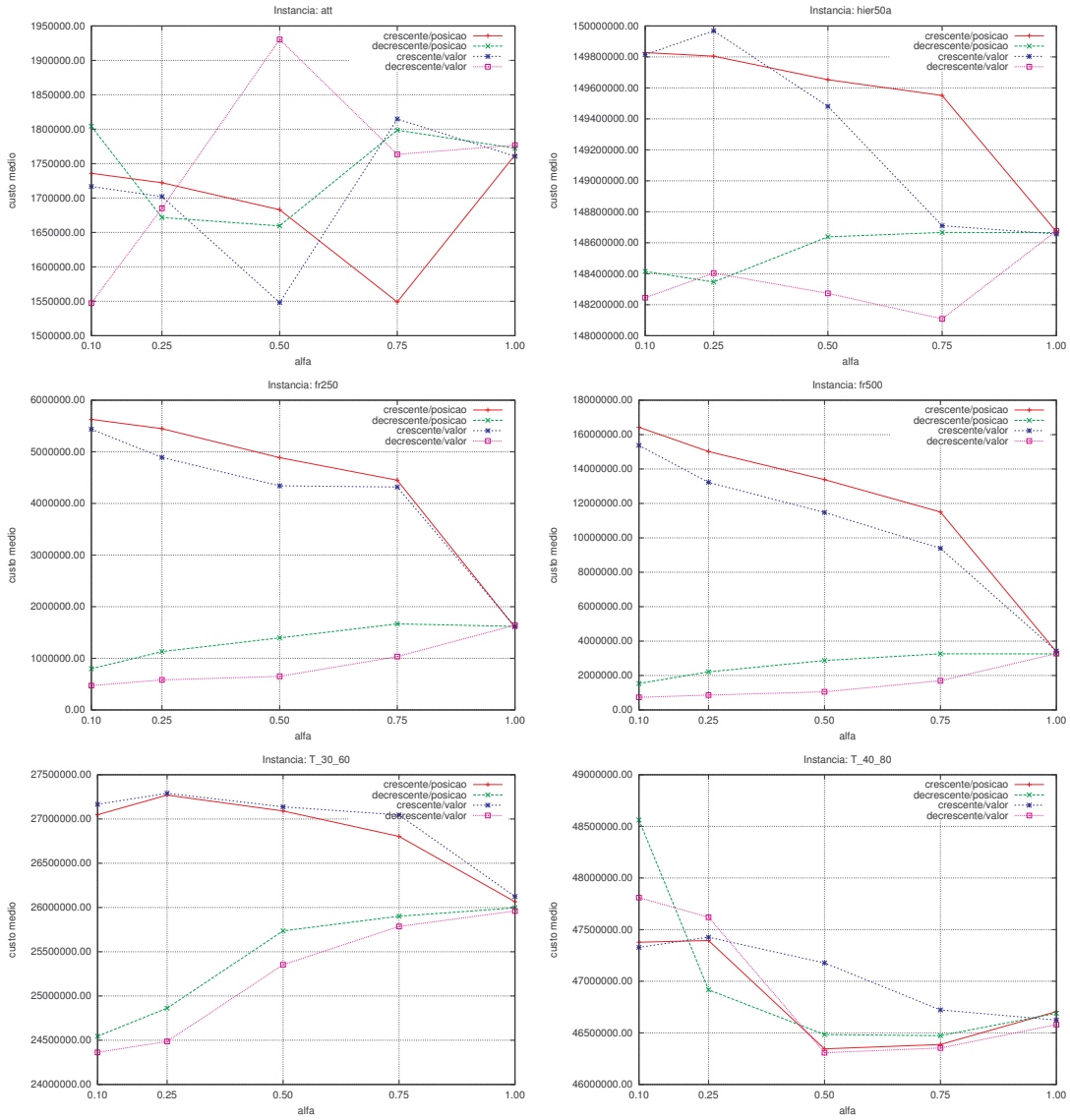

Figura 5.7: Média dos custos obtidos por cada método de acordo com a variação do parâmetro α

5.4 Busca Local

5.4.1 Definindo o método de busca local para o GRASP proposto

Esta bateria de testes foi realizada com o objetivo de determinar, entre os métodos propostos para a fase de busca local, descritos na seção 4.2, aquele que apresenta melhores resultados. Testes com os três métodos foram realizados sobre seis instâncias diferentes, com $\delta = 1$ e o número máximo de caminhos $R = 4$. Cada execução é composta por 200 iterações, onde a fase de construção gera 200 soluções iniciais, e cada método de busca local proposto é aplicado. Realizaram-se cinco execuções para cada instância, cada uma delas com uma semente e com um valor de β diferentes. Os valores utilizados para o parâmetro β nas cinco execuções foram 0.10, 0.25, 0.50, 0.75 e 1.00. Apresenta-se na tabela 5.2, o melhor, o pior e o resultado médio, obtido nas 1000 iterações, distribuídas entre as 5 execuções para cada método. Os resultados destacados em negrito representam o melhor resultado entre os três métodos. Por exemplo, o método de busca local por aresta apresentou o melhor resultado médio para a instância *att*. Observa-se que, o método de *busca local por caminhos* apresentou o melhor desempenho nas instâncias utilizadas. Em seis das oito instâncias este método obteve custos menores, tanto em relação ao melhor valor, quanto ao valor médio, e em sete das oito instâncias obteve o menor custo em relação ao pior valor encontrado. Dessa forma, este foi o método escolhido como procedimento de busca local no GRASP estudado.

Verificou-se que a busca local por aresta, por dar maior importância às arestas de alta utilização, apresentou bom desempenho nas instâncias que possuem gargalos, ou seja, arestas que se caracterizam pelo alto congestionamento. Entretanto, a eficiência deste método diminui quando a utilização das arestas é distribuída de forma mais homogênea. Ao contrário do primeiro método, a busca local por demanda apresentou melhor desempenho quando foi aplicada sobre demandas que não possuíam gargalos, porém sua eficiência se mostrou limitada, não conseguindo superar os outros dois métodos em nenhum dos testes. Isso se deve ao fato de que neste método, a carga

		att			hier50	
Metodos	melhor	medio	pior	melhor	medio	pior
aresta	**187,222.30**	**202,117.29**	**249,450.10**	143,815,202.10	144,842,329.23	145,721,153.30
demanda	189,035.20	203,003.68	251,231.00	143,720,401.30	144,467,475.13	145,568,323.30
caminho	188,113.60	202,677.84	249,541.70	**143,533,144.70**	**144,284,740.97**	**145,374,235.40**
		fr250			fr500	
Metodos	melhor	medio	pior	melhor	medio	pior
aresta	211,876.80	216,008.79	222,700.40	377,993.10	394,222.07	399,899.30
demanda	211,808.50	215,474.44	220,768.40	374,478.00	386,939.89	397,550.70
caminho	**210,208.70**	**215,108.99**	**220,544.40**	**372,853.70**	**385,522.04**	**393,246.20**
		fr750			T_20_40	
Metodos	melhor	medio	pior	melhor	medio	pior
aresta	632,100.30	649,933.23	667,991.20	3,667,912.20	3,842,529.76	4,800,112.10
demanda	628,730.40	643,822.55	663,529.70	3,756,342.80	3,870,115.16	4,927,200.50
caminhos	**626,206.30**	**641,960.14**	**660,056.80**	**3,667,503.30**	**3,834,058.68**	**4,747,457.30**
		T_30_60			T_40_80	
Metodos	melhor	medio	pior	melhor	medio	pior
aresta	19,575,677.40	19,940,588.23	**21,349,002.10**	**30,731,500.70**	**32,401,678.23**	35,097,476.20
demanda	19,939,330.20	19,997,122.47	22,013,454.90	31,642,734.20	33,546,820.42	35,478,429.00
caminho	**19,317,402.20**	**19,910,102.63**	21,653,868.60	30,939,541.70	32,821,624.18	**34,979,871.20**

Tabela 5.2: Resultados dos três diferentes métodos de busca local propostos

de uma determinada demanda deve ser sempre distribuída uniformemente entre todos os seus caminhos, prejudicando assim a melhor distribuição da carga. O terceiro método, a busca local por caminhos é mais versátil, não se restringe com o congestionamento pontual, como a busca local por aresta, nem o ignora como o método por demanda.

5.4.2 Definindo o parâmetro β

Os testes realizados nesta seção têm como objetivo identificar o melhor valor para o parâmetro β. Este parâmetro serve para definir o tamanho da LRC usada na busca local por caminhos, apresentado na seção 4.2. Para cada instância, foram realizadas 5 execuções do GRASP, cada uma com 200 iterações. Em cada uma das 5 execuções, foi atribuído um valor diferente para o parâmetro β: 0.10, 0.25, 0.50, 0.75, 1.00. Os resultados encontrados são apresentados na tabela 5.3, onde para cada instância, existem cinco linhas, uma para cada valor de β, e três colunas incluindo o melhor custo, custo médio e pior custo. Para cada instância, o melhor resultado de cada coluna encontra-se destacado em negrito.

A figura 5.8, apresenta a média das soluções nas 200 iterações obtida para cada valor de β. Observa-se que, para todas as instâncias, a melhor solução *média* foi

	att			hier50		
β	melhor	médio	pior	melhor	médio	pior
0.10	188,113.60	**202,677.84**	249,541.70	**143,533,144.70**	**144,284,740.97**	145,374,235.40
0.25	**188,108.40**	203,008.64	252,340.90	143,718,395.20	144,476,559.33	**145,307,727.10**
0.50	188,880.20	203,431.60	**248,960.90**	143,678,023.30	144,543,686.11	145,451,350.40
0.75	188,699.10	205,258.09	294,126.70	143,588,813.50	144,551,702.92	145,728,707.20
1.00	188,744.90	207,680.12	253,141.70	143,667,095.80	144,547,447.92	145,470,219.00
	fr250			fr500		
β	melhor	médio	pior	melhor	médio	pior
0.10	211,475.30	**215,108.99**	220,544.40	**372,853.70**	**385,522.04**	**393,246.20**
0.25	**210,208.70**	215,474.44	**219,800.20**	375,474.00	386,939.89	398,556.90
0.50	211,989.40	215,923.17	220,653.30	376,057.10	388,012.34	395,157.90
0.75	212,084.50	216,735.89	221,689.40	380,089.80	389,371.29	400,056.00
1.00	213,070.00	217,662.01	222,211.70	381,019.20	390,930.87	398,477.20
	fr750			T_20_40		
β	melhor	médio	pior	melhor	médio	pior
0.10	**626,206.30**	**641,960.14**	**660,056.80**	**3,667,503.30**	**3,834,058.68**	4,747,457.30
0.25	628,140.20	643,869.69	662,433.80	3,670,213.00	3,864,797.02	**4,640,977.60**
0.50	627,706.60	645,884.97	669,982.20	3,671,823.60	3,860,601.60	5,098,216.60
0.75	634,171.80	649,886.53	668,532.00	3,668,639.10	3,866,171.23	4,817,268.80
1.00	633,480.90	652,389.04	671,369.60	3,671,172.00	3,910,139.41	4,867,899.50
	T_30_60			T_40_80		
β	melhor	médio	pior	melhor	médio	pior
0.10	19,375,391.90	**19,910,102.63**	21,653,868.60	**30,939,541.70**	**32,821,624.18**	**34,979,871.20**
0.25	19,333,034.30	19,937,507.93	21,015,296.80	31,248,957.10	32,988,860.76	35,344,193.00
0.50	19,361,917.90	19,922,546.09	**20,686,107.50**	31,361,789.60	33,158,125.38	35,258,164.00
0.75	**19,317,402.20**	19,974,254.41	21,233,515.20	31,503,757.80	33,366,640.58	35,217,819.00
1.00	19,343,862.30	20,086,968.03	21,265,121.60	31,794,226.00	33,522,126.30	35,156,712.80

Tabela 5.3: Resultados da Busca Local com a variação do β

encontrada com $\beta = 0.1$. Sendo assim, define-se como 0.1 o valor de β nas execuções do GRASP.

