Universidade Federal Fluminense

LUANA SABATHA DE SOUZA PEREIRA

Heurísticas para o Problema de Transmissão Multiponto com Minimização de Energia para Redes de Sensores

NITERÓI

2006

LUANA SABATHA DE SOUZA PEREIRA

Heurísticas para o Problema de Transmissão Multiponto com Minimização de Energia para Redes de Sensores

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

Orientador:
Celso da Cruz Carneiro Ribeiro

Universidade Federal Fluminense

NITERÓI

2006

Heurísticas para o Problema de Transmissão Multiponto com Minimização de Energia para Redes de Sensores

Luana Sabatha de Souza Pereira

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

Aprovada por:

Prof. Celso da Cruz Carneiro Ribeiro, D. Sc. / IC-UFF
(Presidente)

Prof. Geraldo Robson Mateus, Ph. D. / UFMG

Prof. Luiz Satoru Ochi, D. Sc. / IC-UFF

Dr. Marcelo Prais, D. Sc. / ONS

Prof[a]. Simone de Lima Martins, D. Sc. / IC-UFF

Niterói, 14 de Agosto de 2006.

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

P 436 Pereira, Luana Sabatha de Souza

Heurísticas para o problema de transmissão multiponto com minimização de energia para redes de sensores / Luana Sabatha de Souza Pereira. - Niterói, RJ : [s.n.],2006.

106 f.

Orientador - Celso da Cruz Carneiro Ribeiro.

Dissertação (Mestrado em Ciência da Computação) - Universidade Federal Fluminense, 2006.

1. Heurística. 2. Otimização combinatória (computação).
3. Algoritmos. 4. Metaheurísticas-GRASP 5. GRASP. I. Título.

CDD 005.136

Ao meu marido, minha mãe e amigos.

Agradecimentos

Agradeço a Deus por iluminar o meu caminho, me amparar nos momentos de dúvida e desânimo e proteger das adversidades da vida.

À minha mãe por ter me ensinado a ser uma pessoa correta, persistente e batalhadora, sempre lutando para alcançar os objetivos.

Aos colegas da UFF, em especial Caroline Thennecy pelo apoio prestado no aprendizado de Programação, indispensável para a conclusão do mestrado.

Aos amigos da GPO2 pela companhia diária e pelo incentivo a buscar o aperfeiçoamento pessoal e profissional.

Ao meu marido pelo companheirismo, paciência e compreensão, me apoiando e incentivando a jamais desistir.

Por fim agradeço a meu orientador Celso Ribeiro pelo ensino e motivação transmitidos durante todo o mestrado e ao restante dos membros da banca examinadora pela presença e contribuições finais ao trabalho.

Resumo

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

Heurísticas para o Problema de Transmissão Multiponto com Minimização de Energia para Redes de Sensores
Luana Sabatha de Souza Pereira
Agosto/2006

Orientador: Celso da Cruz Carneiro Ribeiro
Programa de Pós-Graduação em Computação

O ambiente em redes de sensores sem fio ad hoc apresenta desafios para o estudo de problemas de transmissão multiponto, em que um nó fonte transmite para todos os outros da rede. A limitação de energia das baterias é uma questão crítica neste ambiente pois os nós das redes de sensores são alimentados por baterias, que têm tempo de vida limitado.

O Problema de Transmissão Multiponto com Minimização de Energia para Redes de Sensores consiste de uma rede ad hoc na qual deseja-se estabelecer a comunicação de um nó fonte para todos os outros nós da rede, de forma que todos os nós sejam alcançados e que a potência total de transmissão seja minimizada. Este trabalho cita as principais aplicações das redes de sensores sem fio, bem como introduz e avalia algoritmos para construção de árvores nessas redes.

Propõe-se neste trabalho uma nova heurística, baseada em metaheurísticas, combinando as características do algoritmo GRASP com as da técnica de reconexão por caminhos, para encontrar soluções próximas do ótimo para o problema, explorando a natureza da transmissão em ambientes de comunicação sem fio ad hoc e tratando a necessidade da operação eficiente de energia. O algoritmo proposto apresenta resultados melhores do que os apresentados na literatura, tanto em termos de qualidade da solução quanto em termos de tempo de processamento.

Abstract

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

Heuristics for the Minimum Power Broadcast Problem in Wireless Sensors Networks
Luana Sabatha de Souza Pereira
August/2006

Advisor: Celso da Cruz Carneiro Ribeiro
Department: Computer Science

The wireless sensor network enviroment presents chalenges to the study of *broadcasting* (one source node broadcasts to all others nodes in the network) problems. Batteries's energy limitation is a critical issue in this enviroment, since nodes are powered by batteries with a limited lifetime.

The Minimum Power Broadcast Problem in Wireless Sensors Networks consists in establishing communication from a source node to all other nodes in ad hoc network in such a way that all the nodes are connected and the total broadcast power is minimized. This work presents wireless sensors networks applications, as well as introduces and evaluates algorithms for tree construction in these networks.

A new heuristic is proposed in this work, combining the characteristics of GRASP and path relinking, to find near-optimal solutions to this problem, that exploits the broadcast nature of the wireless ad hoc communication environment and addresses the need for energy-efficient operation. The proposed algorithm obtains better solutions and is less time consuming than the algorithms found in the literature.

Palavras-chave

1. Redes sem Fio
2. Redes Ad Hoc
3. Redes de sensores
4. Transmissão Multiponto
5. Algoritmos
6. Otimização Combinatória
7. Metaheurísticas
8. GRASP com Path Relinking
9. Minimum Energy Broadcast Problem

Glossário

MPB	:	Minimum Power Broadcast
WMA	:	Wireless Multicast Advantage
BLU	:	Broadcast Least-Unicast Cost
SPF	:	Shortest Path First (baseado em Prim [54])
BIP	:	Broadcast Incremental Power (Prim com propriedade WMA)
DSPF	:	Densest Shortest Path First (baseado em BIP)
rSPF	:	Random Shortest Path First (SPF randomizado)
rBIP	:	Random Broadcast Incremental Power (BIP randomizado)
rDSPF	:	Random Densest Shortest Path First (DSPF randomizado)
EWMA	:	Embedded Wireless Multicast Advantage
BLB	:	Busca local best (mais aprimorante com vizinhança completa)
BLB1	:	BLB com vizinhança nós folha
BLB2	:	BLB com vizinhança filhos mais custosos
BLF	:	Busca local first (primeiro aprimorante com vizinhança completa)
BLF1	:	BLF com vizinhança nós folha
BLF2	:	BLF com vizinhança filhos mais custosos
BLFc	:	BLF com circularização
BLF2c	:	BLF2 com circularização
BLFcr	:	BLF com circularização e randomização da ordem dos nós
GRASP	:	Greed Randomized Adaptative Search Procedures;
PR	:	Path Relinking (Reconexão por Caminhos);
GRASPh	:	GRASP com $\beta \in [0,1]$ com dist. probab. concentrada em $[0.4,1]$
GRASPf	:	GRASP com $\beta = 0,6$
GRASPe	:	GRASP com $\beta \in [0,\ 1]$ com dist. probab. equiprovável
GRASPh_gl	:	GRASP com $\beta \in [0.1,\ 1]$ com dist. probab. concentrada em $[0.1, 0.3]$
GRASPf_gl	:	GRASP com $\beta = 0,3$
GRASPh_ew	:	GRASPh com EWMA
HGR_IDU	:	GRASP híbrido com perturbação IDU

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

1.1 Motivação

As redes sem fio dividem-se em dois modelos principais: infra-estruturadas e sem infra-estrutura (redes ad hoc)[22]. No modelo infra-estruturado, uma parte da rede é fixa, garantindo a infra-estrutura necessária para acesso das estações móveis e para interconexão com redes externas. O equipamento que interliga a parte fixa com a móvel da rede é chamado de ponto de acesso ou estação base. A infra-estrutura central fixa interconecta todos os pontos de acesso por ligações cabeadas (*wired links*) de alta-velocidade e cada terminal sem fio comunica-se com um equipamento, responsável por intermediar as comunicações. Todas as comunicações da rede são feitas entre um nó móvel e o ponto de acesso, simplificando o projeto. Os nós desta rede, mesmo dentro do alcance uns dos outros, estão impossibilitados de estabelecer comunicação direta entre si. Neste modelo, se os pontos de acesso são danificados ou tornam-se indisponíveis, os nós têm que procurar outros pontos de acesso para reestabelecer a comunicação. Além disso, uma vez que vários nós se comunicam com o mesmo ponto de acesso, mecanismos de controle devem ser implementados. Da mesma forma, quando um nó móvel se afasta do seu ponto de acesso e a comunicação se torna difícil, ele precisa achar outro ponto de acesso próximo da sua nova localização, transferir a comunicação para este novo ponto e continuar a comunicação sem interrupção ou perda de dados. As redes celulares são o principal exemplo das redes infra-estruturadas.

Por outro lado, as redes *ad hoc* não requerem infra-estrutura prévia para que a comunicação ocorra [52]. Apresenta uma arquitetura mais flexível, onde os nós usuários se

comunicam diretamente uns com os outros se um nó estiver no raio de alcance do outro, senão o uso de nós intermediários pode ser feito para se alcançar o destino. Ao contrário das redes infra-estruturadas, onde as ligações e suas capacidades são determinadas a priori, a rede ad hoc é construída dinamicamente, dependendo de fatores como as distâncias entre os nós e a potência transmitida por eles, esquemas de controle de falha na recepção, interferência de outros usuários e ruídos no sinal. Desta forma, mesmo se a localização dos nós for fixa, muitos dos fatores que afetam a topologia da rede são influenciados pela ação dos nós da rede.

Nas redes ad hoc o encaminhamento dos pacotes de informação é realizado pelos próprios terminais que compõem a rede, sem a necessidade de um ponto centralizador. Nessas redes, uma faixa de transmissão é atribuída a cada estação e a alocação total da faixa determina uma árvore de transmissão, onde uma estação i transmite para outra estação j se, e somente se, j estiver na faixa de transmissão de i. A faixa de transmissão de uma estação depende da potência fornecida para a estação: particularmente, a potência P_i requerida pela estação i para transmitir dados corretamente para outra estação j deve satisfazer a desigualdade

$$P_i \geq \gamma \cdot d_{ij}^{\alpha} \ , \tag{1.1}$$

onde d_{ij} é a distância euclideana entre i e j, $\alpha \geq 1$ é o fator de atenuação do ambiente, e $\gamma \geq 1$ é o parâmetro de ajuste do receptor para detecção de sinal, que é normalmente normalizado como 1. Em um ambiente ideal (isto é, no espaço vazio, sem nenhuma barreira) considera-se $\alpha = 2$, mas o fator de atenuação pode variar de 1 a mais de 6, dependendo das condições ambientais do local onde a rede se localiza [50]. O caso mais estudado é $\alpha = 2$.

Os nós de uma rede ad hoc são tipicamente mais complexos que os de uma rede infra-estruturada, porque eles devem implementar todas as funcionalidades necessárias à operação da rede, enquanto que numa rede infra-estruturada pode-se transferir uma maior parte da complexidade para o ponto de acesso, deixando nos nós apenas suas funções essenciais. Dentre as funcionalidades necessárias aos nós de um rede ad hoc pode-se destacar o roteamento (determinação da melhor rota para que uma mensagem chegue ao seu destino de forma segura e eficiente), já que nós fora do alcance direto uns dos outros devem contar com nós intermediários para se comunicar. Estes nós intermediários precisam ser capazes de encaminhar os pacotes até seu destino. Esta maior complexidade dos nós nas redes ad hoc agrava ainda mais o problema do consumo de energia nestas redes. Por essas razões, o ambiente de redes sem fio ad hoc apresenta desafios não identificados

em redes celulares ou em redes com fio, mesmo se a mobilidade dos nós não é considerada.

A potência total gasta na transmissão de sinais a longo alcance é a alta, uma vez que sinais de rádio têm propriedades não lineares de atenuação, ou seja, a perda de propagação do sinal varia não-linearmente com a distância (conforme apresentado na Inequação 1.1). Wieselthier et al. [64] afirmam que em aplicações ponto a ponto (transmissão de um nó fonte para um nó destino) é melhor (da perspectiva da potência total transmitida) transmitir no menor nível de potência possível, mesmo que isso requeira o uso de vários nós intermediários até alcançar o destino. Entretanto, em aplicações de comunicação por difusão (*broadcast*), não é prudente tirar estas conclusões a priori, pois o uso de uma potência mais alta pode permitir a conectividade simultânea para um grande número de nós, de forma que a potência total requerida para alcançar todos os membros pode ser, na verdade, reduzida.

Redes de sensores sem fio são um tipo especial de rede ad hoc com sensoreamento distribuído, baixa capacidade de processamento e consumo reduzido de energia. Elas podem ser compostas por dezenas ou, até mesmo, centenas de pequenos dispositivos alimentados por baterias, chamados de nós sensores. Estas redes não necessitam de tipo algum de infraestrutura para estabelecer a comunicação entre seus componentes. Os dados são transmitidos através do ar em canais de freqüência de rádio (rádio difusão) ou infravermelho.

As redes de sensores sem fio podem ser utilizadas no monitoramento de condições ambientais, rastreamento de animais, coordenação e processamento de informações e nos casos em que redes cabeadas não estão disponíveis, quando sua instalação não é econômica ou em situações em que é necessária uma infra-estrutura rápida e não existe um sistema base central para transmissão. Com o avanço das tecnologias capazes de suportar este tipo de serviço, o custo dos equipamentos vem caindo bastante ao longo do tempo, tornando o uso de sensores sem fio ainda mais atrativo.

Dentre as questões mais cruciais relacionadas a redes de sensores sem fio ad hoc, cita-se a limitação de energia das baterias, uma vez que os nós dessa rede normalmente são alimentados por baterias e que estas têm tempo de vida limitado. Uma questão também crítica é o equilíbrio entre o alcance da transmissão sem fio (recepção simultânea por muitos nós de uma mesma mensagem transmitida) e a interferência resultante desta transmissão. Outro impacto indesejável do uso da alta potência de transmissão, além da interferência, é que o uso de uma potência alta acarreta o acréscimo da potência total utilizada, aumentando o consumo total de energia.

Desta forma, tendo em vista as grandes perspectivas de aplicações envolvendo redes de sensores sem fio e redes ad hoc, é crescente a demanda por soluções que minimizem o consumo de energia.

1.2 Apresentação do problema

O *Problema de Transmissão Multiponto (de um ponto para todos) com Minimização de Energia em Redes Ad Hoc* consiste em estabelecer a comunicação de um nó fonte para todos os outros nós de uma rede ad hoc, direta ou indiretamente, de forma que a potência total de transmissão (assumindo que nenhuma potência é gasta para recepção/processamento do sinal) seja minimizada e que a conectividade seja mantida (ou seja, todos os nós sejam alcançados pela transmissão do nó fonte). Uma de suas aplicações ocorre no contexto de redes de sensores. Este problema de minimização de energia em redes ad hoc possui diversas denominações na literatura: *Minimum Power Broadcast* (MPB) [26, 27, 28, 47, 48]; *Min-Power Symmetric Connectivity* [5]; *Minimum Energy Consumption Broadcast Subgraph* [16, 21, 22]; *Minimum Energy Broadcast Tree* [6, 13, 17, 64] e *Minimum-Energy Routing* [62]. Ao longo deste trabalho será utilizada a denominação MPB para citar o problema.

Árvores são estruturas apropriadas para representar objetos que possuem um relacionamento hierárquico entre si. A representação esquemática de árvores coloca a raiz no topo e os demais elementos em subárvores da raiz. Uma rede de transmissão pode ser representada por uma árvore, colocando-se o nó fonte da rede na raiz e os demais nós em subárvores. Os nós da árvore de transmissão (com exceção do nó raiz) dividem-se em dois tipos: nós folha e nós retransmissores. Os nós retransmissores são aqueles que transmitem dados para outros nós, intermediando a transmissão entre o nó fonte e os nós destino, enquanto os nós folha são somente receptores de dados. A potência total da árvore de transmissão é simplesmente a soma das potências alocadas a cada um dos nós transmissores da rede (potência alocada ao nó fonte e aos nós retransmissores). A modelagem matemática do problema é apresentada a seguir.

Um grafo dirigido é um objeto formado por dois conjuntos: um conjunto de vértices e um conjunto de arcos que interligam os vértices. Considera-se um grafo dirigido completo $G = (V, E)$, onde o conjunto de vértices, V, representa os nós da rede ($i = 0, \ldots, n-1$) e o conjunto de arcos, E, representa as ligações sem fio entre cada par (i, j) de nós da rede, sendo que o custo c_{ij} de cada arco $(i, j) \in E$, $i, j \in V$, indica a potência necessária para

transmistir do nó i para o nó j. Assume-se que o valor da potência mínima c_{ij} requerida para enviar mensagens de um nó para outro é dado pela distância entre os nós elevada a uma potência α, isto é $c_{ij} = d_{ij}^{\alpha}$. A alocação de potências aos nós corresponde ao nível de potência em que cada nó opera, isto é, transmite mensagens. Assim, dada uma alocação de potências aos nós $w : V \longrightarrow R^{+}$, o grafo G_w é um grafo dirigido que tem o mesmo conjunto de nós de G. Um arco (i, j) pertence a G_w se a potência alocada ao nó i satisfaz $w(i) \geq c_{ij}$. Não considerando obstáculos, a função de custo dos arcos é simétrica (isto é, $c_{ij} = c_{ji}$).

Assume-se que a comunicação entre os nós é uniforme, isto é, α é constante em toda área de transmissão, desprezando-se obstáculos (tais como montanhas e prédios), e que não há nenhuma limitação na direção da transmissão. Considera-se, ainda, que não há restrição sobre a máxima potência de transmissão e os nós podem utilizar diferentes níveis de potência para as várias configurações de rede de transmissão de que participarem. A conectividade da rede depende da potência transmitida.

Considera-se uma implementação centralizada, onde a construção da árvore de transmissão é feita a partir do nó fonte, que tem conhecimento completo de todos os nós da rede. Presume-se, também, ampla disponibilidade de recursos na largura de banda de transmissão, isto é, número ilimitado de frequências, posições de tempo ou códigos ortogonais CDMA (*Code Division Multiple Access* ou Acesso Múltiplo por Divisão de Código é uma tecnologia móvel e padrão de sistema para celulares), e assume-se que as perdas de potência devido à recepção de sinal e processamento são desprezíveis se comparadas ao sinal de transmissão.

Este trabalho considera uma rede ad hoc com nós fixos, como, em geral, ocorre nas redes de sensores (embora seja adotada uma rede de nós fixos, pode-se incorporar o impacto da mobilidade nos modelos apresentados porque a potência de transmissão pode ser ajustada para acomodar os novos locais dos nós, se necessário), e um nó fonte específico que deve transmitir uma mensagem para todos os outros nós da rede. Qualquer nó pode ser usado como nó transmissor intermediário para alcançar outros nós. Todos os nós possuem número suficiente de conversores de sinal, podendo então atender várias sessões de transmissão simultaneamente, e antenas *omnidirecionais* (transmissão do sinal em todas as direções), de forma que, se o nó i transmite para o nó j, todos os nós que estiverem na faixa de comunicação entre i e j também receberão a transmissão. Uma quantidade insuficiente de qualquer um dos recursos citados pode resultar na construção de redes de transmissão que não alcancem todos os seus destinos ou no uso de mais

potência do que necessário.

Embora existam propostas de soluções para o MPB com enfoque no tempo de vida das baterias que alimentam os nós da rede, neste trabalho, por simplificação, o enfoque será na minimização da potência total alocada à rede, sem preocupar-se, especificamente, com o tempo em que os nós estarão transmitindo ou no balanceamento do consumo de potência dos nós da rede.

É importante observar que a propriedade de transmissão das redes de sensores sem fio pode ser explorada em aplicações de transmissão por difusão (*broadcast*, onde um nó transmite para todos os outros nós da rede) e transmissão multiponto (*multicast*, onde um nó fonte transmite para um subconjunto dos outros nós da rede). Considere-se o exemplo ilustrado na Figura 1.1, no qual um subconjunto da árvore de transmissão envolve o nó i que está transmitindo para seus vizinhos, os nós j e k. A potência requerida para alcançar o nó j é P_{ij} e a requerida para alcançar o nó k é P_{ik}. Uma única transmissão na potência $P_{i,(j,k)} = max\{P_{ij}, P_{ik}\}$ é suficiente para alcançar tanto o nó j quanto o nó k, presumindo-se o uso de antenas omnidirecionais (transmissão do sinal em todas as direções). Esta habilidade para explorar esta propriedade da comunicação sem fio ad hoc é denominada por Wieselthier et al. [64] de vantagem das redes de transmissão multiponto sem fio (*Wireless Multicast Advantage* ou WMA).

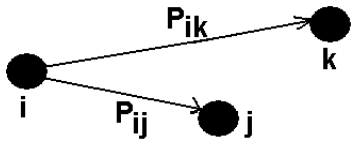

Figura 1.1: Propriedade WMA - $P_{i(j,k)} = max\{P_{ij}, P_{ik}\}$.

O ambiente em redes sem fio ad hoc no qual a transmissão de um nó é capaz de alcançar qualquer outro nó que estiver em sua faixa de comunicação é denominado ambiente baseado em nós (*node-based*) [64]. A potência total requerida para alcançar um conjunto de nós é a máxima requerida para alcançar qualquer um deles individualmente. Nas redes com fio, o ambiente é denominado baseado em ligações (*link-based*), pois o custo da transmissão de i para j e k é a soma dos custos dos nós individuais: $P_{i(j,k)} = P_{ij} + P_{ik}$, ou seja, existe um custo associado a cada ligação e o custo total da rede é a soma dos custos das ligações. Conseqüentemente, o problema de transmissão em redes com fio pode ser formulado como um problema de árvore geradora de custo mínimo e resolvido em tempo polinomial. Já nas redes sem fio ad hoc, o tratamento deve ser diferente devido à pro-

priedade WMA. Os algoritmos para cálculo de árvores geradoras de custo mínimo não capturam a natureza baseada em nós das redes sem fio ad hoc.

Como resultado da propriedade WMA, a idéia correta da comunicação omnidirecional sem fio ad hoc é a de um ambiente baseado em nós caracterizado pelas seguintes propriedades [64]:

- A transmissão de um nó é capaz de alcançar outro nó se o último estiver em sua faixa de comunicação, o que significa que a razão entre o sinal e o ruído excede a um certo limite e que os nós receptores alocaram recursos suficientes de recepção para este propósito.
- A potência total requerida para alcançar um conjunto de nós é a máxima requerida para alcançar qualquer um deles individualmente.

Cagalj et al. [13] descrevem uma prova formal da complexidade de MPB tanto para o caso geral quanto para o problema geométrico (quando os nós são pontos no espaço euclideano e define-se o custo de um arco como a distância euclideana entre i e j elevada a uma potência fixa α, isto é, $c_{ij} = d_{ij}^{\alpha}$), demonstrando que sua versão de decisão é um problema NP-completo através de uma transformação polinomial para o problema de cobertura de conjuntos.

Quando todos os nós estão dispostos em uma linha (problema unidimensional), MPB pode ser resolvido em tempo polinomial e sua solução é trivial, bastando que o nó fonte transmita para o nó mais distante, alcançando implicitamente todos os outros nós da rede. Em [14], os mesmos autores mostram que um algoritmo aproximativo para o problema não pode ter complexidade menor do que $O(\log n)$, onde n é o número total de nós.

1.3 Principais aplicações e trabalhos correlatos

Os dispositivos de comunicação sem fio vêm se tornando mais potentes, menores e mais leves com o decorrer dos anos, além de conter interfaces mais funcionais para os usuários. Para demonstrar o grande potencial das redes de sensores sem fio, citam-se algumas de suas possíveis aplicações [29]:

- **Militares**: a rápida instalação e as características de auto-organização e tolerância a falhas das redes de sensores sem fio (RSSFs) fazem com que elas sejam uma ferramenta de sensoriamento bastante promissora para integrar sistemas militares

de comando, controle, comunicação, computação, inteligência, vigilância, reconhecimento e mira. Como as RSSFs são descartáveis e baseadas na densa instalação de nós de baixo custo, a destruição de alguns nós por ações inimigas não afeta uma operação militar tanto quanto a destruição de um sensor tradicional, de maior custo. Tal característica torna as RSSFs bastante adequadas para uso em campos de batalha. Algumas das aplicações militares de RSSFs são o monitoramento de forças, equipamentos e munições amigos, a vigilância de campo de batalha, o reconhecimento de forças e terrenos inimigos, a avaliação de danos de batalhas, a detecção e o reconhecimento de ataques químicos, biológicos e nucleares. Além disso, sensores podem ser incorporados a sistemas inteligentes de mira [4].

- **Ambientais**: RSSFs podem ser usadas no rastreamento de movimentos de pássaros, insetos e outros pequenos animais, no monitoramento das condições ambientais que afetam a colheita e o gado, no controle de irrigação, no monitoramento em grande escala da Terra e na exploração planetária, na detecção química e biológica, no monitoramento biológico e ambiental de águas, solos e da atmosfera, na detecção de incêndio em florestas, na pesquisa meteorológica e geofísica, na detecção de inundação, no mapeamento de bio-complexidade do ambiente e no estudo da poluição [1, 10, 11, 12, 19, 31, 63]. Diante de calamidades da natureza, como, por exemplo, enchentes, terremotos ou furacões, é possível que a infraestrutura seja avariada, de modo que apenas os dispositivos sem fio e sem infraestrutura continuem operando.

- **Saúde**: RSSFs podem ser usadas no monitoramento de pacientes, na realização de diagnósticos, na administração de drogas em hospitais, no telemonitoramento de dados fisiológicos humanos e no monitoramento de médicos e pacientes dentro de um hospital [49]. Por exemplo, uma ambulância pode enviar informações sobre um paciente para o hospital, visando agilizar o seu acolhimento e tratamento. Pode-se inclusive consultar especialistas para um diagnóstico mais rápido e preciso.

- **Domésticas**: nós sensores e atuadores inteligentes podem ser embutidos em eletrodomésticos, tais como aspiradores de pó, fornos de microondas, geladeiras e vídeocassetes [34]. Esses nós podem interagir entre si e com redes externas via Internet ou satélite, permitindo que usuários gerenciem dispositivos domésticos local ou remotamente, de forma fácil e integrada. Sensores sem fio também podem ser utilizados no projeto de ambientes inteligentes, como descrito em [39]. Nesses ambientes, os nós sensores podem ser embutidos em móveis e eletrodomésticos e comunicar-se uns com os outros e com um nó servidor do aposento. O servidor do aposento pode se

comunicar com outros servidores de outros aposentos e aprender sobre os serviços que eles oferecem, por exemplo, impressão ou fax. Esses servidores e os nós sensores podem ser integrados com dispositivos embutidos existentes e tornarem-se sistemas auto-organizáveis e adaptativos baseados em modelos de teoria de controle [39].

- **Comerciais**: redes de sensores sem fio podem ser usadas no monitoramento de fadiga de material, na atualização de base de dados, no gerenciamento de estoque de fábricas, no monitoramento de qualidade de produtos, na construção de escritórios inteligentes, no controle ambiental de edifícios inteligentes, no controle de robôs, brinquedos interativos, museus interativos e automação, no controle de processos fabris, no monitoramento de áreas de desastres, no diagnósticos de máquinas, na detecção e monitoramento de roubos de automóveis, na detecção de veículos e no recebimento de informações tais como notícias, músicas, condições da estrada, do tempo, etc. Informações de logística podem ser distribuídas pela rede, obtendo-se economia de dinheiro e tempo [32, 55, 56, 63].

- **Substituição de redes cabeadas**: redes sem fio se mostram como boas alternativas para áreas de difícil acesso, prédios históricos que não podem ser danificados com cabeamento, ou mesmo eventos, tais como comícios e espetáculos, que precisem de uma rápida instalação.

A minimização do consumo de energia (controle de potência) em redes sem fio ad hoc recebeu atenção significativa nos últimos anos [5, 6, 13, 16, 17, 21, 22, 26, 27, 28, 45, 47, 48, 61, 62, 64]. Dentre estes trabalhos, destaca-se o de Wieselthier et al. [64] por observar que a aproximação baseada em nós era mais adequada para redes sem fio ad hoc do que os algoritmos previamente adotados, baseados em ligações. Nesse trabalho desenvolveu-se o algoritmo *Broadcast Incremental Power* (BIP), que é uma heurística simples para a construção de árvores de transmissão de mínima potência total. No algoritmo BIP, novos nós são adicionados à árvore com base no mínimo custo incremental, até que todos os nós sejam incluídos. Esse foi o primeiro algoritmo a considerar a propriedade WMA na construção de árvores de transmissão multiponto de mínima potência total.

Subseqüentemente, Wan et al. [62] forneceram os primeiros resultados analíticos para o MPB. Explorando a estrutura da árvore geradora de custo mínimo geométrica, provaram que a razão de aproximação (medida da qualidade da solução obtida pela heurística em relação à solução ótima; por exemplo, uma razão de aproximação igual a 6, indica que o custo da solução produzida pela heurística é no máximo seis vezes o custo da solução ótima) da MST está entre 6 e 12 e a razão de aproximação do algoritmo BIP está entre 13/3

e 12. Também observaram que, para algumas instâncias, BIP desconsidera a propriedade WMA das redes sem fio ad hoc, porque adiciona à rede de transmissão somente um nó em cada iteração, que é aquele que pode ser incluído ao mínimo custo adicional. Então BIP, embora seja centralizado, não usa toda a disponibilidade de informação sobre a rede. Como resultado, pode construir uma árvore de transmissão que coincida com a árvore geradora de custo mínimo em algumas instâncias, ignorando completamente a natureza das redes sem fio ad hoc.

Em [13], Cagalj et al. introduziram uma heurística denominada *Embedded Wireless Multicast Advantage* (EWMA) que calcula soluções de boa qualidade para o MPB. Este algoritmo utiliza como entrada uma árvore geradora e a transforma em uma árvore de transmissão com custo total inferior ao custo da árvore inicial, realizando melhorias locais. Outra heurística chamada *Sweep* foi proposta em [64]. Ela também utiliza como entrada uma árvore obtida por qualquer heurística construtiva e elimina transmissões desnecessárias, reduzindo a potência de transmissão de alguns nós e transferindo a responsabilidade da transmissão para um nó particular. A descrição de uma série de problemas em redes sem fio ad hoc e alguns algoritmos aproximativos podem ser vistos no trabalho de Caragiannis et al. [17].

A influência dos erros nas ligações para a confiabilidade da transmissão foi considerada em Banerjee et al. [7], que apresentaram esquemas para construções de árvores de transmissão por difusão e multiponto para comunicações sem fio confiáveis. O trabalho de Park e Sahni [51] tem enfoque na maximização do tempo de vida em redes sem fio ad hoc. Outras heurísticas construtivas desenvolvidas para o problema podem ser consultadas em [6, 25, 40, 41, 43, 44].

Modelos de programação inteira para solução ótima do MPB foram propostos em [27, 47]. Outras técnicas foram sugeridas para resolver o problema, incluindo uma técnica de transmissão baseada somente nos nós internos da rede proposta por Stojmenović et al. [61], algoritmos localizados, quando não se tem o conhecimento completo da localização de todos os nós da rede, em [18, 65] e um procedimento baseado em colônias de formigas por Das et al. [24] que, posteriormente, foi combinado com um método de clusterização em [26].

Em [28], Das et al. apresentaram uma heurística simples de busca local denominada *"r-shrink"* para melhorar as soluções construídas por outros algoritmos. Essa busca consiste basicamente em reduzir o raio de transmissão dos nós transmissores e tentar reacomodar os nós que se desconectaram devido à essa redução. Montemanni et al. [48] apresentaram um

algoritmo de *simulated annealing* para o MPB, afirmando que o algoritmo proposto é capaz de fornecer soluções de boa qualidade, significativamente melhores do que as provenientes do algoritmo BIP e produzindo os melhores resultados publicados na literatura até então.

1.4 Organização da dissertação

O objetivo dessa dissertação é propor novas heurísticas para o problema de transmissão de um ponto a todos em redes ad hoc com minimização da energia consumida.

O Capítulo 2 traz a descrição dos principais algoritmos construtivos - BLU, BIP, SPF, DSPF - e de melhoria - EWMA - encontrados na literatura, bem como novas heurísticas construtivas (rBIP, rDSPF e rSPF) e de busca local (BLB, BLB1, BLB2, BLF, BLF1, BLF2, BLF2c, BLFc e BLFcr) propostas para resolução do problema.

No Capítulo 3 são propostas novas heurísticas, baseadas em metaheurísticas, combinando as características dos algoritmos GRASP e de reconexão por caminhos com as heurísticas apresentadas no capítulo anterior.

A descrição das instâncias testes, condições dos experimentos e resultados computacionais dos algoritmos apresentados estão no Capítulo 4. Finalmente, o Capítulo 5 traz conclusões e indicações para trabalhos futuros.

Capítulo 2

Heurísticas

A Seção 2.1 ilustrará o funcionamento de uma rede de transmissão composta de um nó fonte e dois nós destinos. As considerações descritas no capítulo anterior, tais como recursos ilimitados, interferência desprezível no sinal, nós fixos e uso de antenas omnidirecionais são adotadas como premissas para o problema. Sem perda de generalidade, assume-se que o nó 0 é sempre o nó fonte. A Seção 2.2 apresenta as heurísticas construtivas existentes na literatura e as propostas neste trabalho. O algoritmo EWMA proposto por Cagalj [13] será descrito na Seção 2.3 e os algoritmos de busca local serão apresentados na Seção 2.4.

2.1 Transmissão de mínima potência total: dois destinos

Considere-se um nó fonte i, localizado na origem, e dois nós destino j (localizado no eixo horizontal, sem perda de generalidade) e k, como ilustrado na Figura 2.1 a seguir. Seja θ o ângulo formado por j e k tendo i como vértice e $\gamma \geq 1$ o parâmetro de ajuste do receptor para detecção de sinal, que é normalmente normalizado como 1. As distâncias entre i e j, i e k e j e k são, respectivamente d_{ij}, d_{ik} e d_{jk}, Assume-se, sem perda de generalidade, que $d_{ik} > d_{ij}$ e define-se:

- $P_{ij} = \gamma \cdot d_{ij}^{\alpha}$: potência necessária para transmitir de i para j;
- $P_{ik} = \gamma \cdot d_{ik}^{\alpha}$: potência necessária para transmitir de i para k; e
- $P_{jk} = \gamma \cdot d_{jk}^{\alpha}$: potência necessária para transmitir de j para k.

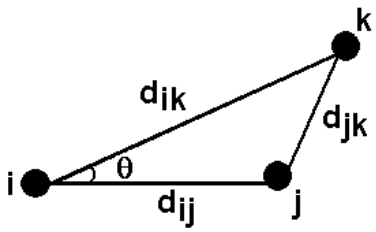

Figura 2.1: Exemplo de transmissão para dois destinos.

Neste simples exemplo, considerando um ambiente sem obstáculos ou interferências, existiriam duas estratégias de transmissão:

1. O nó i transmite utilizando potência P_{ik}: tanto j quanto k são alcançados.
2. O nó i transmite utilizando potência P_{ij}: como somente j é alcançado, este transmite para k usando P_{jk}, resultando na potência total de $P_{ij} + P_{jk}$.

Deseja-se escolher a alternativa que alcance os dois destinos e que resulte na menor potência total. Pela lei dos cossenos $d_{jk}^2 = d_{ik}^2 + d_{ij}^2 - 2d_{ik}d_{ij}\cos\theta$ ou, reescrevendo, $d_{ik}^2 = d_{jk}^2 - d_{ij}^2 + 2d_{ik}d_{ij}\cos\theta$. Para $\alpha = 2$, a potência total usando a primeira estratégia é d_{ik}^2 e, usando a segunda, é $d_{ij}^2 + d_{jk}^2$. Assim, para que a primeira estratégia resulte na menor potência total é necessário que $d_{ik}^2 < d_{ij}^2 + d_{jk}^2$. Substituindo d_{ik}^2 pela expressão da lei dos cossenos obtém-se $d_{jk}^2 - d_{ij}^2 + 2d_{ik}d_{ij}\cos\theta < d_{ij}^2 + d_{jk}^2$, que, simplificando, resulta em $d_{ij} > d_{ik}\cos\theta$.

Logo, no caso da propagação que segue a lei $1/r^2$, onde r é a distância de um nó ao nó fonte, chega-se ao seguinte resultado:

- usar a primeira estratégia, se $d_{ij} > d_{ik}\cos\theta$; ou
- usar a segunda estratégia, caso contrário.

Para o caso geral de propagação, assumindo-se a lei $1/r^\alpha$, o resultado da manipulação algébrica seria o seguinte:

- usar a primeira estratégia, se $x^\alpha - 1 < (1 + x^2 - 2x\cos\theta)^{\alpha/2}$, onde $x = d_{ik}/d_{ij}$; ou
- usar a segunda estratégia, caso contrário.

2.2 Algoritmos construtivos

Heurística é qualquer método aproximado, projetado com base nas características dos problemas, com complexidade reduzida em relação à dos algoritmos exatos e fornecendo, em geral, soluções viáveis de boa qualidade - sem garantia de otimalidade - para problemas NP-difíceis. Algoritmos construtivos são heurísticas que criam uma solução viável para o problema. Algumas das heurísticas construtivas apresentadas por Wieselthier et al. [64] e Athanassopoulos et al. [6] serão descritas na Seção 2.2.1 a seguir. Elas serviram de base para o desenvolvimento de novas heurísticas construtivas para a resolução do MPB. Os novos algoritmos propostos serão apresentados na Seção 2.2.2.

2.2.1 Principais algoritmos construtivos da literatura

Algoritmo BLU

Um aproximação direta, mas longe da ótima, é o uso de árvores de transmissão que consistam na superposição dos melhores caminhos individuais para cada destino [9]. Um algoritmo tal como o de Bellman-Ford [8] ou Dijkstra [30] pode ser usado para calcular os caminhos de mínima distância do nó fonte para qualquer outro nó. O algoritmo BLU (*Broadcast Least-Unicast Cost*) recebe como entrada o grafo $G = (V, E)$ e os custos $c_{ij}, \forall (i, j) \in E$, e estabelece o caminho mais curto do nó fonte para todos os outros nós da rede. A árvore de transmissão consiste na superposição destes caminhos mais curtos. A propriedade WMA é ignorada na construção da árvore, sendo levada em consideração apenas na avaliação de seu custo total (isto é, a potência total necessária para manter a árvore de transmissão conectada, quando todos os nós são alcançados).