5.5 Estudo do comportamento do algoritmo em função do número máximo de caminhos R

O estudo do número máximo de caminhos R é de fundamental importância para o desempenho do algoritmo. O objetivo dos testes desta seção é verificar o comportamento do algoritmo proposto, considerando a variação do número máximo de caminhos, em função tanto da qualidade das soluções, quanto do tempo dispensado para executá-lo. Para esta bateria de testes utilizaram-se 4 instâncias, onde para cada uma delas, foram realizadas 6 execuções do algoritmo GRASP, com 200 iterações cada. Para cada uma das seis execuções, atribuiu-se um valor diferente para R: 1,2,4,8,16,32.

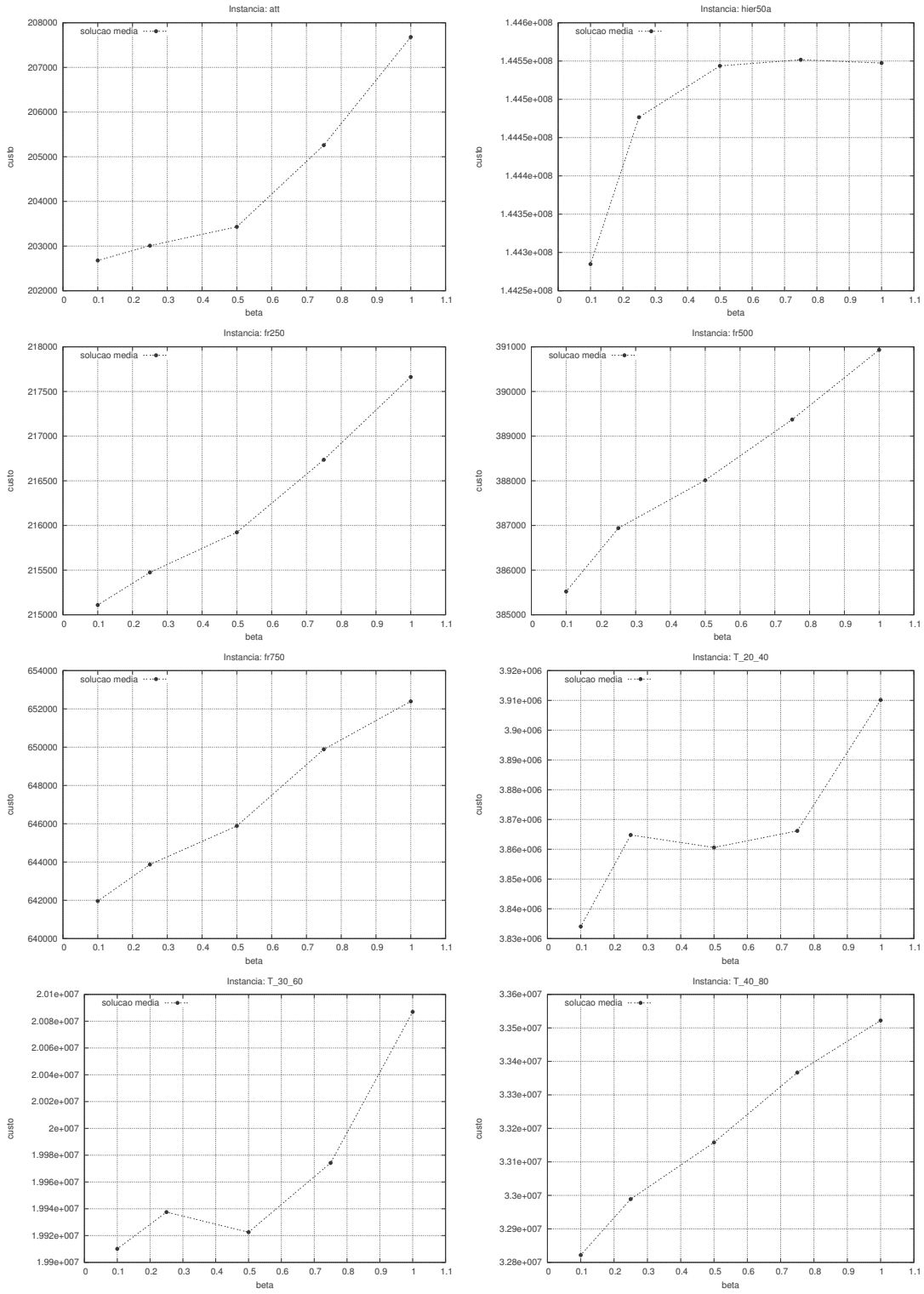

Figura 5.8: Comparação do parâmetro β

A figura 5.9 apresenta os resultados deste teste, onde cada ponto da seqüência "*custo*" representa a média dos custos encontrada nas 200 iterações, para determinado valor de R associado. A seqüência "*tempoutilizado*" representa o tempo médio de uma iteração calculado na execução das 200 iterações para cada valor de R distinto. Neste gráfico, observa-se que a razão entre o custo Φ e o número máximo de caminhos R é inversamente proporcional a razão entre o tempo utilizado e o número máximo de caminhos, ou seja, com o aumento de R, verifica-se a diminuição de Φ e um aumento do tempo utilizado. Uma observação mais criteriosa revela que, quando se utilizam valores de R maiores que 4, o aumento considerável do tempo médio gasto para encontrar determinada solução não justifica a pequena melhora na qualidade desta solução. Por este motivo, utiliza-se R igual a 4 nos próximos testes.

Outra comparação diz respeito ao número de caminhos ativos. Observa-se na figura 5.10, que o número de caminhos ativos Π, ou seja, o número total de caminhos utilizados para rotear todas as demandas é diretamente proporcional ao número máximo de caminhos R. Considerando-se que é desejável o uso do menor número de caminhos possível para se rotear um conjunto de demandas [7], verifica-se que a melhora obtida na qualidade das soluções encontradas para R com valores acima de 4 é pequena em relação ao crescimento do número de caminhos ativos Π verificado.

5.6 Estudo da variação da parâmetro δ

Nesta seção, investiga-se o impacto da variação do parâmetro δ na função objetivo $\phi_{ij} = (1 - \delta) \cdot \phi^a_{ij} + \delta \cdot \phi^c_{ij}$, onde ϕ^a_{ij} representa a componente de atraso de propagação e ϕ^c_{ij} a componente de congestionamento. Foram executadas 2000 iterações para cada uma das quatro instâncias utilizadas, usando 21 valores diferentes para δ, compreendidos entre 0.001 e 1. Nas figuras 5.11 e 5.12, para cada valor de δ são apresentados o custo e a máxima utilização da melhor solução encontrada. A máxima utilização $\Omega = max_{(i,j)}\{u_{ij}\}, \forall (i,j) \in A$, ou seja, a maior utilização u_{ij} encontrada entre todas as arestas $(i,j) \in A$. Observa-se que quanto mais próximo de 0 está o δ, o custo se reduz e a máxima utilização aumenta, por outro lado,

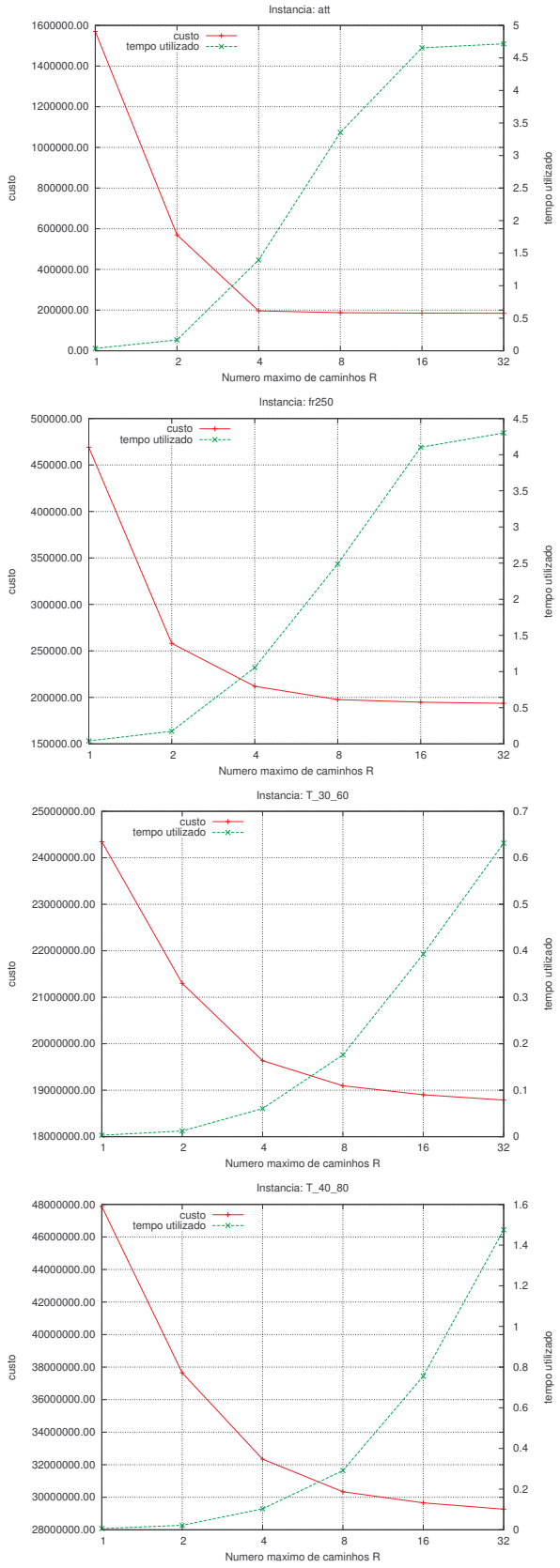

Figura 5.9: Comparação entre o custo Φ e o tempo médio em função da variação do número máximo de caminhos R

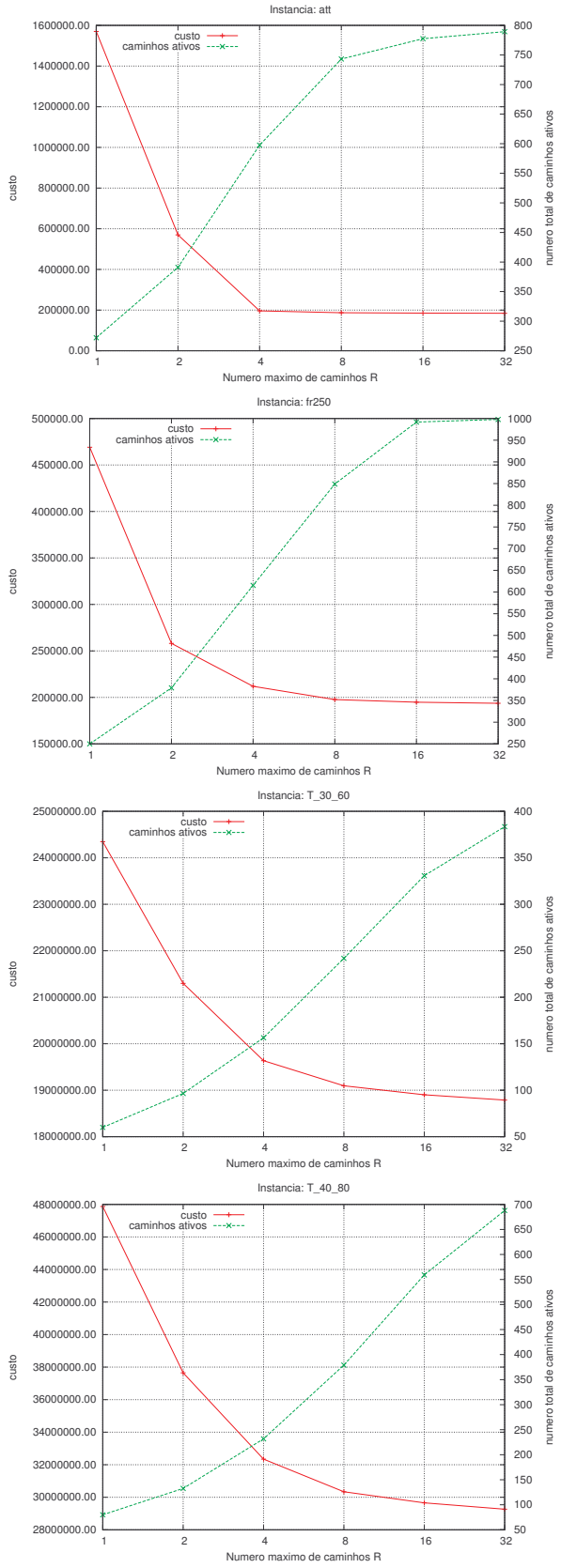

Figura 5.10: Comparação entre o custo Φ e o número de caminhos ativos Π em função da variação do número máximo de caminhos R

quanto mais δ se aproxima de 1, maior é o custo encontrado e menor é a máxima utilização. Este comportamento se deve ao fato de que quando δ aproxima-se de 0, o congestionamento é menos considerado na determinação dos pesos dos enlaces, em conseqüência da diminuição das penalidades inseridas pela função de ϕ_{ij}^{c}. Isto provoca um pior tratamento do congestionamento, resultando em arestas com maiores índices de utilização u_{ij}, por outro lado, como as penalidades da função ϕ_{ij}^{c} são responsáveis por compor boa parte dos valores do custo, este valor tende a diminuir.

Conclui-se que, o δ deverá ser definido de acordo com as características de cada problema apresentado. Como este estudo esta baseado na comparação de resultados entre os métodos apresentados, atribuiu-se o valor 1 a δ, em todos os testes, com o objetivo de simplificar esta comparação.

5.7 Comparação entre o GRASP e o GRASP com reconexão de caminhos

Neste trabalho, são comparados o GRASP puro e quatro métodos de reconexão de caminhos propostos como método de intensificação da busca local. Para todos os algoritmos de reconexão de caminhos utilizados, a política de inclusão de soluções no Repositório, caracteríza-se pela inclusão da solução corrente de menor custo em substituição à solução elite de maior custo. Estes algoritmos serão representados por:

- **GRASP**: GRASP puro sem reconexão de caminhos.
- **RCback**: GRASP com reconexão de caminhos backward.
- **RCfor**: GRASP com reconexão de caminhos forward.
- **RCt50**: GRASP com reconexão de caminhos backward truncado em 50% do caminho, ou seja, o procedimento de reconexão caminha apenas até a metade do caminho que leva até a solução guia.

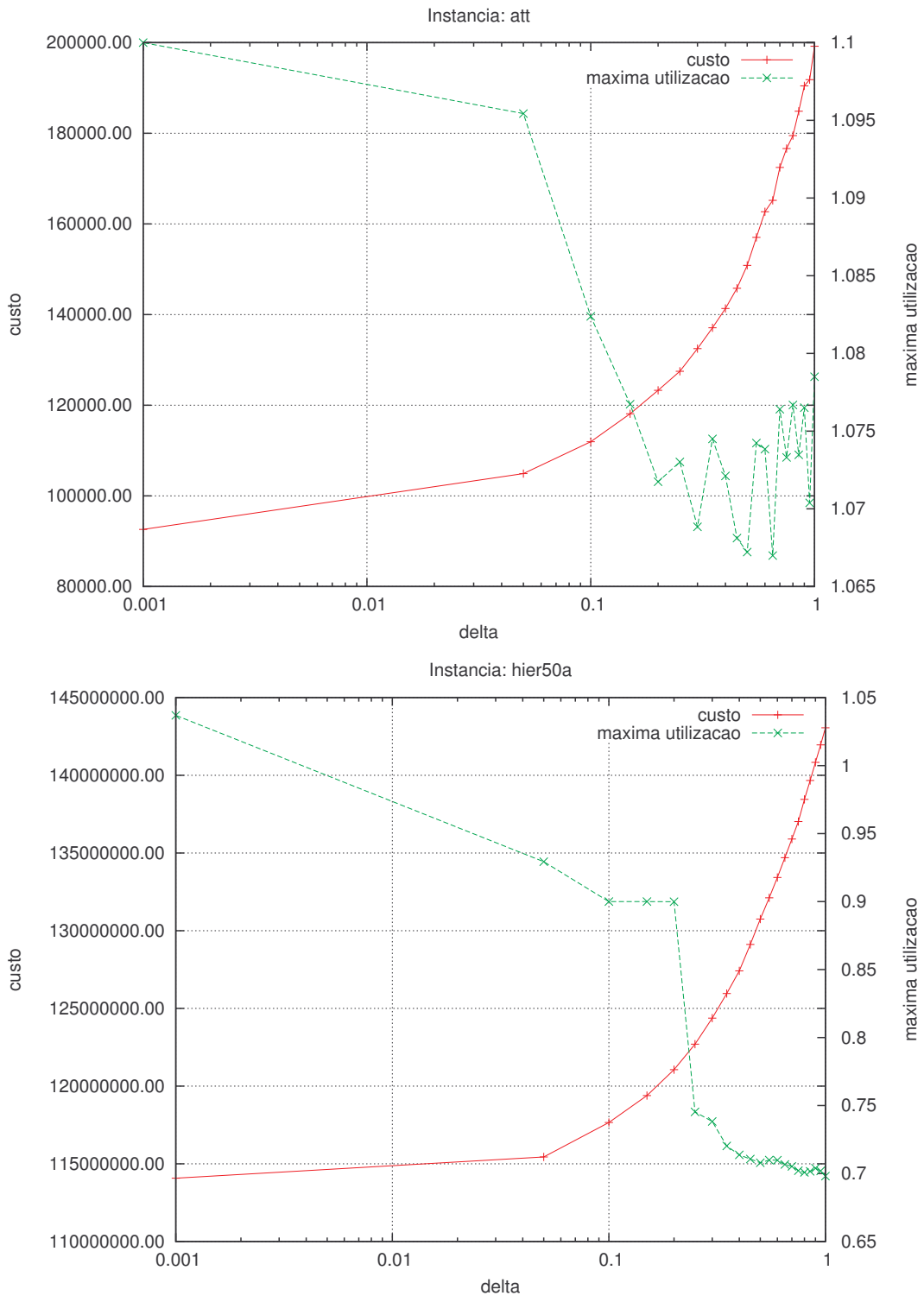

Figura 5.11: Comparação entre o custo Φ e a máxima utilização Ω em função da variação de δ (att e hier50a)

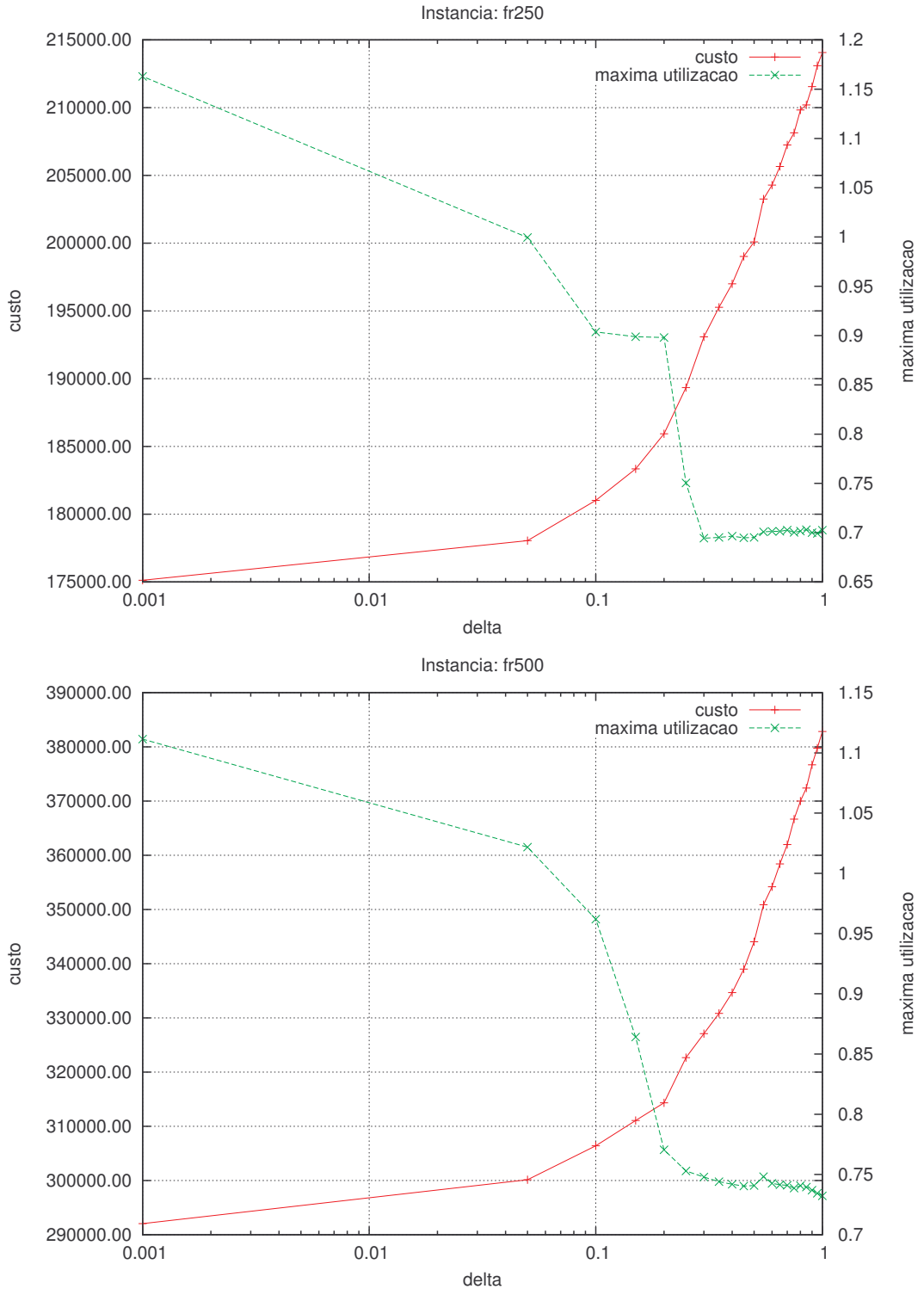

Figura 5.12: Comparação entre o custo Φ e a máxima utilização Ω em função da variação de δ (fr250 e fr500)

- **RCt20**: GRASP com reconexão de caminhos backward truncado em 20% do caminho, ou seja, o procedimento de reconexão caminha apenas até 20% do caminho que vai em direção a solução guia.

Os procedimentos de reconexão de caminhos truncado que foram utilizados no sentido backward, não foram aplicados no sentido forward, pois como a reconexão truncada explora apenas uma parte da trajetória mais próxima da solução inicial, e no método forward a solução inicial é a de pior qualidade, o procedimento tende a explorar uma vizinhança de soluções de menor qualidade.

5.7.1 Definição do tamanho do Repositório

Antes da realização dos testes efetuou-se um estudo sobre a influência do tamanho do repositório na qualidade das soluções geradas pelo algoritmo. O algoritmo *RCt20* foi utilizado, pois mostrou-se de execução mais rápida. Para cada uma das oito instâncias utilizadas, executou-se o procedimento cinco vezes, com valores diferentes para o tamanho do repositório iguais a 1,2,4,8,16. Cada execução processou 200 iterações. Na tabela 5.4, os resultados destes experimentos são apresentados, e as melhores soluções encontram-se em destaque. O repositório de tamanho 4 apresentou os melhores resultados para as instâncias testadas. Em oito instâncias obteve a melhor solução em seis delas, e em quatro, quando se considerando os valores médios, a metade dos melhores valores. Observa-se que, em nenhuma situação, os testes com tamanhos iguais a 8 e 16 apresentaram um desempenho melhor que os demais. Isso se deve ao fato de que, como o repositório possui um número grande de soluções elite, a diversidade fica grande, e isso acarreta na dificuldade das soluções convergirem para soluções de melhor qualidade. Vale ressaltar que esta calibragem do tamanho do repositório é válida para execuções com 200 iterações, uma vez que, aumentando o número de iterações, pode ser interessante aumentar a diversidade entre as soluções elite, assim como, para um número menor de iterações um repositório de tamanho menor deverá apresentar melhores resultados.