A Figura 2.2 a seguir ilustra o pseudocódigo do algoritmo BLU, utilizando o algoritmo de Dijkstra no passo 1. A complexidade do algoritmo BLU é determinada pelo cômputo dos passos 1 a 10. A determinação do caminho mais curto da fonte para cada destino (passo 1) é $O(n^2)$. A inserção dos custos dos caminhos do nó fonte para cada destino na fila (passo 2) e o ciclo dos passos 5 a 9 têm complexidade $O(n \log n)$. A inicialização da árvore (passo 3) e a totalização do custo final (passo 10) são $O(n)$. Logo, a complexidade final de BLU é $O(n^2)$.

O cálculo dos caminhos mais curtos de um nó para todos os outros da rede num grafo dirigido completo pode ser feito em $O(n^2)$ usando uma implementação simples do algoritmo de Dijkstra usando tabelas de dispersão (*heaps*) como filas de prioridade [23].

```
procedimento BLU
1.  Encontrar o caminho mais curto do nó fonte para cada nó destino;
2.  Inserir os nós destino e custos do nó fonte para cada destino na fila Q;
3.  Criar uma árvore de transmissão T sem nó gum;
4.  Inserir o nó fonte na árvore T;
5.  enquanto (Q ≠ ∅)
6.  início
7.     Extrair o nó de custo mínimo da fila Q;
8.     Acrescentar esse nó e o caminho mais curto da fonte até ele à árvore T;
9.  fim-enquanto
10. Retornar a árvore T, totalizando o custo final adotando a propriedade WMA;
Fim BLU
```

Figura 2.2: Pseudocódigo do algoritmo BLU.

Estruturas mais complexas, que geram implementações mais rápidas do algoritmo de Dijkstra (por exemplo, *heaps* de Fibonacci [23]) não diminuem substancialmente o tempo de processamento no caso de o grafo que representa a rede ad hoc ser completo, como nesse trabalho.

A Figura 2.3 mostra um exemplo de árvore de transmissão produzida pelo algoritmo BLU. Inicialmente, somente o nó fonte está na árvore. Calcula-se o caminho mais curto do nó fonte para cada destino. A seguir escolhe-se o nó mais próximo do nó fonte, que é o nó 3 ($c_{03} = 0,65$). Na segunda iteração o nó mais próximo é o nó 2 ($c_{02} = 0,85$); na terceira, é nó 7 ($c_{07} = 2,01$); na quarta, é o nó 6 ($c_{02} + c_{26} = 2,3$); na quinta, é o nó 1 ($c_{07} + c_{71} = 3,51$); na sexta, é o nó 8 ($c_{02} + c_{28} = 4$); na sétima, é o nó 5 ($c_{03} + c_{35} = 4,57$); na oitava, é o nó 4 ($c_{02} + c_{26} + c_{64} = 5,84$) e, por fim, o nó 9 ($c_{02} + c_{26} + c_{69} = 7,36$).

Seja P_i a potência atribuída ao nó i. A potência requerida para manter a árvore somente é totalizada após sua construção, considerando a propriedade WMA. Para cada nó, calcula-se a potência necessária para alcançar todos os seus destinos (filhos) e, a seguir, totaliza-se a potência total da árvore. Neste exemplo, os nós transmissores são os nós $0, 3, 2, 7$ e 6 e a potência total da árvore então é dada por $P_0 + P_3 + P_2 + P_7 + P_6 = \max(c_{02}; c_{03}; c_{07}) + c_{35} + \max(c_{26}; c_{28}) + c_{71} + \max(c_{64}; c_{69}) = \max(0,85; 0,65; 2,01) + 3,92 + \max(1,45; 3,15) + 1,50 + \max(3,54; 5,06) = 15,64$. Se a potência total da árvore fosse contabilizada sem considerar a propriedade WMA, seria a resultante da soma dos custos de todas as arestas da árvore, ou seja, $22,13$.

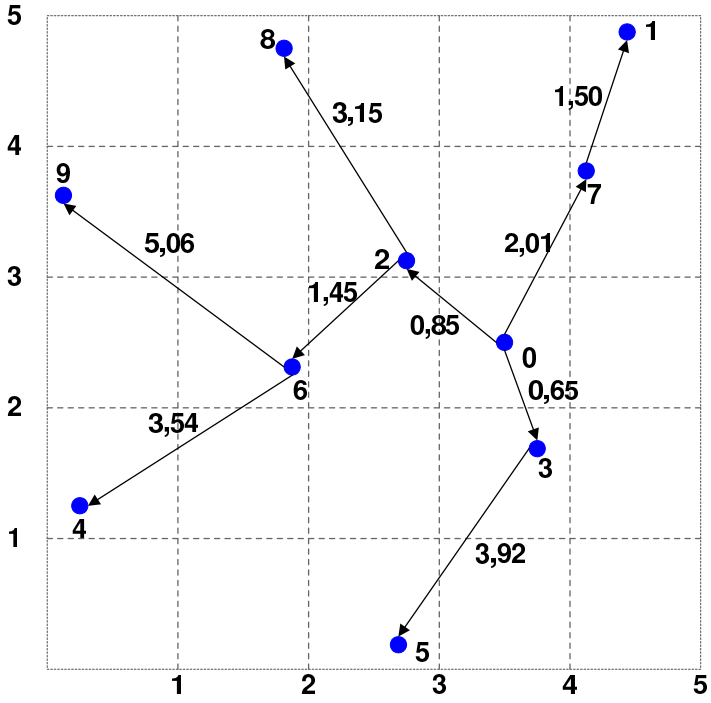

Figura 2.3: Árvore construída usando BLU, cujo custo total é $15,64$ ($\alpha = 2$).

Algoritmo SPF

Este algoritmo é baseado no uso da formulação padrão de árvore geradora mínima (como usado nas redes com fio), na qual o custo da ligação é associado a cada par de nós (isto é, a potência para manter a ligação). O algoritmo SPF (*Shortest Path First*) recebe como entrada o grafo $G = (V, E)$ e os custos $c_{ij}, \forall (i, j) \in E$ (tal como no algoritmo anterior) e, em cada fase, adiciona à árvore de transmissão o caminho mais curto (isto é, o caminho cujo estabelecimento requer a menor potência) que conecta algum nó da árvore a algum nó fora da árvore. O algoritmo termina quando todos os nós estiverem conectados. Assim como no algoritmo anterior, a propriedade WMA também é ignorada na construção da árvore, sendo levada em consideração apenas na avaliação do custo total da árvore.

A complexidade do algoritmo de árvore geradora mínima de Prim [54] depende de como o conjunto de nós a incluir na árvore é implementado. Se o conjunto for implementado como um *heap*, a complexidade do algoritmo será $O(|E| \log |V|)$. Para os testes, foi utilizada uma implementação simples do algoritmo de Prim com complexidade $O(n^3)$. A Figura 2.4 a seguir ilustra o pseudocódigo do algoritmo SPF, utilizando o algoritmo de Prim no passo 1. A complexidade do algoritmo SPF é determinada basicamente pelo passo 1, pois a totalização do custo final (passo 2) é $O(n)$. Logo, a complexidade final desta implementação de SPF é $O(n^3)$.

```
procedimento SPF
 1.  Encontrar a árvore geradora mínima do grafo G;
 2.  Retornar a árvore T, totalizando o custo final adotando a propriedade WMA;
Fim SPF
```

Figura 2.4: Pseudocódigo do algoritmo SPF.

A Figura 2.5 a seguir mostra a árvore de transmissão produzida pelo algoritmo SPF. Inicialmente, somente o nó fonte está na árvore. O algoritmo de árvore geradora mínima (AGM) começa escolhendo o nó mais próximo do nó fonte, que é o nó 3 ($c_{03} = 0,65$). Na segunda iteração, o algoritmo AGM adiciona a ligação de menor custo entre um nó que está dentro e outro que está fora da árvore, que, nesse caso, é nó 2 ($c_{02} = 0,85$); na terceira, o algoritmo AGM adiciona o nó 6 ($c_{26} = 1,45$); na quarta, o nó 7 ($c_{07} = 2,01$); na quinta, escolhe o nó 1 ($c_{71} = 1,5$), na sexta, o nó 8 ($c_{28} = 3,15$); na sétima, o nó 4 ($c_{64} = 3,54$); na oitava, o nó 5 ($c_{35} = 3,92$) e, por fim, o nó 9 ($c_{89} = 4,28$).

Após a construção da árvore geradora mínima do grafo, SPF calcula, para cada nó, a potência necessária para alcançar todos os seus filhos e, a seguir, totaliza a potência total da árvore considerando a propriedade WMA, assim como feito em BLU. Os nós transmissores da árvore são os nós $0, 3, 2, 6, 7$ e 8 e a potência total da árvore então é dada por: $P_0 + P_3 + P_2 + P_6 + P_7 + P_8 = \max(c_{02}; c_{03}; c_{07}) + c_{35} + \max(c_{26}; c_{28}) + c_{64} + c_{71} + c_{89} = \max(0,85; 0,65; 2,01) + \max(1,45; 3,15) + 3,92 + 3,54 + 1,50 + 4,28 = 18,4$. Observe que a árvore produzida por SPF, usando o mesmo exemplo do algoritmo BLU, obteve custo total cerca de 18% maior que este. Sem considerar a propriedade WMA, a potência total da árvore seria $21,35$, resultado da soma dos custos das arestas da árvore.

Algoritmo BIP

O algoritmo BIP (*Broadcast Incremental Power*) [64] é semelhante ao algoritmo de Prim [54] para a formação da árvore geradora mínima, na forma pela qual novos nós são adicionados à árvore um a um (na base do mínimo custo) até que todos os nós sejam incluídos. A implementação do algoritmo é baseada no algoritmo de Prim padrão, mas com uma diferença fundamental. Enquanto no algoritmo de Prim os custos $c_{ij}, \forall (i,j) \in E$, se mantêm constantes durante a execução do algoritmo, BIP deve atualizar dinamicamente os custos das arestas em cada iteração (isto é, quando um novo nó é adicionado à arvore) para refletir o fato de que o custo de adicionar novos nós a uma lista de vizinhos do nó transmissor é apenas o *custo incremental*.

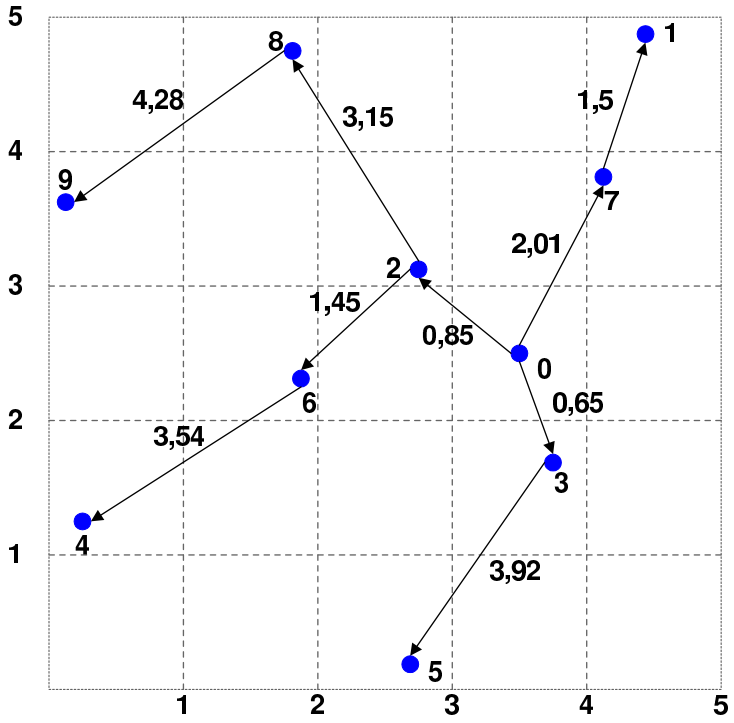

Figura 2.5: Árvore construída usando SPF, cujo custo total é $18,4$ ($\alpha = 2$).

Para exemplificar, considere que o nó i já esteja na árvore (pode ser tanto um nó folha quanto um retransmissor) e o nó j ainda não esteja. Para cada nó j fora da árvore é feita a seguinte atualização:

$$c'_{ij} = \begin{cases} c_{ij} - P_i, & \text{se } c_{ij} > P_i; \\ 0, & \text{caso contrário,} \end{cases}$$

onde c_{ij} é o custo da ligação entre os nós i e j e P_i é o nível de potência em que o nó i já está transmitindo (potência antes de adicionar o nó j; $P_i = 0$ se o nó i é, até o momento, um nó folha). O valor c'_{ij} representa o *custo incremental* associado à adição do nó j ao conjunto de nós para os quais o nó i já transmite. Assim, o custo de transmitir de um nó i, que está na árvore, para um nó j, fora da arvore não é c_{ij}, pois se o nó i já transmite com potência P_i, basta que ele a aumente em $c_{ij} - P_i$ para alcançar j.

O par (i, j) que resulta no custo incremental mínimo é selecionado em cada iteração. O nó i passa a transmitir no nível de potência necessária para alcançar o nó j e é chamado de nó predecessor ou nó pai de j. Um novo nó é adicionado à arvore em cada etapa do algoritmo. BIP não fornece necessariamente árvores de mínimo custo para redes ad hoc, ao contrário do algoritmo de Prim, que garante árvores de custo mínimo para redes com fio. As Figuras 2.6 e 2.7 ilustram o pseudocódigo do algoritmo BIP e da função auxiliar que calcula a ligação mais próxima.

```
procedimento BIP
1.  Criar uma árvore de transmissão T sem nó algum;
2.  Inserir o nó fonte na árvore T;
3.  para i = 0,...,n-1 faça
4.  início
5.     para j = 0,...,n-1 e j ≠ i faça
6.     início
7.        c'_ij = c_ij;
8.     fim - para
9.  fim - para
10. enquanto (|T| < n) faça
11. início
12.    (I,J) ← Calcular_ligação_mais_próxima (T,c');
13.    Atualizar a árvore T, adicionando o nó J;
14.    Atualizar o predecessor do nó J;
15.    para ∀j ∉ T faça
16.    início
17.       se (c_ij > P_i)
18.          c'_ij = c_ij - P_i;
19.       senão
20.          c'_ij = 0;
21.    fim - para
22. fim - enquanto
23. Retornar a árvore T, totalizando seu custo final;
Fim BIP
```

Figura 2.6: Pseudocódigo do algoritmo BIP.

```
procedimento Calcular_ligação_mais_próxima (T,c')
1.  custo_min ← ∞;
2.  para ∀i ∈ T faça
3.  início
4.     para ∀j ∉ T faça
5.     início
6.        se (c'_ij < custo_min) então
7.        início
8.           custo_min ← c'_ij;
9.           I ← i;
10.          J ← j;
11.       fim - se
12.    fim - para
13. fim - para
14. Retornar (I,J);
Fim
```

Figura 2.7: Pseudocódigo do procedimento `Calcular_ligação_mais_próxima`, utilizado no BIP.

O ciclo dos passos 3 a 9 do algoritmo BIP tem complexidade $O(n^2)$ e o dos passos 10 a 22 consiste na execução n vezes da função que calcula a ligação mais próxima (passo 12), cuja complexidade é $O(n^2)$, na atualização da árvore e do predecessor do nó J (passos 13 e 14), ambos com complexidade $O(1)$, e do ciclo dos passos 15 a 21 que tem complexidade $O(n)$. Logo, a complexidade dos passos 10 a 22 é $O(n^3)$. A totalização do custo final (passo 23) tem complexidade $O(n)$. Logo, a complexidade de BIP é $O(n^3)$.

A Figura 2.8 a seguir apresenta um exemplo de construção de uma árvore de transmissão pelo algoritmo BIP. O exemplo consiste de uma rede de 10 nós, na qual o nó 0 é o nó fonte. Assume-se um fator de atenuação $\alpha = 2$. Inicialmente, a árvore consiste somente do nó fonte. Primeiro, determina-se o nó que a fonte pode alcançar com o mínimo gasto de potência. O vizinho mais próximo do nó 0 é o nó 3. Este nó é adicionado à árvore. Neste momento, os nós 0 e 3 estão incluídos na árvore (Figura 2.8 (a)). A seguir, determina-se qual novo nó pode ser adicionado à arvore na base do mínimo custo incremental. Existe duas alternativas: ou o nó 0 aumenta sua potência para alcançar um outro nó, ou o nó 3 pode transmitir para seu vizinho mais próximo que ainda não está na árvore. Neste exemplo, o nó 0 aumenta seu nível de potência para alcançar o nó 2 (Figura 2.8 (b)). O custo associado à adição do nó 2 à árvore é o custo incremental associado ao aumento da potência do nó 0 para o nível suficiente para alcançar o nó 2. O custo de transmissão entre os nós 0 e 3 é $c_{0,3} = 0,65$ e o custo de transmissão entre o nó 0 e o nó 2 é $c_{0,2} = 0,85$. O custo incremental associado à adição do nó 2 à árvore contendo os nós 0 e 3 é $c_{0,2} - c_{0,3} = 0,20$. Neste caso, explora-se a propriedade WMA das redes sem fio ad hoc porque tanto o nó 2 quanto o nó 3 são alcançados quando o nó 0 transmite com potência suficiente para alcançar o nó 2. No terceiro passo existem três nós na rede: os nós 0, 2 e 3. Para cada um destes nós, determina-se o custo incremental para alcançar um novo nó. Como os nós 2 e 3 não estavam transmitindo previamente, seus respectivos custos incrementais serão iguais às suas potências totais de transmissão se eles forem escolhidos para transmitir. Para o nó 0, que já estava transmitindo, seu custo incremental será somente o acréscimo de potência requerido para alcançar mais um nó. O nó 7 é o que pode ser acrescentado à árvore com o mínimo custo incremental (Figura 2.8 (c)). Este procedimento continua até todos os nós serem incluídos na árvore, como ilustrado na Figura 2.8 (d). A ordem na qual os nós foram adicionados foi $0, 3, 2, 7, 6, 1, 8, 9, 5$ e 4.

A potência total requerida para manter esta árvore é a soma das potências máximas de transmissão de cada um dos nós transmissores. Neste exemplo, os nós 0, 2 e 7 transmitem, enquanto os outros nós (folhas) são somente receptores. A potência total de transmissão é dada por $P_0 + P_2 + P_7 = \max(c_{0,2}; c_{0,3}; c_{0,6}; c_{0,7}) + \max(c_{2,4}; c_{2,5}; c_{2,8}; c_{2,9}) + c_{7,1} =$

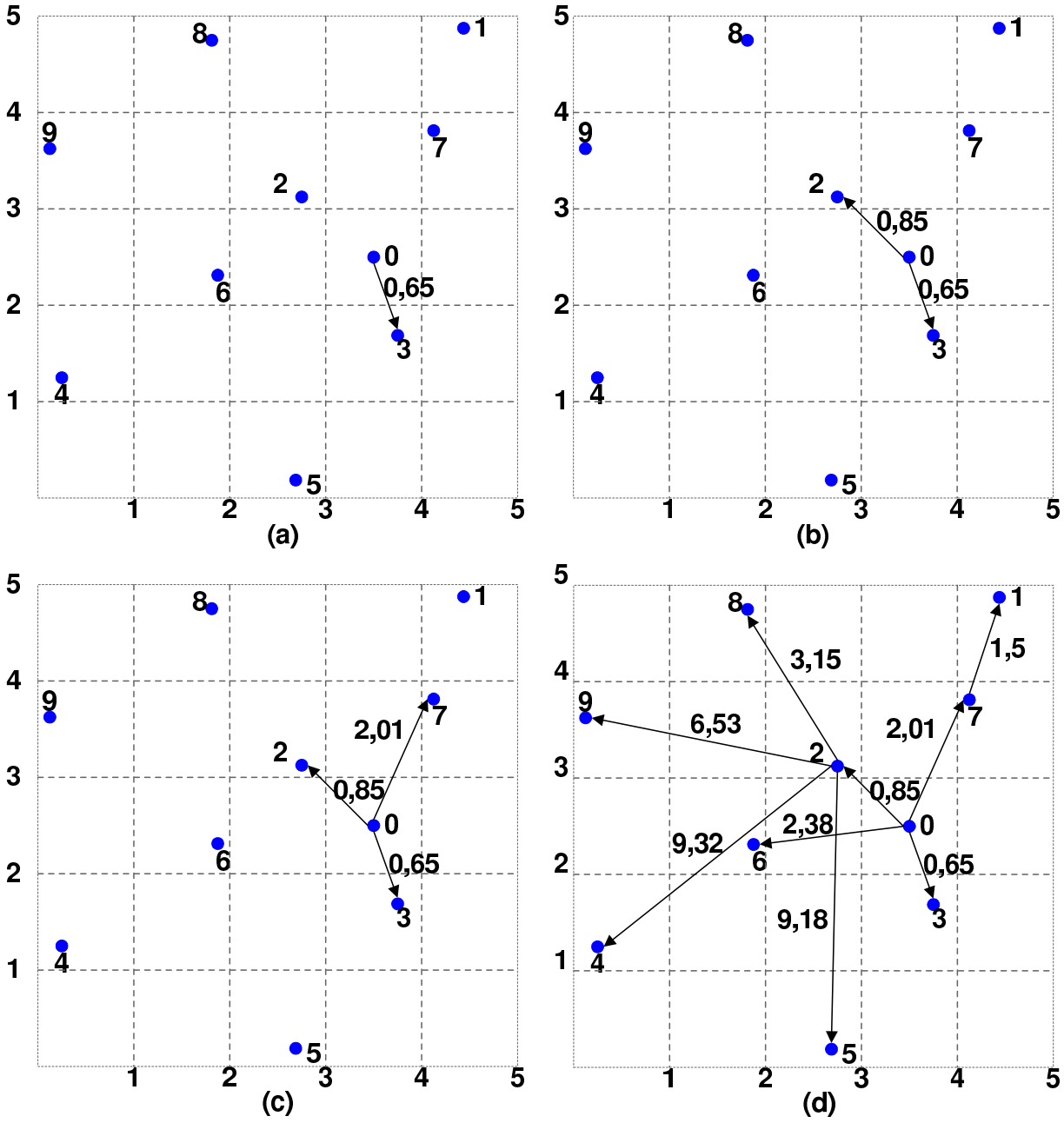

Figura 2.8: Árvore construída pelo BIP ($\alpha = 2$), sendo (a) transmissão de 0 para 3; (b) transmissão de 0 para 2; (c) transmissão 0 para 7 e (d) a árvore final com custo total $13,2$.

$$\max(0,85; 0,65; 2,38; 2,01) + \max(9,32; 9,18; 3,15; 6,53) + 1,50 = 13,2.$$

Para este exemplo, o custo obtido foi $13,2$, 16% menor que o resultado obtido por BLU e 28% menor que o obtido pelo SPF.

Algoritmo DSPF

O algoritmo DSPF (*Densest Shortest Path First*) é bastante semelhante ao algoritmo BIP e foi apresentado inicialmente em [6]. A única diferença entre ele e BIP é que DSPF seleciona, dentre os caminhos mínimos de um nó na árvore para um nó fora da árvore, aquele que minimiza a razão entre a potência adicional e o número de nós que serão adicionados implicitamente, caso esse nó seja escolhido.

As Figuras 2.9 e 2.10 ilustram os pseudocódigos do algoritmo DSPF e da função que calcula a razão mínima. Em DSPF, a complexidade dos passos 3 a 9 é $O(n^2)$ e a do ciclo dos passos 10 a 22 é $O(n^3)$, pois a função que calcula a razão mínima tem complexidade $O(n^2)$, e os demais passos deste ciclo tem complexidade inferior a $O(n^2)$. A totalização do custo final (passo 23) tem complexidade $O(n)$. Logo, a complexidade de DSPF, assim como a do BIP, é $O(n^3)$.

```
procedimento DSPF
 1.  Criar uma árvore de transmissão T sem nó algum;
 2.  Inserir o nó fonte na árvore T;
 3.  para i = 0,...,n-1 faça
 4.  início
 5.      para j = 0,...,n-1 e j ≠ i faça
 6.      início
 7.          c'_ij ← c_ij;
 8.      fim - para
 9.  fim - para
10.  enquanto (|T| < n) faça
11.  início
12.      (I,J) ← Calcular_ligação_com_razão mínima (T,c');
13.      Atualizar a árvore T, adicionando os nós alcançados;
14.      Atualizar o predecessor dos nós adicionados;
15.      para ∀j ∉ T faça
16.      início
17.          se (c_ij > P_i)
18.              c'_ij = c_ij - P_i;
19.          senão
20.              c'_ij = 0;
21.      fim - para
22.  fim - enquanto
23.  Retornar a árvore T, totalizando seu custo final;
Fim DSPF
```

Figura 2.9: Pseudocódigo do algoritmo DSPF.

A Figura 2.11 apresenta um exemplo de construção de uma árvore de transmissão utilizando o algorimo DSPF. Já na primeira iteração de DSPF (Figura 2.11 (a)) observa-se sua diferença em relação a BIP. A transmissão escolhida pelo algoritmo foi a do nó fonte 0 para o nó 2, pois a inclusão de 2 inclui implicitamente o nó 3 (o nó 2 possui a menor razão

```
procedimento Calcular_ligação_com_razão_mínima (T, c')
1.   razao_min ← ∞;
2.   para ∀i ∈ T faça
3.   início
4.      para ∀j ∉ T faça
5.      início
6.         Calcular o caminho mais curto de i para j;
7.      fim - para
8.   fim - para
9.   para ∀j ∉ T faça
10.     Seja i o nó da árvore T cuja distância a j é mínima;
11.     Determinar o total de nós alcançados com a adição de j;
12.     Calcular a razão r_ij entre o custo da adição de j e o total de nós alcançados;
13.     se (r_ij < razao_min) então
14.     início
15.        razao_min ← r_ij;
16.        I ← i;
17.        J ← j;
18.     fim - se
19.  fim - para
20.  Retorne (I, J);
Fim
```

Figura 2.10: Pseudocódigo do procedimento `Calcular_ligação_com_razão_mínima`, utilizado no DSPF.

entre a potência adicional e o número de nós adicionados: $0,85/2 = 0,425$). Na segunda iteração, DSPF escolhe a transmissão do nó 2 para o 8 (razão $3,15/3 = 1,05$), pois ela acarreta a inclusão implícita dos nós 6 e 7. Assim, ao final da segunda iteração de DSPF (Figura 2.11 (b)), existem seis nós adicionados à árvore, enquanto que na segunda iteração do BIP existiam somente três. Na terceira iteração, DSPF opta pela transmissão de 7 para 1 (razão $1,5/1 = 1,5$). O procedimento continua até que todos os nós sejam incluídos na árvore, como ilustrado na Figura 2.11 (d). Para este exemplo foram necessárias apenas quatro iterações para a construção da árvore e os nós adicionados no passo 4 foram, 4, 5 e 9 (razão $(9,32 - 3,15)/3 = 2,06$).

Neste exemplo os nós 0, 2 e 7 transmitem, enquanto os outros nós (folhas) são somente receptores. Assim, a potência total de transmissão da árvore construída por DSPF é $P_0 + P_2 + P_3 = \max(c_{0,2}; c_{0,3}) + c_{7,1} + \max(c_{2,4}; c_{2,5}; c_{2,6}; c_{2,8}; c_{2,9}) = \max(0,85; 0,65) + \max(9,32; 9,18; 1,45; 3,15; 6,53) + 1,50 = 11,67$.

O custo total da solução produzida por DSPF é cerca de 12% menor que o custo total obtido por BIP, 25% menor que o custo total obtido por BLU e 37% menor que o obtido pelo SPF. Este exemplo ilustra como a utilização de DSPF acelera significativamente a

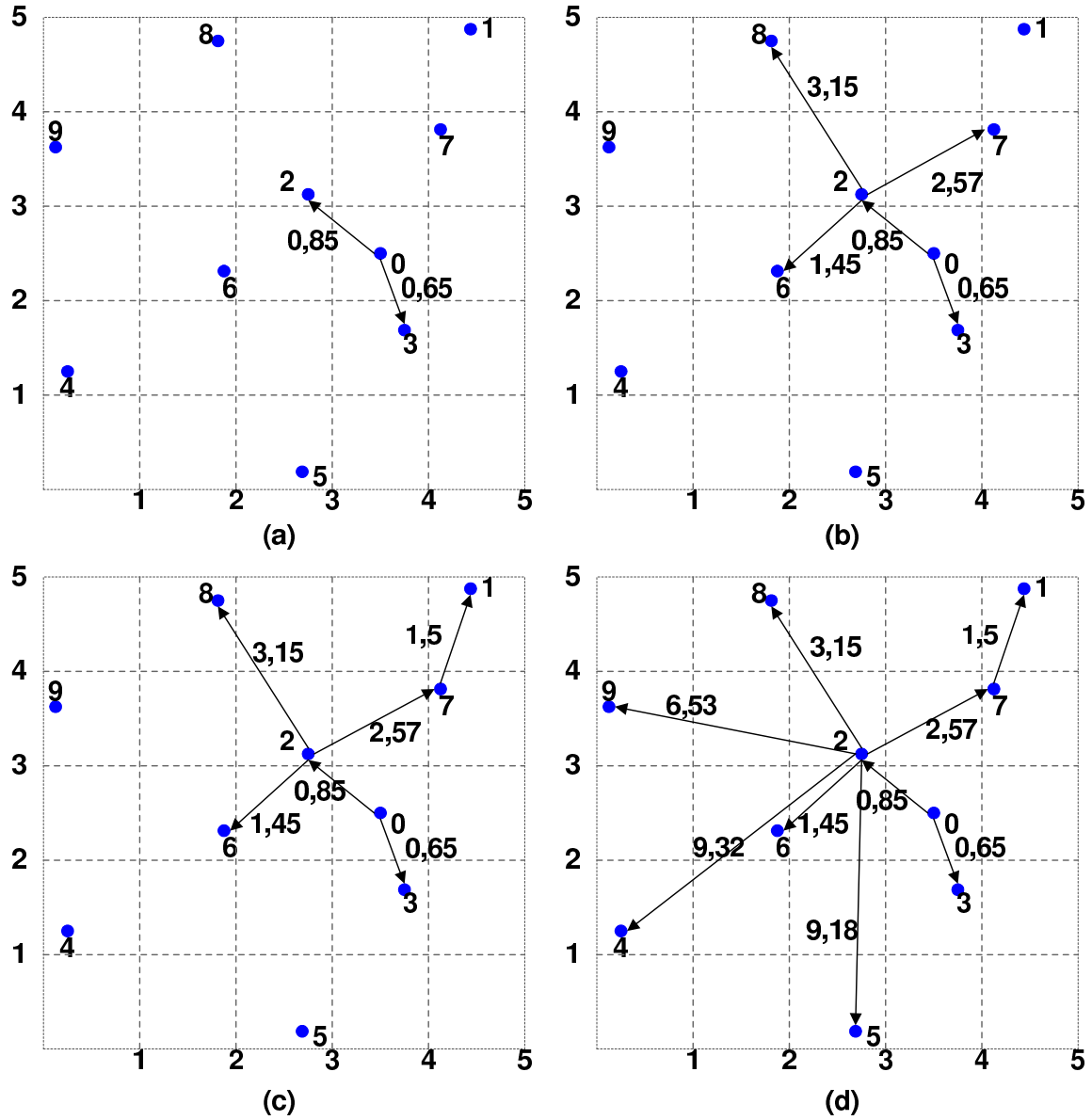

Figura 2.11: Árvore construída usando DSPF ($\alpha = 2$), sendo (a) transmissão de 0 para 2; (b) transmissão de 2 para 8; (c) transmissão de 7 para 1 e (d) a árvore final com custo total 11, 67.

construção da árvore, pois o algoritmo permite a inclusão de vários nós em uma única iteração, ao contrário do BIP, que só inclui um nó em cada iteração.

2.2.2 Novas heurísticas construtivas randomizadas

Algoritmo rSPF

Este algoritmo é baseado no algoritmo SPF descrito na seção anterior. A diferença em relação ao SPF é que rSPF (*Random Shortest Path First*), ao invés de escolher o

candidato de menor custo, constrói uma lista de candidatos a serem incluídos na árvore em ordem crescente de custo e, em cada fase, adiciona à árvore de transmissão um nó escolhido aleatoriamente a partir de um subconjunto restrito formado pelos melhores nós (aqueles com os menores custos) que compõem a lista de candidatos. Este subconjunto recebe o nome de lista de candidatos restrita (LCR). A LCR, a partir da qual um nó será selecionado aleatoriamente, é definida em função de um parâmetro $0 \leq \beta \leq 1$, compreendendo todos os elementos que ainda não fazem parte da solução e cujo impacto no custo total esteja no intervalo $[c_{min}, \beta(c_{max} - c_{min}) + c_{min}]$, onde c_{min} é o custo referente à inclusão do nó de menor custo (correspondente à seleção gulosa) e c_{max} é o custo da inclusão do nó que implica no maior aumento no custo total da solução. Logo, o parâmetro β controla o nível de aleatoriedade da escolha, pois determina o tamanho da lista de candidatos restrita. Observe-se que $\beta = 0$ implica em uma escolha puramente gulosa, enquanto que $\beta = 1$ implica em escolhas totalmente aleatórias. O algoritmo rSPF continua até que todos os nós sejam inseridos na árvore. Assim como em SPF, a propriedade WMA também é ignorada na construção da árvore, sendo levada em consideração apenas na avaliação do custo total da árvore.

A Figura 2.12 a seguir ilustra o pseudocódigo do algoritmo rSPF. A determinação do caminho mais curto para cada destino fora da árvore (linha 5) tem complexidade $O(n^2)$. A montagem da lista ordenada (linha 6) tem complexidade $O(n \log n)$. O passos 7 a 9 têm complexidade $O(1)$. Logo, a complexidade dos passos 3 a 10 do algoritmo é $O(n^3)$. A totalização do custo final (passo 11) tem complexidade $O(n)$. Portanto, a complexidade de rSPF é $O(n^3)$, semelhante à complexidade de sua versão não aleatorizada.

```
procedimento rSPF (β)
 1.  Criar uma árvore de transmissão T sem nó algum;
 2.  Inserir o nó fonte na árvore T;
 3.  enquanto (|T| < n) faça
 4.  início
 5.     Encontrar o caminho mais curto para cada nó destino fora da árvore;
 6.     Montar uma lista ordenada por menor custo dos candidatos;
 7.     Criar a LCR e selecionar aleatoriamente um nó dessa lista;
 8.     Atualizar a árvore T, adicionando o nó escolhido;
 9.     Atualizar o predecessor do nó adicionado;
10.  fim - enquanto
11.  Retornar a árvore T, totalizando seu custo final adotando a propriedade WMA;
Fim rSPF
```

Figura 2.12: Pseudocódigo do algoritmo rSPF.

A Figura 2.13 mostra a árvore de transmissão produzida pelos algoritmos SPF e rSPF. Para este exemplo, a sequência dos nós adicionados à arvore utilizando rSPF foi 0, 7, 2, 6, 4, 9, 1, 8, 3 e 5 e a potência requerida para manter a árvore é $P_0 + P_2 + P_3 + P_6 + P_7 = \max(c_{02}; c_{03}; c_{07}) + \max(c_{26}; c_{28}) + c_{35} + \max(c_{64}; c_{69}) + c_{71} = \max(0,85; 0,65; 2,01) + \max(1,45; 3,15) + 3,92 + \max(3,54; 5,06) + 1,50 = 15,64$, sendo 0 o nó fonte e 2, 3, 6 e 7 os nós retransmissores.

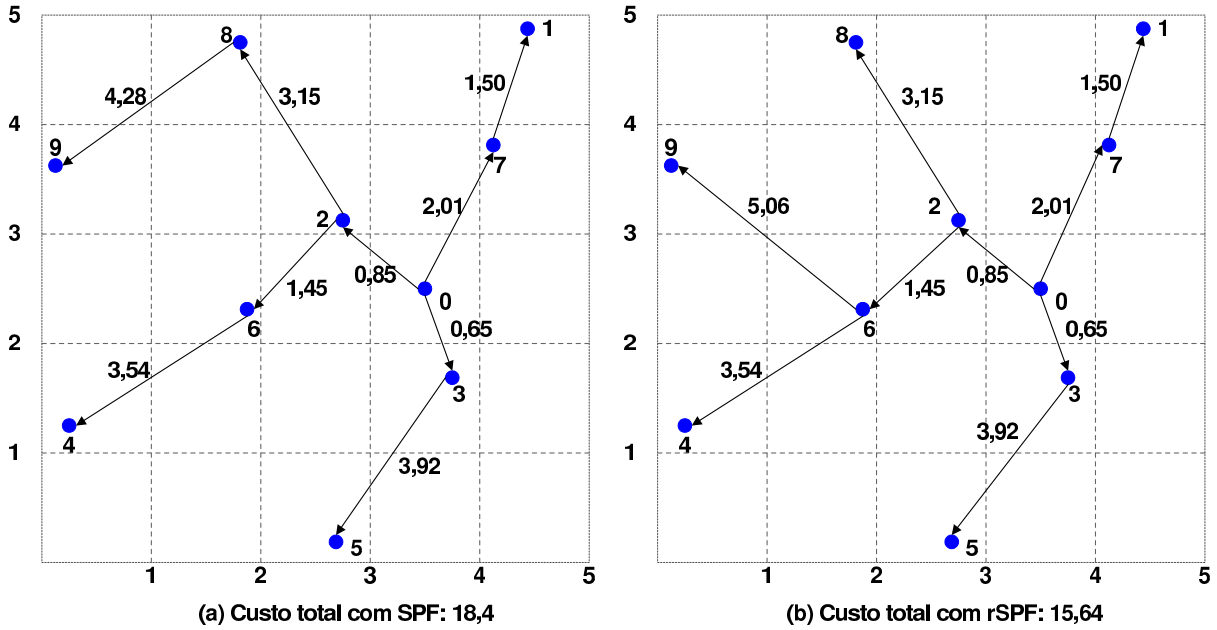

Figura 2.13: Comparação entre a árvore construída por (a) SPF e (b) rSPF com $\alpha = 2$.