Tamanho	att		hier50a	
do Repositório (T)	melhor	médio	melhor	médio
1	185,741.10	**185,892.99**	143,434,698.00	143,503,596.46
2	187,137.80	187,428.01	142,803,214.00	143,036,611.22
4	**185,584.30**	187,010.11	**142,499,736.50**	**142,779,020.45**
8	187,267.50	188,465.26	143,254,409.90	143,337,530.04
16	189,092.60	189,856.40	143,310,241.60	143,508,610.78

Tamanho	fr250		fr500	
do Repositório (T)	melhor	médio	melhor	médio
1	205,965.80	207,316.23	359,724.80	362,925.58
2	205,752.40	**206,625.39**	359,069.30	362,857.40
4	**205,554.90**	207,055.03	**359,062.70**	**362,533.27**
8	207,555.90	209,160.90	361,543.50	366,684.25
16	209,259.40	210,408.14	363,238.50	366,346.95

Tamanho	T_10_20		T_20_40	
do Repositório (T)	melhor	médio	melhor	médio
1	**11,343.80**	**11,442.42**	3,659,043.60	3,662,227.70
2	11,487.10	11,518.79	3,658,517.40	3,665,282.79
4	11,359.70	11,485.96	**3,658,337.00**	**3,662,110.96**
8	11,435.70	11,449.61	3,663,464.20	3,665,740.69
16	11,351.60	11,481.88	3,662,325.70	3,667,780.65

Tamanho	T_30_60		T_40_80	
do Repositório (T)	melhor	médio	melhor	médio
1	19361484.90	19460818.70	30951817.50	**31084639.86**
2	19359724.30	19454249.38	**30921856.00**	31099740.50
4	**19352450.70**	**19385382.05**	30922725.90	31126593.68
8	19389033.80	19513695.64	31177391.10	31470968.91
16	19386085.50	19477431.67	31230257.40	31495352.08

Tabela 5.4: Definição do tamanho do Repositório

5.7.2 Testes Computacionais

Na primeira bateria de testes, são realizados experimentos relacionados a qualidade das soluções e o tempo computacional despendido por cada método. As tabelas 5.5 e 5.6 apresentam dados obtidos em cinco execuções de cada método, sobre cada instância. Cada execução é composta por 200 iterações. Para cada instância, são apresentados os resultados obtidos para cada método. As duas primeiras colunas definem a instância testada e o algoritmo utilizado. A terceira coluna apresenta o custo médio das soluções obtidas em todas as execuções. As próximas cinco colunas(Melhor Custo, distribuição, Π, Ω e mono) apresentam os resultados referentes à solução de menor custo encontrada. A quarta coluna mostra o menor custo encontrado nas cinco execuções, ou seja, nas 1000 iterações. Na quinta coluna (distribuição) encontram-se todas as arestas, distribuídas em classes, de acordo com sua utilização u_{ij}. Existem seis classes definidas pelos intervalos da função linear por partes ϕ^c_{ij}(Eq. 2.4). Na tabela é indicada a quantidade de arestas que pertence a cada classe. Por exemplo, na instância *att*, utilizando-se o método GRASP, existem 222 arestas na primeira classe, que representa a classe de menor congestionamento e é definida pelo intervalo $u_{ij} \in [0, 1/3)$, 41 arestas no intervalo $[1/3, 2/3)$, 8 arestas no intervalo $[2/3, 9/10)$, 3 no intervalo $[1, 11/10)$ e não existem arestas nos intervalos $[9/10, 1)$ e $[11/10, \infty)$. A coluna seguinte, apresenta o número de caminhos ativos(Π) usados na solução. A máxima utilização(Ω) apresentada na próxima coluna indica o maior valor de utilização u_{ij} encontrado nas arestas para esta solução. A oitava coluna (*mono*) apresenta o percentual das demandas que são roteadas por apenas um caminho. A última coluna indica o tempo médio gasto por uma iteração do método associado. De acordo com os resultados apresentados nesta tabela, o método que obteve melhores soluções, em relação ao custo Φ, foi o *RCback*, entretanto, observa-se que foi este o método que consumiu a maior quantidade de tempo por iteração. Em geral, boas soluções são aquelas que apresentam o menor custo, o menor Ω, o menor número de caminhos ativos, o maior percentual de *mono* e a distribuição de arestas mais concentrada nas classes iniciais, onde a utilização é menor, causando assim menos congestionamento. Porém, como está sendo empregado o uso de heurísticas, e o tempo é um fator fundamental para determinar-se a qualidade de uma solução

Instância	Método	Custo Médio	Melhor Custo	distribuição	Π	Ω	mono	Tempo(s)
	GRASP	199,344.2	188,408.5	222/41/8/0/3/0	591	1.075	40.44	1.40
	RCback	**185,385.7**	**185,244.3**	225/41/5/0/3/0	527	1.078	49.26	5.00
att	RCfor	186,648.5	186,399.4	230/35/6/0/3/0	569	1.042	46.69	4.69
	RCt50	185,982.3	185,601.9	227/38/6/0/3/0	510	1.032	53.68	4.07
	RCt20	186,410.7	186,141.5	220/46/5/0/3/0	528	1.065	52.57	2.70
	GRASP	144,325,515.8	143,450,832.9	101/43/4/0/0/0	3744	0.668	70.82	10.56
	RCback	**142,464,507.7**	**142,363,512.8**	99/43/6/0/0/0	3638	0.669	72.04	44.37
hier50a	RCfor	142,858,195.7	142,648,322.2	102/41/5/0/0/0	3825	0.674	69.67	43.19
	RCt50	142,759,558.0	142,574,133.5	100/42/6/0/0/0	3676	0.670	71.80	35.92
	RCt20	142,779,020.4	142,499,736.5	103/40/5/0/0/0	3629	0.670	73.67	23.41
	GRASP	3,128,440,545.2	3,078,103,806.6	53/76/72/26/0/3	3565	1.548	68.29	13.25
	RCback	**3,031,786,480.6**	**3,027,928,546.6**	53/77/71/26/0/3	3521	1.548	69.41	71.64
wax50a	RCfor	3,036,825,894.5	3,031,318,138.5	60/60/79/28/0/3	4114	1.548	59.05	71.34
	RCt50	3,069,535,651.2	3,032,232,705.7	54/72/79/22/0/3	3611	1.548	67.30	57.99
	RCt20	3,081,916,747.4	3,033,902,706.4	56/74/71/26/0/3	3568	1.548	69.28	35.41
	GRASP	214,956.2	210,727.8	617/70/1/0/0/0	579	0.691	38.80	1.09
	RCback	**205,614.9**	**203,509.7**	624/63/1/0/0/0	592	0.691	36.00	7.94
fr250	RCfor	208,858.1	206,209.8	607/80/1/0/0/0	620	0.691	34.40	3.38
	RCt50	207,604.4	205,427.8	625/62/1/0/0/0	608	0.691	34.00	5.91
	RCt20	207,055.0	205,243.5	622/65/1/0/0/0	582	0.691	38.80	3.37
	GRASP	385,440.3	375,358.9	748/155/3/0/0/0	1171	0.741	38.20	2.75
	RCback	355,171.4	352,334.6	750/153/3/0/0/0	1168	0.741	38.60	37.51
fr500	RCfor	376,315.1	374,483.7	717/185/4/0/0/0	1243	0.741	34.00	38.23
	RCt50	**354,958.0**	**349,393.6**	745/159/2/0/0/0	1138	0.724	36.60	29.49
	RCt20	357,659.4	352,927.8	734/170/2/0/0/0	1160	0.724	37.20	15.17
	GRASP	642,409.4	627,427.9	663/327/6/0/0/0	1739	0.741	41.20	3.19
	RCback	612,039.0	600,957.4	675/315/6/0/0/0	1746	0.741	40.67	40.97
fr750	RCfor	627,501.6	621,255.7	621/368/7/0/0/0	1887	0.733	37.47	41.11
	RCt50	609,477.4	601,077.9	663/328/5/0/0/0	1691	0.728	42.80	30.92
	RCt20	**604,656.2**	**592,789.3**	656/335/5/0/0/0	1701	0.728	40.80	16.64

Tabela 5.5: Comparação dos resultados das cinco variantes do GRASP

aproximada, deve-se considerar, o tempo que cada método utilizou para encontrar tais soluções.

Uma outra bateria de testes foi realizada, utilizando-se somente os algoritmos com reconexão de caminhos, permitindo assim, um estudo mais detalhado para se observar o comportamento destes procedimentos. Cada método foi executado uma única vez, com 1000 iterações, e a mesma semente foi utilizada para os quatro métodos, dessa forma, os resultados gerados pela busca local foram os mesmos. O objetivo principal é apresentar o comportamento de cada um dos métodos sob as mesmas soluções geradas pelo GRASP puro. As figuras 5.13, 5.14, 5.15 e 5.16,

Instância	Método	Custo Médio	Melhor Custo	distribuição	Π	Ω	mono	Tempo(s)
	GRASP	17,094.3	11,490.8	19/5/17/1/0/0	44	0.943	50.00	0.005
	RCback	11,438.1	11,392.4	19/6/16/1/0/0	46	0.943	45.00	0.007
T_10_20	RCfor	11,566.5	11,421.7	18/9/13/2/0/0	40	0.943	55.00	0.007
	RCt50	**11,411.7**	**11,362.0**	18/7/16/1/0/0	43	0.943	55.00	0.006
	RCt20	11,498.7	11,424.0	19/5/17/1/0/0	41	0.943	55.00	0.006
	GRASP	3,842,653.3	3,663,967.1	37/15/31/9/0/10	96	2.291	47.50	0.02
	RCback	3,666,577.4	**3,651,953.3**	37/16/27/12/0/10	88	2.217	55.00	0.04
T_20_40	RCfor	3,670,231.5	3,661,691.5	40/16/21/15/0/10	87	2.150	50.00	0.04
	RCt50	3,666,364.9	3,659,648.2	40/18/28/6/0/10	94	2.354	42.50	0.03
	RCt20	**3,665,090.3**	3,660,340.0	39/14/33/6/0/10	90	2.294	55.00	0.03
	GRASP	19,874,345.0	19,373,370.0	50/16/30/18/19/27	144	6.059	40.00	0.045
	RCback	19,346,305.5	**19,227,505.0**	53/25/34/12/10/26	150	6.479	40.00	0.10
T_30_60	RCfor	19,405,765.0	19,319,653.0	57/28/30/16/2/27	150	5.529	36.67	0.10
	RCt50	**19,303,744.1**	19,279,216.7	58/21/34/20/1/26	154	5.550	33.33	0.08
	RCt20	19,335,557.4	19,305,987.2	55/18/37/17/7/26	160	4.769	38.33	0.06
	GRASP	32,754,870.6	30,987,529.8	58/23/37/22/21/53	212	9.895	36.25	0.07
	RCback	30,968,544.8	30,071,266.1	70/29/27/19/18/51	244	11.197	22.50	0.25
T_40_80	RCfor	31,285,932.2	30,300,111.8	66/26/32/8/31/51	218	11.347	33.75	0.25
	RCt50	**30,846,281.8**	**30,029,789.9**	65/28/24/11/45/41	231	11.931	26.25	0.20
	RCt20	31,059,517.2	30,251,603.6	68/29/14/22/25/56	224	12.309	28.75	0.15
	GRASP	9,357,341.6	9,061,337.1	13/9/1/0/0/19	65	4.552	62.22	0.02
	RCback	**9,034,245.3**	**9,033,049.7**	13/9/1/0/0/19	65	4.552	62.22	0.02
T_10_45	RCfor	9,035,811.9	9,033,134.9	13/9/1/0/0/19	64	4.578	66.67	0.02
	RCt50	9,035,738.4	9,033,120.6	13/9/1/0/0/19	65	4.211	66.67	0.01
	RCt20	9,034,690.9	9,033,171.5	13/9/1/0/0/19	62	4.132	68.89	0.01
	GRASP	33,129,027.5	31,521,133.9	26/10/3/5/15/39	169	8.232	46.25	0.05
	RCback	**31,311,354.5**	**31,110,044.5**	25/11/3/5/15/39	162	8.360	46.25	0.08
T_20_80	RCfor	31,498,079.7	31,294,795.9	29/7/5/6/11/40	144	7.656	56.25	0.08
	RCt50	31,418,080.0	31,320,108.4	26/9/2/5/18/38	155	7.949	48.75	0.07
	RCt20	31,696,450.7	31,516,283.0	27/9/3/3/13/43	158	8.207	52.50	0.06
	GRASP	64,117,566.6	62,449,771.0	38/16/14/17/14/55	252	18.381	46.67	0.15
	RCback	**61,472,970.5**	**61,152,530.7**	40/12/18/13/19/52	259	19.036	43.33	0.32
T_30_120	RCfor	61,627,509.0	61,328,154.4	47/13/11/12/22/49	263	20.513	45.83	0.32
	RCt50	61,525,649.1	61,365,465.8	40/16/17/12/19/50	253	17.500	47.50	0.27
	RCt20	61,538,968.2	61,243,751.4	40/15/14/15/23/47	257	18.047	45.83	0.21
	GRASP	121,143,224.7	118,959,530.6	62/23/21/13/16/77	374	14.113	41.25	0.23
	RCback	118,751,614.0	118,088,023.1	62/23/21/15/15/76	373	14.126	41.25	0.72
T_40_160	RCfor	118,559,575.0	118,215,268.0	68/24/14/11/19/76	396	12.961	33.12	0.74
	RCt50	118,274,136.3	118,039,377.2	67/21/19/7/20/78	376	13.787	38.12	0.61
	RCt20	**117,980,201.6**	**117,775,073.4**	67/21/19/14/16/75	380	13.842	34.38	0.41

Tabela 5.6: Comparação dos resultados das cinco variantes do GRASP (continuação)

apresentam os resultados da execução dos quatros métodos aplicados a cada uma das instâncias. Observa-se que, em geral, o procedimento que apresenta melhores resultados, é o *RCback*, seguido do *RCt50*, e o procedimento de pior desempenho verificado foi o procedimento *RCfor*. Nesta bateria de testes, as diferentes estratégias estão sendo comparadas apenas em termos de qualidade, não considerando o tempo utilizado por cada uma.

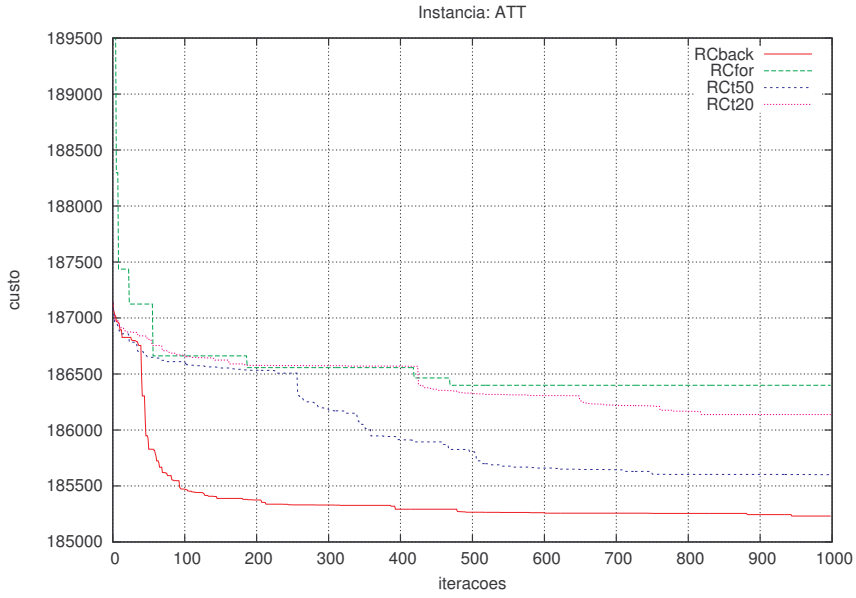

Figura 5.13: Comparação dos quatro métodos de reconexão de caminhos usando a instância att

5.7.3 Testes de Alvo

Nesta seção, utilizando-se o método de análise probabilística apresentado em [2], realiza-se um estudo de convergência de cada algoritmo(GRASP, RCback, RCfor, RCt50 e RCt20). Este método consiste em, definida uma solução alvo, se realizar 200 execuções do algoritmo, onde, em cada execução, são realizadas a quantidade necessária de iterações para que seja encontrada uma solução melhor ou igual a solução alvo. Para cada algoritmo, os tempos de execução para alcançar o alvo são armazenados em um vetor, e ordenados de forma crescente. A cada posição i deste vetor são associados o tempo de execução t_i, em segundos, e a probabilidade $p_i =$

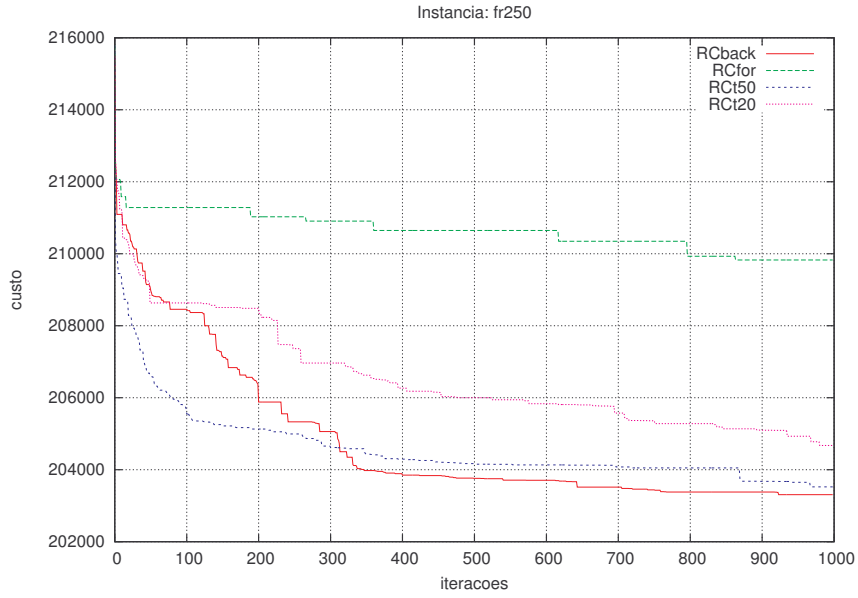

Figura 5.14: Comparação dos quatro métodos de reconexão de caminhos usando a instância fr250

$(i - 0.5)/200$. Dessa forma, os pontos $z_i = (t_i, p_i)$, com $i = 1, ..., 200$, representam a probabilidade p_i que determinado algoritmo possui de alcançar a solução alvo, em no máximo t_i segundos.

Para estes testes, foram utilizadas seis instâncias, e para cada uma, foram definidas duas soluções alvo. As *soluções alvo* foram definidas, executando-se 1000 iterações do GRASP puro e armazenando-se os resultados em um vetor com 1000 posições, ordenadas de forma crescente. Atribuiu-se ao *alvo médio*, a média dos custos encontrados e ao *alvo difícil* o melhor custo encontrado em todas as iterações. As figuras 5.17, 5.18, 5.19, 5.20, 5.21 e 5.22 apresentam os testes realizados, onde se pode observar que o algoritmo *RCt20* é o que apresenta melhor comportamento, pois se verifica que este algoritmo possui uma maior probabilidade de alcançar a solução alvo em um tempo menor que os outros algoritmos. Graficamente, isso pode ser verificado, observando-se que, as curvas referentes a este algoritmo encontram-se mais à esquerda no gráfico, ou seja, possui uma probabilidade maior de alcançar o alvo em um menor intervalo de tempo. Outra característica a ser destacada é o aumento da diferença entre os tempos obtidos pelo GRASP puro e os métodos que

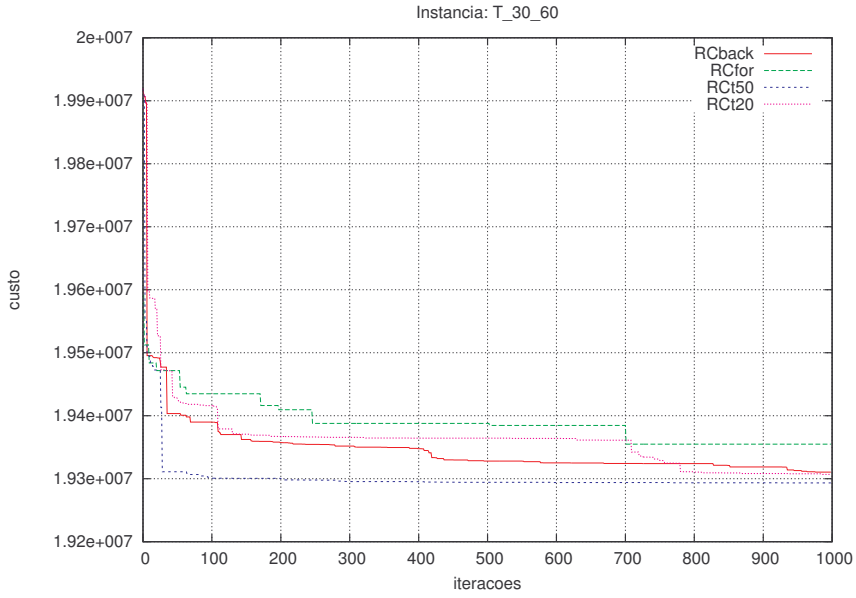

Figura 5.15: Comparação dos quatro métodos de reconexão de caminhos usando a instância T_30_60

utilizam reconexão de caminhos com o aumento da dificuldade do alvo.

5.8 Estudo do impacto de cada fase do GRASP com reconexão de caminhos

Nesta seção, o impacto obtido por cada fase do GRASP com reconexão de caminhos é examinada. Esta comparação foi realizada para avaliar a contribuição de cada fase do algoritmo para a qualidade das soluções geradas, e para a quantidade de tempo computacional dispensada.

Executaram-se 1000 iterações do algoritmo *RCt20* sobre cada uma das instâncias. Para cada instância, toma-se, como objeto de comparação, a melhor solução obtida pelo algoritmo nas 1000 iterações, e calcula-se a média das soluções geradas por cada fase do algoritmo(construção, busca local e reconexão de caminhos). Na figura 5.23 dois gráficos são apresentados. No gráfico superior, é apresentado o percentual de contribuição de cada fase do algoritmo na qualidade da solução. Para

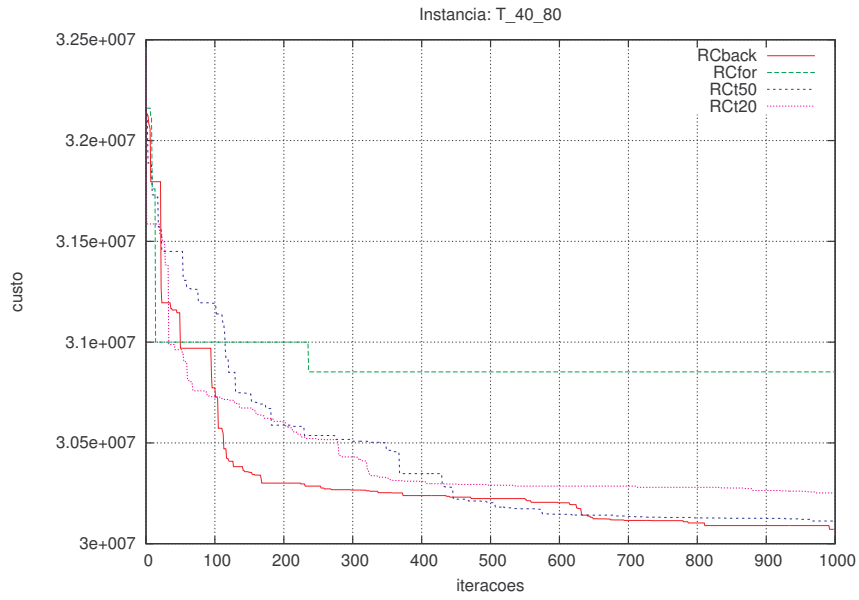

Figura 5.16: Comparação dos quatro métodos de reconexão de caminhos usando a instância T_40_80

isto, calculou-se o percentual do valor médio obtido em cada fase em relação ao menor custo obtido em todas as iterações. Por exemplo, para a instância *att*, a fase de construção contribui em média com 11,69% na qualidade da solução, a fase de busca local contribui com 80,38%, e a reconexão de caminhos com 7,23%. Observa-se também que a média das soluções obtidas pelo algoritmo para esta instância chega a 99,31% da melhor solução encontrada, indicando que, em geral, as soluções geradas estão concentradas próximas da melhor solução. No gráfico inferior, encontra-se o percentual do tempo consumido por cada fase do algoritmo, em relação ao tempo total médio. Mais uma vez usando a instância *att* como exemplo, a fase de construção consome apenas 1,51% do tempo total, enquanto a fase de busca local consome 37,28% e a reconexão de caminho 61,77%. Ambas as figuras citadas apresentam, na sua última coluna, uma média dos custos encontrados e do tempo consumido pelo algoritmo nas 16 instâncias, que pode ser utilizado, para a análise da contribuição de cada fase do algoritmo *RCt20*, para este grupo de instâncias. Observa-se que, em média, a fase de construção é responsável por 63,13% da qualidade da solução final, a fase de busca local contribui com 31,57% e a reconexão de caminhos com 5,3%. Por outro lado, o tempo médio dispensado pela fase de construção foi o menor, com