Algoritmo rBIP

O algoritmo rBIP (*Random Broadcast Incremental Power*) é a versão aleatorizada do algoritmo BIP. A diferença em relação a BIP é que o algoritmo rBIP constrói uma lista de candidatos a serem incluídos na árvore, em ordem crescente dos custos incrementais $c'_{ij} = c_{ij} - P_i$ de inclusão na árvore, cria uma lista de candidatos restrita (LCR) formada pelos melhores nós (aqueles com os menores custos incrementais) e, em cada fase, adiciona à árvore de transmissão um nó escolhido aleatoriamente dentre os candidatos pertencentes à LCR. O tamanho da lista de candidatos restrita é determinado pelo parâmetro β, tal como descrito em rSPF. O algoritmo rBIP também atualiza dinamicamente os custos em cada passo (quando um novo nó é adicionado à arvore), para refletir o fato de que o custo de adicionar novos nós a uma lista de vizinhos do nó transmissor é um valor incremental. O algoritmo continua até que todos os nós sejam inseridos na árvore. A Figura 2.14 a seguir ilustra o pseudocódigo do algoritmo rBIP. A única linha que difere da versão gulosa na Figura 2.6 é a linha 12.

A função auxiliar `Escolher_ligação_dentre_as_mais_próximas` (T, c', β), ao invés de identificar a inserção com o menor custo adicional, determina os caminhos mais curtos para cada nó destino fora da árvore, monta uma lista com todos os candidatos em ordem crescente de custo incremental, cria uma lista de candidatos restrita formada pelos melhores nós e escolhe um nó aleatoriamente dessa lista. Nesta função, a determinação dos caminhos mais curtos para cada nó destino tem complexidade $O(n^2)$, a montagem da lista dos nós candidatos tem complexidade $O(n \log n)$ e a escolha do candidato tem complexidade $O(1)$. Assim, a complexidade da função que escolhe um candidato dentre os de menor custo também é $O(n^2)$. A complexidade dos passos 3 a 9 de rBIP é $O(n^2)$, o ciclo dos passos 10 a 22 do algoritmo tem complexidade $O(n^3)$ e a totalização do custo final (passo 23) tem complexidade $O(n)$. Logo, a complexidade de rBIP é $O(n^3)$, semelhante à complexidade de sua versão não aleatorizada.

```
procedimento rBIP (β)
 1.  Criar uma árvore de transmissão T sem nó algum;
 2.  Inserir o nó fonte na árvore T;
 3.  para i = 0,...,n−1 faça
 4.  início
 5.      para j = 0,...,n−1 e j ≠ i faça
 6.      início
 7.          c'_ij = c_ij;
 8.      fim - para
 9.  fim - para
10.  enquanto (|T| < n) faça
11.  início
12.      (I, J) ← Escolher_ligação_dentre_as_mais_próximas (T, c', β);
13.      Atualizar a árvore T, adicionando o nó J;
14.      Atualizar o predecessor do nó J;
15.      para ∀j ∉ T faça
16.      início
17.          se (c_ij > P_i)
18.              c'_ij = c_ij − P_i;
19.          senão
20.              c'_ij = 0;
21.      fim - para
22.  fim - enquanto
23.  Retornar a árvore T, totalizando seu custo final;
Fim rBIP
```

Figura 2.14: Pseudocódigo do algoritmo rBIP.

A Figura 2.15 mostra a árvore de transmissão produzida pelos algoritmos BIP e rBIP. A sequência dos nós adicionados à arvore por rBIP foi 0, 3, 2, 7, 6, 4, 5, 9, 8 e 1 e a potência requerida para mantê-la é $P_0 + P_6 + P_7 = \max(c_{02}; c_{03}; c_{06}; c_{07}) + \max(c_{64}; c_{65}; c_{68}; c_{69}) +$

$c_{71} = \max(0,85; 0,65; 2,38; 2,01) + \max(3,54; 5,03; 6,42; 5,06) + 1,50 = 10,3$, sendo 0 o nó fonte e 6 e 7 os nós retransmissores.

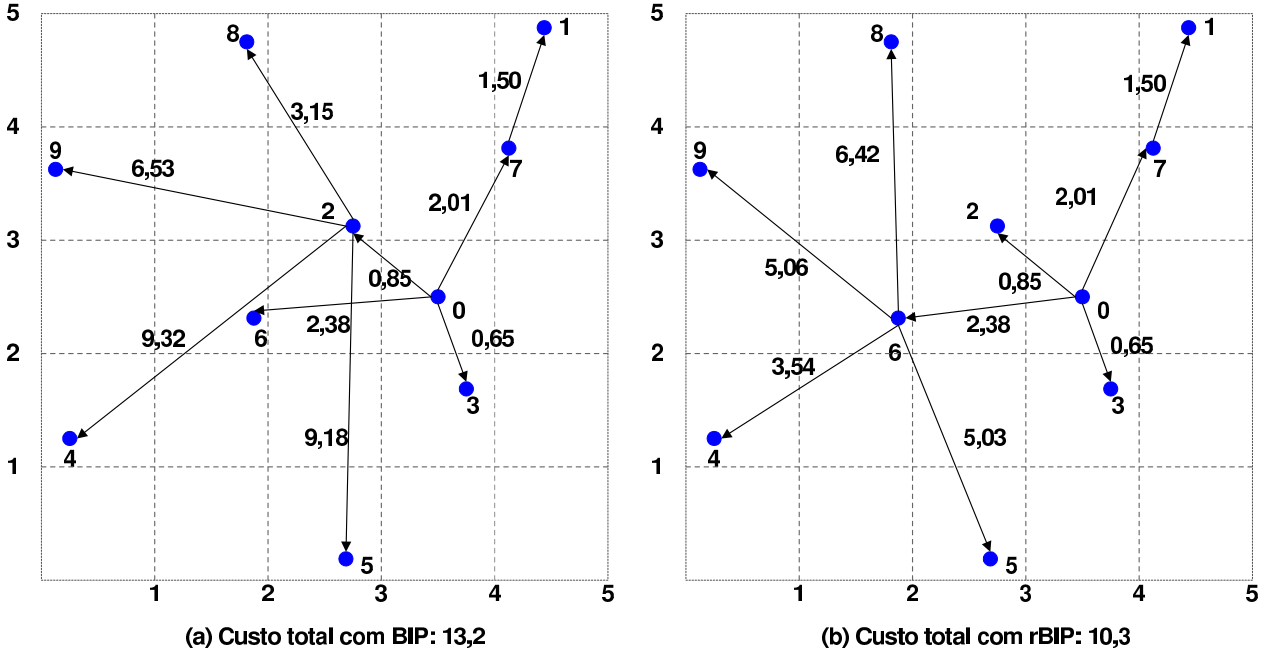

Figura 2.15: Comparação entre a árvore construída por (a) BIP e (b) rBIP com $\alpha = 2$.

Algoritmo rDSPF

O algoritmo rDSPF (*Random Densest Shortest Path First*) é a versão aleatorizada do algoritmo DSPF, descrito na seção anterior. A diferença em relação a DSPF é que rDSPF constrói uma lista de candidatos a serem incluídos na árvore, em ordem crescente da razão entre o custo da inclusão de um nó e o número de novos nós que seriam alcançados caso ele fosse escolhido, constrói uma lista de candidatos restrita, cujo tamanho é determinado pelo parâmetro β, formada pelos nós com as menores razões e, em cada fase, adiciona à árvore de transmissão um nó escolhido aleatoriamente dessa lista de candidatos restrita.

A Figura 2.16 a seguir ilustra o pseudocódigo do algoritmo *rDSPF*. A única diferença em relação à versão gulosa ocorre na linha 12. Enquanto na versão gulosa determina-se a ligação com razão mínima, nesta versão escolhe-se uma ligação aleatoriamente dentre as pertencentes à lista de candidatos restrita formada pelos nós com as menores razões (com complexidade $O(n^2)$). Os passos 3 a 9 têm complexidade $O(n^2)$ e o ciclo dos passos 10 a 22 tem complexidade $O(n^3)$. A totalização do custo final (passo 23) tem complexidade $O(n)$. Logo, a complexidade de rDSPF, assim como a de DSPF, é $O(n^3)$.

A Figura 2.17 mostra a árvore de transmissão produzida pelos algoritmos DSPF e rDSPF. A sequência dos nós adicionados à arvore por rDSPF foi 0, 2, 6, 3, 8, 7, 9, 1, 4

```
procedimento rDSPF (β)
1.  Criar uma árvore de transmissão T sem nó algum;
2.  Inserir o nó fonte na árvore T;
3.  para i = 0,...,n−1 faça
4.  início
5.     para j = 0,...,n−1 e j ≠ i faça
6.     início
7.        c'_ij ← c_ij;
8.     fim - para
9.  fim - para
10. enquanto (|T| < n) faça
11. início
12.    (I,J) ← Escolher_ligação_dentre_as_com_razão_mínima (T,c',β);
13.    Atualizar a árvore T, adicionando os nós alcançados;
14.    Atualizar o predecessor dos nós adicionados;
15.    para ∀j ∉ T faça
16.    início
17.       se (c_ij > P_i)
18.          c'_ij = c_ij − P_i;
19.       senão
20.          c'_ij = 0;
21.    fim - para
22. fim - enquanto
23. Retornar a árvore T, totalizando seu custo final;
Fim rDSPF
```

Figura 2.16: Pseudocódigo do algoritmo rDSPF.

e 5. A potência requerida para mantê-la é $P_2 + P_0 = \max(c_{21}; c_{24}; c_{25}; c_{26}; c_{27}; c_{28}; c_{29}) + \max(c_{02}; c_{02}) = \max(6,45; 9,32; 9,18; 1,45; 2,57; 3,15; 6,53) + \max(0,85; 0,65) = 10,17$, sendo 0 o nó fonte e 2 o nó retransmissor.

2.3 Algoritmo de melhoria EWMA

Cagalj et al.[13] desenvolveram uma heurística denominada EWMA (*Embedded Wireless Multicast Advantage*) que calcula soluções de boa qualidade para o MPB. Este algoritmo utiliza como entrada uma solução viável (uma árvore de transmissão por difusão T), efetua a troca de alguns ramos (um ramo representa uma transmissão de um nó para outro) por novos, de forma que a potência total da nova árvore seja menor que a inicial, transformando-a em uma nova árvore de transmissão. Isto é feito de tal forma que a viabilidade das soluções obtidas permanece intacta.

Antes de descrever o funcionamento do algoritmo é necessário definir os conceitos de filho, pai e ganho. Por definição, todo nó j de uma rede de transmissão (com exceção do

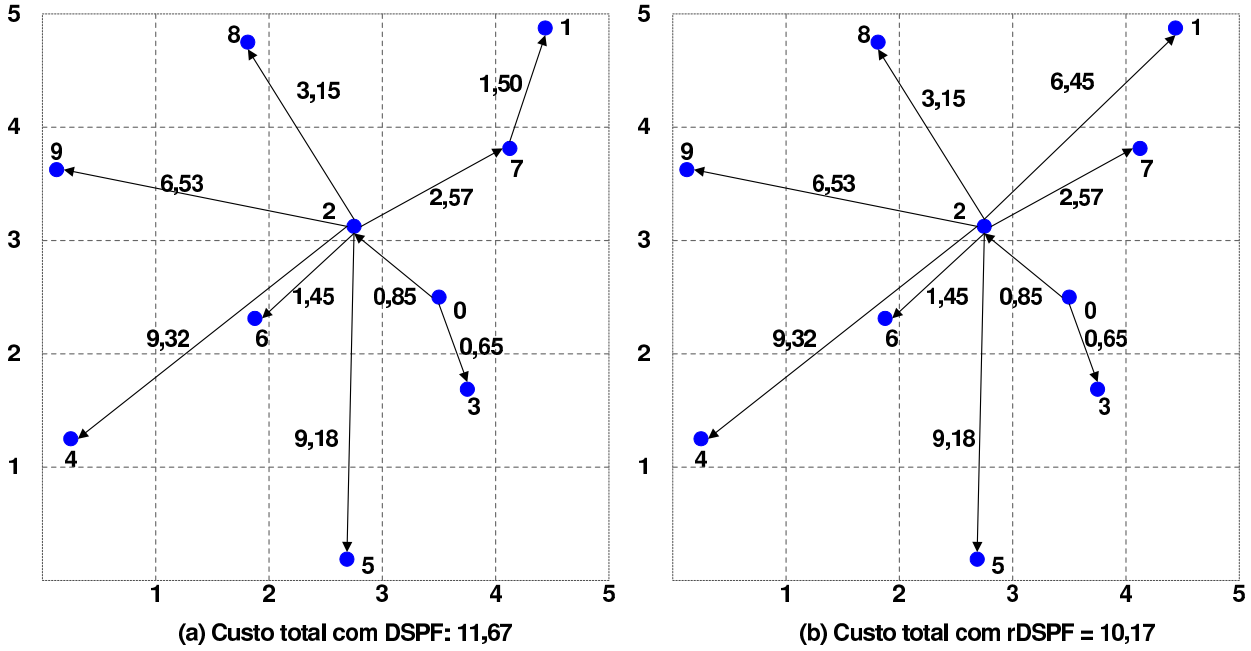

Figura 2.17: Comparação entre a árvore construída por (a) DSPF e (b) rDSPF com $\alpha = 2$.

nó fonte, que não possui pai) só pode ter um pai, que é o nó escolhido para realizar a transmissão para j. O conjunto de filhos de um nó transmissor i é o conjunto de nós que são alcançados pela transmissão por i. Define-se *ganho* como a diferença na potência total da árvore antes e após uma troca de ramos. Em EWMA, essa noção de ganho é usada como critério para a seleção dos nós transmissores na árvore de transmissão. O algoritmo utiliza três conjuntos auxiliares: C, conjunto dos nós alcançados pela rede de transmissão, F, conjunto dos nós transmissores na árvore final, e E, conjunto de nós excluídos (isto é, nós que eram transmissores na árvore inicial e que deixaram de sê-lo na árvore final). Inicialmente, o conjunto C contém apenas o nó fonte e os conjuntos F e E estão vazios.

EWMA começa a reconstruir a árvore a partir dos nós no conjunto $C - F - E$, determinando seus respectivos ganhos. O ganho decorrente do aumento da potência de transmissão de um nó i é obtido calculando-se o decréscimo na potência total da árvore de transmissão resultante da exclusão de nós do conjunto de transmissores na árvore inicial (onde exclusão significa um nó transmissor passar a ser nó folha), em troca do aumento da potência de transmissão do nó i. Este acréscimo de potência deve ser suficiente para alcançar todos os nós que eram cobertos previamente pelos que foram excluídos. Conseqüentemente, a viabilidade da solução é preservada.

O algoritmo inicia com o nó fonte no conjunto C e determina o ganho decorrente da exclusão de cada um dos nós transmissores da solução inicial. Ele seleciona o nó j que resulta no maior ganho positivo e adiciona ao conjunto E todos os nós que o aumento

da potência do nó fonte permite excluir do conjunto de nós transmissores. Em outras palavras, se o aumento da potência do nó fonte, que permite abranger todos os filhos de j, passar também a alcançar os filhos de outros nós transmissores, então estes nós transmissores também podem deixar de transmitir e serão incluídos no conjunto E. A seguir, o nó fonte é adicionado ao conjunto F com a potência que maximiza o ganho.

No segundo estágio, o conjunto C contém o nó fonte e todos o nós que ele alcança com a nova potência. O conjunto F contém o nó fonte e o conjunto E contém os nós que foram excluídos (nós que deixaram de ser transmissores devido ao aumento da potência do nó fonte). Se nenhum dos nós do conjunto $C - F - E$ tem ganho positivo, EWMA seleciona dentre eles o de menor ganho negativo, ou seja, aquele cuja exclusão acarreta o menor acréscimo na potência total da árvore. Esse procedimento repete-se até que todos os nós da rede sejam cobertos. O algoritmo não efetua uma busca exaustiva sobre todas as combinações possíveis de nós transmissores.

Sejam P_i a potência atual de transmissão do nó i e c_{ik} a potência de transmissão necessária para o nó i alcançar o nó k. Considere ΔP_i^j como o acréscimo na potência de transmissão do nó i com a exclusão do nó transmissor j, obtido pela diferença entre a potência de transmissão necessária para alcançar todos os filhos do nó j e a potência atual do nó i. Seja g_i^j o ganho na potência total da árvore decorrente do aumento na potência do nó i necessário para a exclusão do nó j, obtido pela soma das potências dos nós que deixaram de ser transmissores abatida do acréscimo (ΔP_i^j) na potência do nó i.

A árvore construída pelo algoritmo SPF, e apresentada na Figura 2.18 (a), será utilizada para exemplificar o funcionamento do algoritmo. A potência total inicial da árvore é $18,4$. Inicialmente, $C = \{0\}$ e $F = E = \emptyset$. A seguir, calcula-se o ganho do acréscimo da potência de transmissão dos nós pertencentes ao conjunto $C - F - E$, devido ao decréscimo da potência de cada um dos nós transmissores na solução de entrada. Ou seja, calcula-se o ganho do nó 0 (único pertencente ao conjunto $C - F - E$), devido à exclusão dos nós 2, 3, 6, 7 e 8 (nós transmissores na solução inicial).

O acréscimo de potência do nó 0 para alcançar os filhos do nó 2 (nós 6 e 8) é dado por $\Delta P_0^2 = \max_{i \in \{8,6\}} (c_{0,i}) - P_0 = 6,92 - 2,01 = 4,91$. Ou seja, a retirada do nó 2 acarretaria um aumento de $4,91$ na potência do nó fonte 0, de forma que ele possa alcançar os filhos do nó 2 e a árvore continue conectada. Em contrapartida, caso o nó 0 aumente sua potência para $6,92$, ele passará a alcançar, não só os filhos do nó 2, como também os filhos do nó 3 (nó 5) e do nó 7 (nó 1), pois as potências necessárias para transmitir do nó 0 para os nós 5 e 1 são, respectivamente, $6,06$ e $6,81$, ambas inferiores a $6,92$. Assim, o

ganho resultante do aumento de potência do nó 0, devido à exclusão do nó 2, é dado pela soma das potências economizadas (nós que deixam de transmitir) abatida do acréscimo de potência do nó 0: $g_0^2 = P_2 + P_3 + P_7 - \Delta P_0^2 = 3,15 + 3,92 + 1,50 - 4,91 = 3,66$.

Analogamente, o acréscimo da potência do nó 0 com a exclusão do nó 3 (filho: 5) é $\Delta P_0^3 = c_{0,5} - P_0 = 6,60 - 2,01 = 3,59$; com a exclusão do nó 6 (filho: 4) é $\Delta P_0^6 = c_{0,4} - P_0 = 11,41 - 2,01 = 9,40$; com a exclusão do nó 7 (filho: 1) é $\Delta P_0^7 = c_{0,1} - P_0 = 6,81 - 2,01 = 4,80$; e com a exclusão do nó 8 (filho: 9) é $\Delta P_0^8 = c_{0,9} - P_0 = 11,61 - 2,01 = 9,60$.

O cálculo do ganho devido à exclusão do nó 3 é dado por $g_0^3 = P_3 - \Delta P_0^3 = 3,92 - 3,59 = 0,33$; com a exclusão de 6 é $g_0^6 = P_2 + P_3 + P_6 + P_7 - \Delta P_0^6 = 3,15 + 3,92 + 3,54 + 1,50 - 9,40 = 2,71$; com a exclusão de 7 é $g_0^7 = P_3 + P_7 - \Delta P_0^7 = 3,92 + 1,50 - 4,80 = 0,62$; e com a exclusão de 8 é $g_0^8 = P_2 + P_3 + P_6 + P_7 + P_8 - \Delta P_0^8 = 3,15 + 3,92 + 3,54 + 1,50 + 4,28 - 9,60 = 6,78$. O nó que resulta no maior ganho positivo é o nó 8, pois sua exclusão acarreta também a exclusão dos nós 2, 3, 6 e 7 e traz um ganho na potência total da árvore de $6,78$.

Assim, adicionam-se todos os nós que esse nó exclui (nós que deixam de ser transmissores) no conjunto E, ou seja, $E = \{2, 3, 6, 7, 8\}$ e atualiza-se a potência destes nós para zero. Então, o nó fonte é adicionado ao conjunto F com a potência que maximiza o ganho. A seguir, atualiza-se o conjunto C com o nó fonte e todos os nós que ele alcança com a nova potência: $C = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Como ao final desta etapa todos os nós da rede já estão cobertos, o algoritmo termina. A potência total da nova árvore é $11,61$, cerca de 37% menor que a da árvore inicial. As Figuras 2.18 e 2.19 apresentam, respectivamente, a árvore antes e após a execução de EWMA e o pseudocódigo do algoritmo EWMA.

Para determinar a complexidade de EWMA, Cagalj et al.[13] definem d como o tamanho da maior vizinhança (isto é, o grau máximo de um nó ou número máximo de filhos de um nó) e q como o número total de nós transmissores na árvore inicial (isto é, $|F| = q$) e mostram que o tempo de processamento de EWMA é limitado por $O(d^4q^2)$. Isto indica que a complexidade de EWMA é inferior à complexidade dos algoritmos construtivos, fato que poderá ser observado nos resultados experimentais da Seção 4.5.3. O algoritmo EWMA, descrito acima, será utilizado em alguns algoritmos do próximo capítulo.

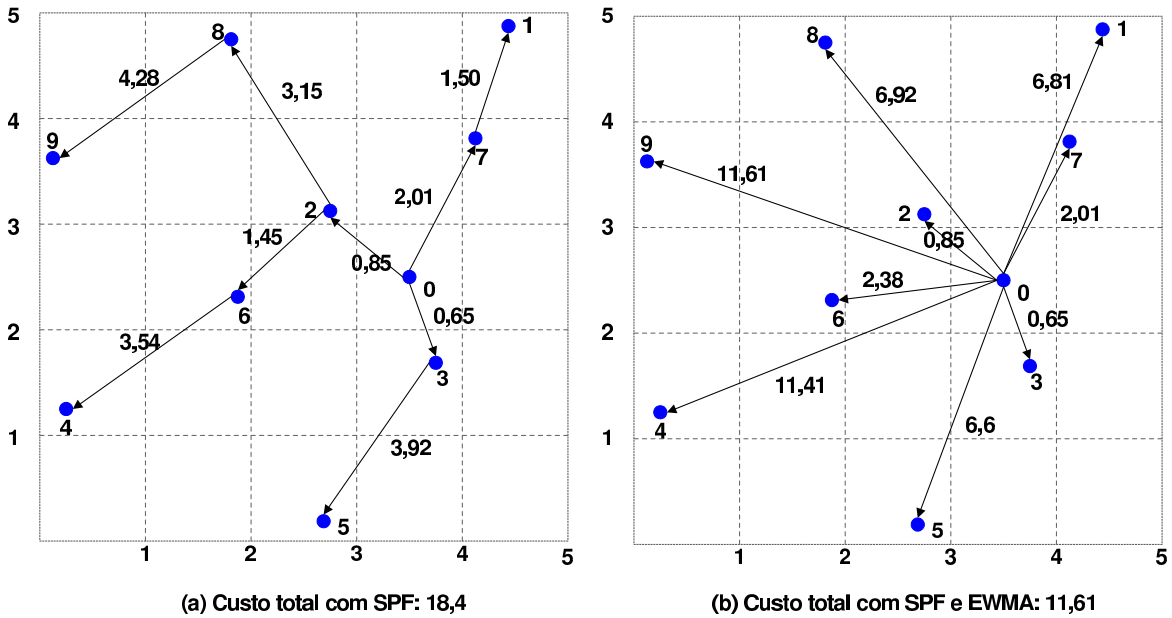

Figura 2.18: Comparação entre a árvore construída por (a) SPF e (b) SPF com EWMA com $\alpha = 2$.

```
procedimento EWMA (T)
1.  Inserir o nó fonte no conjunto C;
2.  Inicializar os conjuntos E e F;
3.  enquanto (|C| < n) faça
4.  início
5.     para todos os nós transmissores j = 1,...q na solução corrente faça
6.     início
7.        Calcular ΔP_i^j e o ganho g_i^j de cada nó i ∈ C − F − E;
8.     fim - para
9.     Escolher o nó j que acarrete o maior ganho do nó i;
10.    Atualizar os conjuntos C, E e F;
11.    Atualizar a potência dos nós;
12. fim - enquanto
13. Retornar a árvore T, totalizando seu custo final;
Fim EWMA
```

Figura 2.19: Pseudocódigo do algoritmo EWMA.

2.4 Algoritmos de busca local

Algoritmos de busca local são construídos como uma estratégia de exploração do espaço de busca. Tomam como partida uma solução inicial, obtida a partir de um método construtivo e buscam, em cada iteração, uma melhoria sucessiva da solução corrente através da exploração da vizinhança. Terminam quando não existe solução vizinha aprimorante.

Na literatura existem poucos trabalhos publicados para o MPB utilizando estratégias

de busca local [28, 48, 64]. Nas seções a seguir serão apresentadas duas novas vizinhanças (além de uma baseada em vizinhança já discutida na literatura [28]) e os algoritmos de busca local propostos para o MPB.

2.4.1 Vizinhanças utilizadas

O fator crítico na modelagem de um algoritmo de busca local é a escolha da estrutura de vizinhanças, isto é, a maneira pela qual as vizinhanças são definidas. Em geral, quanto maior for o número de soluções vizinhas analisadas, maior é a chance de se encontrar soluções de melhor qualidade. Por outro lado, quanto maior é a vizinhança, mais tempo leva para procurar soluções na vizinhança em cada iteração. Por esta razão, uma vizinhança maior não necessariamente produz uma heurística mais efetiva, a menos que se possa procurar soluções numa vizinhança maior de uma forma muito eficiente.

Antes da definição das vizinhanças, faz-se necessário escolher a melhor representação das soluções para o problema. Para o MPB, a solução é representada por uma matriz de n linhas (cada linha representando um nó) por 2 colunas (uma coluna para a informação do predecessor (nó pai) e a outra para a potência de cada nó). O custo total da solução é obtido pela soma das potências de transmissão de todos os nós. Para os algoritmos de busca local, utiliza-se uma estrutura auxiliar adicional, de dimensão $n \times 2$, onde cada linha i representa um nó. A primeira coluna guarda o número de filhos de cada nó. A segunda traz a lista dos nós que são alcançados pela transmissão do nó i em ordem decrescente de custo.

A vizinhança de uma solução é formada pelas soluções alcançáveis a partir desta solução por um movimento, que consiste na mudança de um pequeno conjunto de componentes da solução. Serão implementadas três vizinhanças para os algoritmos de busca local (as duas primeiras são propostas e a última é baseada na literatura [28, 48]):

- *vizinhança completa*: para cada nó experimenta-se trocar seu pai atual (seu predecessor) por qualquer outro nó da rede. Assim, a única diferença entre duas soluções vizinhas é o pai de um único nó. O nó fonte, por definição, não tem pai definido e não participa das trocas de pai. Assim, se a rede possui um total de n nós, tenta-se efetuar $(n-1) \times (n-2)$ trocas, sendo $(n-1)$, o número de nós excetuando o nó fonte e $(n-2)$, o número de candidatos a pai com exceção do próprio nó e seu pai atual;

- *vizinhança nós folha*: é um subconjunto da vizinhança completa, onde as tentativas

de troca de pai nesta vizinhança são realizadas somente para os nós folha da rede (ou nós receptores). Assim, se houverem m nós folha, o tamanho total da vizinhança será $m \times (n-2)$;

- *vizinhança filhos mais custosos*: as tentativas de troca envolvem somente o pai do filho de maior custo (aquele que requer a maior potência do nó pai para alcançá-lo) de cada nó transmissor. Ou seja, para cada nó transmissor reduz-se sua potência de transmissão, de forma que ele alcance um nó a menos do que alcançava (desconectando seu filho mais custoso). Tenta-se realocar o nó que foi desconectado com essa redução a qualquer outro nó da rede. Se houverem q nós transmissores, o tamanho total da vizinhança será $q \times (n-2)$. Essa vizinhança é baseada na estratégia utilizada nos trabalhos [28, 48].

2.4.2 Implementações

Para cada uma das vizinhanças descritas na seção anterior, serão empregados dois procedimentos de busca local:

- Melhoria iterativa ou primeiro aprimorante - em cada iteração é selecionada a *primeira* solução aprimorante encontrada na vizinhança; e
- Descida mais rápida ou mais aprimorante - em cada iteração é selecionada a *melhor* solução aprimorante da vizinhança.

As seções a seguir descreverão os procedimentos efetuados para cada vizinhança e busca local, bem como a complexidade de cada um.

2.4.2.1 Mais aprimorante

Algoritmo BLB

O algoritmo BLB (Busca Local *Best* - busca local mais aprimorante com vizinhança completa) recebe como entrada uma solução fornecida por um algoritmo construtivo. Basicamente, o que o algoritmo faz é realizar trocas de ramos (tentando trocar, em cada iteração, o pai de um nó i por outro nó j). Para cada nó i, exceto o nó fonte (nó 0), e cada nó j, BLB verifica se todas restrições são atendidas com o uso da função auxiliar `Troca_atende_restrições`, conforme a Figura 2.20.

```
procedimento Troca_atende_restrições (i, j)
1.   troca ← falso; gain ← 0;
2.   se (i ≠ j) então
3.      se (pai(i) ≠ j) então
4.         se (i ≠ pai(j)) então
5.            se (troca é benéfica) então
6.               se (árvore é conexa com troca de pai(i) por j) então
7.                  troca ← verdadeiro;
8.                  Calcular gain;
9.   Retornar troca e gain;
Fim Troca atende restrições
```

Figura 2.20: Pseudocódigo do procedimento `Troca_atende_restrições`.

O procedimento `Troca_atende_restrições` efetua os seguintes testes:

- se o candidato j a pai do nó i não é o próprio nó i (linha 2);
- se o candidato j a pai do nó i não é o pai atual do nó i (linha 3);
- se o nó i não é o pai atual de j (linha 4);
- se a troca do pai atual de i por j é benéfica (isto é, se a exclusão de i da lista de filhos do seu pai atual e sua posterior inclusão na lista de filhos de j não acarreta aumento no custo total da árvore de transmissão - linha 5); e
- se a troca do pai de i por j não viola a conectividade da árvore (isto é, se a árvore permanece conexa, se a troca for realizada - linha 6). Caso a troca do pai de i por j satisfaça todas as restrições, calcula-se o ganho ($gain$) na potência total da árvore decorrente da troca (linha 8).

Caso qualquer um dos testes seja violado, a função retorna que a troca não satisfaz as restrições. A seguir, o algoritmo BLB abandona o par (i, j) atual e passa para o próximo candidato a vizinho (isto é, outra tentativa de troca é realizada). Caso a troca não viole nenhuma restrição, são guardados os índices dos elementos ($best\_i$ e $best\_j$) que causam a maior redução na potência total da árvore sem violar as restrições e calcula-se o ganho na potência total da árvore decorrente da troca ($bestgain$). Somente após examinar todos os vizinhos da solução inicial é que são atualizadas a solução e seu custo total, com a melhor troca encontrada na vizinhança. Após a atualização, o algoritmo retoma todo o processo a partir dessa nova solução. O algoritmo é finalizado quando nenhuma solução aprimorante na vizinhança da solução é encontrada. O pseudocódigo do algoritmo BLB é apresentado na Figura 2.21.

A complexidade total da função `Troca_atende_restrições` é $O(n)$, pois os passos 1 a 5 têm complexidade $O(1)$, o passo 6 tem complexidade $O(n)$ e o passo 7 é realizado em $O(1)$. Para determinar a complexidade de cada iteração da busca local BLB, é necessário identificar a complexidade dos passos 5 a 18 e a do ciclo 19 a 23. Os passos 5 a 18 têm complexidade $O(n-1) \times (O(n) \times O(n))$, isto é, $O(n^3)$, pois a complexidade da função auxiliar `Troca_atende_restrições` (linha 9) é $O(n)$. A complexidade do ciclo dos passos 19 a 23 é $O(\log n)$, pois a atualização da solução (linha 21) consiste na exclusão de i da lista de filhos de seu pai ($O(\log n)$), na sua inclusão na lista de filhos de j ($O(\log n)$) e na atualização do pai de i para j ($O(1)$). A atualização do custo total é resultado do custo da solução anterior abatido do ganho (*bestgain*) obtido com a troca ($O(1)$). Portanto, a complexidade de cada iteração da busca local BLB é $O(n^3)$.

```
procedimento BLB (T)
 1.  improve ← 1;
 2.  enquanto (improve = 1)
 3.  início
 4.     improve ← 0; bestgain ← −1;
 5.     para i = 1, ..., n − 1 faça
 6.     início
 7.        para j = 0, ..., n − 1 faça
 8.        início
 9.           troca, gain ← Troca_atende_restrições (i, j);
10.           se (troca) então
11.              se (gain > bestgain) então
12.              início
13.                 bestgain ← gain;
14.                 best_i ← i;
15.                 best_j ← j;
16.              fim - se
17.        fim - para
18.     fim - para
19.     se (bestgain > 0) então
20.     início
21.        Atualizar a árvore T e seu custo total;
22.        improve ← 1;
23.     fim - se
24.  fim - enquanto
25.  Retornar a árvore T;
Fim BLB
```

Figura 2.21: Pseudocódigo do algoritmo BLB.

Algoritmo BLB1

O Algoritmo BLB1 (busca local mais aprimorante com vizinhança nós folha) é semelhante ao algoritmo BLB com vizinhança completa. A diferença entre eles é a vizinhança considerada. Enquanto BLB tenta efetuar trocas de ramos em toda a vizinhança, BLB1 tenta realizar trocas de ramos somente dos nós folha da árvore de entrada (tentando trocar, em cada iteração, o pai de um nó folha). Assim como em BLB, o algoritmo também verifica se todas restrições são atendidas com o uso da função auxiliar `Troca_atende_restrições` descrita no item anterior (Figura 2.20) e, caso não seja violada nenhuma restrição, são guardados o ganho e os índices dos elementos que causam a maior redução na potência total da árvore sem violar as restrições. Caso qualquer um dos testes seja violado, a função auxiliar interrompe os testes e, a seguir, BLB1 abandona o par atual e passa para o próximo candidato a vizinho (isto é, outra tentativa de troca do pai de um nó folha é realizada).

Após examinar todos os vizinhos da solução corrente, são atualizadas a solução e seu custo total, com a melhor troca encontrada na vizinhança. Após a atualização, o algoritmo retoma todo o processo a partir dessa nova solução. O algoritmo é finalizado quando nenhuma solução aprimorante na vizinhança da solução é encontrada. O pseudocódigo do algoritmo BLB1 é apresentado na Figura 2.22.

Seja $folha$, um vetor de m posições, onde cada uma delas contém o índice de um nó folha, considerando-se que a solução corrente possui m nós folha. A complexidade de cada iteração da busca local BLB1 é $O(m) \times (O(n) \times O(n))$ (resultante do cômputo das linhas 5 a 18, pois a complexidade do ciclo dos passos 19 a 23 é a mesma do algoritmo anterior, isto é $O(\log n)$, o que resulta em $O(mn^2 + \log n) = O(mn^2)$. Portanto, a complexidade de cada iteração de BLB1 é $O(mn^2)$.

Algoritmo BLB2

O algoritmo BLB2 (busca local mais aprimorante com vizinhança filhos mais custosos) também é semelhante ao algoritmo BLB com vizinhança completa, com diferença apenas na vizinhança considerada. Enquanto BLB tenta efetuar trocas de ramos em toda a vizinhança, BLB2 tenta realizar trocas somente do ramo de maior custo de cada nó transmissor (tentando trocar, em cada iteração, o pai do filho de maior custo de um nó transmissor i por outro nó j, ou seja, reduz-se a potência do nó transmissor i e tenta-se realocar o nó que se desconectou devido a essa redução a outro nó qualquer). O algoritmo

```
procedimento BLB1 (T)
 1.  improve ← 1;
 2.  enquanto (improve = 1)
 3.  início
 4.     improve ← 0; bestgain ← −1;
 5.     para i = 1, ..., m faça
 6.     início
 7.        para j = 0, ..., n − 1 faça
 8.        início
 9.           troca, gain ← Troca_atende_restrições (i, j);
10.           se (troca) então
11.              se (gain > bestgain) então
12.              início
13.                 bestgain ← gain;
14.                 best_i ← folha[i];
15.                 best_j ← j;
16.              fim - se
17.        fim - para
18.     fim - para
19.     se (bestgain > 0) então
20.     início
21.        Atualizar a árvore T e seu custo total;
22.        improve ← 1;
23.     fim - se
24.  fim - enquanto
25.  Retornar a árvore T;
Fim BLB1
```

Figura 2.22: Pseudocódigo do algoritmo BLB1.

também verifica todas as restrições (usando a mesma função auxiliar da Figura 2.20) e, caso nenhuma seja violada, guarda os índices dos elementos que causam a maior redução na potência total da árvore sem violar as restrições. Caso qualquer um dos testes seja violado, o algoritmo abandona o par atual e passa para o próximo candidato a vizinho.

Após examinar todos os vizinhos da solução inicial, são atualizadas a solução, seu custo total e a lista de nós transmissores, efetuando-se a melhor troca encontrada na vizinhança. Após a atualização, o algoritmo retoma todo o processo a partir dessa nova solução. O algoritmo é finalizado quando nenhuma solução aprimorante na vizinhança da solução é encontrada. O pseudocódigo do algoritmo BLB2 é apresentado na Figura 2.23.

Sejam *transmissor* e $filho$, dois vetores de q posições cada, onde cada posição do primeiro vetor contém o índice de um nó transmissor e cada posição do segundo contém a identificação do filho mais custoso do nó transmissor i, considerando-se que a solução corrente possui q nós transmissores. A complexidade de cada iteração de BLB2 é $O(q) \times$

```
procedimento BLB2 (T)
 1.   improve ← 1;
 2.   enquanto  (improve = 1)
 3.   início
 4.      improve ← 0;  bestgain ← −1;
 5.      para  i = 1, ..., q  faça
 6.      início
 7.         para  j = 0, ..., n − 1  faça
 8.         início
 9.            troca,  gain ← Troca_atende_restrições (i, j);
10.            se (troca) então
11.               se (gain > bestgain)
12.               início
13.                  bestgain ← gain;
14.                  best_i ← filho[transmissor[i]];
15.                  best_j ← j;
16.               fim - se
17.         fim - para
18.      fim - para
19.      se (bestgain > 0) então
20.      início
21.           Atualizar a árvore T, seu custo total e sua lista de transmissores;
22.           improve ← 1;
23.      fim - se
24.   fim - enquanto;
25.   Retornar a árvore T;
Fim BLB2
```

Figura 2.23: Pseudocódigo do algoritmo BLB2.