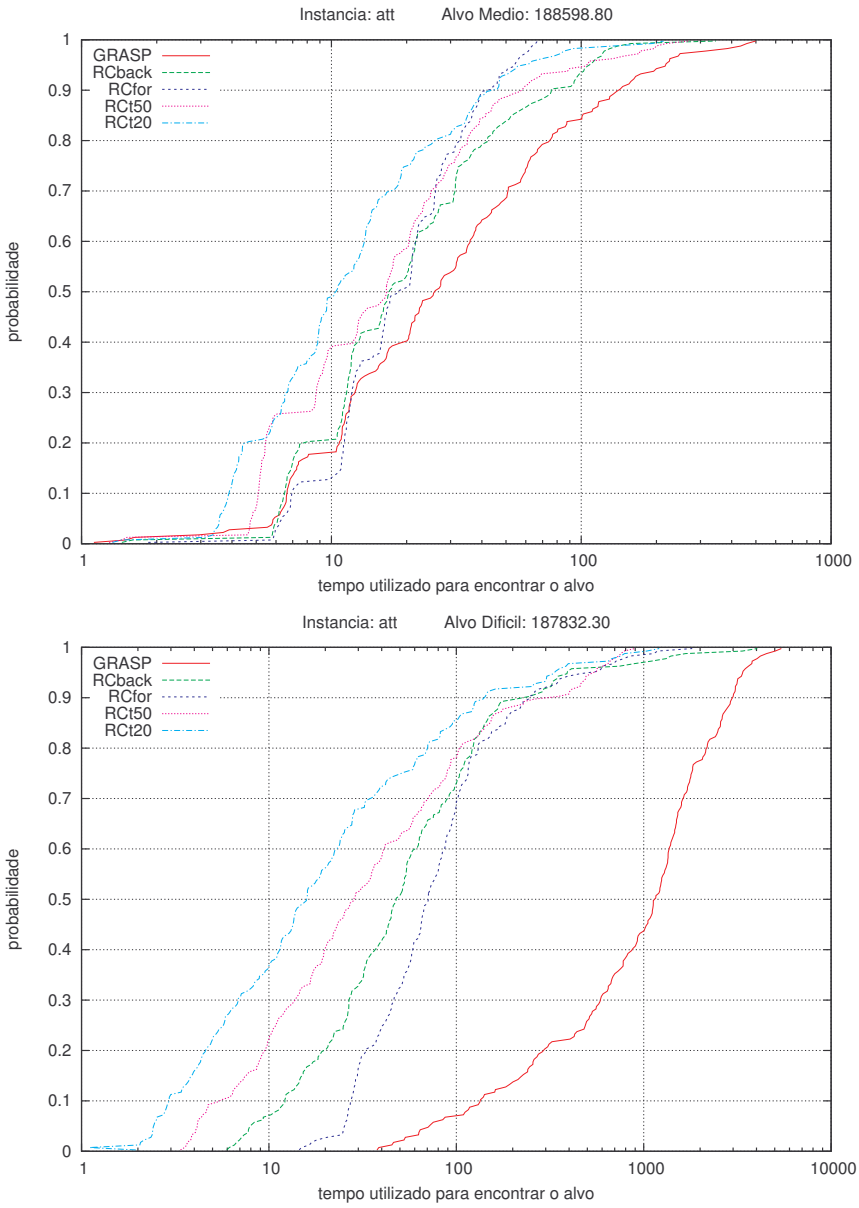

Figura 5.17: Distribuição da probabilidade de se alcançar a solução alvo em função do tempo - Instância: att

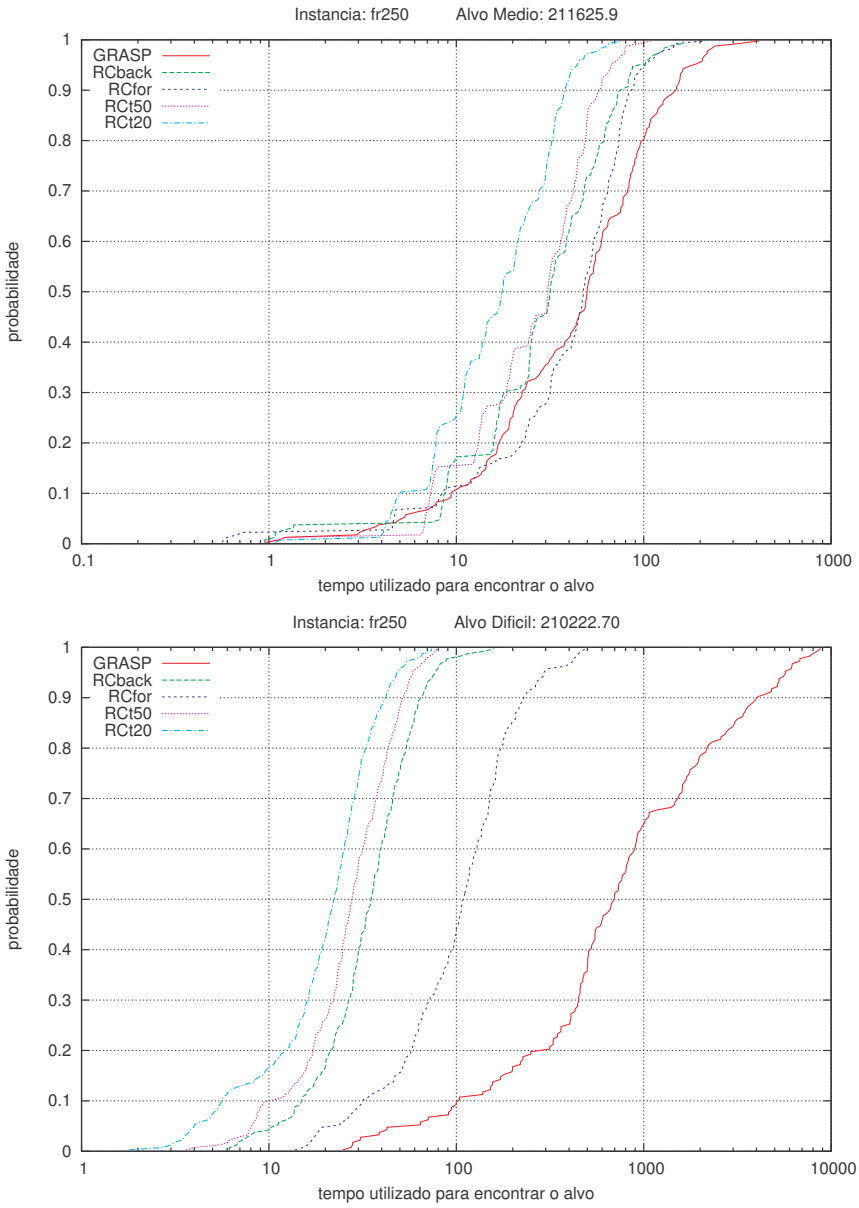

Figura 5.18: Distribuição da probabilidade de se alcançar a solução alvo em função do tempo - Instância: fr250

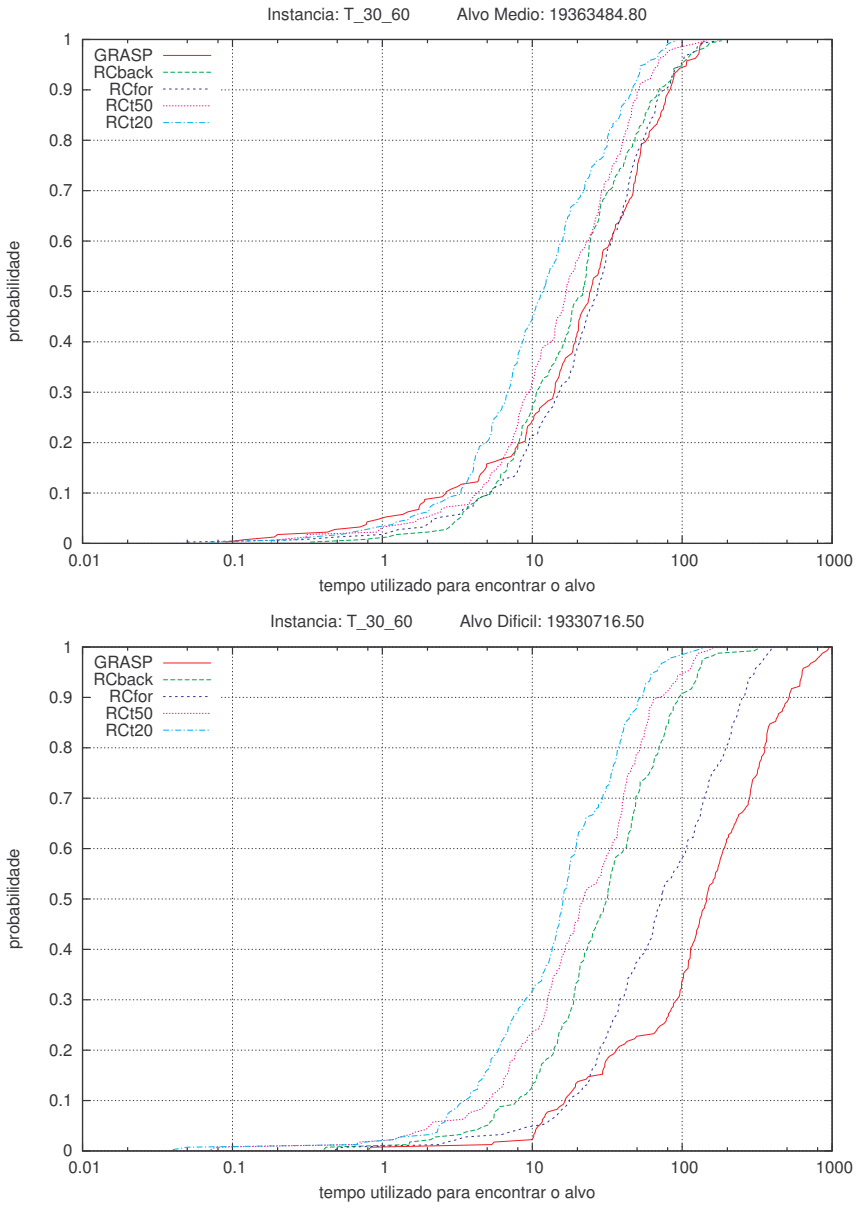

Figura 5.19: Distribuição da probabilidade de se alcançar a solução alvo em função do tempo - Instância: T_30_60

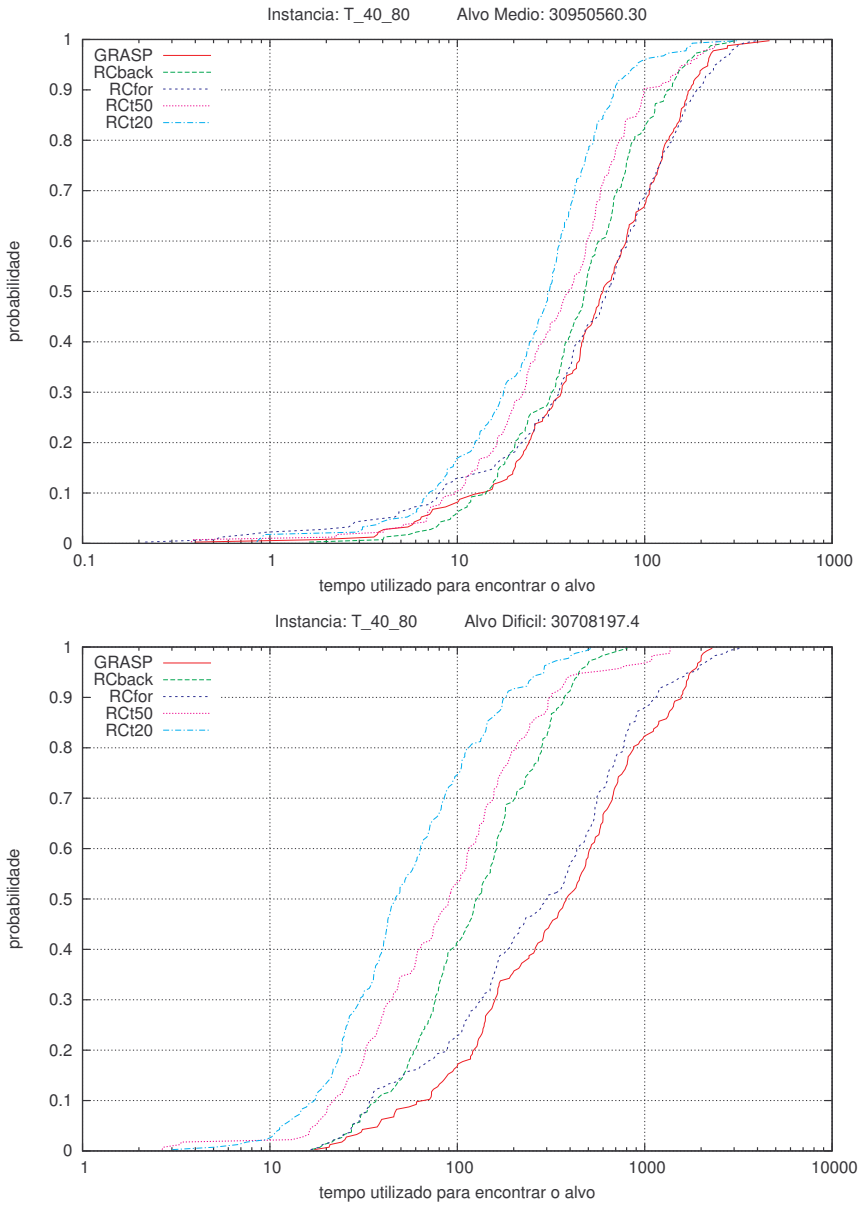

Figura 5.20: Distribuição da probabilidade de se alcançar a solução alvo em função do tempo - Instância: T_40_80

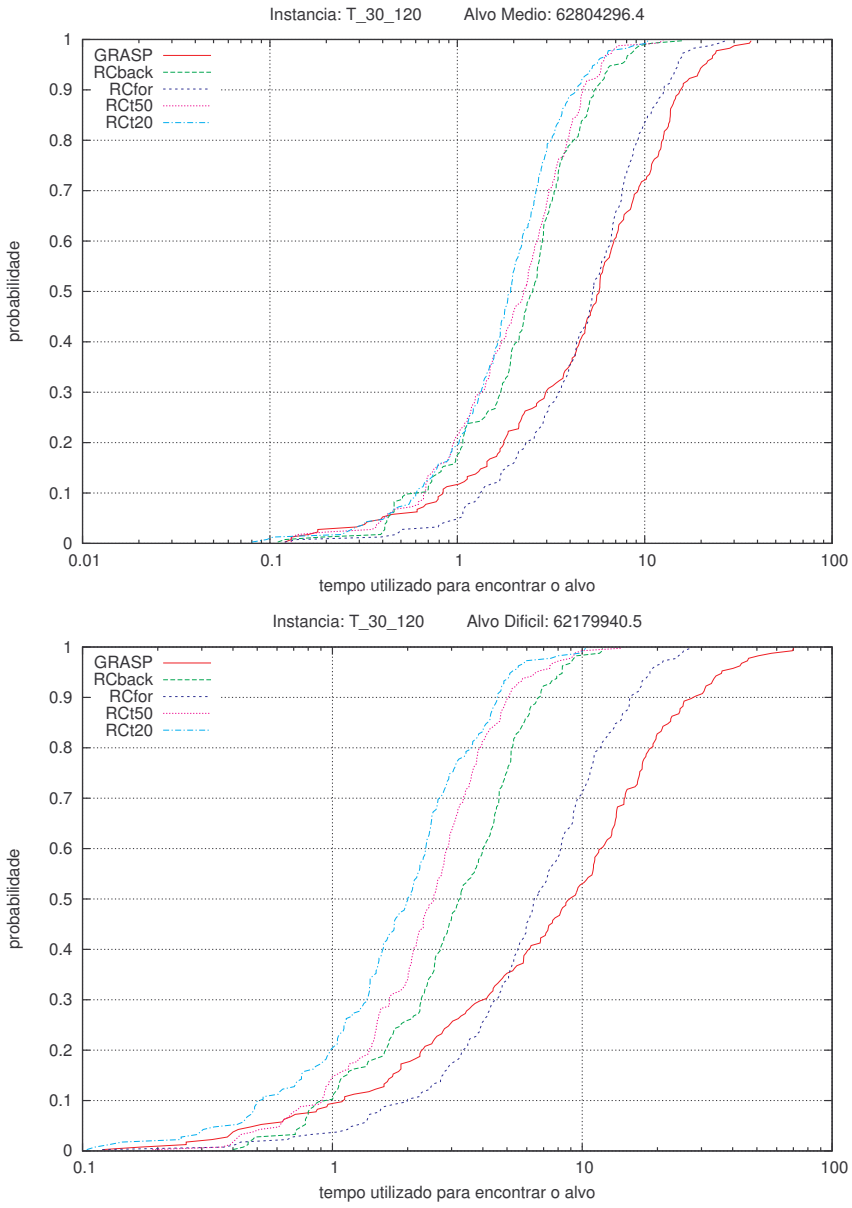

Figura 5.21: Distribuição da probabilidade de se alcançar a solução alvo em função do tempo - Instância: T_30_120

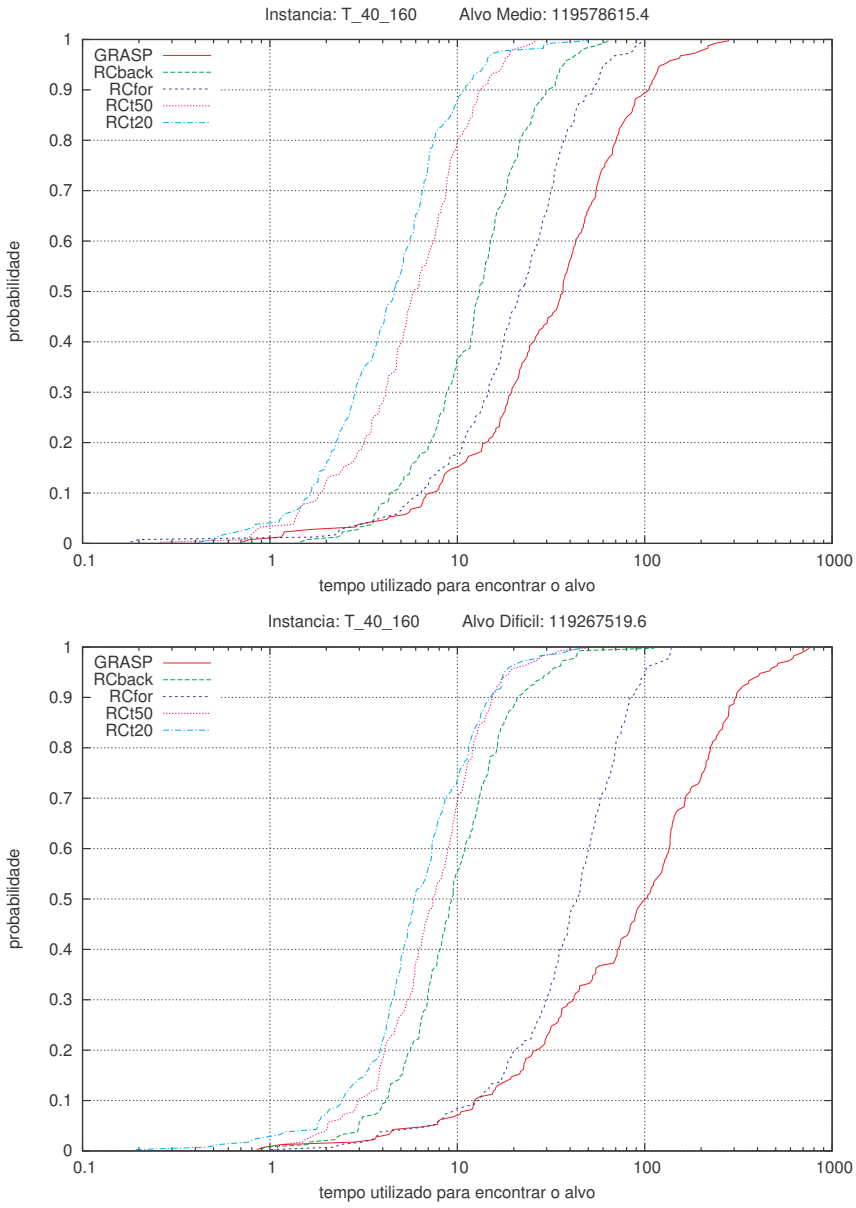

Figura 5.22: Distribuição da probabilidade de se alcançar a solução alvo em função do tempo - Instância: T_40_160

apenas 1,58% do tempo médio total, contra 50,28% utilizado pela fase de busca local e 48,14% consumidos pela reconexão de caminhos.

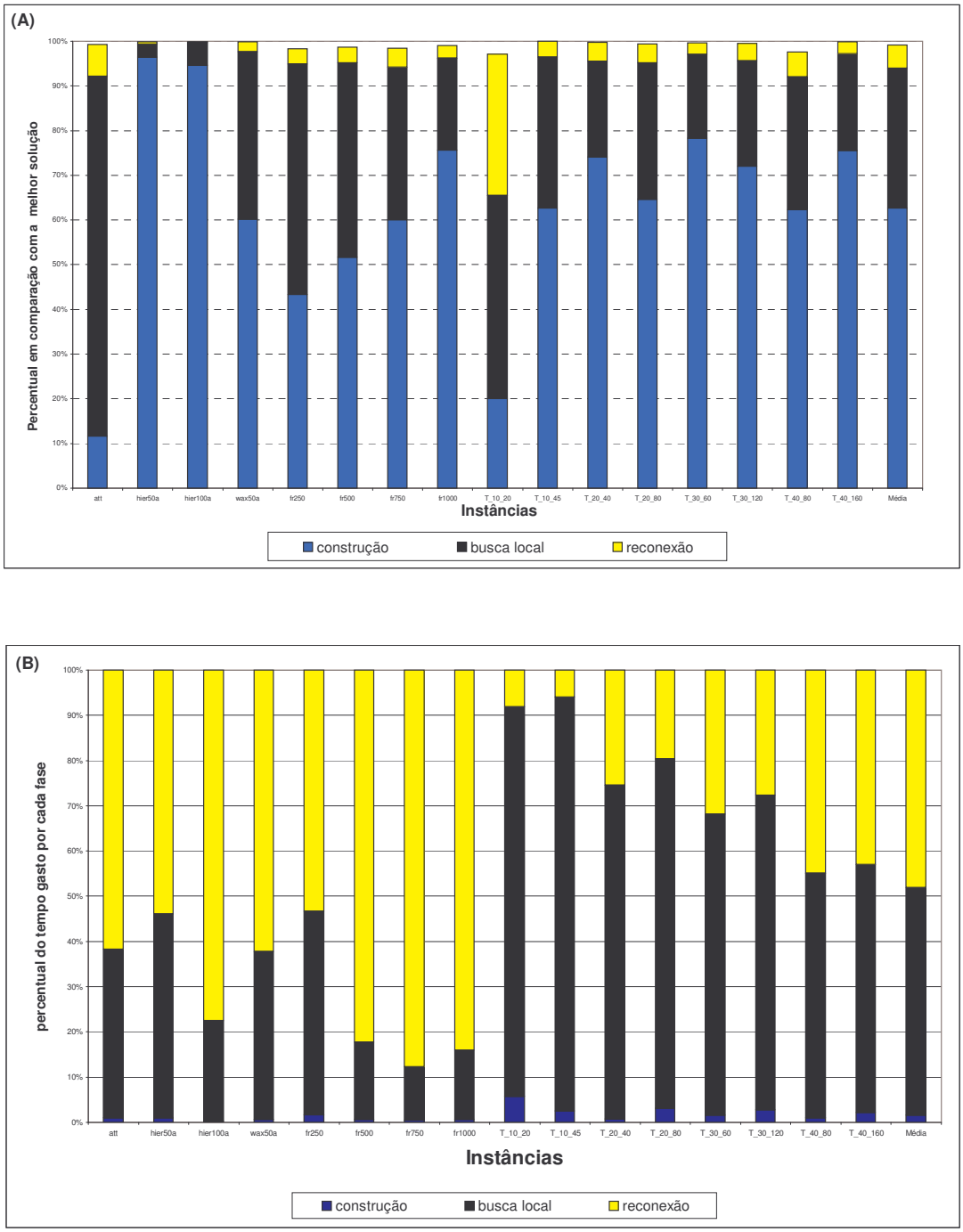

Figura 5.23: (A): Estudo do impacto de cada fase do algoritmo na qualidade da solução, em relação a melhor solução encontrada. (B): Estudo do impacto de cada fase do algoritmo no tempo médio de execução.