$(O(n) \times O(n))$ (linhas 5 a 18). A atualização da solução, do custo total e da lista de transmissores (linha 21) tem complexidade $O(\log n)$. Portanto, a complexidade total de cada iteração de BLB2 é $O(qn^2)$.

2.4.2.2 Primeiro aprimorante

Algoritmo BLF

O algoritmo BLF (Busca Local *First* - busca local primeiro aprimorante com vizinhança completa) é semelhante ao algoritmo BLB. A diferença é que BLB atualiza a solução somente após examinar todos os vizinhos da solução inicial e seleciona o melhor. Em BLF, caso uma troca de ramos não viole nenhuma restrição e melhore a solução atual, a solução e seu custo são atualizados e o algoritmo reinicia o processo de busca local a partir dessa nova solução. Se a troca acarreta violação de alguma restrição, a busca passa

ao próximo candidato a vizinho. O algoritmo termina quando não encontra nenhuma solução aprimorante na vizinhança da solução.

Assim, a diferença básica entre BLF e BLB está no procedimento de retomada da busca. No primeiro algoritmo, basta encontrar um vizinho melhor do que a solução corrente para que a busca local seja realizada a partir desta nova solução, enquanto que, em BLB, percorre-se toda vizinhança para encontrar o vizinho que causa a maior redução no custo total da solução.

Embora tanto BLB quanto BLF tenham a mesma complexidade em cada iteração, $O(n^3)$, o tempo de processamento médio, que será apresentado posteriormente nas Tabelas 4.12 e 4.14, mostra que BLB é mais custoso que BLF, pois a cada iteração BLB escolhe o melhor vizinho para realizar a busca local. Freqüentemente, tanto a melhoria iterativa (primeiro aprimorante) quanto a descida mais rápida (melhor aprimorante) alcançam o mesmo ótimo local, porém a melhoria iterativa tende a levar menos tempo para convergir [58]. O pseudocódigo do algoritmo BLF é apresentado na Figura 2.24 a seguir.

```
procedimento BLF (T)
1.   improve ← 1;
2.   enquanto (improve = 1)
3.   início
4.      improve ← 0;
5.      para i = 1, ..., n − 1 faça
6.      início
7.         para j = 0, ..., n − 1 faça
8.         início
9.            troca, gain ← Troca_atende_restrições (i, j);
10.           se (troca) então
11.              se (gain > 0) então
12.              início
13.                 improve ← 1;
14.                 Atualizar a árvore T e seu custo total;
15.                 Reiniciar a busca a partir da nova árvore;
16.              fim - se
17.           fim - para
18.      fim - para
19.   fim - enquanto
20.   Retornar a árvore T;
Fim BLF
```

Figura 2.24: Pseudocódigo do algoritmo BLF.

Algoritmo BLF1

O algoritmo BLF1 (busca local primeiro aprimorante com vizinhança reduzida) é semelhante ao algoritmo BLF com vizinhança completa, com diferença apenas na vizinhança considerada. Enquanto BLF tenta efetuar trocas de ramos em toda a vizinhança, BLF1 tenta realizar trocas de ramos somente dos nós folha (tentando trocar, em cada iteração, o pai de um nó folha). Com exceção da vizinhança, as demais características de BLF1 são idênticas às de BLF. O pseudocódigo do algoritmo BLF1 será omitido por ser semelhante ao da Figura 2.24, susbstituindo-se apenas os índices de i na linha 5 por $i = 1, \ldots, m$, considerando que a solução possui m nós folha. Assim como BLB1, cada iteração da busca local BLF1 tem complexidade $O(mn^2)$.

Algoritmo BLF2

O algoritmo BLF2 (busca local primeiro aprimorante com vizinhança filhos mais custosos) também é semelhante ao algoritmo BLF com vizinhança completa, diferindo apenas na vizinhança considerada. BLF2 tenta realizar trocas somente dos ramos de maior custo de cada nó transmissor (tentando trocar, em cada iteração, o pai do filho mais custoso de um nó transmissor, assim como BLB2), ao contrário de BLF, que tenta efetuar trocas em toda a vizinhança. O pseudocódigo do algoritmo BLF2 será omitido por ser semelhante ao da Figura 2.24, substituindo-se os índices de i na linha 5 por $i = 1, \ldots, q$, considerando que a solução possui q nós transmissores e acrescentando, na linha 14, a atualização da lista de nós transmissores. Assim como BLB2, cada iteração da busca local BLF2 tem complexidade $O(qn^2)$.

2.4.2.3 Circularização e aleatorização da ordem dos nós

A circularização é uma estratégia que pode ser incorporada ao algoritmos de busca local como forma de aceleração do processo de busca. Em um procedimento usual de busca local primeiro aprimorante, ao encontrar um vizinho aprimorante, a busca local é interrompida, a solução é atualizada e reinicia-se o processo de busca local a partir desta solução, avaliando-se a vizinhança da nova solução. Por exemplo, suponha que o algoritmo BLF encontre o primeiro vizinho aprimorante em $i = 4$ e $j = 3$. Ao reiniciar a busca, após a atualização da solução, o algoritmo recomeça a busca retornando a $i = 1$ e $j = 0$ (início do ciclo da busca local, que começa em $i = 1$, porque o nó fonte, $i = 0$, não pode ter seu pai trocado). Com a circularização, ao invés de recomeçar o processo

de busca local do início, inicia-se de onde se parou (ponto onde foi encontrada a solução aprimorante). No exemplo, a nova busca iniciaria em $i = 4$ e $j = 4$. Caso nenhuma solução aprimorante seja encontrada até o final da vizinhança ($i = n - 1$ e $j = n - 1$), avaliam-se as soluções do início do ciclo da busca local ($i = 1$, $j = 0$) até o ponto em que se parou a busca anterior ($i = 4$ e $j = 3$). Assim, a vizinhança é percorrida de forma circular.

A circularização foi introduzida no algoritmo primeiro aprimorante com vizinhança completa (BLF), resultando no algoritmo BLFc (busca local primeiro aprimorante com vizinhança completa).

A complexidade de cada iteração de BLFc é a mesma de BLF, pois ambos consideram a mesma vizinhança, $O(n^3)$. Entretanto, BLFc na prática converge muito mais rápido para um ótimo local e produz resultados tão bons quanto sua versão não circularizada, BLF. Estes resultados serão apresentados posteriormente nas Tabelas 4.13 e 4.14 da Seção 4.4 desta dissertação.

A circularização também foi introduzida no algoritmo BLF2, resultando no algoritmo BLF2c (busca local primeiro aprimorante circular com vizinhança reduzida). Assim como BLF2, BLF2c também tem complexidade de cada iteração $O(qn^2)$ (q é o número de nós transmissores na solução), pois ambos consideram a mesma vizinhança. Entretanto, BLF2c, na prática, converge muito mais rápido para um ótimo local e produz resultados tão bons quanto sua versão não circularizada, BLF2.

A aleatorização da ordem dos nós é uma estratégia que, ao ser incorporada à busca local, acarreta diversificação na busca pela vizinhança, permitindo visitar soluções que antes não seriam alcançadas caso se atingisse um ótimo local prematuramente. A aleatorização foi incorporada ao algoritmo BLFc e o algoritmo resultante foi denominado BLFcr (busca local primeiro aprimorante circular, com vizinhança completa e aleatorização na ordem de visitação dos nós). A aleatorização consiste em permutar a ordem dos nós, antes de iniciar o ciclo de busca local, ou seja, ao invés de se percorrer a vizinhança na ordem sequencial ($i = 1, \ldots, n - 1$ e $j = 0, \ldots, n - 1$), percorre-se a vizinhança na ordem definida pela permutação dos nós. A complexidade de BLFcr é semelhante à de BLFc, isto é, $O(n^3)$.

Capítulo 3

Metaheurísticas

3.1 GRASP

O procedimento de busca adaptativa gulosa randomizada (GRASP) [58] é uma metaheurística iterativa para problemas de otimização combinatória, composta de duas fases distintas. A primeira fase de cada iteração constrói uma solução viável, cuja vizinhança é explorada, na segunda através de um processo de busca local. A melhor solução obtida ao longo de todas as iterações GRASP efetuadas é retornada como a solução encontrada.

Na fase de construção, uma solução viável é construída, elemento a elemento. A cada iteração deste processo de construção, os próximos elementos candidatos a serem incluídos na solução são colocados em uma lista, seguindo um critério pré-determinado de ordenação. O processo de seleção do próximo elemento da lista a ser incluído na solução que está sendo construída é baseado em uma função adaptativa gulosa, que estima o benefício da seleção de cada um dos elementos. A componente probabilística do procedimento reside no fato de que o elemento é selecionado de forma aleatória, a partir de um subconjunto restrito formado pelos melhores elementos que compõem a lista de candidatos (não necessariamente o melhor deles, que seria aquele selecionado por um algoritmo puramente guloso). Este subconjunto recebe o nome de lista de candidatos restrita. Após o elemento ser incorporado à solução parcial, a lista de candidatos é atualizada e os custos incrementais são reavaliados.

Não se pode garantir a otimalidade local das soluções geradas pela fase de construção. Desta forma, quase sempre há um benefício em se aplicar uma busca local à solução

construída, embora também não se possa garantir sua otimalidade. A busca local atua de forma iterativa, através da substituição sucessiva da solução atual por uma solução melhor em sua vizinhança. O processo de busca local termina quando não há mais soluções aprimorantes na vizinhança sendo visitada. De um modo geral, a eficiência da busca local depende da qualidade da solução construída. O procedimento de construção tem um papel importante na busca local, uma vez que as soluções construídas constituem bons pontos de partida para a busca local, permitindo assim acelerá-la.

Seja β o parâmetro do algoritmo construtivo randomizado, que define o tamanho da lista de candidatos restrita, conforme descrito na Seção 2.2.2. Este é o único parâmetro a ser ajustado na implementação de um algoritmo GRASP. Feo e Resende [33] discutiram o efeito do valor de β na qualidade da solução e na diversidade das soluções geradas durante a fase de construção. Valores selecionados de β que levam a uma lista de candidatos restrita de tamanho muito limitado (ou seja, valor de β muito próximo da escolha gulosa) implicarão em soluções de qualidade muito próxima daquela puramente gulosa, com um esforço computacional proporcionalmente menor. Em contrapartida, provocam uma baixa diversidade de soluções construídas. Já uma escolha de β próxima da seleção aleatória leva a uma grande diversidade de soluções construídas. Por outro lado, muitas destas soluções terão qualidade inferior, tornando mais lento o processo de busca local.

O procedimento GRASP procura, portanto, conjugar bons aspectos dos algoritmos puramente gulosos com aqueles dos algoritmos aleatórios de construção de soluções.

No caso de MPB, sejam T uma árvore de transmissão obtida por um algoritmo construtivo, $f(T)$ seu custo total, $T^\star$ a árvore de transmissão de menor custo e $f^\star$ o custo total de $T^\star$. A Figura 3.1 a seguir ilustra o procedimento GRASP básico para MPB, sob a forma de pseudocódigo.

O algoritmo recebe como entrada o parâmetro do algoritmo construtivo aleatorizado (β) e o número de iterações ($numIter$). Cada iteração do GRASP consiste de uma fase de construção (linha 4), de uma fase de busca local (linha 5) e, se necessário, da atualização da melhor solução encontrada até o momento (linhas 8 e 9). A melhor solução encontrada é armazenada em $T^\star$.

Os algoritmos construtivos aleatorizados apresentados na Seção 2.2.2 podem ser utilizados na fase construtiva do procedimento GRASP, que consiste exatamente na aplicação do procedimento realizado pelos algoritmos construtivos aleatorizados, com β como parâmetro de entrada (em rDSPF, por exemplo, os nós que trazem o maior custo-benefício são os primeiros da lista e um deles é selecionado aleatoriamente em cada iteração dentre

```
procedimento GRASP (β, numIter)
 1.    f* ← ∞
 2.    para i = 1, ..., numIter faça
 3.    início
 4.       T ← Construir_Solução_Aleatória (β);
 5.       T ← Busca_Local (T);
 6.       se (f(T) < f*) então
 7.       início
 8.          f* ← f(T);
 9.          T* ← T;
10.       fim - se
11.    fim - para
12.    Retornar T*;
Fim GRASP
```

Figura 3.1: Pseudocódigo do algoritmo GRASP.

os nós pertencentes à lista de candidatos restrita, cujo tamanho é definido pelo parâmetro β). Caso $\beta = 0$, então o algoritmo construtivo aleatorizado escolherá em cada iteração o melhor elemento da lista de candidatos para entrar na solução, construindo uma solução puramente gulosa (tal com a solução gerada pelo algoritmo guloso DSPF, Seção 2.2). Por outro lado, $\beta = 1$ resultará numa escolha totalmente aleatória na lista dos candidatos que não violam a viabilidade da solução em construção.

Prais e Ribeiro [53] mostraram que o uso de um valor fixo para o parâmetro β freqüentemente impede a construção de soluções de melhor qualidade, que poderiam ser obtidas utilizando-se outros valores de β. Com base nestes resultados, decidiu-se investigar o comportamento do procedimento GRASP em função das seguintes estratégias de variação do parâmetro β:

- β fixo em todas as iterações do GRASP;
- β selecionado aleatoriamente a cada iteração, a partir de uma distribuição de probabilidades equiprovável; e
- β selecionado aleatoriamente a cada iteração, a partir de uma distribuição de probabilidades fixa.

Em todos os testes relatados no Capítulo 4 foi utilizado o algoritmo construtivo rDSPF (descrito na Seção 2.2.2) para a fase construtiva e os algoritmos de busca local BLF2c e BLFc (Seção 2.4.2.2) para a fase de busca local (algoritmos que obtiveram melhor desempenho em tempo de processamento e boa qualidade da solução). Como a complexidade do algoritmo construtivo rDSPF é $O(n^3)$ e a complexidade, em cada iteração, dos algoritmos

de busca local é $O(n^3)$ (BLFc e BLFcr) ou $O(qn^2)$ (BLF2c), a complexidade resultante do algoritmo GRASP em cada iteração (para todos os tipos de estratégia de variação de β considerados) é semelhante à complexidade da busca local utilizada.

3.2 Reconexão por caminhos

O algoritmo de reconexão por caminhos (*path relinking*) foi originalmente proposto por Glover [35] como uma estratégia de intensificação, conectando soluções de elite obtidas por busca tabu ou busca por dispersão (*scatter search*) [36, 37, 38]. Considerando-se duas soluções de elite, identifica-se o conjunto de movimentos que devem ser aplicados a uma solução para se chegar a outra, definindo-se uma trajetória. A exploração da trajetória que conecta soluções de alta qualidade (iniciando-se de uma dessas soluções, chamada de solução inicial, e gerando um caminho que se dirige à outra solução, chamada de solução guia) gera novas soluções, que podem ser de melhor qualidade do que a solução inicial e a guia. A reconexão por caminhos pode ser vista como uma estratégia que procura incorporar atributos de soluções de alta qualidade, favorecendo estes atributos nos movimentos selecionados. Seja $\Delta(X_1, X_2)$ o conjunto de movimentos que devem ser aplicados à solução inicial para alcançar a solução final, $X^\star$ a solução de menor custo, $f^\star$ o custo total de $X^\star$ e η o movimento escolhido a ser aplicado à solução X. A Figura 3.2 ilustra o pseudocódigo da reconexão por caminhos aplicada a um par de soluções (X_1, X_2).

```
procedimento Reconexão_por_caminhos (X1, X2)
1.    Calcular a diferença_simétrica Δ(X1, X2);
2.    f* ← min(f(X1), f(X2));
3.    X* ← argmin(f(X1), f(X2));
4.    X ← X1;
5.    enquanto (Δ(X, X2) ≠ ∅) faça
6.    início
7.       η* ← argmin(f(X ⊕ η), ∀η ∈ Δ(X, X2));
8.       X ← X ⊕ η*;
9.       se (f(X) < f*) então
10.      início
11.         f* ← f(X);
12.         X* ← X;
13.      fim - se
14.   fim - enquanto
15.   Retornar X*;
Fim Reconexão_por_caminhos
```

Figura 3.2: Pseudocódigo da reconexão por caminhos.

O algoritmo inicia calculando a diferença simétrica $\Delta(X_1, X_2)$ entre X_1 e X_2, resultando no conjunto de movimentos que devem ser aplicados à solução inicial para alcançar

a solução final (linha 1). A melhor solução $X^\star$ e seu custo $f^\star$ neste caminho é retornada pelo algoritmo (linha 15). Em cada passo, o algoritmo examina todos os movimentos $\eta \in \Delta(X, X_2)$ a partir da solução atual X e seleciona aquele que resultar na solução de custo mínimo, isto é, aquele que minimiza $f(X \oplus \eta)$, onde $X \oplus \eta$ é a solução resultante da aplicação do movimento η à solução X (linha 7). O melhor movimento $\eta^\star$ é realizado, produzindo a solução $X \oplus \eta^\star$ (linha 8). Se necessário, a melhor solução $X^\star$ e o menor custo $f^\star$ são atualizados (linhas 11 e 12).

Aplicando-se o algoritmo da Figura 3.2 a MPB, o cálculo da diferença simétrica (linha 1) é realizado em $O(n \log n)$, uma vez que, em cada iteração, compara-se o pai de cada nó i em uma solução com o pai da outra e, caso sejam diferentes, armazena-se o movimento e o custo decorrente da troca em tempo $O(\log n)$ numa fila de prioridade (o movimento de menor custo é armazenado na raiz). A escolha do melhor movimento (linha 7) é feita em $O(1)$, pois este é o primeiro elemento da lista de prioridade. A atualização da solução (linha 8) também tem complexidade $O(1)$. Assim, a complexidade total de uma aplicação de reconexão por caminhos é $O(n \log n)$.

Várias alternativas podem ser consideradas na reconexão por caminhos:

- *reconexão periódica*: a reconexão por caminhos não é realizada sistematicamente, mas sim, periodicamente;
- *reconexão para frente*: a reconexão por caminhos é realizada usando a solução de pior custo entre X_1 e X_2 como solução inicial e a outra, como solução alvo;
- *reconexão para trás*: usa-se a solução de melhor custo entre X_1 e X_2 como a solução inicial e a outra, como solução alvo;
- *reconexão para frente e para trás*: duas trajetórias diferentes são realizadas, uma partindo de X_1 e a outra de X_2;
- *reconexão mista*: são explorados dois caminhos simultaneamente, o primeiro partindo de X_1 e o segundo de X_2, até que eles se encontrem em uma solução equidistante de X_1 e X_2;
- *reconexão aleatorizada*: selecionar um movimento aleatoriamente dentre uma lista de candidatos com os melhores movimentos na trajetória entre as soluções inicial e guia, ao invés de selecionar o melhor movimento ainda não realizado; e
- *reconexão truncada*: somente uma parte da trajetória completa entre X_1 e X_2 é analisada.

Todas estas alternativas envolvem compromissos entre tempo computacional e qualidade da solução. Ribeiro et al. [60] observaram que explorar duas trajetórias diferentes para cada par (X_1, X_2) leva aproximadamente o dobro do tempo necessário para explorar uma delas, com ganhos marginais na qualidade da solução. Também observaram que se somente uma trajetória for investigada, soluções melhores são encontradas quando a reconexão por caminhos inicia da melhor solução entre X_1 e X_2. Como a vizinhança da solução inicial é muito mais cuidadosamente explorada do que a da solução guia, iniciar da melhor delas dá ao algoritmo mais chance de investigar com mais detalhes a vizinhança da solução mais promissora. Pela mesma razão, as melhores soluções são normalmente mais próximas da solução inicial do que da solução guia, permitindo finalizar a trajetória de religação antes da solução guia ser alcançada. Neste trabalho serão investigadas as estratégias de reconexão para frente e de reconexão para trás no GRASP híbrido descrito a seguir.

3.3 GRASP híbrido

Até recentemente, a maioria das implementações de GRASP assumia independência de suas iterações, não fazendo uso algum de estruturas de memória. A reconexão por caminhos é um procedimento capaz de adicionar memória ao procedimento GRASP básico, levando a melhorias significativas de qualidade e tempo da solução. O uso da reconexão por caminhos com o procedimento GRASP, como uma estratégia de intensificação aplicada a cada solução localmente ótima, foi primeiramente proposto por Laguna e Martí [42], sendo seguido por várias extensões, melhorias e aplicações bem sucedidas [2, 15, 57, 60]. Normalmente são utilizadas duas estratégias básicas:

- aplicar a reconexão por caminhos a todos os pares de soluções de elite, periodicamente durante as iterações do GRASP ou após a realização de todas as iterações do GRASP, como uma etapa de pós-otimização, ou
- aplicar a reconexão por caminhos a cada ótimo local obtido após a fase de busca local, como uma estratégia de intensificação.

A aplicação da reconexão por caminhos como uma estratégia de intensificação a cada ótimo local parece ser mais efetiva do que usá-la somente numa fase de pós-otimização. Em geral, a combinação entre intensificação e pós-otimização resulta na melhor estratégia. A reconexão por caminhos é aplicada a pares (X_1, X_2) de soluções, onde X_1 é a

solução localmente ótima obtida após a busca local e X_2 é uma solução de elite escolhida aleatoriamente de um conjunto com um número limitado `Tam_Pool` de soluções de elite encontradas ao longo da busca.

Inicialmente, o conjunto de soluções de elite (`Pool`) está vazio. Cada solução localmente ótima obtida pela busca local é considerada como uma candidata a ser inserida no conjunto se ela é diferente de qualquer outra solução atualmente no conjunto. Se o conjunto já tem `Tam_Pool` soluções e a candidata é melhor do que a pior delas, então a primeira substitui a última. Se o conjunto não estiver cheio, então a candidata é simplesmente inserida. A melhor solução encontrada após a busca local é também considerada como uma candidata para inserção no conjunto de soluções de elite.

Para MPB, foi implementado um algoritmo GRASP híbrido que incorpora as características do procedimento GRASP com as da reconexão por caminhos e adota várias estratégias de otimização: a reconexão por caminhos é introduzida após cada fase de busca local do GRASP (estratégia de intensificação), somente após todas as iterações do GRASP (etapa de pós-otimização) ou adotando ambas as estratégias simultaneamente.

Ribeiro et al. [60] propuseram um GRASP híbrido com perturbação dos custos das arestas e reconexão por caminhos para o problema de *Steiner* em grafos. A fase construtiva do GRASP é substituída pelo uso de várias heurísticas construtivas com uma estratégia de perturbação de custos que combina elementos de intensificação e diversificação. A fase construtiva do procedimento híbrido baseia-se na randomização dos custos para construir diferentes soluções em diferentes iterações. Os custos das arestas são perturbados aleatoriamente no grafo original e a heurística construtiva escolhida é aplicada ao grafo original com os custos modificados. Os métodos utilizados para randomização dos custos incorporam mecanismos de aprendizagem associados com estratégias de intensificação e diversificação propostas originalmente no contexto da busca tabu.

Três métodos distintos de randomização dos custos (Intensificação - I, Diversificação - D e Uniforme - U) são aplicados ao longo do algoritmo de Ribeiro et al [60]. Em uma dada iteração $i \in \{1, \ldots, numIter\}$ do GRASP, o custo modificado c_e^i de cada aresta $e \in E$ é selecionado aleatoriamente de uma distribuição uniforme entre c_e e $r_i(e) \cdot c_e$, onde o coeficiente $r_i(e)$ depende do método de randomização do custo aplicado na iteração i. Seja $t_{i-1}(e)$ o número de soluções localmente ótimas no qual a aresta $e \in E$ aparece, após $i-1$ iterações do procedimento GRASP híbrido. A Tabela 3.1 a seguir mostra como os coeficientes $r_i(e)$ são calculados para cada método de randomização.

Método	$r_i(e)$
I	$2 - 0.75 \cdot t_{i-1}(e)/(i-1)$
D	$1.25 + 0.75 \cdot t_{i-1}(e)/(i-1)$
U	2

Tabela 3.1: Coeficientes de randomização.

No método D, valores do coeficiente $r_i(e)$ são maiores para as arestas que aparecem mais freqüentemente nos ótimos locais encontrados. Este esquema leva a uma estratégia de diversificação, uma vez que as arestas mais freqüentemente utilizadas são penalizadas com aumentos maiores. Por outro lado, o método I é uma estratégia de intensificação penalizando arestas menos freqüentes com coeficientes $r_i(e)$ maiores. Finalmente, o método U usa uma estratégia de penalização uniforme, independente da informação de freqüência.

Os custos originais sem penalização são utilizados nas três primeiras iterações, de forma a incorporar soluções sem nenhuma perturbação ao conjunto de soluções de elite. Os métodos de randomização dos custos são então aplicados ciclicamente, um em cada uma das iterações restantes, começando com o método I, depois D, depois U, então I novamente e este ciclo se repete até o final das iterações. A alternância entre iterações de diversificação (método D) e intensificação (método I) caracteriza uma aproximação de oscilação estratégica que apresentou bons resultados no trabalho de Ribeiro et al. [60] e será adotada como estratégia de perturbação dos custos das arestas do grafo que representa a rede de transmissão, nos testes com o GRASP híbrido.

Sejam β o parâmetro do algoritmo construtivo randomizado, $f^\star$ o custo total da árvore de custo mínimo $T^\star$, T e $f(T)$ uma árvore de transmissão e seu custo, G o grafo que representa a rede com os custos das arestas sem nenhuma perturbação e G' o mesmo grafo com os custos das arestas alterados, conforme as estratégias de perturbação utilizadas em [60] e descritas acima. A Figura 3.3 a seguir apresenta o pseudocódigo do algoritmo GRASP híbrido introduzindo a reconexão por caminhos após cada fase de busca local do GRASP (estratégia de intensificação) e após todas as iterações do GRASP (estratégia de pós-otimização).

Nas três primeiras iterações do procedimento GRASP híbrido, a heurística construtiva aleatorizada (linha 8) faz uso dos custos originais das arestas do grafo, sem adotar nenhuma estratégia de perturbação. A partir da quarta iteração, a heurística construtiva aleatorizada faz uso dos custos das arestas modificados, conforme descrito anteriormente. O procedimento de busca local inicia da solução produzida na fase construtiva, mas sempre utiliza os custos originais para pesquisar a vizinhança a partir desta solução (linha

```
procedimento GRASP híbrido (β, numIter, Tam_Pool)
 1.    f* ← ∞; Pool ← ∅;
 2.    para i = 1, ..., numIter faça
 3.    início
 4.       se (i > 3) então
 5.       início
 6.          Perturbar os custos das arestas conforme a estratégia estabelecida;
 7.       fim - se
 8.       T ← Construir_Solução_Aleatória (β);
 9.       T ← Busca_Local (T);
10.       se (i > 3) então
11.       início
12.          Selecionar aleatoriamente uma solução T' ∈ Pool;
13.          T ← Reconexão por caminhos (T', T);
14.       fim - se
15.       Adicionar T a Pool se atender as condições necessárias;
16.       se (f(T) < f*) então
17.       início
18.          f* ← f(T)
19.          T* ← T
20.       fim - se
21.       Atualizar os parâmetros da randomização t_i(e) e r_i(e) para ∀e ∈ E;
22.    fim - para
23.    Aplicar reconexão por caminhos em todos os pares de soluções do conjunto Pool;
24.    Atualizar a melhor solução encontrada T*;
25.    Retornar T*;
Fim GRASP híbrido
```

Figura 3.3: Pseudocódigo do algoritmo GRASP híbrido.

9). A solução obtida pela busca local é combinada com uma solução selecionada aleatoriamente do conjunto `Pool` através da reconexão por caminhos (linhas 12 e 13), a partir da quarta iteração, utilizando a estratégia de reconexão para trás.

A solução obtida após a fase de reconexão por caminhos (ou após a busca local, se $i < 4$) é considerada como candidata a entrar no conjunto de soluções de elite `Pool` (conjunto com as soluções de alta qualidade encontradas durante a execução) se ela é diferente de qualquer outra solução que já esteja no conjunto. Se o conjunto já estiver completo (com `Tam_Pool` soluções) e o candidato for melhor do que a pior solução do conjunto, então o primeiro substitui esta última. Se o conjunto não estiver cheio, o candidato é simplesmente inserido (linha 15). Finalmente, atualiza-se a melhor solução encontrada, caso necessário (linhas 18 e 19).

Após todas as iterações do GRASP, tem-se a fase de pós-otimização (linha 23), na qual as soluções de elite são combinadas umas com as outras utilizando-se reconexão por caminhos para trás entre cada par de soluções de elite (inicial e guia).

A complexidade de cada iteração do algoritmo GRASP híbrido é resultante basicamente do cômputo da perturbação ($O(n^2)$, linha 6), da aplicação da heurística construtiva ($O(n^3)$, linha 8), da busca local (cada iteração é $O(n^3)$, linha 9) e da reconexão por caminhos ($O(n \log n)$, linha 13). Como todos os demais passos têm complexidade inferior a $O(n^3)$, a complexidade de cada iteração de todas as estratégias de GRASP híbrido implementadas, assim como a do GRASP simples, é semelhante à complexidade da busca local.

A Seção 4.5.5 apresentará os resultados obtidos com o GRASP híbrido aplicando-se as seguintes estratégias:

- utilização de algumas estratégias de reconexão por caminhos (intensificação, pós-otimização, reconexões para frente e para trás, Tabela 4.22);
- variação dos procedimentos de busca local na fase de busca local (Tabela 4.23);
- inserção do algoritmo EWMA logo após a fase de busca local (Tabela 4.23);
- utilização de valores para o tamanho do conjunto `Pool` (parâmetro `Tam_Pool`, Tabela 4.24);
- utilização de valores para a diferença mínima entre soluções (parâmetro `DiffSize`, Tabela 4.24); e
- adoção de combinações de estratégias de perturbação nos custos das arestas do grafo que representa a rede de transmissão (Tabela 4.21).

Capítulo 4

Resultados computacionais

4.1 Condições dos experimentos

O computador utilizado nos experimentos computacionais tem um processador AMD Athlon XP 1.7 GHz com 256 megabytes de memória e sistema operacional Microsoft Windows XP Professional, versão 2002. A linguagem de programação utilizada foi C/C++, o ambiente para edição e compilação dos algoritmos foi o Bloodshed Dev-C++ versão 4.9.9.1 e o compilador utilizado foi Mingw32, versão 2.95.2-1. Em todos os algoritmos que utilizam geração de números aleatórios usou-se uma implementação em C do gerador de números aleatórios Mersenne Twister, desenvolvido por Matsumoto e Nishimura [46, 59].

4.2 Instâncias teste

Para a realização dos testes foram construídas instâncias aleatórias utilizando-se o mesmo procedimento de geração de instâncias adotado em vários artigos [6, 24, 25, 26, 48, 64]: redes com número específico de nós $n \in \{10, 25, 50, 75, 100, 200\}$ foram geradas aleatoriamente em um quadrado de lado igual a cinco, isto é, os nós correspondem a pontos de $\mathbb{R}^2$ com coordenadas uniformemente stribuídas no intervalo $[0, 5]$.

É importante observar que as unidades de distâncias utilizadas neste estudo são arbitrárias. Por exemplo, se todas as distâncias na região 5×5 forem dobradas, mantendo-se as distâncias relativas entre o nós (resultando na distribuição dos nós em uma região 10×10), isto resultaria em um aumento nos níveis de potência das ligações por um fator

de 2^{α} (α é o fator de atenuação do ambiente). Como as árvores produzidas pelos algoritmos estudados mantêm-se as mesmas sob qualquer escala de distância, a potência total da árvore também aumentaria pelo mesmo fator 2^{α}. Assim, qualquer razão de comparação entre as potências totais das árvores produzidas pelos algoritmos é independente do fator de escala da distância.

Para cada rede gerada, um dos nós foi escolhido aleatoriamente para ser o nó fonte. Assume-se que o nível de potência máximo de cada nó é suficiente para alcançar toda a rede. Para cada tamanho da rede, foram geradas aleatoriamente 100 instâncias. Para os testes com as heurísticas construtivas, utilizaram-se dois valores para o fator de atenuação do ambiente: $\alpha = 2$ e $\alpha = 4$. Para as demais heurísticas adotou-se $\alpha = 2$, que corresponde ao caso mais estudado.

Após iniciados os testes com as instâncias geradas aleatoriamente, Das [26, 28, 48] forneceu 250 instâncias teste por ele utilizadas (50 instâncias com $n = 10, 25, 50, 75$ e 100 nós). Estas instâncias também foram geradas aleatoriamente em uma região 5×5. Assim, decidiu-se incorporá-las às 600 instâncias testes previamente geradas, totalizando 850 instâncias teste: 150 instâncias com $n = 10, 25, 50, 75$ e 100 nós e 100 instâncias para as redes com $n = 200$ nós.

Os valores médios de potência total mostrados nas tabelas de resultados deste capítulo correspondem à média das potências totais das árvores de transmissão obtidas por cada algoritmo, sobre todas as instâncias do mesmo tamanho de rede. Assim, cada entrada na tabela representa o valor médio dos resultados obtidos para as 150 instâncias de cada tamanho de rede (com exceção da rede de 200 nós, que possui 100 instâncias teste). Desconhecem-se os valores ótimos das instâncias consideradas. Assim, como forma de avaliação da qualidade dos algoritmos, optou-se em observar seu desempenho sempre em relação ao algoritmo BIP [64], que é o algoritmo construtivo de referência na literatura. Além disso, a maioria dos trabalhos da literatura comparam o desempenho dos algoritmos propostos com o BIP, permitindo, então, uma análise comparativa entre os algoritmos propostos neste trabalho e os da literatura. Uma versão do algoritmo BIP foi implementada para que se pudesse realizar a comparação entre os resultados dos algoritmos propostos neste trabalho e os do BIP.

4.3 Resultados das heurísticas construtivas

O objetivo da análise das heurísticas construtivas é identificar a qualidade das soluções por elas encontradas, além de compará-las entre si. Os itens a seguir indicarão os resultados obtidos pelas heurísticas gulosas e randomizadas, além de trazer uma comparação entre elas. Para a heurística com o melhor desempenho, realizou-se, adicionalmente, um histograma de freqüência das soluções para um subconjunto das instâncias teste, de forma a identificar o comportamento do algoritmo com a variação do parâmetro β.

4.3.1 Heurísticas gulosas

A Tabela 4.1 apresenta os valores médios das soluções para cada uma das 850 instâncias teste, com os algoritmos construtivos gulosos descritos na Seção 2.2.1 executados uma única vez para cada instância. Por exemplo, o valor $13,37$ na linha das redes de 10 nós e coluna BLU, foi obtido somando-se os custos totais das soluções obtidas por BLU, utilizando-se $\alpha = 2$, para cada uma das instâncias de dez nós e, a seguir, dividindo-se o montante total pelo número de instâncias de dez nós (150). O valor entre parênteses (10%) indica o quanto esse valor médio se distancia daquele produzido pelo algoritmo BIP, isto é, o valor médio das soluções obtidas pelo BLU, para as instâncias de 10 nós, é 10% maior que o valor médio das soluções obtidas por BIP (lembrando-se que MPB é um problema de minimização de potência total). O valor médio $14,96$, na mesma coluna BLU, é calculado somando-se a potência total da solução de cada uma das 850 instâncias teste obtida por BLU e, a seguir, dividindo-se esse montante pelo número total de instâncias (850).

A análise da Tabela 4.1 permite tirar as seguintes conclusões:

- A utilização do algoritmo BLU na resolução do MPB produz soluções cujo valor médio é 27% maior do que aquele produzido pelo algoritmo BIP, adotando-se $\alpha = 2$. Observe-se que a distância entre os valores médios das soluções obtida por BLU e por BIP aumenta à medida que o número de nós aumenta, indicando perda na qualidade da solução com o aumento da rede. Quando adota-se o fator de atenuação $\alpha = 4$, a distância entre os valores médios das soluções é um pouco menor do que com $\alpha = 2$, mas também é crescente, produzindo soluções cujo valor médio é 9% maior que o obtido por BIP.

- A utilização do algoritmo SPF produz soluções cujo valor médio é 8% maior do que

Nós	$\alpha = 2$				$\alpha = 4$			
	BLU	BIP	SPF	DSPF	BLU	BIP	SPF	DSPF
10	13,37 (10%)	11,85	13,14 (11%)	11,50 (-3%)	49,39 (7%)	46,32	47,85 (3%)	46,16 (0%)
25	15,07 (21%)	12,50	13,48 (8%)	11,84 (-5%)	19,35 (11%)	17,48	17,87 (2%)	17,16 (-2%)
50	15,23 (29%)	11,77	12,71 (8%)	11,23 (-5%)	8,58 (13%)	7,58	7,77 (3%)	7,52 (-1%)
75	15,24 (31%)	11,62	12,46 (7%)	11,07 (-5%)	5,56 (17%)	4,76	4,86 (2%)	4,76 (0%)
100	15,50 (33%)	11,66	12,39 (6%)	11,16 (-4%)	4,06 (19%)	3,41	3,48 (2%)	3,40 (0%)
200	15,57 (40%)	11,14	11,93 (7%)	10,67 (-4%)	1,89 (23%)	1,54	1,58 (3%)	1,54 (0%)
média	14,96 (27%)	11,79	12,73 (8%)	**11,28 (-4%)**	15,56 (9%)	14,22	14,63 (3%)	**14,12 (-1%)**

Tabela 4.1: Valores médios das soluções obtidas pelas heurísticas construtivas gulosas (o valor entre parênteses representa a variação em relação ao valor obtido por BIP).

o produzido por BIP, para $\alpha = 2$. A diferença entre o valor médio das soluções das 850 instâncias teste adotando-se SPF e BIP diminui ao se considerar $\alpha = 4$, mas o valor médio ainda é 3% maior do que o obtido por BIP.

- O algoritmo DSPF produz soluções cujo valor médio é 4% menor que o valor médio obtido por BIP, para $\alpha = 2$. Ao se considerar $\alpha = 4$, a diferença entre BIP e DSPF se reduz, mas o valor médio das soluções ainda é 1% menor do que o obtido por BIP.

A Tabela 4.2 a seguir apresenta os tempos médios de processamento em milisegundos obtidos pelos algoritmos construtivos gulosos. A análise dos tempos médios de processamento permite concluir que os algoritmos construtivos BLU, SPF e DSPF apresentam tempo médio de processamento inferior ao do BIP. O algoritmo com o melhor desempenho, simultaneamente, em tempo médio de processamento e valor médio das soluções, tanto para $\alpha = 2$, quanto para $\alpha = 4$, é o algoritmo DSPF, apresentando tempo médio de processamento para as 850 instâncias 70% menor que o tempo médio obtido por BIP ($\alpha = 2$). Este resultado mostra que a utilização de DSPF acelera significativamente a construção da árvore de transmissão, pois o algoritmo permite a inclusão de vários nós em uma única iteração.

4.3.2 Heurísticas randomizadas

Para cada uma das heurísticas randomizadas rBIP, rSPF e rDSPF, variou-se o parâmetro β de 0 a 1, em intervalos de $0,1$, e utilizou-se a mesma semente inicial em todas as

Nós	$\alpha = 2$				$\alpha = 4$			
	BLU	BIP	SPF	DSPF	BLU	BIP	SPF	DSPF
10	0,3 (-100%)	13,8	12,6 (-9%)	6,1 (-56%)	1,5 (-88%)	12,7	11,8 (-7%)	7,3 (-43%)
25	0,5 (-98%)	33,1	32,3 (-2%)	14,7 (-56%)	1,5 (-96%)	34,8	32,5 (-7%)	13,3 (-62%)
50	0,4 (-100%)	86,1	73,3 (-15%)	31,1 (-64%)	1,6 (-98%)	85,5	72,5 (-15%)	27,9 (-67%)
75	0,3 (-100%)	86,1	123,0 (43%)	54,6 (-37%)	2,1 (-99%)	170,1	122,8 (-28%)	52,4 (-69%)
100	1,3 (-100%)	289,3	194,0 (-33%)	94,5 (-67%)	2,0 (-99%)	314,7	194,3 (-38%)	89,5 (-72%)
200	4,5 (-100%)	1777,1	775,2 (-56%)	499,7 (-72%)	5,2 (-100%)	2065,6	773,2 (-63%)	497,8 (-76%)
média	1,0 (-100%)	312,0	168,0 (-46%)	**94,3 (-70%)**	2,1 (-99%)	352,0	167,5 (-52%)	**92,2 (-74%)**

Tabela 4.2: Tempos médios de processamento das soluções obtidas pelas heurísticas construtivas gulosas em milisegundos (o valor entre parênteses representa a variação em relação ao valor obtido por BIP).

heurísticas simuladas. A semente utilizada para inicializar o gerador de números aleatórios foi idêntica para todas as heurísticas testadas de forma a não influenciar no desempenho dos algoritmos. Para cada uma das 850 instâncias teste foi realizada uma única execução com cada combinação de algoritmo e parâmetro β. Os resultados obtidos foram agrupados por heurística, valor de β e tamanho de instância, e calculou-se o valor médio das soluções obtidas em cada grupo. As Tabelas 4.3 a 4.5 ilustram a evolução do valor médio das soluções obtidas pelas heurísticas construtivas rBIP, rSPF e rDSPF com a variação do parâmetro β, para $\alpha = 2$ e $\alpha = 4$.

		β										
rBIP	Nós	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
$\alpha = 2$	10	11,85	11,85	11,85	11,85	13,19	11,83	11,83	**11,81**	11,84	11,84	11,95
	25	12,50	12,50	12,50	12,50	13,75	12,44	**12,43**	12,50	12,50	12,50	12,83
	50	**11,77**	**11,77**	**11,77**	11,79	13,02	11,86	11,88	11,97	12,02	12,05	12,52
	75	**11,62**	**11,62**	**11,62**	11,63	12,80	11,73	11,80	11,86	11,98	12,05	12,48
	100	**11,66**	**11,66**	**11,66**	11,67	12,79	11,83	11,85	11,92	12,06	12,13	12,56
	200	**11,14**	**11,14**	11,16	11,22	12,46	11,40	11,51	11,59	11,72	11,83	12,24
	média	**11,79**	**11,79**	11,80	11,81	13,03	11,87	11,91	11,96	12,04	12,08	12,44
$\alpha = 4$	10	46,32	46,32	46,32	46,32	46,32	46,32	46,32	**46,31**	**46,31**	**46,31**	46,96
	25	17,48	17,48	17,48	17,48	17,48	17,50	**17,47**	17,51	17,51	17,52	17,77
	50	**7,58**	**7,58**	**7,58**	**7,58**	7,60	7,61	7,64	7,67	7,72	7,72	7,94
	75	4,76	4,76	4,76	**4,75**	4,77	4,79	4,80	4,84	4,87	4,88	5,09
	100	**3,41**	**3,41**	**3,41**	3,42	3,43	3,46	3,47	3,48	3,54	3,55	3,69
	200	**1,54**	**1,54**	**1,54**	1,55	1,56	1,57	1,59	1,60	1,63	1,64	1,70
	média	**14,22**	**14,22**	**14,22**	**14,22**	14,23	14,25	14,25	14,27	14,30	14,31	14,57

Tabela 4.3: Evolução do valor médio das soluções obtidas pelo algoritmo rBIP com o parâmetro β. Os valores em negrito indicam o menor valor médio encontrado por rBIP para cada grupo de instâncias.

Para a heurística rBIP, a variação do valor médio das soluções com β é menor adotando-se $\alpha = 4$ do que com $\alpha = 2$. Para as redes de 50, 75 e 100 nós, adotando-

se $\alpha = 2$, o menor valor médio foi obtido utilizando-se β menor ou igual a $0,2$. Para as redes de 10, 25 nós e 200 nós, os valores de β que resultam no menor valor médio são, respectivamente, $0,7$, $0,6$ e menor ou igual a $0,1$.

Observa-se que, à medida que o número de nós da rede aumenta, o valor de β necessário para encontrar soluções com custo menor diminui, iniciando em $0,7$ e reduzindo-se até $0,1$. A adoção de β próximo à escolha gulosa para redes com até 25 nós não produz os menores valores médios das soluções, pois, para esta dimensão de rede, a diversidade de soluções obtidas é pequena, sendo insuficiente para que se encontre soluções com custo total menor do que o obtido adotando-se a escolha gulosa.

rSPF	Nós	β										
		0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
$\alpha = 2$	10	13,14	**11,95**	13,14	13,14	13,19	13,10	13,12	13,17	13,08	13,30	13,27
	25	13,48	**12,83**	13,58	13,61	13,75	13,80	13,77	13,72	13,77	13,99	14,03
	50	12,71	**12,52**	12,90	12,96	13,02	13,06	13,14	13,15	13,21	13,19	13,34
	75	**12,46**	12,48	12,75	12,81	12,80	12,92	12,97	13,03	13,04	13,08	13,12
	100	**12,39**	12,56	12,69	12,74	12,79	12,89	12,89	13,00	12,94	13,04	13,14
	200	**11,93**	12,24	12,32	12,34	12,46	12,50	12,48	12,58	12,70	12,72	12,71
	média	12,73	**12,44**	12,93	12,97	13,03	13,08	13,10	13,14	13,15	13,25	13,30
$\alpha = 4$	10	47,85	**46,96**	47,85	47,85	47,85	47,85	47,87	47,89	47,89	47,74	48,40
	25	17,87	**17,77**	18,01	18,01	17,99	17,98	18,03	18,10	18,09	18,20	18,27
	50	**7,77**	7,94	7,82	7,86	7,88	7,90	7,85	7,92	7,91	7,92	8,00
	75	**4,86**	5,09	4,92	4,94	4,96	4,96	4,97	5,00	4,98	4,99	4,99
	100	**3,48**	3,69	3,52	3,54	3,53	3,56	3,55	3,58	3,58	3,58	3,63
	200	**1,58**	1,70	1,60	1,61	1,61	1,61	1,62	1,62	1,63	1,63	1,65
	média	14,63	**14,57**	14,68	14,70	14,70	14,70	14,71	14,75	14,74	14,74	14,89

Tabela 4.4: Evolução do valor médio das soluções obtidas pelo algoritmo rSPF com o parâmetro β. Os valores em negrito indicam o menor valor médio encontrado por rSPF para cada grupo de instâncias.

Utilizando-se o algoritmo rSPF, o valor médio das soluções tem tendência ligeiramente crescente à medida que aumenta-se o valor de β. Essa tendência é mais expressiva para $\alpha = 2$ e bem discreta para $\alpha = 4$. Para as redes abaixo de 75 nós, o valor de β que resulta no menor valor médio das soluções é $0,1$ ($\alpha = 2$). Para as redes de 75, 100 e 200 nós, o valor de β que resulta no menor valor médio é 0, ou seja, o algoritmo guloso é quem produz o menor valor médio das soluções. Sendo assim, a randomização do algoritmo SPF não reduz o valor médio das soluções para redes com mais de 50 nós.

Para rDSPF, observa-se uma tendência diferente da obtida com os outros dois algoritmos randomizados, pois a adoção de $\beta = 0,1$ resulta nos maiores valores médios das soluções em todos os tamanhos de instâncias testadas. Os valores de β que produzem soluções com os menores valores médios para as redes de 10, 25, 50, 75, 100 e 200 nós, são, respectivamente, $0,9$, $0,7$, $0,5$, $0,7$, $0,6$ e $0,3$. Como a diversidade das soluções aumenta com o tamanho da rede, em geral, à medida que o número de nós aumenta, requerem-se menores valores para o parâmetro β. Esses resultados mostram que nem

		β										
rDSPF	Nós	0	0,1	0,2	0,3	0,4	0,5	0,6	0,7	0,8	0,9	1
$\alpha = 2$	10	11,50	13,27	11,50	11,50	11,47	11,49	11,45	11,45	11,47	**11,31**	11,44
	25	11,84	14,03	11,83	11,80	11,80	11,76	11,76	**11,67**	11,71	11,80	11,80
	50	11,23	13,34	11,23	11,19	11,23	**11,16**	11,17	11,18	11,20	11,20	11,37
	75	11,07	13,12	11,06	11,08	11,06	11,01	11,09	**11,00**	11,03	11,10	11,14
	100	11,15	13,14	11,07	11,09	11,08	**11,05**	**11,05**	11,06	**11,05**	11,09	11,24
	200	10,67	12,71	10,63	**10,61**	10,62	10,64	10,64	10,64	10,65	10,71	10,82
	média	11,28	13,30	11,26	11,25	11,25	11,22	11,22	**11,20**	11,22	11,23	11,33
$\alpha = 4$	10	46,16	48,40	46,16	46,16	46,12	45,88	46,22	46,03	46,05	**45,76**	45,88
	25	17,16	18,27	17,23	17,19	17,18	17,25	17,19	17,30	17,25	**17,13**	17,15
	50	7,52	8,00	7,54	7,53	7,53	7,52	7,51	**7,48**	7,52	7,52	7,56
	75	4,76	4,99	4,76	**4,73**	4,75	4,74	4,75	**4,73**	4,75	4,75	4,77
	100	3,40	3,63	3,39	3,39	**3,38**	3,40	3,39	3,40	3,40	3,39	3,43
	200	**1,54**	1,65	**1,54**	**1,54**	**1,54**	**1,54**	**1,54**	**1,54**	1,55	1,55	1,57
	média	14,12	14,89	14,14	14,12	14,12	14,09	14,13	14,11	14,12	**14,04**	14,09

Tabela 4.5: Evolução do valor médio das soluções obtidas pelo algoritmo rDSPF com o parâmetro β. Os valores em negrito indicam o menor valor médio encontrado por rDSPF para cada grupo de instâncias.

sempre a adoção de um parâmetro próximo à escolha gulosa produz soluções com menor valor médio.

A Tabela 4.6 apresenta um resumo dos resultados apresentados nas Tabelas 4.3 a 4.5 (são reapresentados somente os valores em negrito das Tabelas 4.3 a 4.5), indicando os menores valores médios das soluções obtidas e os respectivos tempos médios de processamento, adotando-se o parâmetro β que resultou no melhor valor médio para cada tamanho de rede e algoritmo. O valor $11,81$ na coluna do algoritmo rBIP e segunda linha, por exemplo, indica que a adoção de $\beta = 0,7$ no algoritmo rBIP foi a que resultou no menor valor médio das soluções das instâncias de 10 nós e o valor $13,2$, na coluna t (ms) indica o tempo médio de processamento correspondente. O valor entre parênteses na linha valor $(-0,3\%)$ indica que o valor médio da soluções deste conjunto de instâncias é $0,3\%$ menor que o obtido pelo algoritmo BIP e o valor entre parênteses na linha tempo $(-4,3\%)$ indica que o tempo médio de processamento, com a adoção desse algoritmo e β, é $4,3\%$ menor do que o tempo médio de processamento do algoritmo BIP para este mesmo grupo de instâncias. O valor $11,20$ em negrito, na coluna do algoritmo rDSPF, linha média, $\alpha = 2$, indica o valor médio obtido para as 850 instâncias teste utilizando-se $\beta = 0,7$ no algoritmo rDSPF e está em negrito por ser o melhor resultado encontrado para as combinações de parâmetro β e algoritmo construtivo.

A análise dos resultados apresentados na Tabela 4.6 permite as seguintes observações:

- A randomização da heurística BIP, tanto para $\alpha = 2$, quanto para $\alpha = 4$, não acarretou redução no valor médio das soluções e no tempo médio de processamento levando-se em conta a média das 850 instâncias testadas. Para as instâncias de 10 e 25 nós, apenas, observou-se uma pequena redução no valor médio das soluções

	Nós	rBIP			rSPF			rDPSF		
		valor	β	t (ms)	valor	β	t (ms)	valor	β	t (ms)
$\alpha = 2$	10	11,81 (-0,3%)	0,7	13,2 (-4,3%)	11,95 (0,8%)	0,1	13,5 (-2,2%)	11,31 (-4,6%)	0,9	6,5 (-52,9%)
	25	12,43 (-0,6%)	0,6	33,8 (2,1%)	12,83 (2,6%)	0,1	34,1 (3,0%)	11,67 (-6,6%)	0,7	13,8 (-58,3%)
	50	11,77 (0%)	0	86,1 (0%)	12,52 (6,4%)	0,1	84,3 (-2,1%)	11,16 (-5,2%)	0,5	27,9 (-67,6%)
	75	11,62 (0%)	0	160,8 (0%)	12,46 (7,2%)	0	123,0 (-23,5%)	11,00 (-5,3%)	0,7	51,1 (-68,2%)
	100	11,66 (0%)	0	289,3 (0%)	12,39 (6,3%)	0	194 (-32,9%)	11,05 (-5,2%)	0,5	83 (-71,3%)
	200	11,14 (0%)	0	1777,1 (0%)	11,93 (7,1%)	0	775,2 (-56,4%)	10,61 (-4,8%)	0,3	408 (-77,0%)
	média	11,79 (0%)	0	312,0 (0%)	12,44 (5,5%)	0,1	328,8 (5,4%)	**11,20** (-5,0%)	0,7	78,3 (-74,9%)
$\alpha = 4$	10	46,31 (0%)	0,7	13,9 (9,4%)	46,96 (1,4%)	0,1	12,6 (-0,8%)	45,76 (-1,2%)	0,9	6 (-52,8%)
	25	17,47 (-0,1%)	0,6	34,1 (-2,0%)	17,77 (1,7%)	0,1	34 (-2,3%)	17,13 (-2,0%)	0,9	34 (-2,3%)
	50	7,58 (0%)	0	85,5 (0%)	7,77 (2,5%)	0	72,5 (-15,2%)	7,48 (-1,3%)	0,7	26,3 (-69,2%)
	75	4,75 (-0,2%)	0,3	176,7 (3,9%)	4,86 (2,1%)	0	122,8 (-27,8%)	4,73 (-0,6%)	0,3	50,1 (-70,5%)
	100	3,41 (0%)	0	314,7 (0%)	3,48 (2,1%)	0	194,3 (-38,3%)	3,38 (-0,9%)	0,4	80,7 (-74,4%)
	200	1,54 (0%)	0	2065,6 (0%)	1,58 (2,6%)	0	773,2 (-62,6%)	1,54 (0%)	0,3	407,5 (-80,3%)
	média	14,22 (0%)	0	352 (0%)	14,57 (2,5%)	0,1	372,3 (5,8%)	**14,04** (-1,3%)	0,9	75,2 (-78,6%)

Tabela 4.6: Melhores resultados dos algoritmos construtivos randomizados.

(rBIP construiu soluções cujo valor médio é $0,6\%$ menor que o valor médio obtido por BIP para as redes de 25 nós).

- para SPF, a randomização trouxe uma pequena redução no valor médio das soluções, se comparado ao resultado de SPF (valor médio $2,5\%$ menor que o valor médio de SPF, com $\alpha = 2$). A versão gulosa produzia soluções cujo valor é 8% maior que o valor médio das soluções obtidas por BIP e agora o percentual é de $5,5\%$. Entretanto, mesmo com a randomização, rSPF ainda apresenta valores médios maiores que os obtidos pela heurística BIP, para ambos os valores de α. O parâmetro β que resulta no menor valor médio das soluções é $0,1$ para instâncias com até 50 nós. Para as instâncias acima de 50 nós, a versão gulosa ($\beta = 0$) é a que produz soluções com o menor valor médio dentre os resultados com rSPF.

- rDSPF é a heurística com os melhores resultados, tanto em relação ao valor médio das soluções quanto em relação ao tempo médio de processamento. O valor médio das soluções obtidas por rDSPF para as 850 instâncias teste é 5% menor do que

aquele obtido por BIP, adotando-se $\alpha = 2$. Como o valor médio das soluções produzidas por DSPF era 4% menor que o valor obtido por BIP, conclui-se que a randomização permitiu reduzir em 1% o valor médio das soluções obtidas pela versão gulosa. Com $\alpha = 4$, assim como em DSPF, a diferença em relação a BIP é pequena (valor médio $1,3\%$ menor que o do BIP), mas é discretamente melhor do que a versão puramente gulosa. Na média das 850 instâncias, o parâmetro β que resulta no menor valor médio é $0,7$, mas o valor de β necessário para se obter soluções com menor valor médio decresce à medida que o número de nós aumenta, variando de $0,9$ (redes com até 25 nós) até $0,3$ (rede com 200 nós).

Como rDSPF foi a heurística construtiva com o melhor desempenho, tanto em termos de tempo médio de processamento, quanto em valor médio das soluções, na próxima seção será realizada uma análise mais detalhada do comportamento do valor da solução com o parâmetro β.

4.3.3 Histogramas para rDSPF

As Tabelas 4.7 a 4.9 apresentadas nesta Seção correspondem aos resultados de testes específicos, adotando-se apenas $\alpha = 2$, realizados com um pequeno subconjunto das instâncias teste de forma a observar a influência do parâmetro β na diversidade das soluções obtidas pela heurística construtiva rDSPF. Foram utilizadas três instâncias para cada tamanho de rede e β foi variado de 0 a 1, em intervalos de $0,1$. Para cada instância e cada valor de β foram realizadas 1000 execuções independentes (número escolhido arbitrariamente), com sementes distintas. A seguir, contabilizou-se a freqüência das soluções para cada valor de β, em cada intervalo de custo total da solução. Na primeira faixa do intervalo de custo total (terceira linha das Tabela 4.7, 4.8 e 4.9) encontra-se o custo total mínimo obtido para aquela instância com a heurística rDSPF e na última faixa (última linha), o custo total máximo obtido para aquela instância com rDSPF. Os demais valores das faixas de custo foram calculados de forma a serem equidistantes entre si e estarem entre os custos máximo e mínimo. Embora os testes tenham sido realizados com β em intervalos de $0,1$, optou-se por simplificar sua apresentação fornecendo os resultados em intervalos de $0,2$.

Por exemplo, na Tabela 4.7, para a instância 10a1, na primeira e última faixas $(0,\ 11,49]$ e $(13,91,\ 14,04]$ (terceira e última linhas das tabelas) estão, respectivamente, o menor e o maior custo total obtido para essa instância em 1000 execuções de rDSPF. O valor 1000 na coluna $\beta = 0,1$ indica que a adoção de $\beta = 0,1$ resultou em 1000 so-

luções de custo total maior que 12,51 e menor ou igual a 12,64. Para esse exemplo, todas as 1000 soluções tiveram custo total idêntico e igual a 12,53 (ou seja, no intervalo (12,51, 12,64]). O valor 39 na coluna $\beta = 0,9$, para a mesma instância, indica que das 1000 soluções obtidas, 39 apresentaram custo total maior que 11,75 e menor ou igual a 11,87 (neste intervalo todas as 39 soluções apresentaram custo igual a 11,78). Embora três instâncias de cada tamanho tenham sido testadas, para que o texto não se torne repetitivo, somente serão apresentados os resultados para uma instância de cada tamanho, pois os histogramas são semelhantes entre as instâncias de mesma dimensão.

instância 10a1 (10 nós)						instância 25a1 (25 nós)					
custo	$\beta =$ 0,1	0,3	0,5	0,7	0,9	custo	$\beta =$ 0,1	0,3	0,5	0,7	0,9
(0,00, 11,49]				68	145	(0,00, 10,22]			3	1	8
(11,49, 11,62]						(10,22, 10,44]			2	6	10
(11,62, 11,75]					28	(10,44, 10,66]			16	15	9
(11,75, 11,87]					39	(10,66, 10,88]			15	15	37
(11,87, 12,00]						(10,88, 11,10]				3	6
(12,00, 12,13]						(11,10, 11,32]		283	261	231	250
(12,13, 12,26]						(11,32, 11,55]	1000	580	282	223	178
(12,26, 12,38]						(11,55, 11,77]		78	49	31	44
(12,38, 12,51]						(11,77, 11,99]			5	11	51
(12,51, 12,64]	1000	1000	503	561	445	(11,99, 12,21]				1	17
(12,64, 12,77]			103	66	195	(12,21, 12,43]				23	56
(12,77, 12,89]					6	(12,43, 12,65]				19	47
(12,89, 13,02]			185	173	65	(12,65, 12,88]			9	6	31
(13,02, 13,15]						(12,88, 13,10]			10	32	36
(13,15, 13,28]						(13,10, 13,32]			10	16	28
(13,28, 13,40]						(13,32, 13,54]			7	19	8
(13,40, 13,53]						(13,54, 13,76]			30	8	16
(13,53, 13,66]			124	105	69	(13,76, 13,98]			6	5	1
(13,66, 13,79]						(13,98, 14,20]		4	216	278	146
(13,79, 13,91]						(14,20, 14,43]		55	79	48	20
(13,91, 14,04]			85	27	8	(14,43, 14,65]				9	1

Tabela 4.7: Histograma de frequência das soluções para alguns valores de β (instâncias teste de 10 e 25 nós).

A análise dos histogramas apresentados na Tabela 4.7 permite observar que a diversidade das soluções para a instância de 10 nós é pequena para valores de β menores ou iguais a 0,5 (estratégias mais próximas à escolha gulosa). Mesmo para $\beta = 0,9$, o custo total das soluções está distribuído em apenas nove faixas distintas de custo. Em 1000 execuções da heurística rDSPF, adotando-se $\beta = 0,5$ a instância de 10 nós apresenta soluções com custo total em apenas cinco faixas distintas. Para essa instância, somente com $\beta \geq 0,7$ é que consegue-se obter a solução com o menor custo total (considerando-se todas as heurísticas construtivas, rDSPF é o único que obtém uma solução com esse custo). Para a instância de 25 nós, somente a partir de $\beta = 0,5$ é que se começa a observar um aumento da diversidade de soluções com custos distintos. Observa-se também que, para este valor de β, em apenas três das 1000 execuções independentes obtém-se a solução com o menor custo total. O que deseja-se ilustrar com esses dois exemplos de instâncias é que há pequena diversidade de soluções com custos distintos para redes com até 25 nós

quando adota-se β menor ou igual a $0,5$ e essa baixa diversidade é a justificativa para que rDSPF só encontre boas soluções adotando-se β maior ou igual a $0,5$.

instância 50a1 (50 nós)						instância 75a1 (75 nós)					
custo	$\beta =$ 0,1	0,3	0,5	0,7	0,9	custo	$\beta =$ 0,1	0,3	0,5	0,7	0,9
(0,00, 8,75]				1		(0,00, 9,26]				1	
(8,75, 8,92]		6	14	5	2	(9,26, 9,38]					
(8,92, 9,09]	173	86	66	22	7	(9,38, 9,49]				1	
(9,09, 9,27]	134	237	238	116	53	(9,49, 9,61]				3	1
(9,27, 9,44]	92	248	192	91	36	(9,61, 9,72]				1	5
(9,44, 9,61]	125	206	110	50	24	(9,72, 9,84]			2	4	11
(9,61, 9,78]	191	96	47	46	39	(9,84, 9,95]			14	26	56
(9,78, 9,96]	120	47	25	33	42	(9,95, 10,07]		4	31	47	81
(9,96, 10,13]	114	41	18	37	41	(10,07, 10,18]	1	46	117	117	138
(10,13, 10,30]	42	23	17	40	70	(10,18, 10,30]	26	172	208	188	190
(10,30, 10,47]	9	4	41	80	93	(10,30, 10,41]	165	329	261	218	171
(10,47, 10,65]		1	54	115	137	(10,41, 10,53]	455	306	201	148	141
(10,65, 10,82]		1	47	92	115	(10,53, 10,65]	286	113	101	133	109
(10,82, 10,99]		2	26	80	113	(10,65, 10,76]	67	27	38	62	53
(10,99, 11,16]		2	28	58	84	(10,76, 10,88]		2	16	32	28
(11,16, 11,34]			35	61	67	(10,88, 10,99]		1	9	15	10
(11,34, 11,51]			17	37	40	(10,99, 11,11]			2	3	4
(11,51, 11,68]			11	15	21	(11,11, 11,22]				1	2
(11,68, 11,85]			10	10	11	(11,22, 11,34]					
(11,85, 12,03]			3	7	5	(11,34, 11,45]					
(12,03, 12,20]			1	4		(11,45, 11,57]					

Tabela 4.8: Histograma de frequência das soluções para alguns valores de β (instâncias teste de 50 e 75 nós).

Para as instâncias de 50 e 75 nós (Tabela 4.8), observa-se um aumento na diversidade das soluções em relação à obtida para 10 e 25 nós. Para estas duas instâncias, somente utilizando-se $\beta = 0,7$ é que obtém-se a solução com o menor custo total em apenas uma das 1000 execuções de rDSPF.

instância 100a1 (100 nós)						instância 200_1 (200 nós)					
custo	$\beta =$ 0,1	0,3	0,5	0,7	0,9	custo	$\beta =$ 0,1	0,3	0,5	0,7	0,9
(0,00, 10,76]			1			(0,00, 10,35]			1		
(10,76, 10,84]					3	(10,35, 10,42]		1	3	4	1
(10,84, 10,92]			3	1	9	(10,42, 10,49]	1	4	4	6	1
(10,92, 10,99]		2	4	6	22	(10,49, 10,56]	11	15	12	18	11
(10,99, 11,07]		8	12	24	45	(10,56, 10,63]	34	30	26	35	18
(11,07, 11,15]	3	25	33	55	68	(10,63, 10,70]	68	76	57	44	36
(11,15, 11,23]	25	37	76	94	101	(10,70, 10,77]	151	107	112	85	54
(11,23, 11,31]	70	96	121	117	108	(10,77, 10,84]	197	162	165	145	70
(11,31, 11,39]	138	163	158	146	119	(10,84, 10,91]	183	175	189	184	119
(11,39, 11,47]	182	157	146	130	126	(10,91, 10,98]	157	175	162	151	132
(11,47, 11,55]	222	154	119	111	107	(10,98, 11,05]	101	140	126	135	140
(11,55, 11,63]	197	129	118	87	84	(11,05, 11,12]	66	64	70	74	112
(11,63, 11,71]	84	80	82	86	63	(11,12, 11,19]	23	33	42	56	116
(11,71, 11,79]	57	59	57	57	57	(11,19, 11,26]	4	8	25	33	75
(11,79, 11,87]	20	42	32	38	44	(11,26, 11,34]	4	7	6	21	51
(11,87, 11,95]	2	32	22	29	20	(11,34, 11,41]		1		8	30
(11,95, 12,03]		6	11	10	10	(11,41, 11,48]		2			14
(12,03, 12,11]		4	4	5	7	(11,48, 11,55]				1	10
(12,11, 12,19]		4	1	3	4	(11,55, 11,62]					7
(12,19, 12,27]		1		1	2	(11,62, 11,69]					1
(12,27, 12,35]		1			1	(11,69, 11,76]					2

Tabela 4.9: Histograma de frequência das soluções para alguns valores de β (instâncias teste de 100 e 200 nós).

Para as instâncias de 100 e 200 nós (Tabela 4.9), já observa-se uma diversidade maior

de soluções a partir de $\beta = 0,3$. Somente para $\beta = 0,5$ é que obtém-se a solução de menor custo para estas duas instâncias específicas, considerando-se os resultados obtidos por todas as heurísticas construtivas.

A análise destes histogramas permite corroborar os resultados apresentados na Seção 4.3.2 para a heurística rDSPF e podem ser justificados pela característica peculiar do algoritmo. Como rDSPF pode agregar vários elementos em cada iteração, a utilização de um parâmetro β próximo à escolha gulosa não traz a diversidade necessária para que se encontre soluções de boa qualidade.

A partir destas observações, pode-se inferir que a avaliação dos resultados do algoritmo randomizado com o parâmetro β é importante para identificar quais valores de β devem ser utilizados para gerar as melhores soluções iniciais para os algoritmos de busca local e as metaheurísticas. Para a heurística rDSPF, em particular, observa-se que a adoção de β próximo à escolha gulosa só resulta nos melhores resultados para redes acima de 100 nós. Assim, é necessário avaliar o comportamento de um algoritmo construtivo antes de adotar uma estratégia próxima à escolha gulosa.

4.3.4 Comparação entre as heurísticas gulosas e randomizadas

A Tabela 4.10 resume os melhores resultados para $\alpha = 2$ (menores valores médios e respectivos tempos médios de processamento) obtidos pelas heurísticas construtivas gulosas e randomizadas, conforme já apresentado nas Tabelas 4.1, 4.2 e 4.6 (valores em negrito). O valor 11,20 da coluna rDSPF refere-se ao valor médio das soluções obtidas para as 850 instâncias teste adotando-se $\beta = 0,7$ em rDSPF (valor que resultou no menor valor médio das soluções, conforme apresentado na Tabela 4.6).

	BIP	DSPF	rDSPF
valor médio	11,79	11,28 (-4,3%)	11,20 **(-5,0%)**
tempo (ms)	312,0	94,3 (-69,8%)	78,3 **(-74,9%)**

Tabela 4.10: Resumo com os melhores resultados dos algoritmos construtivos ($\alpha = 2$).

A heurística construtiva rDSPF foi a heurística com os melhores resultados, tanto em termos de tempo médio de processamento quanto em valor médio das soluções, produzindo soluções cujo valor médio é 5% menor do que o valor obtido pelo algoritmo BIP e com redução de quase 75% no tempo médio de processamento de BIP. Devido a este desempenho, rDSPF será utilizado como heurística construtiva nos demais algoritmos deste

trabalho.

Conforme comentado na Seção 4.2, foram considerados dois valores para o fator de atenuação do ambiente $\alpha = 2$ e $\alpha = 4$ nos testes das heurísticas construtivas. Para as demais heurísticas será adotado apenas $\alpha = 2$, que corresponde ao caso mais estudado e onde existe a maior variação entre os valores das soluções obtidas pelas heurísticas construtivas.

4.4 Resultados das heurísticas de busca local

Para a comparação de desempenho entre as heurísticas de busca local implementadas, utilizou-se a heurística rDSPF para gerar a solução de partida de cada busca local, adotando-se $\alpha = 2$, a mesma semente inicial de geração de números aleatórios e o mesmo parâmetro $\beta = 0,6$, garantindo que todos os algoritmos de busca local tenham como partida a mesma solução inicial. Embora o valor $\beta = 0,7$ tenha sido o que resultou no melhor resultado médio para rDSPF, foi escolhido $\beta = 0,6$ para os testes com as buscas locais, porque, para redes com mais de 75 nós, $\beta = 0,7$ não produziu os melhores resultados. Foi utilizado apenas um valor de β fixo porque, nesta etapa, o jetivo é a avaliação e comparação dos algoritmos de busca local.

4.4.1 Resultados para a busca local mais aprimorante

A Tabela 4.11 a seguir apresenta os valores médios obtidos em uma execução para cada uma das 850 instâncias teste, com os algoritmos de busca local mais aprimorante descritos na Seção 2.4.2.1. O valor $11,45$ na linha das redes de 10 nós e coluna rDSPF corresponde ao valor médio das soluções obtidas para as 150 instâncias de dez nós, utilizando-se $\alpha = 2$ e apenas o algoritmo construtivo rDSPF (valor médio encontrado antes da busca local), enquanto o valor $10,76$ nessa mesma linha, coluna BLB, corresponde ao valor médio das soluções utilizando-se o algoritmo de busca local BLB logo após o algoritmo construtivo rDSPF. O valor entre parênteses $(-9,2\%)$ indica o quanto o valor médio das soluções se distancia daquele produzido pelo algoritmo BIP, ou seja, o valor médio das soluções obtidas após a busca local BLB é $9,2\%$ menor que o valor médio das soluções obtidas utilizando-se apenas o algoritmo construtivo BIP, para instâncias de 10 nós. O valor $10,67$, na linha média e mesma coluna BLB, representa o valor médio das soluções obtidas em todas as 850 instâncias teste utilizando-se BLB.

Em relação aos valores médios das soluções obtidas pelos algoritmos de busca local

Nós	BIP	rDSPF	BLB	BLB1	BLB2
10	11,85	11,45 (-3,3%)	10,76 (-9,2%)	11,10 (-6,3%)	10,76 (-9,2%)
25	12,50	11,75 (-6,0%)	11,13 (-10,9%)	11,48 (-8,1%)	11,14 (-10,9%)
50	11,77	11,17 (-5,1%)	10,62 (-9,8%)	10,99 (-6,6%)	10,62 (-9,8%)
75	11,62	11,09 (-4,6%)	10,58 (-8,9%)	10,90 (-6,2%)	10,58 (-8,9%)
100	11,66	11,05 (-5,3%)	10,54 (-9,6%)	10,87 (-6,8%)	10,55 (-9,5%)
200	11,14	10,64 (-4,5%)	10,21 (-8,4%)	10,48 (-5,9%)	10,21 (-8,4%)
média	11,79	11,22 (-4,8%)	**10,67 (-9,6%)**	11,00 (-6,7%)	10,67 (-9,5%)

Tabela 4.11: Valores médios das soluções obtidas com os algoritmos de busca local mais aprimorante ($\alpha = 2$).

mais aprimorante (Tabela 4.11), pode-se afirmar que o algoritmo que adota a vizinhança completa (BLB) apresenta resultados bastante semelhantes aos obtidos com a adoção da vizinhança de filhos mais custosos (BLB2), apresentando valor médio das soluções $9,6\%$ menor que o valor médio das soluções obtidas por BIP. Caso somente o algoritmo construtivo rDSPF fosse utilizado, o valor médio das soluções seria apenas $4,8\%$ menor, corroborando a importância da adoção de buscas locais após o algoritmo construtivo com o objetivo de reduzir o custo total da solução. Entretando, caso seja adotada a vizinhança reduzida BLB1 (consideração apenas dos nós folha), os resultados não são tão bons quanto os das outras duas, produzindo valor médio das soluções 2% menor do que o obtido pelo algoritmo construtivo rDSPF.

A Tabela 4.12 a seguir apresenta os tempos médios de processamento em milisegundos obtidos pelos algoritmos de busca local mais aprimorante. O valor entre parênteses indica a redução em relação ao algoritmo de busca local BLB.

O algoritmo de busca local com o menor tempo médio de processamento é BLB1, com valor 82% menor do que o tempo médio de processamento da busca local BLB (vizinhança completa). Entretanto, como o valor médio das soluções por ele obtidas é bastante acima do valor médio obtido com o algoritmo BLB2, pode-se afirmar que, em termos de desempenho total (tempo de processamento e valor médio das soluções), o melhor algoritmo de busca local mais aprimorante é o BLB2, com tempo médio de processamento 41% menor que o tempo médio de BLB e valor médio das soluções semelhante ao de BLB.

Nós	BIP	rDSPF	BLB	BLB1	BLB2
10	14	6	176	85 (-51%)	101 (-43%)
25	33	14	2517	738 (-71%)	1431 (-43%)
50	86	27	17157	3518 (-79%)	10085 (-41%)
75	161	50	55228	11053 (-80%)	31788 (-42%)
100	289	82	125034	23558 (-81%)	72635 (-42%)
200	1777	396	903061	155427 (-83%)	529850 (-41%)
média	312	78	141556	25160 (-82%)	82813 (-41%)

Tabela 4.12: Tempos médios de processamento em milisegundos obtidos pelos algoritmos de busca local mais aprimorante.

4.4.2 Resultados para a busca local primeiro aprimorante

A Tabela 4.13 a seguir apresenta os valores médios obtidos em uma execução para cada uma das 850 instâncias teste, com os algoritmos de busca local primeiro aprimorante descritos na Seção 2.4.2.2, adotando-se como solução de partida, em cada uma das instâncias teste, a mesma solução inicial adotada nos algoritmos da seção anterior. O valor entre parênteses indica o quanto o valor médio das soluções é menor do que aquele obtido pelo algoritmo BIP.

Nós	BIP	rDSPF	BLF	BLF1	BLF2	BLF2c	BLFc	BLFcr
10	11,85	11,45 (-3,3%)	10,77 (-9,1%)	11,10 (-6,3%)	10,76 (-9,2%)	10,83 (-8,6%)	10,77 (-9,1%)	10,79 (-8,9%)
25	12,50	11,76 (-6,0%)	11,14 (-10,9%)	11,48 (-8,1%)	11,14 (-10,9%)	11,23 (-10,1%)	11,13 (-10,9%)	11,14 (-10,9%)
50	11,77	11,17 (-5,1%)	10,63 (-9,7%)	10,99 (-6,6%)	10,64 (-9,7%)	10,71 (-9,0%)	10,62 (-9,8%)	10,63 (-9,7%)
75	11,62	11,09 (-4,6%)	10,59 (-8,9%)	10,90 (-6,2%)	10,60 (-8,8%)	10,66 (-8,3%)	10,59 (-8,9%)	10,59 (-8,9%)
100	11,66	11,05 (-5,3%)	10,56 (-9,5%)	10,87 (-6,8%)	10,56 (-9,5%)	10,62 (-8,9%)	10,55 (-9,5%)	10,55 (-9,5%)
200	11,14	10,64 (-4,5%)	10,22 (-8,3%)	10,48 (-5,9%)	10,22 (-8,2%)	10,29 (-7,7%)	10,22 (-8,3%)	10,22 (-8,3%)
média	11,79	11,22 (-4,8%)	10,67 **(-9,5%)**	11,00 (-6,7%)	10,68 (-9,4%)	10,75 (-8,9%)	10,67 **(-9,5%)**	10,68 (-9,4%)

Tabela 4.13: Valores médios das soluções obtidas com os algoritmos de busca local primeiro aprimorante ($\alpha = 2$).

A avaliação da Tabela 4.13 permite tecer os seguintes comentários:

- Assim como o resultado obtido nos algoritmos de busca local mais aprimorante, a adoção da vizinhança de nós folha na busca local primeiro aprimorante (BLF1)

trouxe a menor redução no valor médio das soluções, produzindo valor médio 6,7% menor do que o obtido pelo algoritmo BIP, apenas 2% menor que o valor médio das soluções obtidas pelo algoritmo construtivo rDSPF.

- BLF e BLFc diferem apenas na forma de visitação da vizinhança. Enquanto em BLF inicia-se uma nova busca de solução aprimorante do início da vizinhança, BLFc percorre a vizinhança de forma circular, iniciando uma nova busca do ponto em que se parou a busca anterior. Observa-se que a circularização não acarreta redução no valor médio das soluções em relação ao algoritmo BLF, produzindo soluções cujo valor médio, considerando-se as 850 instâncias, é semelhante ao valor médio obtido com o BLF, ou seja, 9,5% menor do que o obtido por BIP.

- A randomização da ordem de visitação dos nós em BLFcr (única característica diferente de BLFc) produziu valor médio das soluções semelhante ao obtido utilizando-se BLFc. Em outras palavras, as estratégias de circularização e randomização da ordem dos nós em BLF, produzem soluções tão boas quanto às produzidas pelo algoritmo BLF.

- A utilização da vizinhança de filhos mais custosos (BLF2) produz soluções cujo valor médio é quase idêntico ao valor médio das soluções obtidas com a adoção da vizinhança completa (BLF).

- O algoritmo BLF2c, versão circularizada do algoritmo BLF2, não produziu soluções com valor médio semelhante ao da versão não circularizada. O valor médio de suas soluções é 0,6% maior que o valor obtido com BLF2.

A Tabela 4.14 a seguir apresenta os tempos médios de processamento em milisegundos obtidos pelos algoritmos de busca local primeiro aprimorante. O valor entre parênteses indica a redução em relação ao algoritmo BLF. O valor em negrito indica o algoritmo com o menor tempo médio de processamento.

A análise dos tempos médios de processamento apresentados na Tabela 4.14 permite fazer as seguintes observações:

- O tempo médio de processamento do algoritmo BLF1 foi o segundo menor de todos os algoritmos de busca local primeiro aprimorante. Entretanto, como o valor médio das soluções por ele obtidas é o maior de todos, ele será descartado a partir desta etapa.

Nós	BIP	rDSPF	BLF	BLF1	BLF2	BLF2c	BLFc	BLFcr
10	14	6	130	68	69	63	117	118
				(-47%)	(-47%)	(-52%)	(-10%)	(-9%)
25	33	14	1619	545	872	584	1164	1201
				(-66%)	(-46%)	(-64%)	(-28%)	(-26%)
50	86	27	10398	2436	5985	2828	5743	5794
				(-77%)	(-42%)	(-73%)	(-45%)	(-44%)
75	161	50	32451	7152	17887	6754	14496	14424
				(-78%)	(-45%)	(-79%)	(-55%)	(-56%)
100	289	82	69678	14471	39060	12618	27378	27057
				(-79%)	(-44%)	(-82%)	(-61%)	(-61%)
200	1777	396	491839	92512	282981	53140	122290	122184
				(-81%)	(-42%)	(-89%)	(-75%)	(-75%)
média	312	78	78030	15238	44563	**10284**	23016	22950
				(-80%)	(-43%)	**(-87%)**	(-71%)	(-71%)

Tabela 4.14: Tempos médios de processamento em milisegundos obtidos pelos algoritmos de busca local primeiro aprimorante.

- A estratégia de circularização de BLF2 (algoritmo BLF2c) acarretou redução de quase 77% no tempo médio de processamento de BLF2, com um pequeno aumento no valor médio das soluções (menos de 1%).
- Em BLFc (versão circularizada de BLF), a redução do tempo médio de processamento foi de 71% em relação ao do BLF, semelhante ao desempenho de BLFcr (versão circularizada e com randomização da ordem dos nós de BLF).

Diante destes resultados, pode-se afirmar que as heurísticas BLF2c, BLFc e BLFcr são aquelas como o melhor desempenho tanto em termos de tempo médio de processamento e valor médio das soluções.

4.4.3 Comparação entre todas as buscas locais

A Tabela 4.15 resume os melhores resultados obtidos com as heurísticas de busca local primeiro e mais aprimorante (apresentados nas Tabelas 4.11 a 4.13). O valor entre parênteses na linha valor médio indica o percentual de redução em relação à BIP e o valor entre parênteses na linha tempo indica a variação do tempo médio de processamento em relação ao tempo médio do algoritmo de busca local BLF. Os valores em negrito da Tabela 4.15 indicam os resultados dos algoritmos de busca local que serão utilizados na fase de busca local do algoritmo GRASP e GRASP híbrido cujos resultados serão apresentados nas seções a seguir. A escolha dessas buscas locais para as demais etapas baseou-se, principalmente, no baixo tempo computacional e qualidade da solução.

Em relação aos resultados apresentados na Tabela 4.15, pode-se fazer as seguintes

	BIP	BLB2	BLF2c	BLFc	BLFcr
valor médio	11,79	10,67 (-9,5%)	**10,75 (-8,9%)**	**10,67 (-9,5%)**	10,68 (-9,4%)
tempo (ms)	312	82813 (6,1%)	**10284** (-86,8%)	**23016** (-70,5%)	22905 (-70,6%)

Tabela 4.15: Resumo com os melhores resultados dos algoritmos de busca local.

observações:

- A heurística de busca local mais aprimorante com o menor valor médio foi BLB2. Entretanto, observa-se que seu tempo médio de processamento é muito maior que os tempos médios dos algoritmos de busca local primeiro aprimorante e 6, 1% maior que o tempo médio da busca local BLF (cujo tempo médio é cerca de 78 segundos).

- A heurística BLF com circularização (BLFc) foi a busca local mais aprimorante com o melhor desempenho em relação ao valor médio das soluções, produzindo soluções cujo valor médio é semelhante àquele obtido com BLF, mas com tempo médio de processamento 71% menor que este. Por este motivo, BLFc será utilizado nos algoritmos GRASP e GRASP híbrido das próximas seções. O desempenho de BLFcr foi bastante simular ao de BLFc, mostrando que a estratégia de randomização da ordem dos nós não trouxe nenhuma redução adicional ao valor médio das soluções obtidas apenas com a circularização.

- BLF2c foi o algoritmo com o melhor desempenho em termos de tempo médio de processamento em relação a todas as heurísticas de busca local. A adoção da vizinhança de filhos mais custosos acrescida de circularização trouxe redução no tempo médio de processamento de BLF de quase 87%. Embora ele não tenha apresentado o mesmo desempenho que BLFc e BLFcr, em termos de valor médio das soluções, esse algoritmo também será utilizado nas próximas etapas do trabalho, pois a diferença entre o valor médio das soluções obtidas por ele e o valor médio das outras buscas locais (BLFc e BLFcr) é menor que 1% e a redução no tempo médio de processamento é bastante significativa.

4.5 Resultados obtidos com a metaheurística GRASP

A Seção 4.5.1 a seguir traz a descrição do gráfico de tempo para atingir o alvo que será utilizado para comparar diferentes versões do algoritmo GRASP. Na Seção 4.5.2, serão apresentados os gráficos construídos a partir dos resultados obtidos para algumas

instâncias teste com o algoritmo GRASP (adotando os algoritmos BLFc, BLF2c ou BLFcr na fase de busca local).

A Seção 4.5.3 traz a descrição dos testes efetuados com o algoritmo GRASP e apresenta os resultados experimentais em termos de valor médio das soluções e tempo médio de processamento para todas as estratégias de busca local utilizadas.

Um resumo com os melhores resultados do algoritmo GRASP antes da implementação do GRASP híbrido está na Seção 4.5.4.

A Seção 4.5.5 apresenta os resultados experimentais do algoritmo GRASP híbrido com a inserção da reconexão por caminhos e a adoção de algumas estratégias de perturbação e variação de parâmetros, concluindo a etapa de apresentação de resultados deste trabalho.

4.5.1 Gráfico de tempo para valor alvo

Aiex et al. [3] estudaram o comportamento do tempo de processamento do algoritmo GRASP para encontrar uma solução menor ou igual a um valor pré-determinado (valor alvo). Para a investigação, eles selecionaram quatro problemas teste da literatura e, para cada uma destas instâncias, determinaram três valores alvo para as soluções entre os valores mínimo e máximo produzidos pelo GRASP. Para cada valor alvo, mediram os tempos de processamento para encontrar uma solução tão boa quanto o alvo e estudaram sua distribuição de probabilidade. Para comparar as distribuições de probabilidade empírica e teórica do tempo de processamento do GRASP para atingir o alvo, seguiram uma metodologia gráfica padrão para análise de dados [20], que será descrita a seguir. A principal conclusão dos experimentos de Aiex et al. [3] é que o tempo para alcançar o valor alvo pelo algoritmo GRASP se aproxima de uma distribuição exponencial de dois parâmetros, cuja função de probabilidade é dada por $f(t) = \frac{1}{\lambda}e^{(t-\mu)/\lambda}$, onde λ é a média da distribuição dos tempos de processamento e μ é a medida de quanto a distribuição se afasta do eixo das ordenadas. Eles mostraram que a aproximação é ainda melhor à medida que aumenta a dificuldade de se encontrar uma solução para um dado valor alvo.

A Figura 4.1 a seguir mostra distribuições empíricas da variável aleatória *tempo para valor alvo*. Para construir o gráfico da distribuição empírica, fixa-se um valor alvo para a solução de uma instância e executa-se cada algoritmo N vezes independentes, armazenando-se o tempo em que uma solução com custo no mínimo tão bom quanto o valor alvo é encontrada. A seguir, ordenam-se os tempos obtidos em ordem crescente. Para cada algoritmo, associa-se ao i-ésimo tempo de processamento ordenado (t_i) uma

probabilidade $p_i = (i - \frac{1}{2})/N$ e constrói-se um gráfico com os pontos $z_i = (t_i, p_i)$, para $i = 1, \ldots, N$. A Figura 4.1 ilustra o gráfico de distribuição de probabilidade cumulativa do tempo para um valor alvo obtido para uma instância teste.

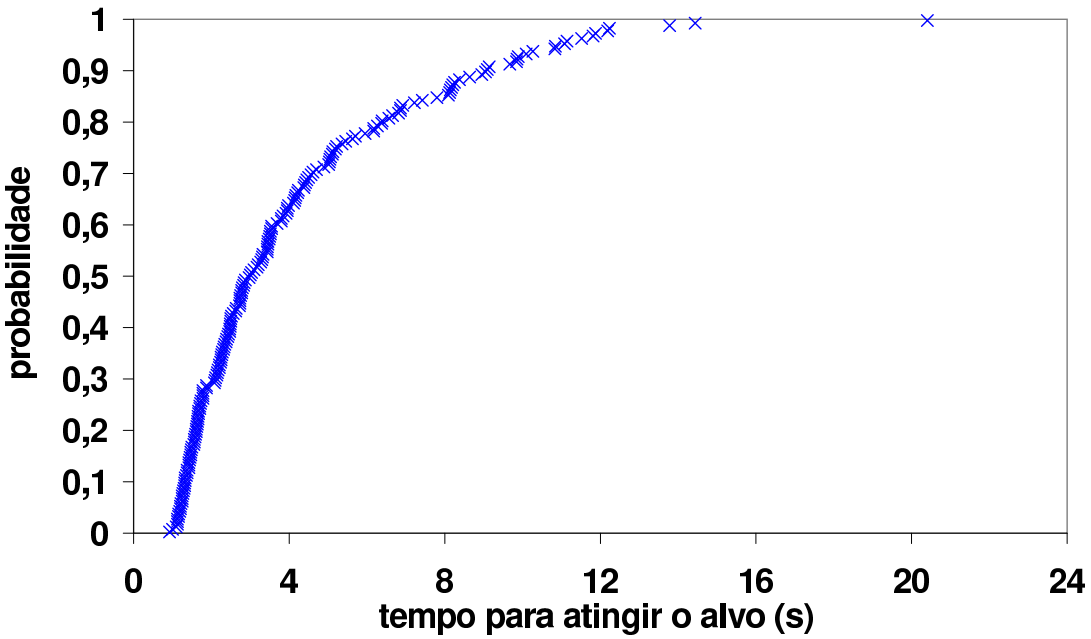

Figura 4.1: Distribuição de probabilidade cumulativa dos tempos medidos.

Para estimar os parâmetros de uma distribuição exponencial de dois parâmetros [20], primeiro constrói-se o gráfico teórico Quantil-Quantil (ou gráfico Q-Q) para os tempos obtidos. Para descrever os gráficos Q-Q, é necessário indicar que a função de distribuição cumulativa para uma distribuição exponencial de dois parâmetros é dada por $F(t) = 1 - e^{(t-\mu)/\lambda}$. Para cada valor de p_i, $i = 1, \ldots, N$, associa-se um p_i-quantil $Qt(p_i)$ da distribuição teórica. Para cada p_i-quantil tem-se, por definição, que $F(Qt(p_i)) = p_i$. Daí, $Qt(p_i) = F^{-1}(p_i)$ e tem-se, então, para a distribuição exponencial de dois parâmetros $Qt(p_i) = -\lambda \ln(1 - p_i) + \mu$.

Os quantis de uma distribuição empírica são simplesmente os tempos de processamento obtidos e ordenados. O gráfico teórico Quantil-Quantil é obtido traçando-se os quantis dos tempos de um distribuição empírica contra os quantis de uma distribuição teórica. Isto envolve três etapas. Primeiro, os tempos medidos são ordenados em ordem crescente. Segundo, são obtidos os quantis da distribuição exponencial teórica. Finalmente, constrói-se o gráfico dos tempos medidos contra os quantis teóricos.

Na situação onde a distribuição dos tempos teórica é uma boa aproximação para a distribuição empírica, os pontos no gráfico Q-Q terão uma configuração quase linear. Se os parâmetros λ e μ da distribuição teórica que melhor se ajustam aos tempos medidos puderem ser estimados a priori, os pontos no gráfico Q-Q tendem a seguir a reta $x = y$.

Alternativamente, em um gráfico dos tempos medidos contra uma distribuição exponencial com $\lambda = 1$ e $\mu = 0$ para o quantil teórico, os pontos tenderão a seguir a linha $y = \hat{\lambda}x + \hat{\mu}$ (onde $\hat{\lambda}$ e $\hat{\mu}$ são os parâmetros estimados após a construção do gráfico Q-Q com os resultados experimentais). Conseqüentemente, os parâmetros λ e μ de uma distribuição exponencial de dois parâmetros podem ser estimados, respectivamente pela inclinação $\hat{\lambda}$ e pela interseção $\hat{\mu}$ da reta descrita no gráfico Q-Q. A Figura 4.2 a seguir mostra o gráfico dos tempos obtidos para um dado valor alvo de uma instância teste, no eixo vertical, contra os quantis de uma distribuição exponencial de dois parâmetros com $\lambda = 1$ e $\mu = 0$ no eixo horizontal, dada por $Qt(p_i) = -\lambda \ln(1 - p_i) + \mu = -\ln(1 - p_i)$, para $i = 1, \ldots, N$.

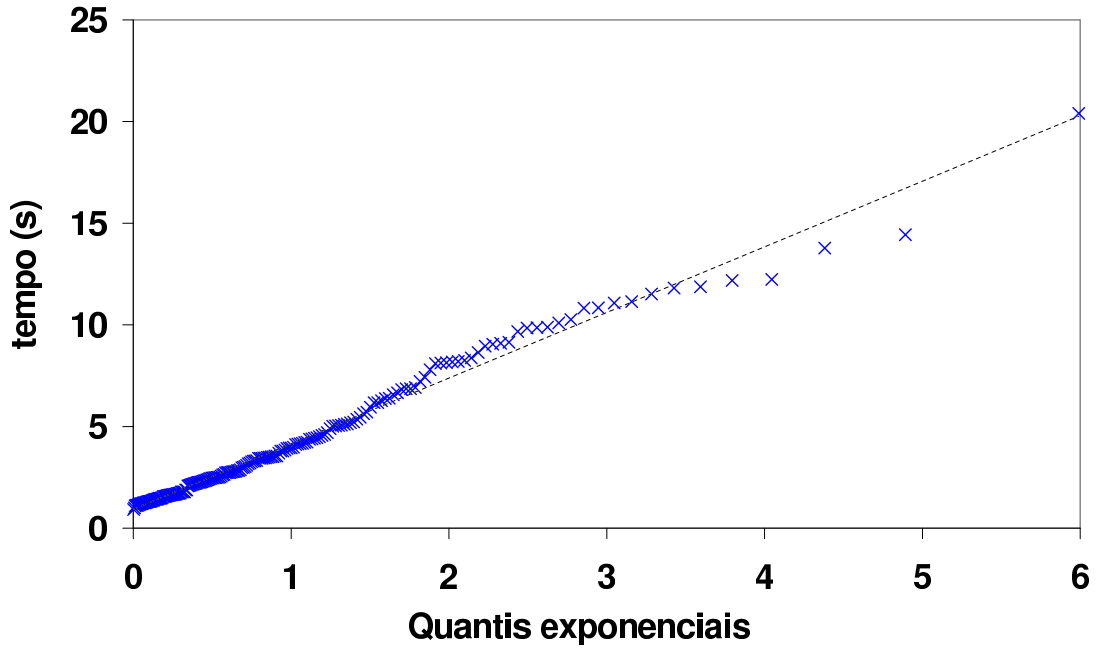

Figura 4.2: Gráfico Q-Q dos tempos medidos × quantil da distribuição exponencial.

Uma vez estimados os dois parâmetros da distribuição ($\hat{\lambda}$ e $\hat{\mu}$) (utilizando-se, por exemplo, o método dos mínimos quadrados com os tempos medidos), pode-se fazer um gráfico superposto das distribuições empírica e teórica. A Figura 4.3 ilustra as distribuições empírica e teórica obtidas superpostas para uma mesma instância teste.

O gráfico de tempo para valor alvo descrito nesta Seção será utilizado para comparar as diferentes versões do algoritmo GRASP implementadas.

4.5.2 Gráfico GRASP com BLFc, BLFcr e BLF2c

Para comparar o comportamento do algoritmo GRASP (descrito na Seção 3.1) com a adoção de diferentes buscas locais (BLFc, BLFcr e BLF2c), foram construídos gráficos que mostram as distribuições empíricas da variável aleatória *tempo para valor alvo* para

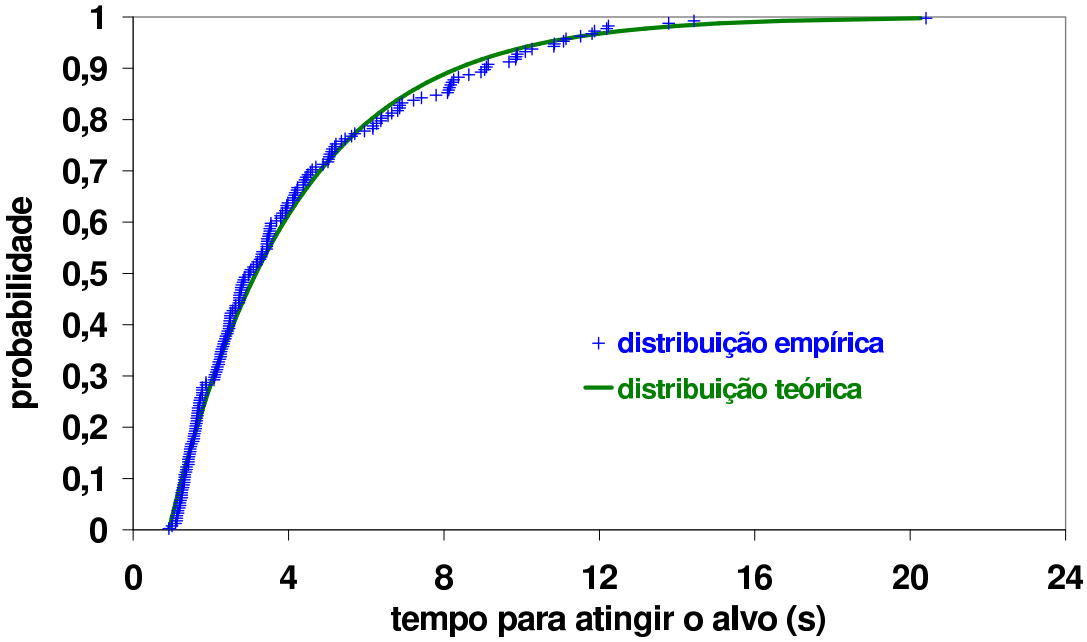

Figura 4.3: Distribuições empírica e teórica superpostas.

cada um dos algoritmos, assim como descrito na seção anterior. Foram escolhidas cinco instâncias teste (uma de cada tamanho) para estes testes.

Para construir o gráfico da distribuição empírica, foram determinados três valores alvo para a solução entre os valores mínimos e máximo produzidos pelo GRASP, para as instâncias de 25, 50 e 75 nós: um de dificuldade fácil, um de média e outro de alta, próximo do melhor obtido após 50 iterações do algoritmo GRASP com rDSPF na fase construtiva e BLF2c na fase de busca local. Para as redes de 100 e 200 nós, escolheu-se apenas um valor alvo, de dificuldade média. Executou-se cada algoritmo 200 vezes independentes, para cada uma das instâncias escolhidas e cada valor alvo, armazenando-se o tempo quando uma solução com custo no mínimo tão bom quanto o valor alvo fosse encontrada. Os gráficos das distribuições empíricas foram construídos após estimativa dos parâmetros $\hat{\lambda}$ e $\hat{\mu}$, utilizando-se o método dos mínimos quadrados, de cada gráfico Q-Q construído com os tempos obtidos experimentalmente, tal como descrito na Seção anterior. As Figuras 4.4 a 4.6 trazem os resultados obtidos para a instância de 25 nós para os três valores alvo fixados.

Para todas as instâncias teste, o comportamento da variável tempo para atingir o valor alvo é bastante semelhante entre o GRASP com a adoção da busca local BLFc e o com a adoção de BLFcr. Caso se considere a busca local BLF2c, a curva situa-se mais próxima ao eixo vertical, indicando que o tempo para atingir o alvo para essa heurística é bem menor do que o dos outros dois.

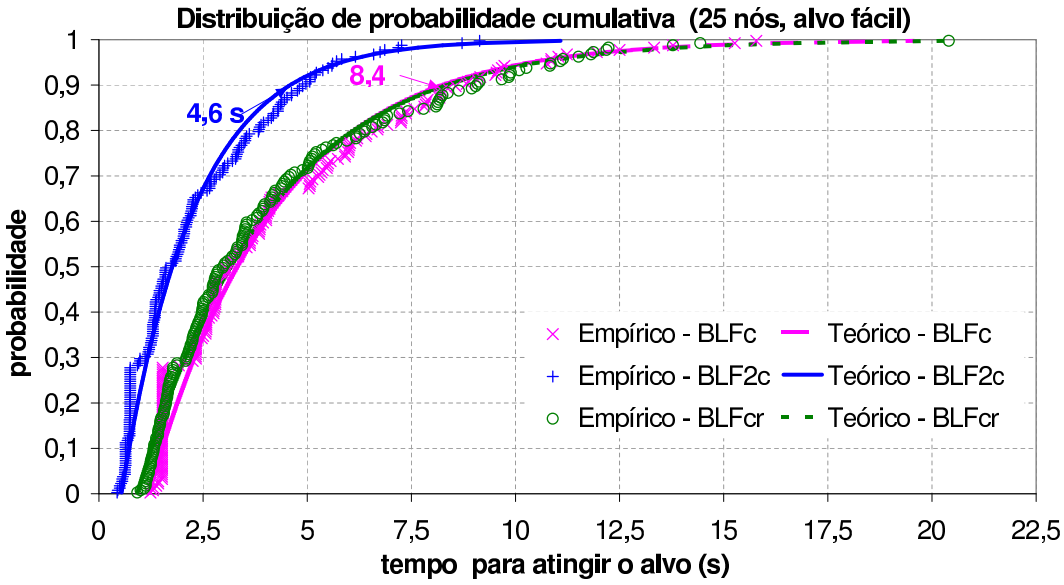

Figura 4.4: Tempo de processamento do GRASP para alvo fácil (25 nós).

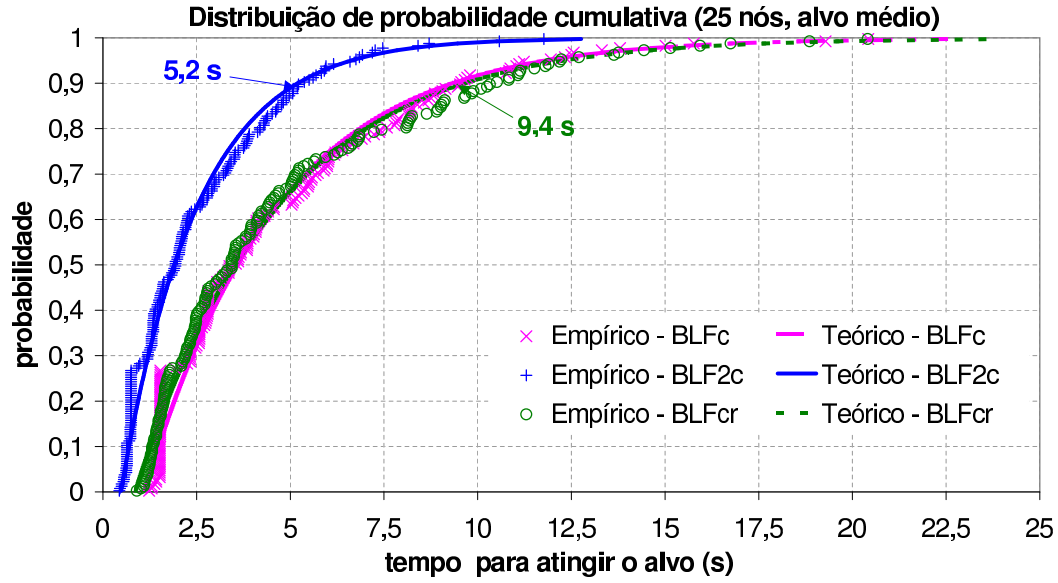

Figura 4.5: Tempo de processamento do GRASP para alvo médio (25 nós).

Para a instância de 25 nós, fixando-se um alvo fácil (Figura 4.4) por exemplo, observa-se que em 90% das 200 execuções (equivalente ao valor 0,9 no eixo vertical) o algoritmo GRASP com BLF2c atinge o valor alvo em aproximadamente 4,6 segundos, enquanto que os algoritmos GRASP com BLFc e com BLFcr levam aproximadamente 8,4 segundos para alcançar esse mesmo valor alvo. Para um alvo de dificuldade média (Figura 4.5), os tempos em 90% das 200 execuções para GRASP com BLF2c e GRASP com BLFc são, respectivamente, 5,2 e 9,4 segundos e, para um alvo difícil (com custo próximo ao menor obtido para esta instância, Figura 4.6), os tempos para GRASP com BLF2c e GRASP com BLFc são, respectivamente, 74 e 106 segundos. Pode-se observar também que a curva com os dados empíricos é bem próxima da curva teórica construída a partir dos

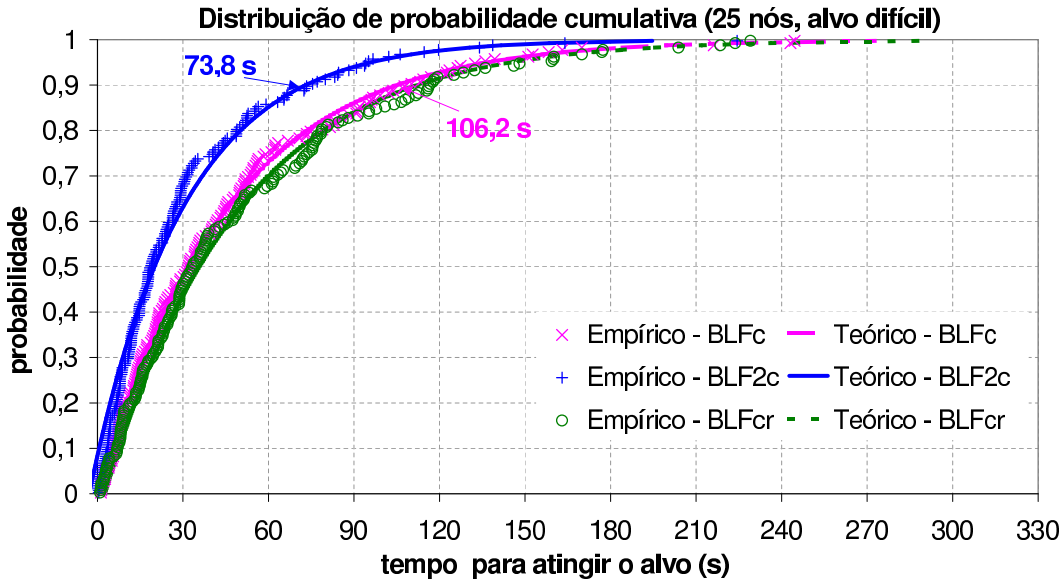

Figura 4.6: Tempo de processamento do GRASP para alvo difícil (25 nós).

parâmetros $\hat{\lambda}$ e $\hat{\mu}$, estimados pelos tempos experimentais.

As Figuras 4.7 a 4.9 trazem os resultados obtidos para as instâncias de 50 nós para os três valores alvo fixados.

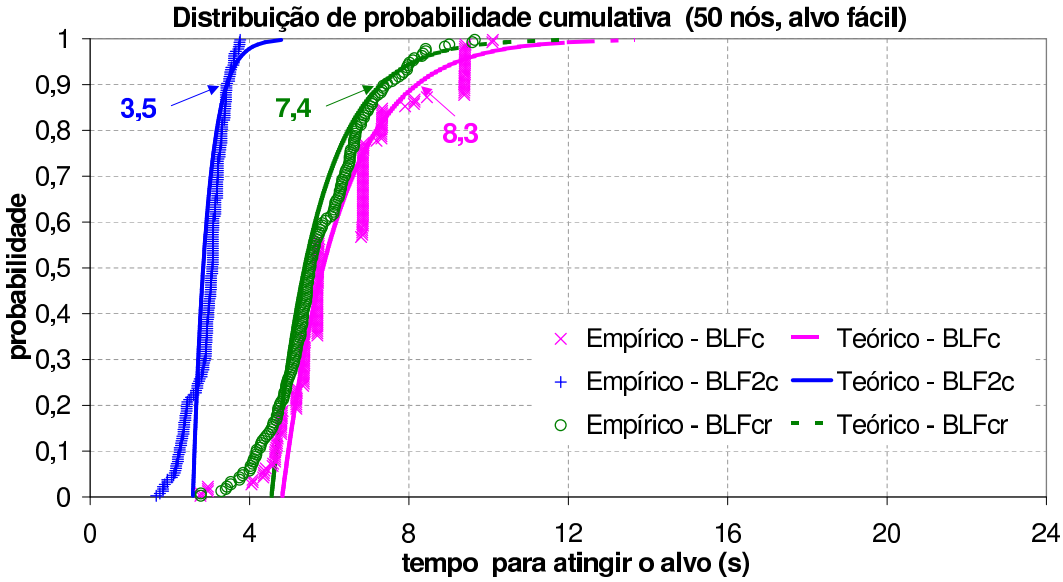

Figura 4.7: Tempo de processamento do GRASP para alvo fácil (50 nós).

Para a instância de 50 nós, fixando-se um alvo fácil (Figura 4.7), observa-se que, em 90% das 200 execuções, o algoritmo GRASP com BLF2c atinge o valor alvo em aproximadamente $3,5$ segundos, enquanto que os algoritmos GRASP com BLFcr e com BLFc levam aproximadamente $7,4$ segundos e $8,3$ segundos, respectivamente, para alcançar esse mesmo valor alvo. Uma outra observação em relação ao alvo de dificuldade fácil é que, na maioria das 200 execuções, BLFc, BLF2c e BLFcr atingem o alvo em poucas iterações

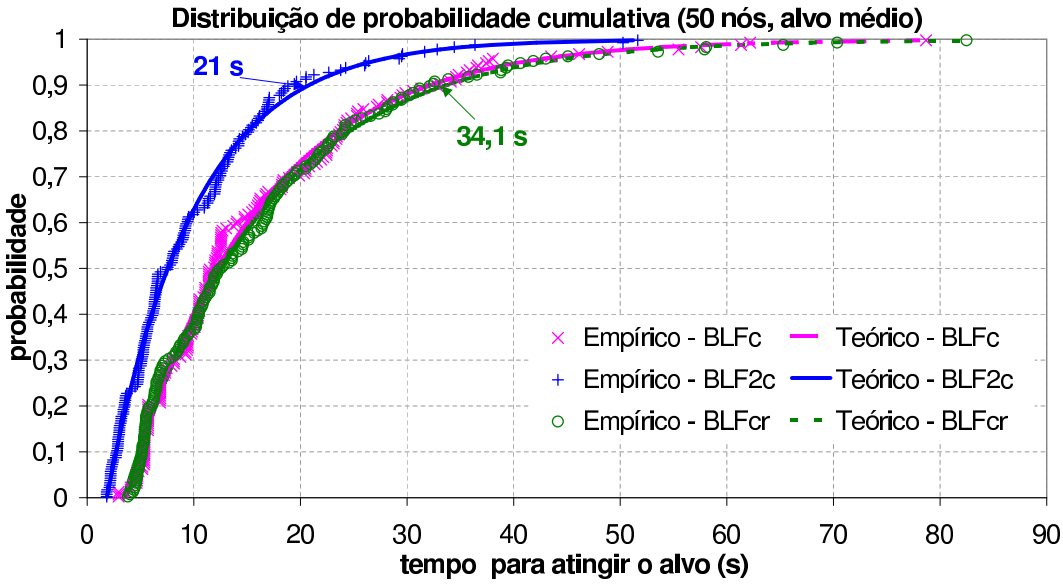

Figura 4.8: Tempo de processamento do GRASP para alvo médio (50 nós).

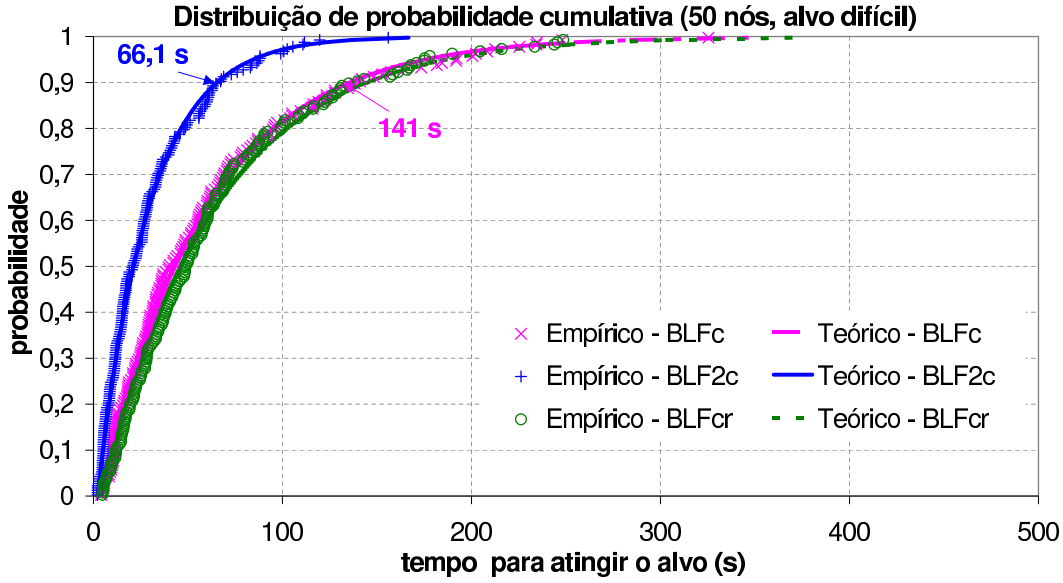

Figura 4.9: Tempo de processamento do GRASP para alvo difícil (50 nós).

do GRASP. Isso justifica o fato de o tempo experimental para atingir o alvo ser quase idêntico em muitas das 200 execuções (pontos que formam linhas quase paralelas ao eixo vertical indicando tempo de processamento idêntico). Assim, para um alvo muito fácil, a curva da distribuição exponencial de dois parâmetros teórica não é uma boa aproximação para os tempos experimentais. Para um alvo de dificuldade média (Figura 4.8), os tempos em 90% das 200 execuções para GRASP com BLF2c e GRASP com BLFc (ou com BLFcr, pois são muito próximos) são, respectivamente, 21 e 34,1 segundos. Para um alvo difícil (Figura 4.9), os tempos para GRASP com BLF2c e GRASP com BLFc são, respectivamente, 66 e 141 segundos. Observa-se que a curva com os dados empíricos torna-se mais próxima da curva teórica, construída com os parâmetros $\hat{\lambda}$ e $\hat{\mu}$ estimados a

partir dos tempos experimentais, à medida que o valor alvo torna-se mais difícil.

As Figuras 4.10 a 4.12 trazem os resultados obtidos para as instâncias de 75 nós para os três valores alvo fixados.

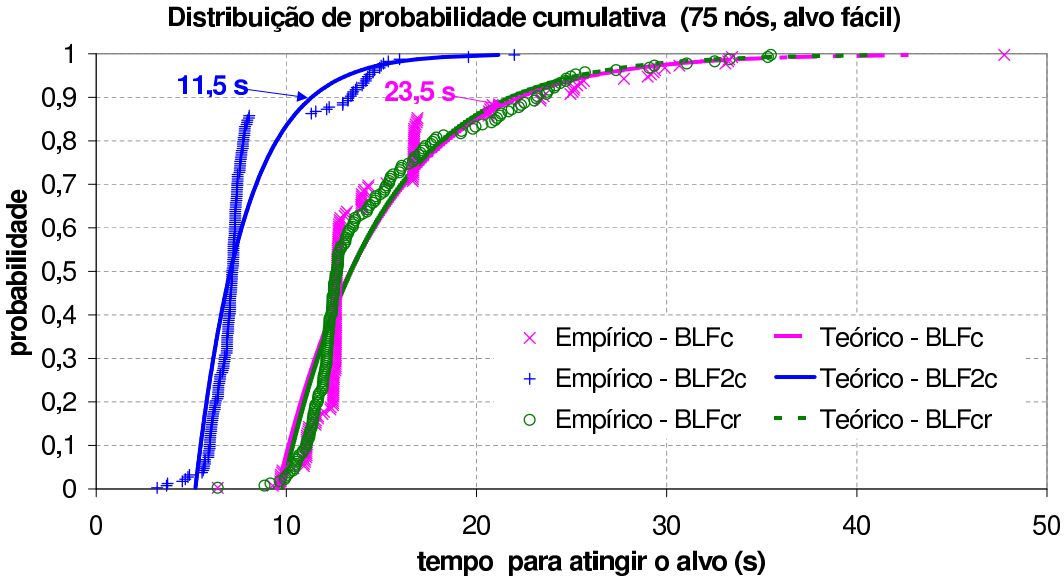

Figura 4.10: Tempo de processamento do GRASP para alvo fácil (75 nós).

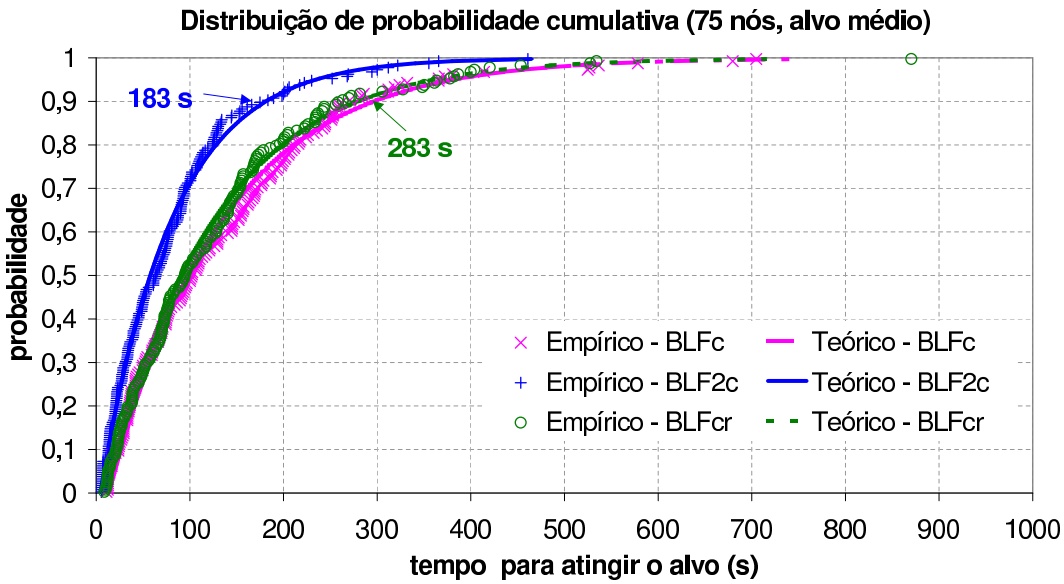

Figura 4.11: Tempo de processamento do GRASP para alvo médio (75 nós).

Para a instância de 75 nós, fixando-se um alvo fácil (Figura 4.10), observa-se que em 90% das 200 execuções o algoritmo GRASP com BLF2c atinge o valor alvo em aproximadamente 11,5 segundos, enquanto o GRASP com BLFc (ou com BLFcr, pois são muito próximos) leva aproximadamente 23,5 segundos para alcançar esse mesmo valor alvo. Na maioria das 200 execuções, BLFc, BLF2c e BLFcr atingem o alvo em poucas iterações do

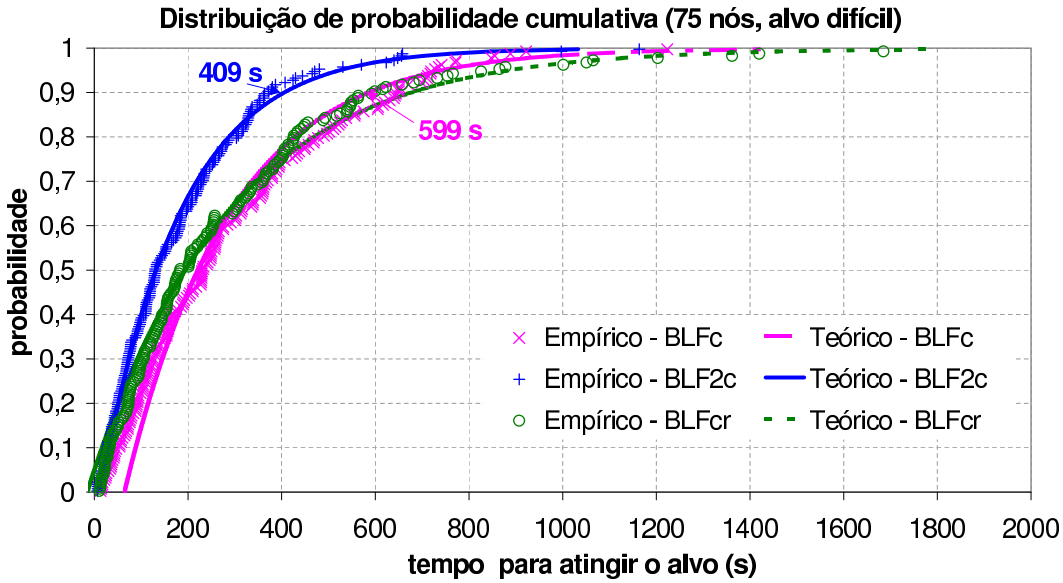

Figura 4.12: Tempo de processamento do GRASP para alvo difícil (75 nós).

GRASP, justificando tempos experimentais para atingir o alvo idênticos em muitas das 200 execuções (pontos que formam linhas quase paralelas ao eixo vertical). Assim, para a instância de 75 nós, a curva da distribuição exponencial de dois parâmetros teórica para um alvo fácil também não é uma boa aproximação para os tempos experimentais, assim como o ocorrido com a instância de 50 nós. Para um alvo de dificuldade média (Figura 4.11), os tempos em 90% das 200 execuções para GRASP com BLF2c e GRASP com BLFc são, respectivamente, 183 e 283 segundos e, para um alvo difícil (Figura 4.11), os tempos para GRASP com BLF2c e GRASP com BLFc são, respectivamente, 409 e 599 segundos. Também observa-se que a curva com os dados empíricos torna-se mais próxima da curva teórica à medida que o valor alvo torna-se mais difícil.

As Figuras 4.13 a 4.14 trazem os resultados obtidos para as instâncias de 100 e 200 nós para um valor alvo de dificuldade média.

Para a instância de 100 nós (Figura 4.13), os valores de tempo para o alvo médio em 90% das execuções são abaixo de 45 e 88 segundos para GRASP com BLF2c e com BLFc, respectivamente. Para a instância de 200 nós (Figura 4.14), os valores de tempo para o alvo em 90% das execuções são abaixo de 764 e 1087 segundos para GRASP com BLF2c e com BLFc, respectivamente.

Diante destes resultados e daqueles já apresentados na seção anterior, optou-se em prosseguir os testes com o GRASP adotando-se apenas as buscas locais BLF2c e BLFc. Ou seja, não serão realizados testes com a utilização de BLFcr no GRASP, por apresentar comportamento semelhante ao do GRASP com BLFc.

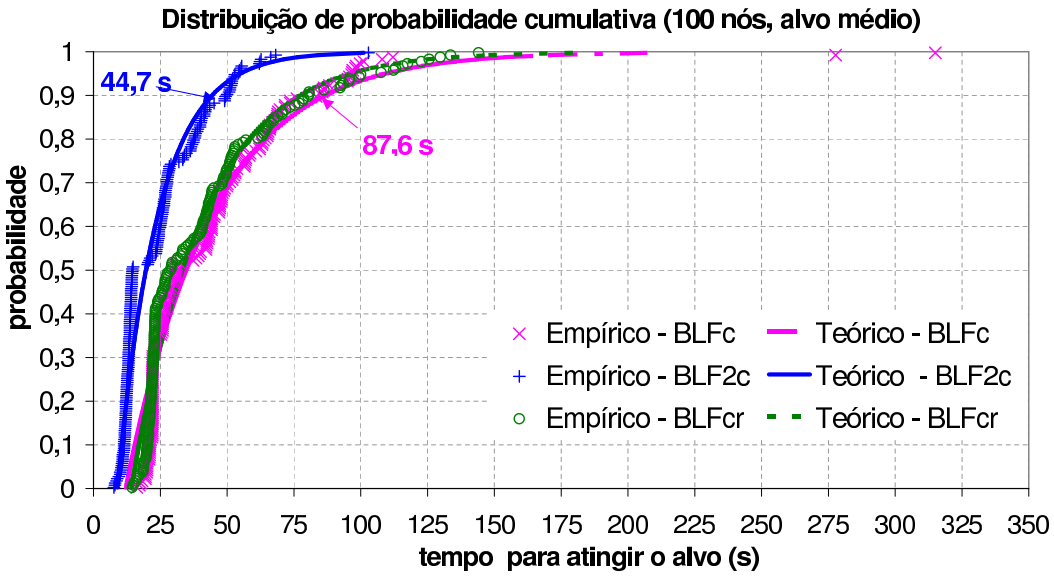

Figura 4.13: Tempo de processamento do GRASP para alvo médio (100 nós).

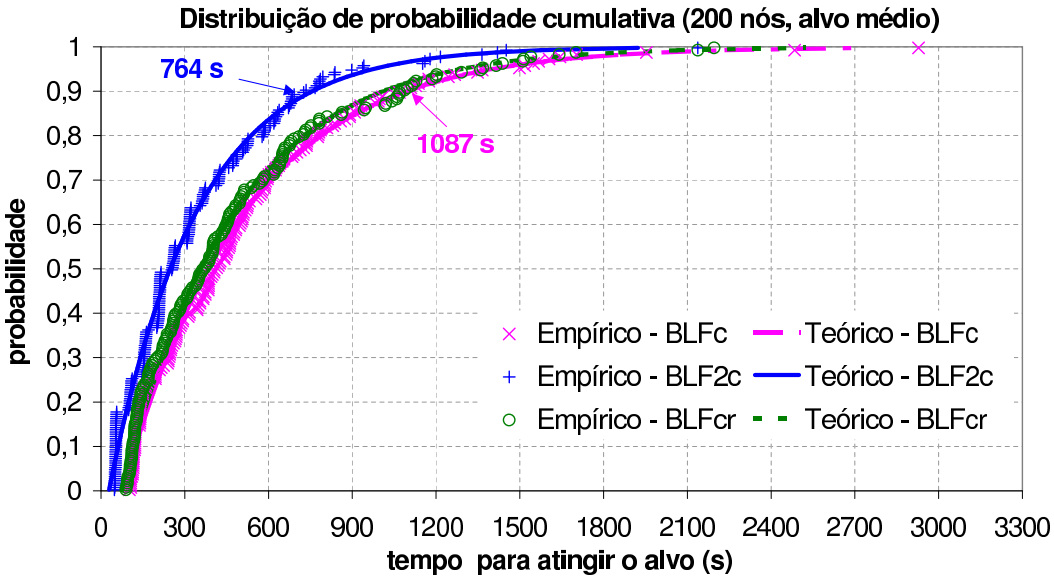

Figura 4.14: Tempo de processamento do GRASP para alvo médio (200 nós).

4.5.3 GRASP

Foram empregadas as seguintes estratégias para o algoritmo GRASP descrito na Seção 3.1:

- GRASPh - o parâmetro β é selecionado aleatoriamente a cada iteração com distribuição de probabilidades fixa e mais concentrada nos percentuais em que obteve-se os melhores resultados experimentais (as probabilidades foram escolhidas após observação dos resultados experimentais do algoritmo rDSPF, procurando favorecer o valores de β que produziram os melhores valores médios das soluções):

$p(\beta = 0) = 3\%$, $p(\beta = 0,1) = 3\%$, $p(\beta = 0,2) = 6\%$, $p(\beta = 0,3) = 5\%$, $p(\beta = 0,4) = 8\%$, $p(\beta = 0,5) = 12\%$, $p(\beta = 0,6) = 12\%$, $p(\beta = 0,7) = 13\%$, $p(\beta = 0,8) = 13\%$, $p(\beta = 0,9) = 13\%$ e $p(\beta = 1) = 12\%$.

- GRASPf - parâmetro β fixo e igual a $0,6$ (não foi escolhido o valor $0,7$, porque não produziu os melhores resultados para redes com mais de 75 nós).

- GRASPe - o parâmetro β é selecionado aleatoriamente a cada iteração com distribuição de probabilidades equiprovável no intervalo $[0,1]$.

- GRASPh_gl - o parâmetro β é selecionado aleatoriamente a cada iteração com distribuição de probabilidades fixa e mais concentrada no intervalo $[0,1,\ 0,3]$:

  $p(\beta = 0\%) = 0\%$, $p(\beta = 0,1) = 25\%$, $p(\beta = 0,2) = 37,5\%$, $p(\beta = 0,3) = 25\%$, $p(\beta = 0,4) = 3\%$, $p(\beta = 0,5) = 3\%$, $p(\beta = 0,6) = 3\%$, $p(\beta = 0,7) = 1\%$, $p(\beta = 0,8) = 1\%$, $p(\beta = 0,9) = 1\%$ e $p(\beta = 1) = 0,5\%$.

- GRASPf_gl - parâmetro β fixo e igual a $0,3$ (utilizado para corroborar que valores de β próximos da escolha gulosa não produzem os melhores resultados).

- GRASPh_ew - o parâmetro β é selecionado aleatoriamente a cada iteração, com a mesma distribuição de probabilidades de GRASPh e com a inserção do algoritmo EWMA em cada iteração, logo após a busca local BLFc (EWMA não foi empregado com as outras estratégias de variação de β, porque GRASPh é a que apresenta os melhores resultados, conforme será apresentado na Tabela 4.16).

Embora o procedimento correto para a comparação entre diferentes versões de um algoritmo GRASP seja fixar o tempo de processamento e verificar o resultado obtido para cada algoritmo, para estes testes, por simplificação, a comparação foi realizada para um número fixo de iterações. A Tabela 4.17 traz o tempos de processamento médios de cada versão implementada do GRASP e mostram que eles são bastantes próximos entre si, pois as variações do parâmetro β não alteram muito os tempos de processamento do GRASP (variação máxima inferior a 2%). Por esse motivo, não faz muita diferença fazer a comparação dos algoritmos com um número fixo de iterações ou com um tempo de processamento fixo.

A Tabela 4.16 apresenta os resultados obtidos para as 850 instâncias testes, utilizando o algoritmo rDSPF na fase construtiva e o algoritmo BLFc (busca local primeiro aprimorante circular com vizinhança completa) na fase de busca local. Para estes testes

adotou-se o número de iterações igual a 50 e a mesma semente inicial para a geração dos números pseudo-aleatórios.

Nós	BIP	GRASPh	GRASPf	GRASPe	GRASPh_gl	GRASPf_gl	GRASPh_ew
10	11,85	10,33 (-12,8%)	10,49 (-11,4%)	10,32 (-12,9%)	10,53 (-11,1%)	10,67 (-9,9%)	10,24 (-13,5%)
25	12,50	10,42 (-16,6%)	10,48 (-16,2%)	10,42 (-16,7%)	10,82 (-13,4%)	10,73 (-14,1%)	10,34 (-17,3%)
50	11,77	10,00 (-15,0%)	10,02 (-14,9%)	10,02 (-14,9%)	10,40 (-11,7%)	10,13 (-13,9%)	9,90 (-15,9%)
75	11,62	9,97 (-14,2%)	10,00 (-13,9%)	10,01 (-13,8%)	10,46 (-9,9%)	10,07 (-13,3%)	9,88 (-14,9%)
100	11,66	10,05 (-13,8%)	10,06 (-13,7%)	10,08 (-13,5%)	10,50 (-9,9%)	10,11 (-13,3%)	9,95 (-14,7%)
200	11,14	9,82 (-11,8%)	9,82 (-11,9%)	9,83 (-11,7%)	10,26 (-7,9%)	9,84 (-11,7%)	9,85 (-11,6%)
média	11,79	10,12 (-14,2%)	10,16 (-13,8%)	10,13 (-14,1%)	10,51 (-10,9%)	10,28 (-12,8%)	**10,04 (-14,9%)**

Tabela 4.16: Valores médios das soluções obtidas pelas diferentes versões do algoritmo GRASP utilizando a busca local BLFc.

Em relação aos resultados apresentados na Tabela 4.16, podem-se fazer as seguintes observações:

- Caso seja adotada a estratégia fixa para β, observa-se que o valor médio das soluções considerando $\beta = 0,6$ (GRASPf) é menor do que com $\beta = 0,3$ (GRASPf_gl), comportamento que já havia sido identificado nos resultados adotando-se apenas a heurística construtiva rDSPF. A adoção de uma parâmetro mais próximo à escolha gulosa ($\beta = 0,3$) em rDSPF não traz a diversidade necessária para encontrar soluções de boa qualidade.

- Comparando-se GRASPh com GRASPh_gl, que diferem apenas na concentração da distribuição de probabilidades de β, observa-se resultados melhores com a adoção da distribuição de probabilidades de GRASPh. O valor médio das soluções obtidas com GRASPh é $14,2\%$ menor que o obtido pelo algoritmo BIP, enquanto que o valor médio de GRASPh_gl, é $10,9\%$ menor que o do BIP.

- Uma distribuição de probabilidades equiprovável de β (GRASPe) também produz resultados tão bons quanto os produzidos por GRASPh, obtendo soluções cujo valor médio é $14,1\%$ menor do que o obtido por BIP.

- A estratégia com o melhor desempenho foi a que utilizou β com distribuição de probabilidades fixa e mais concentrada nos percentuais em que obteve-se os melhores resultados experimentais, associada à inserção da heurística EWMA (descrita

na Seção 2.3) após a fase de busca local (GRASPh_ew), produzindo soluções cujo valor médio é 14,9% menor do que o obtido por BIP. Comparando-se GRASPh e GRASPh_ew, que diferem apenas na inserção do EWMA, verifica-se que a inserção do EWMA acarreta redução de 0,7% no valor médio das soluções obtidas por GRASPh.

A Tabela 4.17 apresenta os tempos médios de processamento em segundos, obtidos pelos algoritmos GRASP descritos acima (com BLFc na fase de busca local). Observa-se

Nós	BIP	GRASPh	GRASPf	GRASPe	GRASPh_gl	GRASPf_gl	GRASPh_ew
10	0,014	5,07	5,08	5,13	5,33	5,35	5,23
25	0,033	43,32	43,02	43,83	45,41	44,82	43,43
50	0,086	211,93	210,26	213,53	218,42	216,13	212,19
75	0,16	548,32	545,9	554,33	563,05	557,58	549,54
100	0,29	1071,59	1066,56	1082,6	1097,2	1091,06	1075,48
200	17,77	5284,75	5264,95	5318,25	5370,26	5337,86	5361,48
média	0,312	954,41	950,3	962,02	973,99	967,45	963,57

Tabela 4.17: Tempos médios de processamento em segundos obtidos pelos algoritmos GRASP utilizando-se a busca local BLFc.

que a estratégia $\beta = 0,6$ (GRASPf) é a que obtém o menor tempo médio, seguida pela estratégias GRASPh, equiprovável (GRASPe), GRASPh com EWMA (GRASPh_ew), $\beta = 0,3$ (GRASPf_gl) e GRASPh com distribuição de probabilidades de β mais concentrada em $[0,1,\ 0,3]$. A diferença percentual entre o maior e o menor tempo médio é inferior a 2%, assim, o valor médio das soluções é o principal fator determinamente na identificação do algoritmo com o melhor desempenho. A inserção do algoritmo EWMA em cada iteração, logo após a fase de busca local (característica que diferencia GRASPh de GRASPh_ew) acarreta acréscimo de apenas 1% no tempo médio de processamento de GRASPh, sendo a estratégia que produz soluções com o menor valor médio.

A Tabela 4.18 traz os resultados obtidos para as 850 instâncias testes, utilizando o algoritmo rDSPF na fase construtiva e o algoritmo BLF2c (busca local primeiro aprimorante circular com vizinhança filhos mais custosos) na fase de busca local. Assim como no GRASP com BLFc, também adotou-se, arbitrariamente, o número fixo de iterações igual a 50 e a mesma semente inicial para a geração dos números pseudo-aleatórios utilizada para as simulações com BLFc. Foram empregadas as mesmas estratégias de variação do parâmetro β descritas no início desta seção, apenas alterando-se o algoritmo de busca local para BLF2c.

Em relação aos resultados apresentados na Tabela 4.18, podem-se fazer as seguintes observações:

Nós	BIP	GRASPh	GRASPf	GRASPe	GRASPh_gl	GRASPf_gl	GRASPh_ew
10	11,85	10,33 (-12,8%)	10,50 (-11,3%)	10,32 (-12,9%)	10,51 (-11,3%)	10,74 (-9,4%)	10,25 (-13,5%)
25	12,50	10,46 (-16,3%)	10,50 (-16,0%)	10,43 (-16,6%)	10,65 (-14,8%)	10,77 (-13,9%)	10,29 (-17,7%)
50	11,77	10,06 (-14,5%)	10,08 (-14,4%)	10,08 (-14,4%)	10,19 (-13,5%)	10,19 (-13,4%)	9,82 (-16,6%)
75	11,62	10,04 (-13,6%)	10,06 (-13,4%)	10,07 (-13,3%)	10,12 (-12,9%)	10,13 (-12,8%)	9,76 (-16,0%)
100	11,66	10,11 (-13,3%)	10,12 (-13,2%)	10,14 (-13,0%)	10,18 (-12,6%)	10,17 (-12,8%)	9,81 (-15,8%)
200	11,14	9,88 (-11,3%)	9,88 (-11,3%)	9,90 (-11,2%)	9,91 (-11,0%)	9,90 (-11,1%)	9,67 (-13,2%)
média	11,79	10,16 (-13,8%)	10,21 (-13,4%)	10,17 (-13,8%)	10,28 (-12,8%)	10,34 (-12,3%)	**9,95 (-15,6%)**

Tabela 4.18: Valores médios das soluções obtidas pelas diferentes versões do algoritmo GRASP utilizando a busca local BLF2c.

- Entre GRASPf_gl ($\beta = 0,3$) e GRASPf ($\beta = 0,6$), observa-se uma diferença entre os valores médios das soluções de $0,6\%$, corroborando o resultando já obtido com a busca local BLFc, de que a utilização de $\beta = 0,6$ produz valores médios menores do que aqueles obtidos com a adoção de $\beta = 0,3$ para todos os tamanhos de rede testados.

- Comparando-se o algoritmo GRASPh, com distribuição de probabilidades mais concentrada nos resultados experimentais, e aquele com distribuição mais concentrada em $[0,1,\ 0,3]$ (GRASPh_gl), ambos utilizando BLF2c, observa-se uma diferença de resultado ainda maior do que a encontrada com BLFc. GRASPh produz soluções cujo valor médio é $13,8\%$ menor do que aquele obtido por BIP, enquanto que GRASPh_gl produz soluções cujo valor médio é $12,8\%$ menor.

- O algoritmo GRASPh_ew (distribuição de probabilidades mais concentrada nos resultados experimentais e EWMA após a busca local) novamente é aquele com melhor desempenho, produzindo soluções cujo valor médio é cerca de 16% menor do que aquele produzido por BIP. Comparando-se GRASPh e GRASPh_ew verifica-se que a inserção do EWMA acarreta redução de 2% no valor médio das soluções obtidas por GRASPh. Observa-se que, embora BLF2c não tenha tido desempenho tão bom quanto BLFc nos resultados das heurísticas de busca local, o valor médio das soluções adotando-se GRASPh_ew com BLF2c foi menor do que o obtido com BLFc.

A Tabela 4.19 apresenta os tempos médios de processamento em segundos obtidos pelos algoritmos GRASP descritos adotando-se BLF2c na fase de busca local. A estratégia

Nós	BIP	GRASPh	GRASPf	GRASPe	GRASPh_gl	GRASPf_gl	GRASPh_ew
10	0,014	3,63	3,64	3,67	3,81	3,83	3,75
25	0,033	30,42	30,21	30,78	31,89	31,47	30,5
50	0,086	144,81	143,66	145,9	149,24	147,68	144,99
75	0,161	345,81	344,28	349,6	355,1	351,65	346,58
100	0,29	632,47	629,51	638,99	647,59	643,97	634,77
200	1,78	2652,22	2644,02	2666,74	2693,61	2677,11	2690,39
média	0,312	516,23	514,23	520,01	526,48	522,94	521,33

Tabela 4.19: Tempos médios de processamento em segundos obtidos pelos algoritmos GRASP utilizando-se a busca local BLF2c.

de variação de β com o menor tempo médio de processamento é GRASPf, seguida por GRASPh, GRASPe, GRASPh_ew, GRASPf_gl, GRASPh_gl. A inserção do EWMA ao GRASPh com BLF2c acarretou aumento no tempo médio de processamento de cerca de 1%, entretanto o valor médio das soluções caiu de cerca de 2%, compensando o aumento no tempo médio de processamento. Ou seja, a utilização do GRASPh com a adoção do algoritmo EWMA após a busca local mostra-se benéfica tanto adotando-se a busca local BLFc quanto BLF2c, com um acréscimo inferior a 1% no tempo médio de processamento do algoritmo sem EWMA (GRASPh).

4.5.4 Comparação entre as variantes de GRASP com BLFc e BLF2c

A Tabela 4.20 a seguir traz um resumo dos melhores resultados médios encontrados para as diferentes estratégias de variação de β e de busca local utilizadas no algoritmo GRASP. Estes resultados foram apresentados nas Tabelas 4.16 a 4.19 (valores em negrito das Tabelas 4.16 e 4.18 e seus respectivos tempos de processamentos nas Tabelas 4.17 e 4.19).

Busca Local		GRASPh	GRASPh_ew
BLF2c	valor médio	10,16	**9,95**
	redução BIP	(-13,8%)	**(-15,6%)**
	tempo (s)	516,2	521,3
BLFc	valor médio	10,12	10,04
	redução BIP	(-14,2%)	(-14,9%)
	tempo (s)	954,4	963,6

Tabela 4.20: Comparação entre os resultados médios dos melhores algoritmos GRASP com BLF2c e com BLFc.

O algoritmo GRASPh com a adoção de BLF2c apresenta soluções cujo valor médio é maior do que aquele considerando-se BLFc. A diferença entre o valor médio das soluções é de apenas $0,4\%$, mas o tempo médio de processamento de GRASPh com BLF2c,

entretanto, é quase 46% menor que o do GRASPh com BLFc. Ou seja, é compensador adotar BLF2c na busca local do GRASP, pois o ganho no tempo médio de processamento poderia permitir realizar mais iterações e, eventualmente, encontrar soluções tão boas ou melhores do que as encontradas com BLFc.

O algoritmo GRASPh com a inserção de EWMA após a busca local BLF2c (GRASPh_ew) produziu soluções com valor médio menor do que sua versão sem EWMA (cerca de 15,6% menor do que o obtido pelo algoritmo BIP) e com tempo médio de processamento bastante inferior ao que seria obtido caso se adotasse apenas GRASPh com BLFc. Devido a esses resultados, o algoritmo EWMA será utilizado após a fase de busca local do algoritmo GRASP híbrido que será apresentado na Seção 4.5.5 a seguir.

4.5.5 GRASP híbrido

Os testes a seguir apresentarão os resultados provenientes da adoção de diferentes estratégias de variações de parâmetros para o GRASP híbrido apresentado na Seção 3.3, conforme descrito abaixo.

- A lista de soluções de elite (`Pool`) é um conjunto formado por soluções distintas e de alta qualidade que são utilizadas na reconexão por caminhos. O parâmetro `Tam_Pool` define o número máximo de soluções distintas na lista. Foram realizados testes utilizando-se os valores 10 e 12 para o parâmetro `Tam_Pool`.

- A lista de soluções de elite armazena apenas soluções diferentes. O parâmetro `DiffSize` refere-se ao número mínimo de elementos que devem diferir entre duas soluções existentes no conjunto de soluções elite. Ou seja, se duas soluções tiverem menos que `DiffSize` elementos diferentes entre elas, não são consideradas suficientemente diferentes para estarem simultaneamente no conjunto de soluções de elite. Foram realizados testes utilizando-se 4, 5 e 6 para o parâmetro `DiffSize`, para as redes com mais de dez nós. Para as redes de dez nós, consideraram-se dois valores para `DiffSize`: 2 e 3. A adoção dos valores 4, 5 ou 6 para `DiffSize` nas redes de dez nós, na maioria dos casos, resultava em conjuntos de elite compostos por soluções de baixa qualidade, indicando que, para redes pequenas, o parâmetro deveria assumir um menor valor.

- Foram adotadas as estratégias de reconexão para frente (estratégia F), quando a solução de partida é a de maior custo total dentre as duas, e reconexão para trás (estratégia B), quando a solução inicial é a de menor custo.

- O algoritmo de reconexão por caminhos foi utilizado de duas formas: como estratégia de intensificação, aplicado logo após a fase de busca local (utilizando uma solução escolhida aleatoriamente do conjunto de soluções elite e aquela obtida após a busca local), e como estratégia de pós-otimização, na qual as soluções de elite são combinadas umas com as outras utilizando-se a reconexão por caminhos entre cada par de soluções de elite.

  Caso opte-se em utilizar ambas as formas de reconexão, intensificação e pós-otimização, FF, FB, BB e BF referem-se, respectivamente, à intensificação e pós-otimização para frente, intensificação para frente e pós-otimização para trás, intensificação e pós-otimização para trás e intensificação para trás e pós-otimização para frente. Se apenas uma forma de reconexão for utilizada, B–, F–, –B e –F, referem-se, respectivamente, às estratégias: intensificação para trás, intensificação para frente, pós-otimização para trás e pós otimização para frente. Caso não se utilize a reconexão por caminhos no GRASP, a coluna correspondente ao algoritmo, na linha *reconexão* das tabelas, estará assinalada com —.

Foram aplicadas as seguintes estratégias de perturbação dos custos originais das arestas do grafo que representa a rede sem fio ad hoc:

- IDU: adotam-se as estratégias I, D e U na perturbação dos custos originais, a partir da quarta iteração. A perturbação IDU é aplicada ciclicamente, começando com o método de intensificação I, depois a diversificação D e, por último, o método uniforme U, repetindo-se esse ciclo até o final das iterações.

- ID: adotam-se as estratégias de intensificação I e diversificação D, a partir da quarta iteração, aplicadas ciclicamente, começando com o método I, depois uma iteração sem perturbação e, a seguir, uma perturbação D até o final das iterações.

- I: adota-se somente a estratégia de intensificação I, a partir da quarta iteração, aplicada ciclicamente de 3 em 3 iterações até o final das iterações.

- D: adota-se somente a estratégia de diversificação D, a partir da quarta iteração, aplicada ciclicamente de 3 em 3 iterações até o final das iterações.

A Tabela 4.21 mostra o impacto da adoção de diferentes estratégias de perturbação nos custos do grafo original no valor médio das soluções e no tempo médio de processamento das 850 instâncias teste, mantendo-se todos os demais parâmetros fixos, ou seja, adotando-se o mesmo algoritmo construtivo (rDSPF com mesma semente inicial), mesma busca

local (BLF2c com EWMA), mesmo número de iterações (50), mesmo tamanho da lista de soluções elite (`Tam_Pool` = 12), mesma diferença mínima (`DiffSize` = 5) entre duas soluções no `Pool` e mesma estratégia de reconexão por caminhos (BB).

Experimento	I	O	P	Q	R
Algoritmo construtivo	rDSPFh				
Busca local	BLF2c e EWMA				
Iterações	50				
`Tam_Pool`	12				
`DiffSize`	5				
Reconexão por caminhos	BB				
Perturbações	IDU	-	I	D	ID
valor médio	9,935	**9,894**	9,955	9,956	9,938
redução BIP	(-15,75%)	**(-16,10%)**	(-15,58%)	(-15,57%)	(-15,73%)
tempo (s)	300,9	532,0	293,7	285,0	291,1

Tabela 4.21: Avaliação das estratégias de perturbação aplicadas ao GRASP híbrido.

Conforme descrito acima, foram adotadas quatro estratégias diferentes de perturbação no GRASP híbrido. Comparando-se os resultados obtidos pelo GRASP híbrido com as perturbações I e D, observa-se que elas produzem resultados quase idênticos, embora o tempo médio de processamento no experimento Q tenha sido cerca de 9 segundos menor do que no experimento P. O uso das perturbações IDU e ID também produziu soluções com valor médio semelhante, ligeiramente menor do que o valor médio das soluções utilizando-se somente a perturbação I ou D.

Observou-se que o melhor resultado, em termos de valor médio das soluções, foi encontrado quando não utilizou-se perturbação alguma no GRASP híbrido (experimento O). Observa-se que seu tempo médio de processamento é 2% superior ao do algoritmo GRASPh_ew (conforme apresentado na Tabela 4.19). A diferença entre eles é apenas na aplicação da reconexão por caminhos como estratégia de intensificação e pós-otimização. A perturbação é uma estratégia que permite alterar significativamente os elementos de uma solução, permitindo, na maioria das vezes, chegar a ótimos locais mais rapidamente do que a estratégia sem perturbação, o que, conseqüentemente, reduz o tempo total de processamento do algoritmo. A diversidade das soluções obtidas com as perturbações é o principal motivo para os experimentos com perturbação apresentarem tempo médio de processamento bastante inferior ao do experimento sem perturbação. As soluções obtidas após a perturbação quando submetidas à busca local chegam mais rapidamente a um ótimo local do que aquelas que não sofreram nenhuma perturbação, porque a perturbação incorpora memória ao GRASP. Por outro lado, a qualidade das soluções obtidas utilizando a perturbação IDU, por exemplo, é um pouco pior (valor médio $0,4\%$ maior do que o do experimento O). Assim, conclui-se que a perturbação IDU encontra soluções cujo

valor médio é quase tão bom quanto o obtido sem perturbação, mas em um tempo médio 43% inferior, o que indica que a perturbação dos custos originais das arestas do grafo é benéfica principalmente quando deseja-se encontrar uma solução de boa qualidade em tempo reduzido. A redução do tempo médio de processamento pode então ser aproveitada para se realizar mais iterações com o algoritmo com perturbação, caso deseje-se alcançar melhores resultados no custo total da solução. Os resultados discriminados por tamanho da rede, que serão apresentados na Tabela 4.25, mostrarão que o algoritmo GRASP híbrido com perturbação produz soluções cujo valor médio é menor do que os obtidos sem perturbação para redes de até 25 nós.

Para avaliar o impacto das diferentes estratégias de reconexão por caminhos, fixaram-se todos os demais parâmetros no GRASP híbrido, variando-se somente as estratégias de reconexão por caminhos. A Tabela 4.22 a seguir traz os resultados obtidos, utilizando-se as 850 instâncias teste, adotando-se cada uma das estratégias de reconexão por caminhos.

Experimento	A	B	C	D	E	F	G	H	I
Alg. construtivo	rDSPFh								
Busca local	BLF2c e EWMA								
Perturbação	IDU								
Iterações	50								
`Tam_Pool`	12								
`DiffSize`	5								
Reconexão	—	F–	–F	FF	B–	–B	FB	BF	BB
valor médio	10,04	10,01	9,99	9,972	9,977	9,966	9,943	9,95	**9,935**
redução BIP	(-14,8%)	(-15%)	(-15,3%)	(-15,44%)	(-15,4%)	(-15,49%)	(-15,7%)	(-15,63%)	**(-15,75%)**
tempo (s)	294,7	338,3	338,7	347,7	291,4	293,4	346,7	346,7	300,9

Tabela 4.22: Avaliação das estratégias de reconexão por caminhos.

Em relação as estratégias de reconexão por caminhos adotadas no GRASP híbrido, conforme Tabela 4.22, pode-se chegar às seguintes conclusões:

- Caso opte-se pela adoção da reconexão por caminhos ao GRASP híbrido somente como estratégia de intensificação, aplicado logo após a fase de busca local (experimentos B e E), observa-se que a alternativa de reconexão para trás produz soluções cujo valor médio é, aproximadamente, $0,3\%$ menor do que o produzido adotando-se a reconexão para frente e com tempo médio de processamento cerca de 47 segundos inferior.

- A adoção da reconexão por caminhos somente como estratégia de pós otimização, adotando-se reconexão para trás (experimento F), produz soluções cujo valor médio é menor do que aquele obtido através do experimento E. Comparando-se os resultados dos experimentos F e C, que diferem somente na direção de reconexão, novamente a alternativa de reconexão para trás é melhor tanto em tempo médio de processamento quanto em valor médio das soluções.

- A combinação das estratégias de intensificação e pós-otimização que produz o melhor resultado é BB, ou seja, a adoção de ambas as estratégias de reconexão por caminhos com a alternativa de reconexão para trás. A diferença entre os resultados produzidos por este experimento e aqueles produzidos pelo GRASP sem reconexão por caminhos (experimento A) é de apenas 6 segundos a mais no tempo médio de processamento e com redução de aproximadamente 1% no valor médio das soluções.

A Tabela 4.23 traz o resultado obtido utilizando-se as 850 instâncias teste para algumas estratégias de busca local utilizadas no GRASP híbrido (apresentado na Seção 3.3), mantendo-se todos os demais parâmetros fixos. Comparando-se os resultados dos experimentos I e K, observa-se que a adoção de BLF2c com EWMA produz soluções cujo valor médio é semelhante ao de BLFc com EWMA, mas com tempo médio de processamento cerca de 44% menor que o de GRASP com BLFc e EWMA, corroborando os resultados apresentados na Seção 4.5.4. A inserção do EWMA ao algoritmo BLF2c no GRASP híbrido (que difere do GRASP simples por considerar perturbação e reconexão por caminhos) produz resultados ainda melhores do que os obtidos para o GRASP simples (conforme apresentado na Tabela 4.20), produzindo soluções cujo valor médio é quase 16% menor do que o produzido por BIP. Estes resultados comprovam o ganho no valor médio das soluções ao se adotar o algoritmo EWMA associado à busca local.

Experimento	I	J	K
Algoritmo construtivo	rDSPFh		
Perturbações	IDU		
Iterações	50		
`Tam_Pool`	12		
`DiffSize`	5		
Reconexão por caminhos	BB		
Busca local	BLF2c e EWMA	BLF2c	BLFc e EWMA
valor médio	**9,935**	10,248	9,942
redução BIP	**(-15,75%)**	(-13,10%)	(-15,69%)
tempo (s)	**300,9**	291,7	539,1

Tabela 4.23: Avaliação da busca local e EWMA aplicadas ao GRASP híbrido.

A Tabela 4.24 mostra o impacto da variação no tamanho do conjunto de soluções de elite (`Tam_Pool`) e da diferença mínima (`DiffSize`) entre soluções do conjunto de soluções de elite (`Pool`) no valor médio das soluções e no tempo médio de processamento das 850 instâncias teste, mantendo-se todos os demais parâmetros fixos. Mantendo-se fixo o tamanho do conjunto de soluções de elite (`Tam_Pool`) e variando-se apenas o parâmetro `DiffSize`, observa-se que `DiffSize` $= 5$ produz os melhores resultados. Os valores apresentados para `DiffSize` na Tabela 4.24 foram utilizados para as redes acima de dez nós. Para as redes de dez nós, utilizou-se 2 para este parâmetro, com exceção do experimento

N, que utilizou `DiffSize`= 3 para as redes de dez nós.

Experimento	I	L	M	N
Algoritmo construtivo	rDSPFh			
Busca local	BLF2c e EWMA			
Perturbações	IDU			
Iterações	50			
Reconexão por caminhos	BB			
`Tam_Pool`	12	10	10	12
`DiffSize`	5	4	5	6
valor médio	**9,935**	9,959	9,942	9,938
redução BIP	**(-15,75%)**	(-15,55%)	(-15,69%)	(-15,73%)
tempo (s)	300,9	296,7	299,0	305,3

Tabela 4.24: Avaliação do tamanho do `Pool` (`Tam_Pool`) e `DiffSize` no GRASP híbrido.

Em relação ao tempo médio de processamento, observa-se um pequeno aumento à medida que `DiffSize` aumenta. Comparando-se os experimentos M e I, que diferem apenas no tamanho do conjunto de soluções de elite, observa-se que a consideração do tamanho máximo igual a 12 produz o menor valor médio das soluções, com acréscimo de cerca de $1,9$ segundos em relação ao tempo médio de processamento do experimento M.

Esses resultados são esperados, já que quanto maior é o tamanho do conjunto de soluções elite, maior é a possibilidade de se encontrar combinações entre soluções que originem melhores soluções. Entretanto, deve-se observar o aumento no tempo médio de processamento antes de decidir qual tamanho máximo deve-se adotar. Foram escolhidos os valores 5 e 12 para os parâmetros `DiffSize` e `Tam_Pool`, respectivamente.

A Tabela 4.25 resume os melhores resultados dos testes, discriminados por tamanho de rede. Todos estes testes utilizaram a mesma heurística construtiva rDSPFh, busca local BLF2c com EWMA, estratégia de reconexão por caminhos B-B, `Tam_Pool`= 12 e fixando-se, arbitrariamente, o número de iterações igual a 50.

Os resultados encontrados para a rede de dez nós indicam que o uso da perturbação IDU e o valor 2 para o parâmetro `DiffSize` no GRASP híbrido é a estratégia que produz soluções com o menor valor médio, em um tempo médio de processamento de $5,3$ segundos. A adoção de `DiffSize`= 3 (experimento I) para a rede de dez nós acarreta em um acréscimo de aproximadamente $0,1\%$ no valor médio das soluções e $0,5$ segundos no tempo médio de processamento. A utilização dos valores 4, 5 ou 6 para `DiffSize` nas redes de dez nós, na maioria dos casos, resultava em conjuntos de elite compostos por soluções de baixa qualidade, indicando que, para redes pequenas, o parâmetro deveria assumir um menor valor.

	Experimentos	I	O	N	R
Nós	`DiffSize`	5**	5*	6*	5*
	Perturbação	IDU	—	IDU	ID
10	valor médio	10,20	10,25	**10,19**	10,19
	redução BIP	(-13,9%)	(-13,5%)	**(-14,0%)**	(-14,0%)
	tempo (s)	5,8	4,6	5,3	5,6
25	valor médio	**10,17**	10,26	10,18	10,21
	redução BIP	**(-18,6%)**	(-17,9%)	(-18,6%)	(-18,3%)
	tempo (s)	25,8	34,3	25,6	26,3
50	valor médio	9,80	**9,77**	9,81	9,80
	redução BIP	(-16,8%)	**(-17,0%)**	(-16,7%)	(-16,7%)
	tempo (s)	94,4	153,1	95,7	99,1
75	valor médio	9,77	**9,71**	9,78	9,77
	redução BIP	(-15,9%)	**(-16,5%)**	(-15,8%)	(-15,9%)
	tempo (s)	207,1	358,1	210,2	205,9
100	valor médio	9,85	**9,68**	9,84	9,84
	redução BIP	(-15,5%)	**(-17,0%)**	(-15,6%)	(-15,6%)
	tempo (s)	369,1	648,3	374,6	350,7
200	valor médio	9,78	**9,60**	9,77	9,75
	redução BIP	(-12,3%)	**(-13,8%)**	(-12,3%)	(-12,4%)
	tempo (s)	1504,1	2724,6	1527,8	1439,9
média	valor médio	9,94	**9,89**	9,94	9,94
	redução BIP	(-15,7%)	**(-16,1%)**	(-15,7%)	(-15,7%)
	tempo (s)	300,9	532,0	305,3	291,1

Tabela 4.25: Resumo com os melhores resultados dos testes com GRASP híbrido. *`DiffSize`= 2 para redes de 10 nós. **`DiffSize`= 3 para redes de 10 nós.

O algoritmo que produziu o melhor resultado para a rede de 25 nós foi o GRASP híbrido com `DiffSize`= 5 e perturbação IDU, produzindo soluções cujo valor médio é cerca de 19% menor do que o produzido por BIP e com tempo médio de processamento de aproximadamente 26 segundos. Observa-se que os experimentos sem perturbação (experimento O) ou apenas com a perturbação ID (experimento R) produziram soluções cujo valor médio é ligeiramente maior que o observado no experimento I, ilustrando a importância da perturbação IDU para redes com até 25 nós.

Para as redes a partir de 50 nós, o experimento O, sem perturbação dos custos originais das arestas do grafo que representa a rede, foi o que produziu os menores valores médios. Entretanto, o tempo médio de processamento do algoritmo sem perturbação, para as 850 instâncias, é aproximadamente 231 segundos maior que o tempo médio do experimento com perturbação IDU. Os experimentos I, N e R apresentaram resultados semelhantes em relação ao valor médio das soluções, sendo que o experimento R, com a perturbação ID, foi o que apresentou o menor tempo médio de processamento, aproximadamente 291 segundos. Em termos gerais, pode-se afirmar que o experimento I foi o que obteve o melhor desempenho em relação a tempo de processamento e qualidade das soluções.

A Tabela 4.26 mostra a evolução no valor médio das soluções e no tempo médio de processamento de alguns algoritmos implementados neste trabalho. HGR_IDU e HGR_s/pert indicam, respectivamente, o algoritmo GRASP híbrido com perturbação IDU (experimento I) e GRASP híbrido sem perturbação (experimento O).

Nós		BIP	rDSPF	BLF2c	GRASPh_ew	HGR_s/pert	HGR_IDU
10	valor médio	11,85	11,31	10,83	10,25	10,25	10,20
	redução BIP	-	(-4,5%)	(-8,6%)	(-13,5%)	(-13,5%)	(-13,9%)
	tempo (s)	0,014	0,007	0,06	3,8	4,6	5,8
25	valor médio	12,50	11,67	11,23	10,29	10,26	10,17
	redução BIP	-	(-6,6%)	(-10,1%)	(-17,7%)	(-17,9%)	(-18,6%)
	tempo (s)	0,033	0,014	0,584	30,5	34,3	25,8
50	valor médio	11,77	11,16	10,71	9,82	9,77	9,80
	redução BIP	-	(-5,2%)	(-9,0%)	(-16,6%)	(-17,0%)	(-16,8%)
	tempo (s)	0,086	0,028	2,8	144,8	153,1	94,4
75	valor médio	11,62	11,00	10,66	9,76	9,71	9,77
	redução BIP	-	(-5,3%)	(-8,3%)	(-16,0%)	(-16,5%)	(-15,9%)
	tempo (s)	0,161	0,051	6,8	346,6	358,1	207,1
100	valor médio	11,66	11,05	10,62	9,81	9,68	9,85
	redução BIP	-	(-5,2%)	(-8,9%)	(-15,8%)	(-17,0%)	(-15,5%)
	tempo (s)	0,289	0,083	12,6	634,8	648,3	369,1
200	valor médio	11,14	10,61	10,29	9,67	9,60	9,78
	redução BIP	-	(-4,8%)	(-7,7%)	(-13,2%)	(-13,8%)	(-12,3%)
	tempo (s)	1,8	0,408	53,1	2690,4	2724,6	1504,1
média	valor médio	11,79	11,20	10,75	9,95	9,89	9,94
	redução BIP	-	(-5,0%)	(-8,9%)	(-15,6%)	(-16,1%)	(-15,7%)
	tempo (s)	0,312	0,078	10,3	521,3	532,0	300,9

Tabela 4.26: Evolução dos resultados dos algoritmos implementados.

É bastante interessante observar a variação do valor médio das soluções à medida que os diferentes algoritmos foram implementados. Observa-se, por exemplo, que o valor médio das soluções obtidas pelo GRASP híbrido com perturbação IDU (HGR_IDU) é 11% menor do que o valor médio das soluções obtidas utilizando-se apenas rDSPF. Também verifica-se que a transição que acarreta a maior redução no valor médio das soluções é a do algoritmo de busca local BLF2c para o algoritmo GRASPh_ew. A redução no valor médio das soluções passa então a ser marginal comparando-se o GRASP híbrido com GRASPh_ew, sendo a variação no tempo médio de processamento então, mais significativa do que a variação no valor médio das soluções. Observa-se que o GRASP híbrido sem perturbação (HGR_s/pert) possui tempo médio de processamento semelhante ao do GRASP simples (GRASPh_ew), mas com redução de $0,5\%$ no valor médio das soluções. Por outro lado, o algoritmo HGR_IDU apresenta tempo médio de processamento 42% menor do que HGR_s/pert. A estratégia de perturbação dos custos originais das arestas do grafo mostrou-se mais efetiva nas redes de menor porte (até 25 nós), mas também é uma alternativa para as redes maiores por acrescentar diversificação das soluções, permitindo chegar a ótimos locais mais rapidamente, ou seja, acelerando o tempo de processamento

do algoritmo. Observa-se que, à medida que o tamanho da rede aumenta, maior é a redução do tempo de processamento de HGR_IDU em relação a HGR_s/pert.

Para ilustrar o comportamento das três melhores variantes do algoritmos GRASP (GRASPh_ew, HGR_IDU e HGR_s/pert), construiu-se um gráfico que mostra distribuições empíricas e teóricas da variável aleatória *tempo para valor alvo* para cada um dos algoritmos. Foram utilizadas para estes testes as mesmas instâncias escolhidas e apresentadas na Seção 4.5.2. Fixou-se um valor para a solução alvo de dificuldade média para cada instância testada, próximo do melhor obtido após 50 iterações do algoritmo GRASP com rDSPF na fase construtiva e BLF2c com EWMA na fase de busca local. Executou-se cada algoritmo 200 vezes independentes, armazenando-se o tempo quando uma solução com custo menor ao igual ao valor alvo fosse encontrada. As Figuras 4.15 e 4.16 trazem os resultados obtidos para as instâncias de 25 e 50 nós.

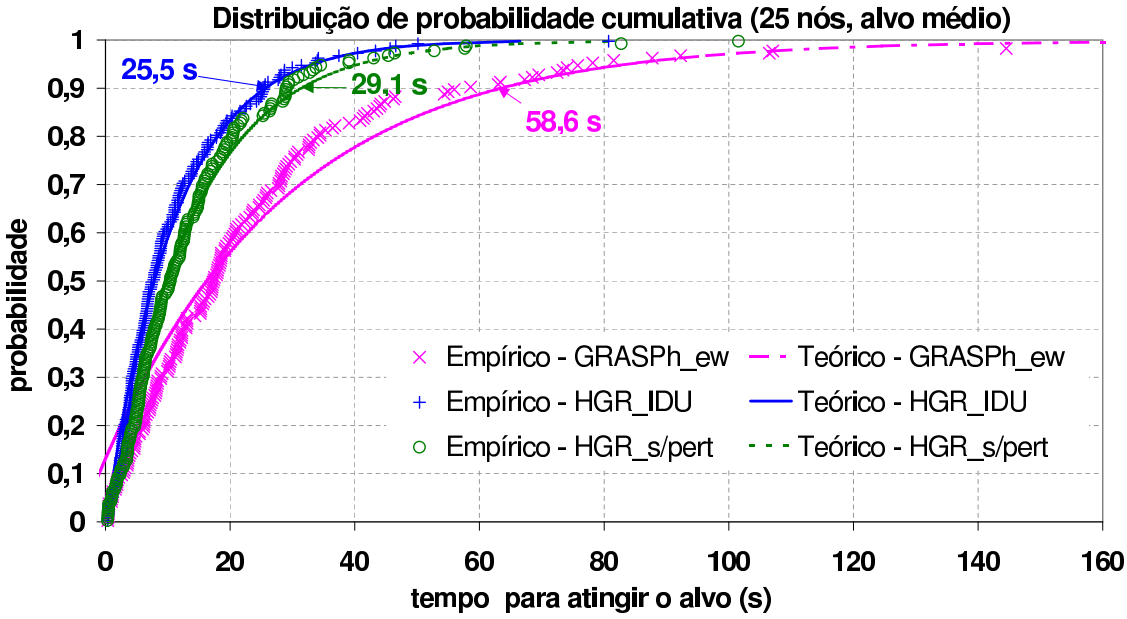

Figura 4.15: Distribuição de probabilidade do tempo para atingir o alvo de GRASPh_ew, HGR_IDU e HGR_s/pert para alvo médio (25 nós).

Tanto para o exemplo com 25 nós quanto para o de 50 nós, observa-se que o tempo para atingir o alvo do algoritmo GRASPh_ew é superior ao algoritmo GRASP híbrido, tanto para a estratégia sem perturbação (HGR_s/pert), quanto para a com perturbação (HGR_IDU). Para GRASPh_ew, instância de 25 nós, o tempo para atingir o alvo em 90% das 200 execuções é de aproximadamente 58 segundos, enquanto para HGR_IDU e HGR_s/pert é de aproximadamente 25 e 29 segundos, respectivamente. Para 50 nós, o tempo para atingir o alvo em 90% das 200 execuções de GRASPh_ew, HGR_IDU e HGR_s/pert são, respectivamente 144, 65 e 78 segundos. Estes resultados corroboram os já apresentados na Tabela 4.25, que indicavam que o algoritmo GRASP híbrido com

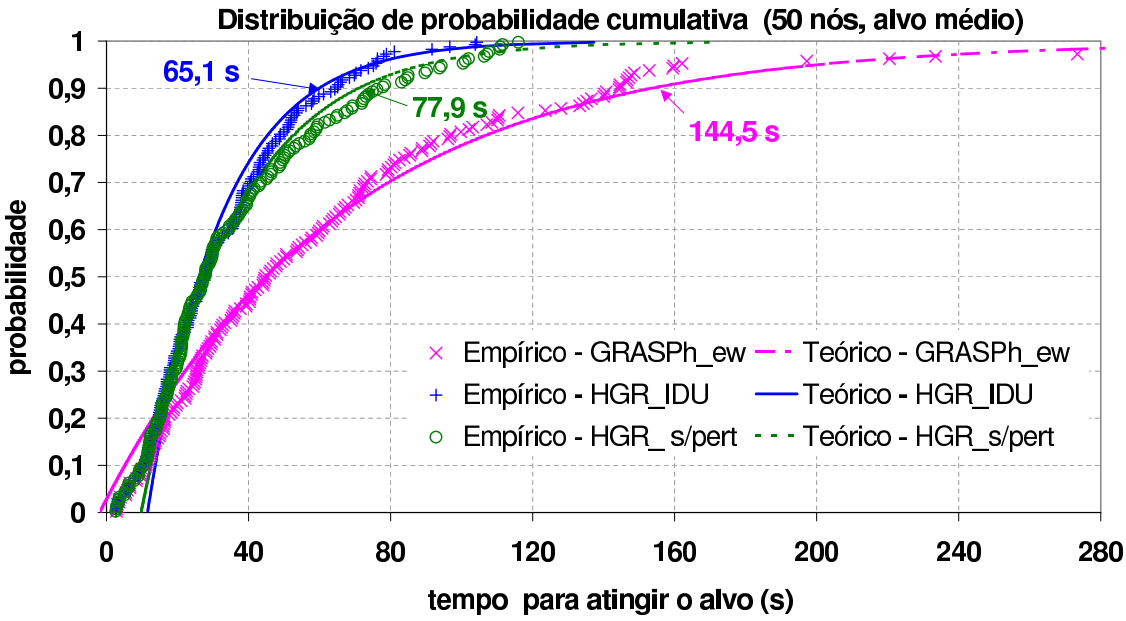

Figura 4.16: Distribuição de probabilidade do tempo para atingir o alvo de GRASPh_ew, HGR_IDU e HGR_s/pert para alvo médio (50 nós).

perturbação apresentava o menor tempo médio de processamento.

A Tabela 4.27 traz uma comparação entre os melhores resultados encontrados neste trabalho e o melhor publicado na literatura, considerando-se apenas o subconjunto de instâncias teste fornecido por Das [48]. Nesse trabalho não foram relatados os tempos de processamento do algoritmo *simulated annealing*.

Nós		HGR_s/pert	HGR_IDU	*simulated annealing* [48]
25	valor médio	10,23	10,12	9,95
	redução BIP	(-17,9%)	(-18,7%)	(-20,1%)
50	valor médio	9,75	9,76	9,65
	redução BIP	(-16,3%)	(-16,3%)	(-17,3%)
75	valor médio	**9,68**	**9,72**	9,74
	redução BIP	**(-16,7%)**	**(-16,4%)**	(-16,2%)
100	valor médio	**9,61**	**9,71**	9,82
	redução BIP	**(-17,1%)**	**(-16,2%)**	(-15,3%)

Tabela 4.27: Comparação com os resultados para as instâncias teste fornecidas por Das [48].

A comparação dos melhores resultados apresentados neste trabalho com o melhor apresentado na literatura até o momento (*simulated annealing* [48]) , considerando-se apenas o mesmo subconjunto de instâncias teste utilizado em [48], permite afirmar que o algoritmo GRASP híbrido (tanto para a estratégia sem perturbação quanto para a estratégia IDU) supera os melhores resultados apresentado até o momento para MPB em redes com mais de 50 nós.

Para as redes de 25 e 50 nós, utilizando-se 50 iterações de HGR_IDU, observa-se que o valor médio das soluções (considerando-se apenas as 100 instâncias fornecidas pelo autor - 50 de 25 nós e 50 de 50 nós) é aproximadamente 1% maior que o valor obtido com o *simulated annealing* [48]. Entretanto, quando aumenta-se o número de iterações de HGR_IDU de 50 para 200, os valores médios encontrados para as instâncias de 25 e 50 nós são semelhantes aos produzidos pelo *simulated annealing*.

Capítulo 5

Conclusões e trabalhos futuros

Redes de sensores sem fio são um tipo especial de rede ad hoc com sensoreamento distribuído, baixa capacidades de processamento e consumo reduzido de energia. Elas podem ser compostas por dezenas ou, até mesmo, centenas de pequenos dispositivos alimentados por baterias, chamados de nós sensores. Estas redes não necessitam de tipo algum de infraestrutura para estabelecer a comunicação entre seus componentes. Elas podem ser utilizadas no monitoramento de condições ambientais, rastreamento de animais, coordenação e processamento de informações e nos casos em que redes cabeadas não estão disponíveis, quando sua instalação não é econômica ou em situações em que é necessária uma infra-estrutura rápida e não existe um sistema base central para transmissão. Com o avanço das tecnologias capazes de suportar este tipo de serviço, o custo dos equipamentos vem caindo bastante ao longo do tempo, tornando o uso de sensores sem fio ainda mais atrativo.

O problema de transmissão multiponto com minimização de energia para redes de sensores consiste de uma rede ad hoc na qual deseja-se estabelecer a comunicação de um nó fonte para todos os outros nós da rede, direta ou indiretamente, de forma que a potência total de transmissão (assumindo que nenhuma potência é gasta para recepção/processamento do sinal) seja minimizada e que a conectividade seja mantida (ou seja, todos os nós sejam alcançados pela transmissão do nó fonte).

Este trabalho cita as principais aplicações das redes de sensores sem fio, introduz e avalia algoritmos eficientes para a construção de árvores de transmissão nessas redes. Como forma de avaliação da qualidade dos algoritmos, optou-se em observar seu desempenho sempre em relação ao algoritmo BIP [64], que é o algoritmo construtivo de referência

na literatura.

Dentre as heurísticas construtivas implementadas, destaca-se o algoritmo rDSPF, que leva em consideração a propriedade WMA [64] das redes sem fio ad hoc durante suas iterações. Este algoritmo possui uma característica diferente de outros algoritmos construtivos aleatorizados em relação ao parâmetro β. A adoção de β próximo à escolha gulosa, para redes com até 100 nós, não produz os menores valores médios das soluções, pois, para esta dimensão de rede, a diversidade de soluções obtidas é pequena, sendo insuficiente para que se encontre soluções com custo total menor do que o obtido adotando-se a escolha gulosa. Esta é a heurística construtiva com os melhores resultados, tanto em relação ao valor médio das soluções quanto em relação ao tempo médio de processamento. O valor médio das soluções obtidas por rDSPF, para as 850 instâncias teste, é 5% menor do que aquele obtido por BIP. Seu tempo médio de processamento é quase 75% menor que o de BIP.

Para os algoritmos de busca local, foram adotadas três vizinhanças nos procedimentos de melhoria iterativa e descida mais rápida. O algoritmo de busca local com o melhor desempenho, em termos de valor médio das soluções, foi aquele que conjugou o procedimento de melhoria iterativa com a vizinhança completa e com a adoção da estratégia de circularização (BLFc). O valor médio das soluções por ele obtidas é $9,5\%$ menor que o obtido por BIP. O algoritmo de busca local por melhoria iterativa com vizinhança de filhos mais custosos e circularização (BLF2c) obteve o melhor desempenho em termos de tempo médio de processamento e também foi considerado na fase de busca local do GRASP, por apresentar redução no tempo de processamento bastante significativa apesar do pequeno acréscimo no valor médio das soluções (inferior a 1%).

O algoritmo GRASP implementado com o melhor desempenho foi aquele que adotou, na fase construtiva, o algoritmo rDSPF e, na fase de busca local, o algoritmo BLF2c associado ao algoritmo de melhoria EWMA (GRASPh_ew). O valor médio das soluções encontradas por GRASPh_ew é $15,6\%$ menor do que o valor médio das soluções obtidas por BIP.

Finalmente, foi proposto um algoritmo GRASP híbrido, que combina as características dos algoritmos GRASP e de reconexão de caminhos. Várias estratégias de variação de parâmetros foram testadas. Os parâmetros que produziram as melhores soluções foram a consideração do tamanho máximo do conjunto de soluções de elite igual a 12, do número mínimo de elementos diferentes entre cada par de soluções do conjunto de elite igual a 5, usar a reconexão por caminhos tanto como estratégia de intensificação quanto de

pós otimização (adotanto, em ambas as estratégias, a reconexão para trás) e utilizar a estratégia de perturbação IDU. O valor médio das soluções encontradas por HGR_IDU é $15,7\%$ menor do que o valor médio das soluções obtidas por BIP.

A comparação dos melhores resultados apresentados neste trabalho com o melhor apresentado na literatura até o momento (*simulated annealing* [48]) permite afirmar que o algoritmo GRASP híbrido supera os melhores resultados apresentados até o momento para MPB em redes com mais de 50 nós. Para as redes com até 50 nós, os resultados, adotando-se 200 iterações, são tão bons quanto os existentes na literatura.

Como sugestão para trabalhos futuros, recomenda-se um estudo mais profundo dos efeitos do balanceamento do consumo de potência dos nós da rede de sensores sem fio, levando-se em conta, por exemplo, o tempo de transmissão dos nós e o tempo de vida das baterias que alimentam os nós. A avaliação do impacto da mobilidade dos nós também é outra questão que pode ser levada em consideração. A definição de um modelo que leve em conta perdas de potência, devido à recepção de sinal e processamento, ou que considere uma implementação localizada, onde não se tem conhecimento completo da localização de todos os nós da rede, também seria uma proposta de trabalhos futuros. Outra extensão deste trabalho consiste na utilização de outras metaheurísticas na resolução do problema e na proposta de novas estratégias para a redução do tempo médio de processamento do GRASP híbrido.

Referências

[1] Agre, J., e Clare, L. An integrated architecture for cooperative sensing networks. *IEEE Computer Magazine 33* (2000), 106–108.

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

[3] Aiex, R. M., Ribeiro, C. C., e Resende, M. G. C. Probabilitity distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics 8* (2002), 343–373.

[4] Akyildiz, I., Su, W., Sankarasubramaniam, Y., e Cayirci, E. Wireless sensor networks: A survey. *Computer Networks 38* (2002), 393–422.

[5] Althaus, E., Calinescu, G., Mandoiu, I. I., Prasad, S., Tchervenski, N., e Zelikovsky, A. Power efficient range assignment in ad-hoc wireless networks. Em *Proceedings of the IEEE Wireless Communications and Networking Conference* (New Orleans, 2003), vol. 4, pp. 1889–1894.

[6] Athanassopoulos, S., Caragiannis, I., Kaklamanis, C., e Kanellopoulos, P. Experimental comparison of algorithms for energy-efficient multicasting in ad hoc networks. Em *Proceedings of the 3rd International Conference for Ad Hoc Networks and Wireless* (Vancouver, 2004), I. Nikolaidis, M. Barbeau, e E. Kranakis, Eds., vol. 3158 de *Lecture Notes in Computer Science*, Springer, pp. 183–196.

[7] Banerjee, S., Misra, A., Yeo, J., e Agrawala, A. Energy-efficient broadcast and multicast trees for reliable wireless communication. Em *Proceedings of IEEE Wireless Communications and Networking Conference* (New Orleans, 2003), vol. 4, pp. 660–667.

[8] Bellman, R. E. On a routing problem. *Quart. Appl. Math 16* (1958), 87–90.

[9] Bertsekas, D., e Gallager, R. *Data Networks.* Prentice Hall, Englewood Cliffs, 1992.

[10] Bhardwaj, M., Garnett, T., e Chandrakasan, A. P. Upper bounds on the lifetime of sensor networks. Em *Proceedings of the IEEE International Conference on Communications* (Helsinki, 2001), vol. 3, pp. 785–790.

[11] Bonnet, P., Gehrke, J. E., e Seshadri, P. Querying the physical world. *IEEE Personal Communications 7* (2000), 10–15.

[12] Bulusu, N., Estrin, D., Girod, L., e Heidemann, J. Scalable coordination for wireless sensor networks: self-configuring localization systems. Em *Proceedings of the 6th International Symposium on Communication Theory and Applications* (Ambleside, 2001).

[13] Cagalj, M., Hubaux, J.-P., e Enz, C. Minimum-energy broadcast in all-wireless networks: NP-completeness and distribution issues. Em *Proceedings of the 8th annual International Conference on Mobile Computing and Networking* (Atlanta, 2002), ACM Press, pp. 172–182.

[14] Cagalj, M., Hubaux, J.-P., e Enz, C. Energy-efficient broadcasting in all-wireless networks. *Wireless Networks 11* (2005), 177–188.

[15] Canuto, S. A., Resende, M. G. C., e Ribeiro, C. C. Local search with perturbations for the prize collecting Steiner tree problem in graphs. *Networks 38* (2001), 50–58.

[16] Caragiannis, I., Kaklamanis, C., e Kanellopoulos, P. New results for energy-efficient broadcasting in wireless networks. Em *Proceedings of the 13th International Symposium on Algorithms and Computation* (Vancouver, 2002), P. Bose e P. Morin, Eds., vol. 2518 de *Lecture Notes in Computer Science*, Springer, pp. 332–343.

[17] Caragiannis, I., Kaklamanis, C., e Kanellopoulos, P. Energy-efficient wireless network design. Em *Proceedings of the 14th Annual International Symposium on Algorithms and Computation* (Kyoto, 2003), T. Ibaraki, N. Katoh, e H. Ono, Eds., vol. 2906 de *Lecture Notes in Computer Science*, Springer, pp. 585–594.

[18] Cartigny, J., Simplot, D., e Stojmenović, I. Localized minimum-energy broadcasting in ad-hoc networks. Em *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies* (San Francisco, 2003), vol. 3, pp. 2210–2217.

[19] Cerpa, A., Elson, J., Estrin, D., Girod, L., Hamilton, M., e Zhao, J. Habitat monitoring: Application driver for wireless communications technology. Em *Proceedings of the 2001 ACM SIGCOMM Workshop on Data Communications in Latin America and the Caribbean* (San Jose, 2001), pp. 20–41.

[20] Chambers, J., Cleveland, W., Kleiner, B., e Tukey, P. *Graphical Methods for Data Analysis*. Chapman & Hall, New York, 1983.

[21] Clementi, A. E., Huiban, G., e Penna, P. On the approximation ratio of the MST-based heuristic for the energy-efficient broadcast problem in static ad-hoc radio networks. Em *Proceedings of the International Parallel and Distributed Processing Symposium* (Nice, 2003), p. 222a.

[22] Clementi, A. E. F., Crescenzi, P., Penna, P., Rossi, G., e Vocca, P. On the complexity of computing minimum energy consumption broadcast subgraphs. Em *Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science* (Dresden, 2001), A. Ferreira e H. Reichel, Eds., vol. 2010 de *Lecture Notes in Computer Science*, Springer, pp. 121–131.

[23] Cormen, T. H., Leiserson, C. E., Rivest, R., e Stein, C. *Introduction to Algorithms*. The MIT Press, Cambridge, 2001.

[24] Das, A. K., II, R. J. M., El-Sharkawi, M., Arabshabi, P., e Gray, A. The minimum power broadcast problem in wireless networks: An ant colony system approach. Em *Proceedings of the IEEE CAS Workshop on Wireless Communications and Networking* (Pasadena, 2002).

[25] Das, A. K., II, R. J. M., El-Sharkawi, M., Arabshabi, P., e Gray, A. Minimum power broadcast trees for wireless networks: Optimizing using the viability lemma. Em *Proceedings of the IEEE International Symposium on Circuits and Systems* (Scottsdale, 2002), vol. 1, pp. 273–276.

[26] Das, A. K., II, R. J. M., El-Sharkawi, M., Arabshahi, P., e Gray, A. A cluster-merge algorithm for solving the minimum power broadcast problem in large scale wireless networks. Em *Proceedings of IEEE Military Communications Conference* (Boston, 2003), vol. 1, pp. 416–421.

[27] Das, A. K., II, R. J. M., El-Sharkawi, M., Arabshahi, P., e Gray, A. Minimum power broadcast trees for wireless networks: Integer programming formulations. Em *Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies* (San Francisco, 2003), vol. 2, pp. 1001–1010.

[28] Das, A. K., II, R. J. M., El-Sharkawi, M., Arabshahi, P., e Gray, A. r-shrink: A heuristic for improving minimum power broadcast trees in wireless networks. Em *Proceedings of the IEEE Global Telecommunications Conference* (San Francisco, 2003), vol. 1, pp. 523–527.

[29] Delicato, F. C. Middleware baseado em serviços para redes de sensores sem fio. Dissertação de Doutorado, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2005.

[30] Dijkstra, E. W. A note on two problems in conexion with graphs. *Numerische Math. 1* (1959), 269–271.

[31] Estrin, D., Govindan, R., e Heidemann, J. Embedding the internet. *Communications of the ACM 43* (2000), 39–41.

[32] Estrin, D., Govindan, R., Heidemann, J., e Kumar, S. Next century challenges: Scalable coordination in sensor networks. Em *Proceedings of the 5th annual International Conference on Mobile Computing and Networking* (Seattle, 1999), pp. 263–270.

[33] Feo, T. A., e Resende, M. G. C. Greedy randomized adaptive search procedures. *Journal of Global Optimization 6* (1995), 109–133.

[34] Garlan, D., Siewiorek, D., Smailagic, A., e Steenkiste, P. Project Aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing 21* (2002), 22–31.

[35] Glover, F. Tabu search and adaptive memory programing: Advances, applications and challenges. Em *Interfaces in Computer Science and Operations Research*, R. S. Barr, R. V. Helgason, e J. L. Kennington, Eds. Kluwer, 1996, pp. 1–75.

[36] Glover, F. Multi-start and strategic oscillation methods: Principles to exploit adaptive memory. Em *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, M. Laguna e J. L. Gonzáles-Velarde, Eds. Kluwer, 2000, pp. 1–24.

[37] Glover, F., e Laguna, M. *Tabu Search*. Kluwer, Norwell, 1997.

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

[39] Herring, C., e Kaplan, S. Component-based software systems for smart environments. *IEEE Personal Communications 7* (2000), 60–61.

[40] Kang, I., e Poovendran, R. A novel power-efficient broadcast routing algorithm exploiting broadcast efficiency. Em *Proceedings of IEEE 58th Vehicular Technology Conference* (Orlando, 2003), vol. 5, pp. 2926–2930.

[41] Kang, I., e Poovendran, R. COBRA: Center-oriented broadcast routing algorithms for wireless ad hoc networks. Em *Proceedings of IEEE Wireless Communications and Networking Conference* (Atlanta, 2004), vol. 5, pp. 813–818.

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

[43] Li, D., Jia, X., e Liu, H. Energy efficient broadcast routing in static ad hoc wireless networks. *IEEE Transactions on Mobile Computing 3* (2004), 144–151.

[44] Li, F., Mannor, S., e Lippman, A. Random tree optimization for energy-efficient broadcast in all-wireless networks. Em *Proceedings of 1st IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks* (Santa Clara, 2004).

[45] Liang, W. Constructing minimum-energy broadcast trees in wireless ad hoc networks. Em *Proceedings of the 3rd ACM international Symposium on Mobile ad hoc networking & computing* (Lausanne, 2002), ACM Press, pp. 112–122.

[46] Matsumoto, M., e Nishimura, T. Mersene Twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation 8* (1988), 3–30.

[47] Montemanni, R., e Gambardella, L. M. An exact algorithm for the min-power symmetric connectivity problem in wireless networks. Technical report, Istituto Dalle Molle di Studi sull Intelligenza Artificiale, Manno-Lugano, 2003.

[48] Montemanni, R., Gambardella, L. M., e Das, A. K. The minimum power broadcast problem in wireless networks: A simulated annealing approach. Em *Proceedings of the IEEE Wireless Communications and Networking Conference* (New Orleans, 2005), vol. 4, pp. 2057–2062.

[49] Noury, N., Hervé, V., Virone, G., e Mercier, E. Monitoring behavior in-home using a smart fall sensor and position sensors. Em *Proceedings of the IEEE-EMBS Special Topic Conference on Microtechnologies in Medicine and Biology* (Lyon, 2000), pp. 607–610.

[50] Pahlavan, K., e Levesque, A. *Wireless Information Networks.* Wiley-Interscience, New York, 1995.

[51] Park, J., e Sahni, S. Maximum lifetime broadcasting in wireless networks. *IEEE Transactions on Computers 54* (2005), 1081–1090.

[52] Pires, A. A. Controle de potência em redes ad hoc. Dissertação de Mestrado, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2005.

[53] Prais, M., e Ribeiro, C. C. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing 12* (1998), 164–176.

[54] Prim, R. C. Shortest connection networks and some generalizations. *Bell System Technical Journal 36* (1957), 1389–1401.

[55] Priyantha, N., Chakraborty, A., e Balakrishnan, H. The Cricket location-support system. Em *Proceedings of the 6th annual International Conference on Mobile Computing and Networking* (Boston, 2000), pp. 32–43.

[56] Rabaey, J., Ammer, J., da Silva Jr., J. L., e Patel, D. Pico-radio: Ad-hoc wireless networking of ubiquitous low energy sensor/monitor nodes. Em *Proceedings of the IEEE Computer Society Annual Workshop on VLSI* (Orlando, 2000), pp. 9–12.

[57] Resende, M. G. C., e Ribeiro, C. C. A GRASP with path relinking for private virtual circuit routing. *Networks 41* (2003), 104–114.

[58] Resende, M. G. C., e Ribeiro, C. C. Greedy randomized adaptive search procedures (GRASP). Em *Handbook of Metaheuristics*, F. Glover e G. Kochenberger, Eds. Kluwer, 2003, pp. 219–249.

[59] Ribeiro, C. C., Souza, R. C., e Vieira, C. E. C. A comparative computational study of random number generators. *Pacific Journal of Optimization 1* (2005), 565–578.

[60] Ribeiro, C. C., Uchoa, E., e Werneck, R. F. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing 14* (2002), 228–246.

[61] Stojmenović, I., Seddigh, M., e Zunic, J. D. Internal nodes based broadcasting in wireless networks. Em *Proceedings of the 34th Hawaii International Conference on System Sciences* (Maui, 2001), vol. 9, IEEE Computer Society, p. 9005.

[62] Wan, P.-J., Calinescu, G., Li, X., e Frieder, O. Minimum-energy broadcast routing in static ad hoc wireless networks. Em *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies* (Anchorage, 2001), vol. 2, pp. 1162–1171.

[63] Warneke, B., Last, M., Liebowitz, B., e Pister, K. S. J. Smart dust: Communicating with a cubic-millimeter computer. *IEEE Computer 34* (2001), 44–51.

[64] Wieselthier, J. E., Nguyen, G. D., e Ephremides, A. On the construction and energy-efficient broadcast and multicast trees in wireless networks. Em *Proceedings of the 19th Annual Joint Conference of the IEEE Computer and Communications Societies* (Tel-Aviv, 2000), vol. 2, pp. 585–594.

[65] Wu, J., e Dai, F. Broadcasting in ad hoc networks based on self-pruning. *International Journal of Foundations of Computer Science 14* (2003), 201–221.