Capítulo 6

Conclusões

Neste trabalho, desenvolveu-se um algoritmo GRASP para o problema de roteamento em rede com número limitado de caminhos, onde o objetivo é minimizar tanto o congestionamento, quanto o atraso de propagação na rede.

No desenvolvimento da fase de construção, quatro estratégias de definição do tamanho da *LRC* foram testadas, e a que apresentou melhores resultados para as instâncias utilizadas foi escolhida. Para a fase de busca local, foram sugeridos três métodos de investigação da vizinhança: a busca local por aresta, por demanda e por caminho. Verificou-se que a busca local por caminho apresentou melhor desempenho em relação aos outros métodos. Isso se deve a sua característica de considerar tanto as arestas de alto congestionamento, quanto as arestas com níveis de congestionamento mais baixos, para definição dos caminhos a serem utilizados por cada demanda.

Um procedimento de reconexão de caminhos usado como dispositivo de intensificação da busca local também foi desenvolvido. Quatro estratégias foram propostas, e todas produziram melhoras significativas nas soluções do GRASP puro. O procedimento *RCback* apresentou o melhor desempenho em relação a qualidade das soluções geradas, dentre os procedimentos, quando todos se submeteram ao teste com um número fixo de iterações. Verificando-se qual o método que converge mais rapidamente para determinada solução alvo, os procedimentos truncados apresentaram

melhor desempenho, com o *RCt20* destacando-se. Em média, o procedimento de reconexão de caminhos *RCt20* melhorou as soluções geradas pelo GRASP em 5,22%, chegando a 31,77% na instância T_10_20. Em duas instâncias, *hier50a* e *hier100a*, a contribuição da reconexão de caminhos na melhora dos resultados obtidos foi extremamente pequena. Suspeita-se que, nesses casos, as soluções encontradas pelo GRASP sejam muito próximas da ótima.

Uma proposta para este GRASPde trabalho futuro seria o uso de alguma técnica que diminuísse o tempo computacional da fase de busca local, pois esta fase consome em média 96,96% do tempo total de execução do algoritmo, contra apenas 3,03% da fase de construção. Uma sugestão seria a utilização de um filtro de soluções, para melhorar a qualidade das soluções iniciais da busca local. Outra proposta interessante seria o desenvolvimento de uma versão do GRASP com reconexão de caminhos em um ambiente de processamento paralelo, permitindo assim obter resultados para instâncias maiores e oferecendo maior poder computacional para investigação das instâncias atuais. Algumas mudanças mais impactantes também podem ser implementadas, como construir uma solução inicial usando o número máximo de caminhos para rotear cada demanda, e na fase de busca local tentar eliminar alguns caminhos. Outra estratégia que também pode ser experimentada é o uso de VNS, combinando o uso de dois ou três métodos de busca local.

A partir da resolução deste problema, pode-se pensar em variantes que poderiam utilizar as idéias apresentadas neste trabalho. Uma restrição que poderia ser inserida no problema é que os caminhos utilizados para rotear cada demanda sejam disjuntos, pois esta restrição implica na obtenção de um sistema mais tolerante a falhas. Outra abordagem que pode ser realizada é utilizar uma formulação para o problema com dois objetivos: minimizar o congestionamento e/ou o atraso de propagação, e ao mesmo tempo, minimizar o número de caminhos utilizados por cada demanda.

Futuros trabalhos também podem inserir restrições de qualidade de serviço (QoS), como níveis de perdas e de ruídos máximos permitidos.

Referências Bibliográficas

[1] Abrahamsson, H., Ahlgren, B., Alonso, J., Andersson, A., e Kreuger, P. A multi path routing algorithm for ip networks based on flow optimisation. In *Proceedingns of QofIS'02, Zurich, Switzerlan. to appear.*, Octuber, 2002.

[2] Aiex, R., Resende, M., e Ribeiro, C. Probability distribution of solution time in GRASP: An experimental investigation. *J. of Heuristics 8*, 2002, 343–373.

[3] Aiex, R. M., Pardalos, P. M., Resende, M. G. C., e Toraldo, G. GRASP with path relinking for the three-index assignment problem. *INFORMS J. on Computing 17*, 2005, 224–247.

[4] Baier, G., Kohler, E., e Skutella, M. The k-splittable flow problem. *Algorithmica 42*, 2005, 231–248.

[5] Calvert, K., Doar, M., e Zegura, E. W. Modeling internet topology. *IEEE Communications Magazine 35*, 1997, 160–163.

[6] Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. Algoritmos: Teoria e prática. *Tradução da Segunda Edição Americana - Editora Campus*, 2002.

[7] Duhamel, C., e Mahey, P. Multicommodity flow problems with a bounded number of paths: a flow deviation approach. Relatório técnico, LIMOS/RR05-01, 2005.

[8] Duhamel, C., Vatinlen, B., Mahey, P., e Chauvet, F. Minimizing congestion with a bounded number of paths. In *Proceedings of 5eme Algotel*, Banyuls, 2003.

[9] Feo, T., e Resende, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6*, 109–133.

[10] Fortz, B., e Thorup, M. Internet traffic engineering by optimizing OSPF weights. In *IEEE INFOCOM 2000 - The Conference on Computer Communications, Tel-Aviv*, 2000, 519–528.

[11] Fortz, B., e Thorup, M. Increasing internet capacity using local search. *Computational Optimization and Applications. 29*, 2004, 13–48.

[12] Fortz, B., e Thorup, M. Robust optimization of ospf/is-is weights in a changing world. *IEEE Journal on Selected Areas in Communications 20*, Maio, 2002, 756–767.

[13] Ghamlouche, I., Crainic, T. G., e Gendreau, M. Path relinking, cycle-based neighbourhoods and capacitated multicommodity network design. *Annals of Operations Research 131*, 2004, 109–133.

[14] Glover, F. Tabu search and adaptive memory programing - advances, applications and challenges. *In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, Interfaces in Computer Science and Operations Research. Kluwer*, 1996, 1–75.

[15] Glover, F., e Laguna, M. Tabu search. *Kluwer*, 1997.

[16] Glover, F., Laguna, M., e Martí, R. Fundamentals of scatter search and path relinking. *Control and Cybernetics 39*, 2000, 653–684.

[17] Guerreiro, F., e Mancini, M. Experimental evaluation of solution approaches for the k-route maximum flow problem. *Computer & Operations Research 32*, 2005, 2453–2478.

[18] Kleinberg, J. Approximation algorithms for disjoint paths problem. *Tese de Doutorado. Massachusetts Institute of Technology*, 2003.

[19] Kurose, J. F., e Ross, K. W. Redes de computadores e a internet: uma nova anbordagem. *Tradução da Primeira Edição - São Paulo, Addison Wesley*, 2003.

[20] Mahey, P., Ouorou, A., LeBlanc, L., e Chifflet, J. A new proximal decomposition algorithm for routing in telecommunication networks. *Networks 31*, 1998, 227–238.

[21] Mahey, P., Ouorou, A., e Vial, J.-P. A survey of algorithms for convex multicommodity flow problems. *Management Science 46*, 2000, 126–147.

[22] Martí, R., e Laguna, M. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing 11*, 1999, 44–52.

[23] Martens, M., e Skutella, M. Flows on few paths: Algorithms and lower bounds. In *Proceedings of the 12th Annual European Symposium on Algorithms*, 2004, 520–531.

[24] Martins, S. L., Ochi, L. S., e Silva, G. C. Experimental comparison of greedy randomized adaptative search procedures for the maximum diversity problem. *Third Internacional Workshop on Efficient and Experimental Algorithms. Lecture Notes in Computer Science 3059*, 2005, 498–512.

[25] Pitsoulis, L. S., e Resende, M. G. C. Greedy randomized adaptive search procedures. In *Handbook of Applied Optimization*, P. M. Pardalos and M. G. C. Resende, Eds. Oxford University Press, 2002, 168–183.

[26] Prais, M., e Ribeiro, C. C. Reactive GRASP: An application to a matrix decomposition problem in tdma traffic assignment. *INFORMS Journal on Computing 12*, 2000, 164–176.

[27] Resende, L. I. P., e Resende, M. G. C. A GRASP for frame relay permanent virtual circuit routing. In *Proceedings of the III Metaheuristics International Conference (MIC99)*, 1999, P. Hansen and C. Ribeiro, Eds., 397–401.

[28] Resende, M., Pitsoulis, L., e Pardalos, P. FORTRAN subroutines for computing approximate solutions of weighted MAX-SAT problems using GRASP. *Discrete Applied Mathematics 100*, 2000, 95–113.

[29] Resende, M. G. C., e Ribeiro, C. C. Greedy randomized adaptive search procedures. In *Handbook of Metaheuristics*, F. Glover and G. Kochenberger, Eds. Kluwer Academic Publishers, 2002, 219–249.

[30] Resende, M. G. C., e Ribeiro, C. C. A GRASP with path-relinking for private virtual circuit routing. *Networks*, 1, 2003, 104–114.

[31] Resende, M. G. C., e Ribeiro, C. C. GRASP and path-relinking: Recent advances and applications. In *Metaheuristics: Progress as Real Problem Solvers*, 2005, T. Ibaraki, K. Nonobe, and M. Yagiura, Eds., 29 – 63.

[32] Resende, M. G. C., e Werneck, R. F. A GRASP with path-relinking for the p-median problem. Relatório técnico, Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ, 2002. TD-5E53XL.

[33] Rosseti, I. C. M. Heurísticas para o problema de síntese de redes a 2-caminhos. *Tese de Doutorado. PUC-RIO - Pontifícia Universidade Católica do Rio de Janeiro*, 2003.

[34] Shahrokhi, F., e Matula, D. W. The maximum concurrent flow problem. *Journal of the Computing Machinery 37*, 1990, 318–334.

[35] Srivastava, S., Agrawal, G., Medhi, D., e Pioro, M. Determining link weight systems under various objectives for ospf networks using lagrangian relaxation-based approach. In *IEEE e-Transactions on Network and Service Management*, Novembro, 2005, vol. 2.

[36] Thottan, M., Mirrokni, V. S., Uzunalioglu, H., e Paul, S. A simple polynomial time framework for reduced path decomposition in multi-path routing. In *INFOCOM*, 2004.

[37] Waxman, B. M. Routing of multipoint connections. *IEEE J. Selected Areas in Communications 6*, 1998, 1617–1622.

[38] Zegura, E. W. *GT-ITM*: Georgia tech internetwork topology models (software). Relatório técnico, Georgia Institute of Technology, www.cc.gatech.edu/fac/Ellen.Zegura/gt-itm/gt-itm.tar.gz, 1996.

[39] Zegura, E. W., Calvert, K. L., e Bhattacharjee, S. How to model an internetwork. In *IEEE Infocom*, San Francisco, CA, March 1996, vol. 2, IEEE, 594–602.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo