Universidade Federal Fluminense

RENATO ELIAS NUNES DE MORAES

Otimização de Potência em Redes Ad Hoc Sem Fio sob Restrições de Conexidade

NITERÓI

2009

Universidade Federal Fluminense

RENATO ELIAS NUNES DE MORAES

Otimização de Potência em Redes Ad Hoc Sem Fio sob Restrições de Conexidade

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Doutor. Área de concentração: Otimização Combinatória e Inteligência Artificial.

Orientador:
Prof. Celso da Cruz Carneiro Ribeiro, Dr. Hab.

NITERÓI

2009

Otimização de Potência em Redes Ad Hoc
Sem Fio sob Restrições de Conexidade

Renato Elias Nunes de Moraes

Tese de Doutorado submetida ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do título de Doutor. Área de concentração: Otimização Combinatória e Inteligência Artificial.

Aprovada por:

Prof. Celso da Cruz Carneiro Ribeiro, D.Hab. / IC-UFF
(Orientador)

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

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

Profa. Arlindo Gomes de Alvarenga, D.Sc. / DI-UFES

Prof. Geraldo Robson Mateus, D.Sc. / DCC-UFMG

Prof. Reinaldo Vallejos Campos, D.Sc. / UTFSM-Chile

Niterói, 21 de Dezembro de 2009.

Catalogação na publicação elaborada pela Biblioteca Central / UVV-ES

M827o Moraes, Renato Elias Nunes de

Otimização de potência em redes Ad Hoc sem fio sobre restrições de conexidade / Renato Elias Nunes de Moraes. – 2009.
141 f. ; il.

Orientador: Prof. Celso da Cruz Carneiro Ribeiro, Dr. Hab.

Tese de Doutorado – Universidade Federal Fluminense.
Inclui bibliografias.

1. Redes de computadores. 2. Sistemas de comunicação sem fio. 3 Otimização matemática. 4. Programação linear. 5. Algoritmos. I. Moraes, Ribeiro, Celso da Cruz Carneiro. II. Universidade Federal Fluminense. III. Título.

CDD 004.6

"O churrasco é infinito."

Marcelo Nunes Biccas.

Agradecimentos

Agradeço à minha família: meus pais Maria José e Humberto, meu irmão Humberto Júnior, minha cunhada Flávia e meus sobrinhos Gabriel e Vinícius, fontes de carinho e motivação nesta jornada. Aos meus pais pela dedicação incondicional.

Agradeço também à minha família de Niterói: os meus tios, verdadeiros pais, Zelson e Maria Elvira, os meus primos/amigos/irmãos Juninho, Eduardo e Marcelo e os grandes amigos que aqui fiz.

Ao professor Celso, pela acolhida na UFF e pela orientação desde o início deste trabalho. Obrigado por acreditar em mim e por participar ativamente desta pesquisa. Mais que um orientador, um amigo e um pesquisador exemplar.

Aos colegas e amigos, em especial aos que acompanharam de perto as dificuldades deste tempo e muito me ajudaram: Renatha, Luciana, Klauko, Stênio, Rodrigo, Cristiano e Luciano.

Um especial agradecimento para Andrezza, minha companheira deste tempo, uma pessoa fantástica que me incentivou constantemente, compreendeu e batalhou junto em todos os momentos difícies que a dedicação aos meus estudos nos proporcionaram. Eterna e incomparável amiga.

Finalmente, meu sincero agradecimento aos professores e colegas da Universidade Federal Fluminense, componentes mais importantes deste agradável e produtivo ambiente. Obrigado aos Professores Mahey e Christophe que me acolheram e muito contribuiram para este trabalho.

Resumo

Uma rede *ad hoc* é uma coleção de dispositivos computacionais interligados por conexões sem fio estabelecidas pela potência de transmissão de seus rádios comunicadores. Redes sem fio defrontam-se com uma variedade de problemas que não ocorrem com redes ligadas através de cabos. Entre eles, destacam-se os problemas de conservação de energia. Um nó (dispositivo computacional e rádio comunicador) de uma rede sem fio é tipicamente mantido com energia de baterias que, em geral, possuem alto custo de manutenção ou que não podem ser recarregadas.

Uma técnica usada em redes ad hoc para melhor aproveitamento da energia disponível é o controle de topologia. Algoritmos de controle de topologia reduzem a potência de transmissão em cada nó, sem que haja quebra de conexidade no grafo de comunicação estabelecido. Entretanto, ao diminuir o número de ligações entre os nós, a rede resultante torna-se mais susceptível à perda de conexidade por falhas.

Nesta tese são apresentados modelos de programação linear inteira para quatro variantes do problema de minimização de potência em redes ad hoc garantindo que o grafo de comunicação seja k-conexo. Uma rede ad hoc é dita tolerante a falhas se $k > 1$. Para problemas com restrição de biconexidade ($k = 2$) são também desenvolvidas heurísticas baseadas em GRASP com reconexão por caminhos. Os algoritmos têm seu comportamento determinado e comparado por testes computacionais.

Palavras-chave: Redes ad hoc, conexidade, controle de topologia, otimização de potência, programação inteira, metaheurísticas, complexidade computacional, otimização matemática, otimização combinatória, programação linear, algoritmos.

Abstract

An *ad hoc network* consists of a collection of transceivers linked by wireless connections. The communication links are stablished by the transceivers transmission power. Wireless networks face a variety of problems that do not happen in wired networks. One important problem is reduce energy consumption by power transceivers minimization. Nodes (computer device and radio communicator) in a wireless network are typically battery-powered, and it is expensive and sometimes infeasible to recharge the device.

Topology Control is a technique used in wireless ad hoc networks to reduce energy consumption by power transceivers minimization. The topology control algorithm adjusts the transmission power of each node maintaining connectivity properties of the communication graph. However, reducing the number of links between nodes result in a network more susceptible to system faults and disconnection.

In this work, we develop mathematical models sufficiently general to encompass four problem variants of the k-connected minimum power consumption problem, which consists in finding a power assignment to the nodes of a wireless network so as that the resulting network topology be k-vertex connected and the total power consumption be minimum. An ad hoc network is said to be fault tolerant if $k > 1$. For the problem with biconnectivity constraints ($k = 2$) we also develop GRASP based heuristics with path relinking. Computational results illustrating the algorithms' behavior are reported and discussed.

Keywords: Ad hoc networks, connectivity, topology control, power optimization, integer programming, metaheuristics, computacional complexity, mathematical programming, linear programming, combinatorial optimization, algorithms.

Abreviações

CMPkC	:	problema de consumo mínimo de potência com restrição de k-conexidade
CMP2C	:	problema de consumo mínimo de potência com restrição de 2-conexidade
CMP2C-AB	:	problema de consumo mínimo de potência com restrição de 2-conexidade com entrada assimétrica e topologia bidirecional
GDTE	:	gráfico de distribuição de tempos de execução

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Avanços na tecnologia de comunicação sem fio provocaram um grande aumento no número e no tipo de aplicações de redes de computadores que se utilizam dessa tecnologia. As redes sem fio podem ser divididas em duas categorias: redes com infra-estrutura (como redes locais sem fio) e redes sem infra-estrutura (também conhecidas como ad hoc).

Enquanto que nas redes com infra-estrutura os dispositivos se comunicam através de estações base, nas redes ad hoc seus nós trocam informações usando seus próprios comunicadores. A comunicação direta evita a necessidade de uma infra-estrutura fixa. A troca de mensagens pode ser feita diretamente, de nó para nó ou através de um caminho com múltiplos nós.

As redes ad hoc encontram larga aplicação, por exemplo em comunicação em campos de batalha, em ambientes de catástrofes naturais e em redes de sensores, entre outras. O termo "ad hoc" significa o estabelecimento de uma rede para um motivo especial, temporário e com propriedades específicas para o objetivo em questão. Uma rede ad hoc tipicamente é colocada em funcionamento por um período limitado de tempo. Seus algoritmos e protocolos são desenvolvidos para a aplicação em particular, por exemplo, enviar um fluxo de mídia através de um campo de batalha ou estabelecer uma vídeo conferência entre participantes de uma operação de resgate.

Redes ad hoc podem ser representadas por um conjunto V de nós (dispositivos computacionais conectados a um rádio comunicador), numerados como $0, 1, \ldots, |V|-1$, juntamente com sua localização ou com as distâncias entre eles. Uma potência de transmissão p_u é associada a cada nó $u \in V$. Para cada par ordenado de nós (u, v), com $u, v \in V$, é conhecido um custo não negativo $e(u, v)$ tal que um sinal transmitido pelo nó u é recebido pelo nó v se e somente se a potência de transmissão de u é pelo menos igual a $e(u, v)$, ou seja, uma ligação é estabelecida do nó u para o nó v se $p_u \geq e(u, v)$. O conjunto de

ligações entre os nós estabelece um *grafo de comunicação*.

Redes ad hoc enfrentam uma variedade de restrições que não ocorrem em redes com infra-estrutura. Os nós de uma rede ad hoc são tipicamente alimentados por baterias que em geral possuem alto custo de manutenção ou que não podem ser recarregadas, tornando a conservação de energia um aspecto crítico do funcionamento dessas redes. Os problemas tratados nessa tese estão focados na economia de energia pela minimização do consumo de potência pelos rádios comunicadores, pois esses dispositivos tendem a ser a maior origem de dissipação de energia em redes sem fio [55].

Uma técnica usada em redes ad hoc para melhorar o aproveitamento da energia disponível é o *controle de topologia*. Ao invés de transmitir com potência máxima, os algoritmos de controle de topologia ajustam a potência de transmissão de cada nó, diminuindo a dissipação de energia e reduzindo o número de ligações entre os nós. Entretanto, a redução do número de ligações entre os nós aumenta a possibilidade de perda de conexidade por falhas. Devido ao ambiente crítico de utilização das redes ad hoc, muitas falhas podem ocorrer como resultado, por exemplo, de defeitos e falta de energia.

Um grafo de comunicação conexo, no qual existe pelo menos um caminho entre cada par de seus nós, é usualmente assumido como o requisito mínimo de conexidade por algoritmos sendo executados em diferentes camadas da rede como, por exemplo, protocolos de roteamento [32]. Porém, com um único caminho entre cada par de nós, a falha de apenas um nó (ou ligação) resultará em desconexidade. Portanto, topologias com múltiplos caminhos disjuntos entre qualquer par de nós podem ser requeridas [34].

O grafo de comunicação $G(p) = (V, E(p))$, onde $E(p) = \{(u, v) : u \in V, v \in V, p_u \geq e(u, v)\}$ é dito ser k-vértice-conexo se para quaisquer dois nós $u, v \in V$ existem k caminhos vértice-disjuntos conectando u a v. Dois caminhos são vértices-disjuntos se não possuem vértices em comum, exceto suas extremidades. Em outras palavras, o grafo $G(p)$ é k-vértice conexo se ele permanece conexo mesmo após a remoção de qualquer subconjunto formado por até $k - 1$ de seus vértices. Como um grafo k-vértice conexo também é k-aresta conexo, mas o contrário não é necessariamente verdadeiro, diz-se que um grafo é k-conexo quando ele é k-vértice conexo.

Dados o conjunto de nós V, os custos não negativos dos arcos $e(u, v)$ para qualquer $u, v \in V$, e o parâmetro $k \geq 1$, o *problema de consumo mínimo de potência com restrição de k-conexidade* (CMPkC) consiste em encontrar uma atribuição ótima de potências de transmissão $p_u : V \rightarrow R+$ para todo nó $u \in V$, tal que o consumo de potência total $\sum_{u \in V} p_u$ seja minimizado e o grafo de comunicação resultante $G(p) = (V, E(p))$ seja k-

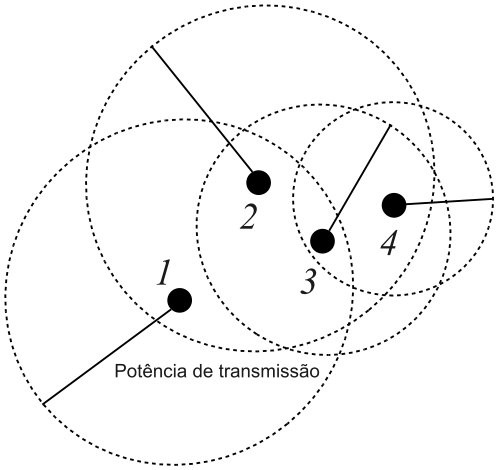


Figura 1.1: Exemplo de uma rede ad hoc em duas dimensões com estações equipadas com uma antena *omnidirectional*.

conexo. Esse problema foi provado ser NP-árduo para $k = 1$ em [9] e para $k = 2$ em [6]. Como o *problema de subgrafo gerador k-conexo de custo mínimo* é NP-árduo mesmo para $k = 2$ [17], o problema de consumo mínimo de potência com restrição de k-conexidade é conjecturado ser NP-árduo [25] para qualquer inteiro positivo k.

1.1 Formulação do Problema

As redes ad hoc sem fio consideradas neste trabalho são constituídas de nós (ou estações) equipadas com um dispositivo computacional, um rádio comunicador, uma fonte de energia suficiente para permitir ao rádio comunicador transmitir com potência máxima e uma antena omnidirecional responsável por enviar e receber sinais. A comunicação é estabelecida pela potência de transmissão em cada estação. A Figura 1.1 mostra um exemplo de uma rede ad hoc onde uma estação pode se comunicar diretamente com todas as demais que estão dentro do seu alcance de transmissão.

Cada nó de uma rede ad hoc pode ajustar sua potência de transmissão baseado na sua distância aos nós receptores e nos níveis de ruído e interferência do ambiente. No modelo de atenuação de potência mais comumente utilizado [43], o sinal de potência diminui com $1/d^{\varepsilon}$, onde d é a distância do transmissor para o receptor e ε é o expoente de perda no caminho, com valores entre 2 e 4. Com esse modelo, a potência requerida pelo nó u para estabelecer uma transmissão através de uma ligação de u para v é dada por

$$p_u \geq d_{uv}^{\varepsilon}.q_v, \tag{1.1}$$

onde d_{uv} é a distância euclidiana entre o transmissor u e o receptor v, e q_v é o limiar de potência do receptor para detecção do sinal, o qual é usualmente normalizado para 1.

A primeira variante do problema de consumo mínimo de potência k-conexo é definida para *entradas simétricas*. Nesse caso, assume-se que a potência necessária (também referenciada como custo do arco (u, v)) para estabelecer uma transmissão entre os nós u e v separados pela distância d_{uv} é $e(u, v) = e(v, u) = d_{uv}^{\varepsilon}$. Apesar da variante simétrica ser aceita como razoável, a relação (1.1) pode ser utilizada somente para ambientes abertos, sem obstáculos entre os comunicadores/receptores. Ela não considera a possível ocorrência de reflexões, espalhamento e difração causados por construções e pela geografia do terreno.

Na prática, a potência necessária para dois nós u e v se comunicarem pode ser assimétrica por várias razões. Por exemplo, custos assimétricos dos arcos podem ser usados para modelar baterias com diferentes níveis de energia [4] ou nós heterogêneos [46]. Além disso, os níveis de ruído do ambiente das regiões onde se localizam os diferentes nós podem ser diferentes [28]. Assim, também é tratada nessa tese a variante mais geral com *entradas assimétricas* do CMPkC. Nesse caso, podem haver pares de nós $u, v \in V$ tais que $e(u, v) \neq e(v, u)$.

A comunicação do nó u para o nó v estará habilitada sempre que $p_u \geq e(u, v)$. Dessa forma, o grafo de comunicação associado a uma atribuição de potência p_u para cada nó $u \in V$ é definido como o grafo direcionado $G(p) = (V, E(p))$, onde $E(p) = \{(u, v) : u \in V, v \in V, p_u \geq e(u, v)\}$. Duas diferentes topologias do grafo podem ser usadas para fazer cumprir a restrição de k-conexidade.

Em uma *topologia unidirecional*, todos os arcos (ver Figura 1.2) estabelecidos pela atribuição de potência no grafo de comunicação $G(p) = (V, E(p))$ são considerados para satisfazer a restrição de k-conexidade.

Apesar da implementação de ligações sem fio unidirecionais ser tecnicamente possível [48] e a imposição de requisitos de bidirecionalidade incorrer em considerável custo adicional, a vantagem da utilização de ligações unidirecionais é questionável. A grande probabilidade de erros de transmissão e de perda de pacotes em redes reais gera a necessidade de retransmissões e de mensagens de reconhecimento [9]. Além disso, os atrasos provocados pela manipulação de ligações unidirecionais pelos algoritmos de roteamento superam os benefícios que essas ligações podem fornecer e melhor desempenho pode ser alcançado simplesmente evitando-as [35]. Portanto, para aumentar o desempenho da rede, ligações bidirecionais são implicitamente assumidas por vários protocolos de rotea-

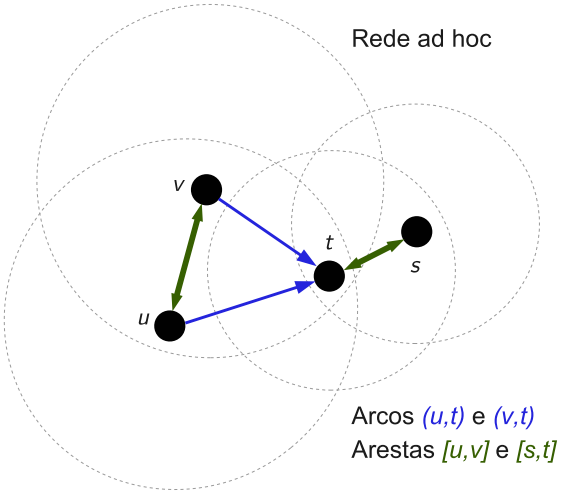


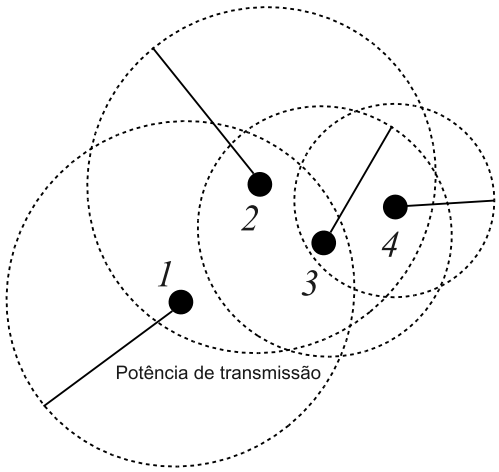
Figura 1.2: Diferença entre arcos unidirecionais e arestas bidirecionais.

mento [26].

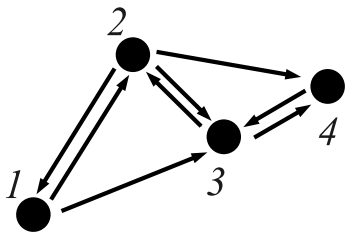
Em uma *topologia bidirecional*, a aresta bidirecional (ver Figura 1.2) $[u, v]$ é utilizada como uma ligação de comunicação para estabelecer a restrição de k-conexidade se v está dentro do alcance de transmissão de u e se u também está dentro do alcance de transmissão de v. Nesse caso, o conjunto de arestas consideradas para fazer cumprir a restrição de k-conexidade no grafo de comunicação $G(p) = (V, E(p))$ está restrito a $B(p) = \{(u, v) : u \in V, v \in V, p_u \geq e(u, v), p_v \geq e(v, u)\} \subseteq E(p)$. As Figuras 1.3(b) e 1.3(c) mostram, respectivamente, as topologias unidirecional e bidirecional estabelecidas pelas potências atribuídas à rede ad hoc da Figura 1.3(a). Tanto o problema de consumo mínimo de potência com restrição de k-conexidade com topologia unidirecional [6, 9, 25] como com topologia bidirecional [3, 5, 6, 10] são NP-árduos.

Como o principal objetivo do controle de topologia é alcançado pela definição da potência de transmissão de cada nó, ele deve se situar acima da camada MAC. Além disso, o controle de topologia fornece a topologia requerida pelos algoritmos de roteamento, devendo assim ficar abaixo da camada de rede. Essa configuração pode ser vista na Figura 1.4. São os protocolos da camada MAC (*Medium Access Control*) os responsáveis por evitar colisões que resultam em retransmissões e que consomem energia e degradam o desempenho. Já a camada de rede é responsável por encontrar e manter os caminhos virtuais entre os pares origem e destino da rede e encaminhar os pacotes até seu destino pelos nós intermediários.

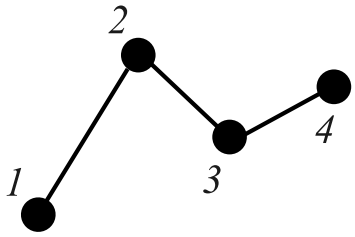
Outro benefício da diminuição da potência de transmissão dos nós está no aumento do desempenho da rede pela redução de interferência, colisões e ruídos provocados pela



(a) Rede ad hoc



(b) Topologia unidirecional



(c) Topologia bidirecional

Figura 1.3: Exemplo de uma rede ad hoc em duas dimensões com estações equipadas com uma antena *omnidirectional*.

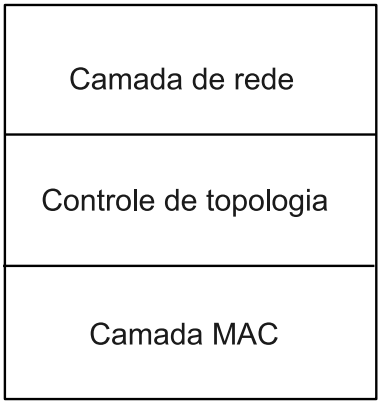


Figura 1.4: Posicionamento do controle de topologia na pilha de protocolos de rede.

superposição de várias ondas de rádio compartilhando o mesmo meio [49].

1.2 Estado da Arte

São consideradas neste trabalho quatro variantes do problema de consumo mínimo de potência com restrição de k-conexidade:

- Entrada simétrica com topologia unidirecional,
- Entrada simétrica com topologia bidirecional,
- Entrada assimétrica com topologia unidirecional, e
- Entrada assimétrica com topologia bidirecional.

O estado da arte é apresentado em função dessa abordagem, identificando os trabalhos desenvolvidos para cada variante. Os algoritmos discutidos são soluções centralizadas principalmente desenvolvidas para redes ad hoc em seu momento estacionário, por exemplo, após um reposicionamento. Algoritmos distribuídos podem ser encontrados em [2, 29, 30, 45, 47, 53]. Apesar dos algoritmos não centralizados terem a vantagem de trabalhar com informação localizada, o consumo de potência das soluções encontradas pode ser arbitrariamente maior que o das soluções ótimas [20]. No entanto, as redes seguem um princípio básico dos elementos serem autônomos.

1.2.1 Entrada Simétrica com Topologia Unidirecional

A variante simétrica do problema de minimização do consumo de potência para a construção de um grafo de comunicação unidirecional satisfazendo a restrição de 1-conexidade (topologia unidirecional e $k = 1$) foi provada ser NP-árdua por Chen e Huang [9], que também apresentaram um algoritmo 2-aproximativo baseado na árvore geradora mínima para sua solução. Kirousis et al. [27] provaram que o problema é NP-árduo no espaço euclideano tridimensional e descreveram o mesmo algoritmo 2-aproximativo baseado na árvore geradora mínima apresentado em [9]. Também foi apresentado em [27] um algoritmo de programação dinâmica com complexidade $O(n^4)$ para o caso particular em que os nós são co-lineares. Clementi et al. [11] provaram que o mesmo problema é NP-árduo no espaço euclideano bidimensional.

A variante simétrica do problema de consumo mínimo de potência sob restrição de biconexidade (ou 2-conexidade) foi discutida em [6], onde o problema foi demonstrado ser NP-árduo e apresenta-se um algoritmo 4-aproximativo para sua solução.

Para a variante simétrica do problema de consumo mínimo de potência com restrição de k-conexidade e topologia unidirecional, foi apresentado em [50] um método de solução exato para nós uniformemente distantes sobre uma reta, um algoritmo com fator constante de aproximação para o caso mais geral de redes ad hoc lineares e um algoritmo $O(k^2)$-aproximativo para o caso planar. Carmi et al. [8] desenvolveram um algoritmo $O(k)$-aproximativo de tempo polinomial baseado em árvores geradoras mínimas para a mesma versão do problema de consumo mínimo de potência com restrição de k-conexidade no espaço bidimensional.

1.2.2 Entrada Simétrica com Topologia Bidirecional

Os primeiros resultados para entradas simétricas com topologia bidirecional foram alcançados em [42] para a minimização da potência máxima de transmissão da rede considerando restrições de 1-conexidade e biconexidade sobre o grafo de comunicação. Para cada versão do problema foram descritos algoritmos centralizados polinomiais e heurísticas para implementação distribuída. Lloyd et al. [31] generalizaram os resultados de [42] pela demonstração de que o problema de minimização da potência máxima de transmissão pode ser resolvido em tempo polinomial para restrições verificáveis também em tempo polinomial.

O problema de consumo mínimo de potência com topologia bidirecional e restrição de 1-conexidade para grafos de entrada simétricos foi proposto em [3, 5], onde foi provado que sua versão de decisão é um problema NP-completo. Blough et al. [3] determinaram limites assintóticos para o custo da solução em instâncias randômicas. Cheng et al. [10] mostraram a importância do problema no caso de redes de sensores, provaram sua NP-completude e propuseram duas heurísticas.

O algoritmo 2-aproximativo apresentado em [27] soluciona a versão simétrica do problema de consumo mínimo de potência com topologia bidirecional e restrição de 1-conexidade. Calinescu et al. [5] diminuíram a taxa de aproximação, através da exploração das similaridades com o problema da árvore mínima de Steiner, apresentando um algorimo com taxa de 15/8 de aproximação. Calinescu et al. [5] também apresentaram um esquema de aproximação em tempo polinomial total de $7/4 + \epsilon$. Esses fatores de aproximação foram melhorados por Althaus et al. [1] para 11/6 e $5/3 + \epsilon$, respectivamente.

Eles também desenvolveram um algoritmo exato do tipo *branch-and-cut* baseado em uma nova formulação por programação inteira. Outro algoritmo exato foi apresentado em [36]. Das et al. [12] desenvolveram um modelo em programação inteira mista para o problema com antenas setoriais, apresentaram uma heurística centralizada baseada no algoritmo de Kruskal para o problema de árvore geradora mínima e discutiram uma heurística simples de inserção e remoção de arestas para melhorar a topologia gerada pela heurística baseada no algoritmo de Kruskal.

Lloyd et al. [31] estudaram o problema de consumo mínimo de potência com topologia bidirecional e restrição de biconexidade. Eles apresentaram um algoritmo com taxa de aproximação $2(2 - 2/n)(2 + 1/n)$. Calinescu e Wan [6] provaram que o problema é NP-árduo e desenvolveram um algoritmo 4-aproximativo.

Para o caso de valores quaisquer de k, algoritmos com fator de aproximação $O(k)$ para o problema de consumo mínimo de potência com topologia bidirecional k-conexa foram apresentados em [20]. Esse fator de aproximação foi reduzido para $O(\log^4 n)$ em [21]. Jia et al. [25] apresentaram, entre outros resultados, um algoritmo $3k$-aproximativo para $k \geq 3$ e um algoritmo 6-aproximativo para $k = 3$. Das e Mesbahi [13] propuseram um procedimento heurístico aplicando uma visão algébrica da conexidade definida pela matriz de Laplace do grafo. A matriz de Laplace de um grafo é uma matriz de representação construída como a diferença da matriz de graus pela matriz de adjacência desse grafo.

1.2.3 Entrada Assimétrica com Topologia Unidirecional

Enquanto que a versão simétrica do problema de consumo mínimo de potência com restrições de conexidade recebeu significante atenção, somente poucos algoritmos aproximativos foram propostos para o caso de requisitos de potência assimétricos, como descrito a seguir.

Krumke et al. [28] consideraram a versão assimétrica do problema de minimização do consumo de potência com restrição de 1-conexidade e topologia unidirecional. Eles mostraram que um algoritmo $\Omega(\log n)$-aproximativo só pode existir se $P = NP$ e apresentaram um algoritmo $O(\log n)$-aproximativo. Independentemente, Calinescu et al. [4] alcançaram um limite aproximativo similar utilizando um algoritmo que constrói uma árvore incrementalmente. Caragiannis et al. [7] também obtiveram um algoritmo $O(\log n)$–aproximativo.

Wang et al. [52] apresentaram um algoritmo aproximativo para o problema assimétrico de minimização do consumo de potência com restrição de k-conexidade e topologia

unidirecional. O algoritmo possui fator de aproximação $O(k + \Delta^-)$, onde Δ^- é o grau de saída máximo dos nós de um subgrafo direcionado com caminhos k-vértice disjuntos entre um determinado nó raiz r qualquer e todos os demais nós.

1.2.4 Entrada Assimétrica com Topologia Bidirecional

Althaus et al. [1] provaram, para a variante assimétrica do problema de minimização do consumo de potência com restrição de 1-conexidade e topologia bidirecional, que um algoritmo $\Omega(\log n)$-aproximativo só pode existir se $P = NP$. Caragiannis et al. [7] desenvolveram um algoritmo $O(1.35 \ln n)$-aproximativo para o mesmo problema. Um algoritmo $O(\ln n)$-aproximativo foi independentemente obtido em [4] usando diferentes técnicas.

1.3 Objetivos da Tese

Na literatura, as principais técnicas adotadas para a resolução dos problemas de otimização de potência sob restrições de conexidade em redes ad hoc são construídas utilizando-se abordagens por algoritmos de aproximação [1, 4, 5, 6, 7, 8, 9, 20, 21, 25, 27, 28, 31, 50, 52]. No caso de problemas de otimização de potência sob restrições de 1-conexidade também são propostos algoritmos exatos [1, 12, 36].

Nessa tese, serão apresentadas formulações exatas para a solução de problemas de minimização de potência sob restrições de k-conexidade. Como os métodos exatos nem sempre conseguem obter soluções ótimas em tempo hábil para problemas reais ou de grande porte, também estudam-se nessa tese abordagens heurísticas para a solução de problemas de minimização de potência sob restrições de biconexidade ($k = 2$). O termo heurísticas se refere, de uma forma geral, a algoritmos construtivos, de busca local ou baseados em metaheurísticas.

O Capítulo 2 apresentará três formulações de programação linear inteira para solucionar as quatro variantes do CMPkC. Os três modelos propostos foram solucionados por um resolvedor de problemas de programação linear inteira e suas soluções comparadas. O caso mais interessante na prática correspondente a $k = 2$ será investigado mais detalhadamente neste e nos próximos capítulos.

O Capítulo 3 descreverá algoritmos gulosos para a variante assimétrica do problema de minimização do consumo de potência sob restrições de biconexidade e topologia bidirecional. Os algoritmos propostos estão baseados em algoritmos de construção de grafos

conexos e no algoritmo de Tarjan [51], que identifica as componentes biconexas de um grafo não direcionado. Também serão apresentados resultados computacionais comparando os algoritmos gulosos propostos e um algoritmo da literatura. Ao final do capítulo, serão desenvolvidos algoritmos gulosos randomizados.

No Capítulo 4 serão desenvolvidos algoritmos de busca local que, combinados com os algoritmos gulosos randomizados do capítulo anterior, serão utilizados na implementação de heurísticas do tipo GRASP (*Greedy Randomized Adaptive Search Procedure*) para o problema de minimização do consumo de potência sob restrições de biconexidade e topologia bidirecional. Serão incorporadas ao GRASP estratégias do tipo VND e de reconexão por caminhos. Resultados experimentais são apresentados ilustrando o ajuste dos parâmetros das heurísticas, o comportamento de cada algoritmo e a qualidade das soluções fornecidas por cada um.

Por último, o Capítulo 5 mostrará as conclusões da tese e possíveis extensões como trabalhos futuros.

Capítulo 2

Modelos de Programação Inteira Mista

A resolução exata do problema de consumo mínimo de potência sob restrição de k-conexidade parece ter sido abordada anteriormente na literatura apenas para o caso particular onde os nós estão dispostos uniformemente distantes sobre uma reta [50]. A maioria dos trabalhos que apresentam soluções exatas resolvem apenas o problema de consumo mínimo de potência sob restrição de 1-conexidade.

Althaus et al. [1] apresentaram um algoritmo exato do tipo *branch and cut* que soluciona instâncias de até 40 nós em uma hora do problema de consumo mínimo de potência sob restrição de 1-conexidade. Em [36] foram desenvolvidos algoritmos exatos que superam os resultados de [1] para instâncias similares. Das et al. [12] desenvolveram um modelo de programação inteira mista para o problema de consumo mínimo de potência sob restrição de 1-conexidade com até 50 nós, cada um com três antenas setoriais. Os resultados ótimos são usados na avaliação de heurísticas.

Nesse capítulo, três modelos de programação inteira mista baseados em fluxos de multicomodidades são apresentados para solucionar as quatro variantes, como definidas na Seção 1.2, do problema de consumo mínimo de potência sob restrição de k-conexidade para qualquer valor de $k \geq 1$.

2.1 Modelo com Potências Contínuas

Uma maneira de formular o problema de consumo mínimo de potência com restrição de k-conexidade, consiste em definir k comodidades com uma unidade de demanda cada uma, que devem ser enviadas de cada um dos $|V|$ nós para todos os demais $|V| - 1$ nós. Essa formulação envolve portanto, um grande número $k|V|(|V| - 1)$ de comodidades. Entretanto, Raghavan [41] demonstrou, no contexto do problema de projeto de redes com

requisitos de conexidade [33], que um modelo mais compacto pode ser formulado usando um grafo não direcionado k-conexo de requisitos $G^k = (V, E^k)$ com um número mínimo $|E^k| = \lceil k|V|/2 \rceil$, $k > 1$, de arestas [24], construído como se segue:

- Se k é par, acrescenta-se a aresta $[i, j]$ em E^k para todo $i, j \in V$ sempre que $(i - j) \bmod |V| \leq k/2$.
- Se k é ímpar e $|V|$ é par, deve-se construir o grafo G^{k-1} e, em seguida, obter E^k de E^{k-1} através da adição ao último das arestas $[i, i + |V|/2]$ para $i = 0, \ldots, |V|/2$.
- Senão, deve-se construir o grafo G^{k-1} e, em seguida, obter E^k de E^{k-1} através da adição ao último das arestas $[0, (|V| - 1)/2)]$, $[0, (|V| + 1)/2]$ e $[i, i + (|V| + 1)/2]$, para $i = 1, \ldots, (|V| - 1)/2$.

O conjunto C de comodidades é construído da seguinte maneira. Seja $[i, j]$ uma aresta de E^k. Se o problema exige uma topologia unidirecional, cria-se k comodidades do nó i para j e outras k comodidades do nó j para i, ambas com demanda de uma unidade. Caso contrário, cria-se k comodidades apenas do nó i para j com demanda de uma unidade, escolhendo-se arbitrariamente um dos nós como origem e o outro como destino. Esse procedimento gera um modelo de fluxo multicomodidades para o problema de consumo mínimo de potência com restrição de k-conexidade unidirecional usando somente $2 \cdot \lceil k|V|/2 \rceil$ comodidades, que é um número menor que $k|V|(|V| - 1)$. A formulação para topologia bidirecional utiliza a metade do número de comodidades da formulação para topologia unidirecional.

Para cada comodidade $c \in C$, representa-se por $o(c)$ sua origem e $d(c)$ seu destino. Para qualquer nó $i \in V$ e qualquer comodidade $c \in C$, define-se $D_c(i) = -k$ se $i = o(c)$, $D_c(i) = +k$ se $i = d(c)$, $D_c(i) = 0$ caso contrário. A variável discreta f_{ij}^c e a variável contínua p_i representam, respectivamente, o fluxo da comodidade c através do arco (i, j) e a potência atribuída ao nó i. A variável binária f_{ij}^c é igual a um se o arco (i, j) é usado pela comodidade c para comunicação do nó i para j, zero caso contrário.

O programa inteiro misto PC definido pela função objetivo (2.1) e pelas restrições (2.2)-(2.6), como mostrado na Figura 2.1, é uma formulação válida para o caso de topologia unidirecional, tanto para a variante com entrada simétrica ($e(u, v) = e(v, u)$) como para a variante com entrada assimétrica ($e(u, v)$ não necessariamente igual a $e(v, u)$) de CMPkC:

As restrições (2.2) são as equações de conservação de fluxo. As inequações (2.3) asseguram que os caminhos estabelecidos serão vértice-disjuntos. As inequações (2.4)

$$\min \sum_{i \in V} p_i \tag{2.1}$$

sujeito a:

$$\sum_{j \in V} f_{ji}^c - \sum_{l \in V} f_{il}^c = D_c(i), \qquad \forall c \in C, \forall i \in V \tag{2.2}$$

$$\sum_{j \in V} f_{ij}^c \leq 1, \qquad \forall c \in C, \forall i \in V : i \neq o(c), i \neq d(c) \tag{2.3}$$

$$p_i \geq e(i,j) \cdot f_{ij}^c, \qquad \forall i,j \in V, \forall c \in C \tag{2.4}$$

$$f_{ij}^c \in \{0,1\}, \qquad \forall i,j \in V, \forall c \in C \tag{2.5}$$

$$p_i \geq 0, \qquad \forall i \in V \tag{2.6}$$

Figura 2.1: Modelo com potências contínuas (PC).

estabelecem que o arco (i,j) deve ser usado se existe um fluxo positivo através dele. Se o arco (i,j) é usado, a potência p_i atribuída ao nó i deve ser pelo menos tão grande quanto o requisito $e(i,j)$. As restrições (2.5) e (2.6) expressam os requisitos de integralidade e a não negatividade das variáveis f e p, respectivamente.

Sempre que uma topologia bidirecional é exigida, as restrições

$$p_i \geq e(i,j) \cdot f_{ji}^c, \qquad \forall i,j \in V, \forall c \in C \tag{2.7}$$

são adicionadas, garantindo que uma aresta $[i,j]$ seja utilizada se existe fluxo do nó i para j ou do nó j para i.

2.2 Modelo com Potências Discretas

Todo nó pode se comunicar com os demais nós da rede por meio de sucessivos incrementos em sua potência de transmissão. Apesar da potência de transmissão de um nó ser uma grandeza contínua, é possível determinar um número finito e discreto de níveis de potência nos quais sua operação é eficiente. Para qualquer nó $i \in V$, os valores de potência onde sua transmissão é eficiente correspondem ao custo $e(i,j)$, para algum $j \in V \setminus \{i\}$. Dessa forma, cada nível de potência de transmissão que um nó pode assumir alcança um número adicional de nós em relação ao nível imediatamente inferior. Isso não ocorre necessariamente quando o nó transmite com outras potências, o que torna esses valores ineficientes.

Após uma fase de pré-processamento dos dados, serão determinados os níveis eficientes de potência em cada nó $i \in V$, que serão os único considerados. Seja então $P_i = [p_i^1, \ldots, p_i^{\phi(i)}]$ uma lista finita e em ordem crescente, cujos elementos representam os níveis de potências eficientes e relevantes para o nó $i \in V$. O nível de potência atribuído ao nó i determina os nós alcançados a partir de i. Denota-se por p_i^1 a potência mínima que permite que transmissões do nó i alcancem algum nó em $V \setminus \{i\}$. Além disso, $\phi(i) \leq |V| - 1$ e $p_i^{\ell+1} > p_i^\ell$ para qualquer $\ell = 1, \ldots, \phi(i) - 1$. Define-se também S_i^ℓ como o conjunto de nós alcançáveis pelo nó i com a atribuição da potência $p_i = p_i^\ell$, para qualquer $\ell = 1, \ldots, \phi(i)$, como mostrado na Figura 2.2. Observa-se que $S_i^{\phi(i)} = V \setminus \{i\}$ é o conjunto total de nós alcançáveis pelo nó i. Assume-se que $S_0 = \emptyset$.

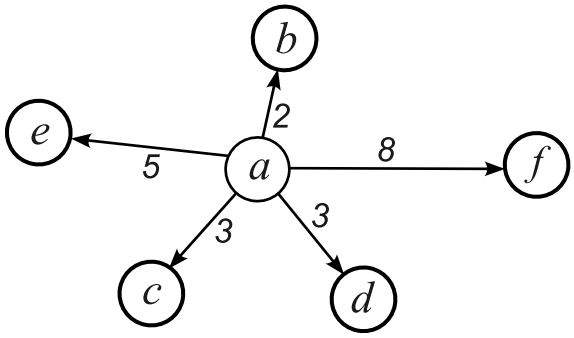


Figura 2.2: $P_a = [2, 3, 5, 8]$ e $S_a^1 = \{b\}$, $S_a^2 = \{b, c, d\}$, $S_a^3 = \{b, c, d, e\}$ e $S_a^4 = \{b, c, d, e, f\}$.

A variável discreta f_{ij}^c é definida como na seção anterior e representa o fluxo da comodidade c através do arco (i, j), para $c \in C$ e para $i, j \in V$. Para qualquer $\ell = 1, \ldots, \phi(i)$, a variável binária w_i^ℓ é igual a um se existe um nó $j \in S_i^\ell$ tal que o arco (i, j) é utilizado para comunicação do nó i para j, zero caso contrário. Define-se $\bar{\ell}(i) \in \{1, \ldots, \phi(i)\}$ de maneira que $|S_i^{\bar{\ell}(i)-1}| < k \leq |S_i^{\bar{\ell}(i)}|$. Portanto, para qualquer nó $i \in V$, a potência $p_i^{\bar{\ell}(i)}$ representa um limite inferior para o nível de transmissão do nó i que permite satisfazer a restrição de k-conexidade, ou seja, é o menor nível de potência do nó i que permite alcançar pelo menos k outros nós. Define-se também $\ell'(i, j) = 1$ se $p_i^1 = e(i, j)$; $\ell'(i, j) \in \{2, \ldots, \phi(i)\}$ se $p_i^{\ell'(i,j)-1} < e(i, j) \leq p_i^{\ell'(i,j)}$, ou seja, $p_i^{\ell'(i,j)}$ é o menor nível de potência necessário para estabelecer comunicação do nó i para j.

O programa inteiro misto PD definido pela função objetivo (2.8) e pelas restrições (2.9)-(2.15), como mostrado na Figura 2.3, também é uma formulação válida para o caso de topologia unidirecional, tanto para a variante simétrica como para a variante assimétrica do CMPkC.

As restrições (2.9), (2.10) e (2.13) desta formulação são as mesmas restrições (2.2),

$$\min \sum_{i \in V} \sum_{\ell=1}^{\phi(i)} p_i^\ell \cdot w_i^\ell \tag{2.8}$$

sujeito a:

$$\sum_{j \in V} f_{ji}^c - \sum_{l \in V} f_{il}^c = D_c(i), \qquad \forall c \in C, \forall i \in V \tag{2.9}$$

$$\sum_{j \in V} f_{ij}^c \leq 1, \qquad \forall c \in C, \forall i \in V : i \neq o(c), i \neq d(c) \tag{2.10}$$

$$\sum_{\ell=\bar{\ell}(i)}^{\phi(i)} w_i^\ell = 1, \qquad \forall i \in V \tag{2.11}$$

$$w_i^\ell = 0, \qquad \forall i \in V, \ell = 1, \ldots, \bar{\ell}(i) - 1 \tag{2.12}$$

$$\sum_{\ell=\max(\ell'(i,j),\bar{\ell}(i))}^{\phi(i)} p_i^\ell \cdot w_i^\ell \geq e(i,j) \cdot f_{ij}^c, \qquad \forall c \in C, \forall i,j \in V \tag{2.13}$$

$$f_{ij}^c \in \{0,1\}, \qquad \forall i,j \in V, \forall c \in C \tag{2.14}$$

$$w_i^\ell \in \{0,1\}, \qquad \forall i \in V, \ell = 1, \ldots, \phi(i). \tag{2.15}$$

Figura 2.3: Modelo com potências discretas (PD).

(2.3) e (2.4) da formulação anterior, respectivamente. Um nó com potência nula não pode transmitir ou encaminhar mensagens. Já que o grafo de comunicação $G(V, E(p))$ tem que ser k-conexo, cada nó precisa estar habilitado a se comunicar com pelo menos k outros nós. Assim, a potência a ele atribuída precisa ser suficiente para alcançar pelo menos os k nós mais próximos a ele. As restrições (2.11) estabelecem que apenas um único nível de potência é atribuído a cada nó e asseguram que esse nível de potência é capaz de alcançar pelo menos k nós. As restrições (2.12) complementam as restrições (2.11), desconsiderando os níveis de potência que se apresentam incapazes de alcançar pelo menos k nós. Como as restrições (2.11) garantem que somente um único nível de potência é atribuído a cada nó, as inequações (2.13) determinam que somente os níveis de potência maiores que o requisito de potência $e(i,j)$ são aceitáveis. As restrições (2.14) e (2.15) estabelecem os requisitos de integralidade das variáveis f e w.

Esta formulação fornece a solução exata para o caso de topologia unidirecional. Se for exigida uma topologia bidirecional, basta acrescentar as restrições

$$\sum_{\ell=\max(\ell'(i,j),\bar{\ell}(i))}^{\phi(i)} p_i^\ell \cdot w_i^\ell \geq e(i,j) \cdot f_{ji}^c, \qquad \forall c \in C, \forall i,j \in V \tag{2.16}$$

que garantirão a existência de um arco em cada direção.

2.3 Modelo com Potências Incrementais

Seja $Q_i = [q_i^1, \ldots, q_i^{\phi(i)}]$ uma lista finita, cujos elementos são incrementos sucessivos e cumulativos dos níveis de potência que podem ser atribuídos ao nó $i \in V$, com $\ell = 1, \ldots, \phi(i) - 1$. Considere também T_i^ℓ o conjunto de novos nós alcançáveis a partir do nó i quando o incremento q_i^ℓ é adicionado à potência corrente atribuída ao nó i. Com respeito à notação definida na seção anterior, $q_i^1 = p_i^1$, $T_i^1 = S_i^1$, $q_i^\ell = p_i^\ell - p_i^{\ell-1}$ e $T_i^\ell = S_i^\ell - S_i^{\ell-1}$ para qualquer $\ell = 2, \ldots, \phi(i)$, como mostrado na Figura 2.4.

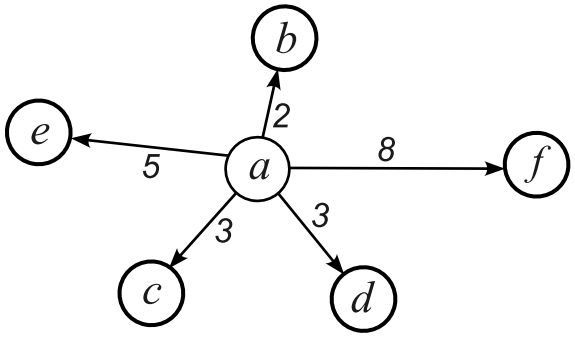


Figura 2.4: $Q_a = [2, 1, 2, 3]$ e $T_a^1 = \{b\}$, $T_a^2 = \{c, d\}$, $T_a^3 = \{e\}$ e $T_a^4 = \{f\}$.

A variável discreta f_{ij}^c representa o fluxo da comodidade c através do arco (i, j). A variável binária x_i^ℓ assume o valor um se existe um nó $j \in T_i^\ell$ tal que (i, j) é usado para comunicação do nó i para j, zero caso contrário.

O programa inteiro misto PI definido pela função objetivo (2.17) e pelas restrições (2.18)-(2.24), como mostrado na Figura 2.5, também é uma formulação válida para o caso de topologia unidirecional, tanto para a versão simétrica como para a versão assimétrica do CMPkC.

As restrições (2.18), (2.19) e (2.23) desta formulação são as mesmas restrições (2.2), (2.3) e (2.4) da primeira formulação, respectivamente. As inequações (2.20) estabelecem que x_i^ℓ precisa ser igual a um se existe um nó $j \in T_i^\ell$ tal que o arco (i, j) é usado para comunicação do nó i para j pela comodidade c. As restrições (2.21) asseguram que $x_i^{\ell+1}$ é necessariamente igual a zero se o incremento prévio não foi utilizado, ou seja, se $x_i^\ell = 0$. As restrições (2.22) atribuem valor um às potências incrementais necessárias para alcançar pelo menos os k nós mais próximos a cada nó i. As restrições (2.23) e (2.24) especificam os requisitos de integralidade das variáveis f e x, respectivamente.

$$\min \sum_{i \in V} \sum_{\ell=1}^{\phi(i)} q_i^\ell \cdot x_i^\ell \tag{2.17}$$

sujeito a:

$$\sum_{j \in V} f_{ji}^c - \sum_{l \in V} f_{il}^c = D_c(i), \qquad \forall c \in C, \forall i \in V \tag{2.18}$$

$$\sum_{j \in V} f_{ij}^c \leq 1, \qquad \forall c \in C, \forall i \in V : i \neq o(c), i \neq d(c) \tag{2.19}$$

$$x_i^\ell \geq f_{ij}^c, \qquad \forall i \in V, \forall c \in C, \forall j \in T_i^\ell, \ell = 1, \ldots, \phi(i) \tag{2.20}$$

$$x_i^{\ell+1} \leq x_i^\ell, \qquad \forall i \in V, \ell = 1, \ldots, \phi(i) - 1 \tag{2.21}$$

$$x_i^\ell = 1, \qquad \forall i \in V, \ell = 1, \ldots, \bar{\ell}(i) \tag{2.22}$$

$$f_{ij}^c \in \{0, 1\}, \qquad \forall i, j \in V, \forall c \in C \tag{2.23}$$

$$x_i^\ell \in \{0, 1\}, \qquad \forall i \in V, \ell = 1, \ldots, \phi(i). \tag{2.24}$$

Figura 2.5: Modelo com potências incrementais (PI).

Quando for exigida uma topologia bidirecional, basta acrescentar as restrições

$$x_i^\ell \geq f_{ji}^c, \qquad \forall i \in V, \forall c \in C, \forall j \in T_i^\ell, \ell = 1, \ldots, \phi(i) \tag{2.25}$$

para garantir a existência de um arco em cada direção.

No caso acima, observa-se que a potência atribuída ao nó i deve ser suficiente para estabelecer ligações bidirecionais sempre que uma topologia bidirecional for exigida. As inequações

$$x_j^m \geq x_i^\ell - x_i^{\ell+1}, \qquad \forall i, j \in V, \ell = 1, \ldots, \phi(i) - 1, \\ m = 1, \ldots, \phi(j) : i \in T_j^m, j \in T_i^\ell, |T_i^\ell| = 1 \tag{2.26}$$

implicam que a potência de transmissão do nó i é determinada de maneira a alcançar o nó $j \in T_i^\ell$ como o mais distante, ou seja, $x_i^\ell = 1$ e $x_i^{\ell+1} = 0$. Elas implicam na existência da aresta $[i, j]$. Dessa forma, elas forçam x_j^m a ser igual a um, para $i \in T_j^m$.

Pode-se também substituir as restrições unidirecionais (2.20) e (2.25) pela restrição bidirecional

$$x_i^\ell \geq f_{ij}^c + f_{ji}^c, \qquad \forall i \in V, \forall c \in C, \forall j \in T_i^\ell, \ell = 1, \ldots, \phi(i). \tag{2.27}$$

Toda solução que satisfizer as restrições (2.27) também satisfaz as restrições (2.20) e (2.25).

A fim de demonstrar que o inverso também é válido, basta provar que f_{ij}^c e f_{ji}^c não podem ser iguais a um simultaneamente. Isso não pode ocorrer pois, caso ocorresse, haveria um ciclo da comodidade c através dos nós i e j.

O modelo PI estendido com o conjunto de inequações (2.26) e (2.27) é referenciado como *modelo com potência incremental bidirecional* (PI-B).

2.4 Resultados Computacionais

Os experimentos computacionais foram realizados com duas classes de instâncias assimétricas geradas randomicamente, contendo de 10 a 50 nós. Para cada tamanho e variante do problema CMPkC, 15 instâncias testes foram geradas:

1. Instâncias euclideanas EU: geradas em uma grade de lado unitário com $|V| \in [10, 50]$. Para cada instância, os nós foram aleatoriamente posicionados na grade. Para gerar valores assimétricos, o custo do arco $e(u, v)$ é dado pelo produto de uma perturbação aleatória $F \in [0,8; 1,2]$ pelo quadrado da distância euclideana $d_{u,v}$ entre u e v, portanto $e(u, v) = Fd_{uv}^2$.
2. Instâncias randômicas RD: instâncias aleatoriamente geradas com $|V| \in [10, 50]$ e custos dos arcos $e(u, v)$ gerados no intervalo $(0, 1]$ segundo uma distribuição uniforme.

Todos os experimentos foram realizados em uma máquina Intel Core 2 Quad com relógio de 2.40 GHz e 8 Gbytes de memória RAM com sistema operacional GNU/Linux 2.6.24. O resolvedor utilizado para solucionar as formulações exatas por programação inteira foi o CPLEX versão 11.0.

2.4.1 Primeiro Experimento: Restrição de k-conexidade

No primeiro conjunto de experimentos foram utilizadas as instâncias euclideanas para comparar os tempos computacionais dos diferentes modelos para valores do parâmetro k variando de 2 a $|V| - 1$.

As Tabelas 2.1 e 2.2 mostram o tempo computacional em segundos e os valores da solução ótima para as instâncias euclideanas simétricas e assimétricas com 15 nós, respectivamente, obtidos pelas formulações PC, PD e PI. As Tabelas 2.3 e 2.4 mostram os mesmos resultados para as instâncias com 20 nós obtidos pelas formulações PD e PI. A

	Topologia unidirecional				Topologia bidirecional				
	Tempo (s)			Valor	Tempo (s)				Valor
k	PC	PD	PI	ótimo	PC	PD	PI	PI-B	ótimo
2	(12) 2909,30	29,72	20,98	1,52	400,55	20,54	10,28	7,55	1,57
3	(7) 5056,50	76,20	34,46	2,30	2137,02	54,14	30,01	17,78	2,40
4	(3) 7640,35	103,22	37,02	3,17	(12) 2471,24	63,72	34,38	17,12	3,27
5	(1) 8564,12	142,30	65,34	4,08	(9) 5309,48	85,43	39,08	27,16	4,22
6	(0) –	109,55	41,06	5,03	(4) 6766,61	81,92	34,02	23,14	5,18
7	(0) –	105,35	38,22	5,99	(7) 3906,37	76,01	35,36	22,08	6,18
8	(0) –	48,28	18,59	6,99	(1) 2871,09	39,46	20,28	12,71	7,20
9	(0) –	19,16	11,17	7,93	(1) 5368,02	34,05	11,05	9,29	8,18
10	(0) –	12,91	7,12	8,87	(2) 8209,58	16,96	7,19	4,70	9,15
11	(1) 5580,70	6,03	2,44	9,91	(4) 5278,28	12,49	4,78	5,46	10,14
12	(3) 6391,45	7,23	1,59	10,91	(9) 4466,20	10,35	1,87	2,05	11,10
13	6,19	1,67	0,59	11,99	4,88	1,36	0,36	0,32	12,10
14	0,23	0,33	0,23	13,06	0,15	0,28	0,16	0,12	13,06

Tabela 2.1: Tempo computacional (em segundos) e valores médios da solução ótima: entrada assimétrica com $|V| = 15$.

	Topologia unidirecional				Topologia bidirecional				
	Tempo (s)			Valor	Tempo (s)				Valor
k	PC	PD	PI	ótimo	PC	PD	PI	PI-B	ótimo
2	(12) 2298,42	27,04	15,57	1,63	410,63	20,88	12,25	7,24	1,67
3	(5) 3052,69	64,11	21,46	2,47	1901,08	43,12	18,95	9,95	2,54
4	(1) 3414,39	76,32	29,53	3,40	(12) 2616,42	51,40	31,94	23,74	3,48
5	(0) –	98,84	39,38	4,40	(7) 4621,02	75,22	33,20	19,51	4,49
6	(0) –	114,04	58,04	5,43	(2) 6286,75	81,15	32,43	21,56	5,50
7	(0) –	87,15	48,43	6,45	(2) 4644,15	57,77	26,90	21,42	6,54
8	(0) –	69,79	28,76	7,53	(2) 6841,71	43,20	24,56	12,10	7,64
9	(0) –	28,72	16,92	8,57	(0) –	40,45	12,36	12,17	8,67
10	(0) –	16,51	11,01	9,62	(2) 4688,78	18,06	5,78	6,83	9,74
11	(0) –	9,07	6,16	10,71	(6) 4997,13	13,08	3,79	3,78	10,78
12	(4) 3917,60	6,24	1,63	11,76	(6) 4975,63	9,66	1,43	1,59	11,82
13	10,16	1,92	0,77	12,85	6,37	1,49	0,45	0,39	12,89
14	0,19	0,31	0,20	13,85	0,14	0,24	0,13	0,11	13,85

Tabela 2.2: Tempo computacional (em segundos) e valores médios da solução ótima: entrada simétrica com $|V| = 15$.

formulação PC não obteve o valor da solução ótima dentro do tempo limite de três horas de processamento para instâncias com 20 nós. Todos os valores apresentados nessas tabelas são os resultados médios de uma execução do algoritmo sobre cada uma das quinze instâncias euclideanas de mesmo tamanho.

Foram descartadas as instâncias cuja solução ótima não foi encontrada por uma determinada formulação em três horas de processamento. Nesse caso, as Tabelas 2.1 a 2.4 apresentam entre parênteses o número de instâncias solucionadas de forma exata por cada formulação e usadas nos cálculos dos valores médios da respectiva formulação. A formulação PC foi a única que não foi capaz de encontrar soluções ótimas dentro do tempo limite de três horas, tanto para a topologia unidirecional como para a topologia bidirecional,

	Topologia unidirecional			Topologia bidirecional			
	Tempo (s)		Valor	Tempo (s)			Valor
k	PD	PI	ótimo	PD	PI	PI-B	ótimo
2	268,79	177,59	1,35	160,83	72,89	67,79	1,39
3	1090,84	779,51	2,08	1018,91	426,89	290,34	2,17
4	1828,26	1104,11	2,89	1458,10	671,47	420,07	2,98
5	1818,49	1479,14	3,73	1760,31	1051,78	641,60	3,89
6	1757,40	1287,72	4,61	1263,74	680,89	562,82	4,82
7	1928,14	895,80	5,56	1579,53	626,97	391,30	5,80
8	2067,12	720,45	6,48	1627,56	571,64	380,74	6,69
9	(14) 1866,11	397,01	7,32	2047,16	598,10	473,43	7,65
10	1405,65	421,43	8,32	1609,07	377,42	269,88	8,63
11	(14) 1326,30	317,64	9,23	2300,35	473,91	388,87	9,68
12	952,18	208,39	10,37	1257,34	169,95	140,80	10,71
13	412,91	122,25	11,38	1248,65	186,67	142,22	11,76
14	158,40	65,66	12,42	377,77	44,29	51,74	12,74
15	107,51	36,71	13,45	131,35	36,26	24,10	13,78
16	79,16	30,06	14,47	140,39	35,86	32,85	14,80
17	45,10	8,65	15,66	73,56	5,45	5,78	15,91
18	11,50	3,34	16,79	9,34	1,97	1,59	16,92
19	1,24	0,73	17,92	0,98	0,50	0,38	17,92

Tabela 2.3: Tempo computacional (em segundos) e valores médios da solução ótima: entrada assimétrica com $|V| = 20$.

	Topologia unidirecional			Topologia bidirecional			
	Tempo (s)		Valor	Tempo (s)			Valor
k	PD	PI	ótimo	PD	PI	PI-B	ótimo
2	277,16	179,02	1,46	226,05	54,82	47,26	1,48
3	615,13	464,32	2,23	592,78	236,07	160,49	2,30
4	1200,44	732,42	3,09	1030,55	404,07	310,09	3,17
5	1564,31	1405,94	4,02	1645,62	779,98	649,54	4,13
6	1761,35	1154,83	4,98	1409,22	605,27	492,14	5,11
7	2042,92	795,34	5,97	1279,66	577,23	276,39	6,11
8	1933,01	499,87	6,92	774,83	325,02	309,01	7,06
9	1440,98	524,45	7,92	1413,48	390,23	245,79	8,09
10	2031,82	495,39	8,95	1645,91	412,01	288,60	9,11
11	(13) 1018,45	184,48	9,93	2196,05	356,81	291,57	10,21
12	1745,10	299,14	11,14	1094,11	234,45	181,36	11,33
13	986,46	187,22	12,24	1163,28	173,41	121,32	12,42
14	367,52	119,45	13,29	605,58	68,08	50,82	13,44
15	89,85	60,20	14,40	138,23	25,48	22,89	14,56
16	73,23	34,18	15,54	105,32	18,20	14,85	15,68
17	58,82	16,18	16,74	85,87	8,92	13,00	16,83
18	11,51	3,47	17,88	9,31	2,05	1,62	17,92
19	1,23	0,72	18,95	1,00	0,50	0,36	18,95

Tabela 2.4: Tempo computacional (em segundos) e valores médios da solução ótima: entrada simétrica com $|V| = 20$.

para instâncias com $|V| = 15$ (ver Tabelas 2.1 e 2.2).

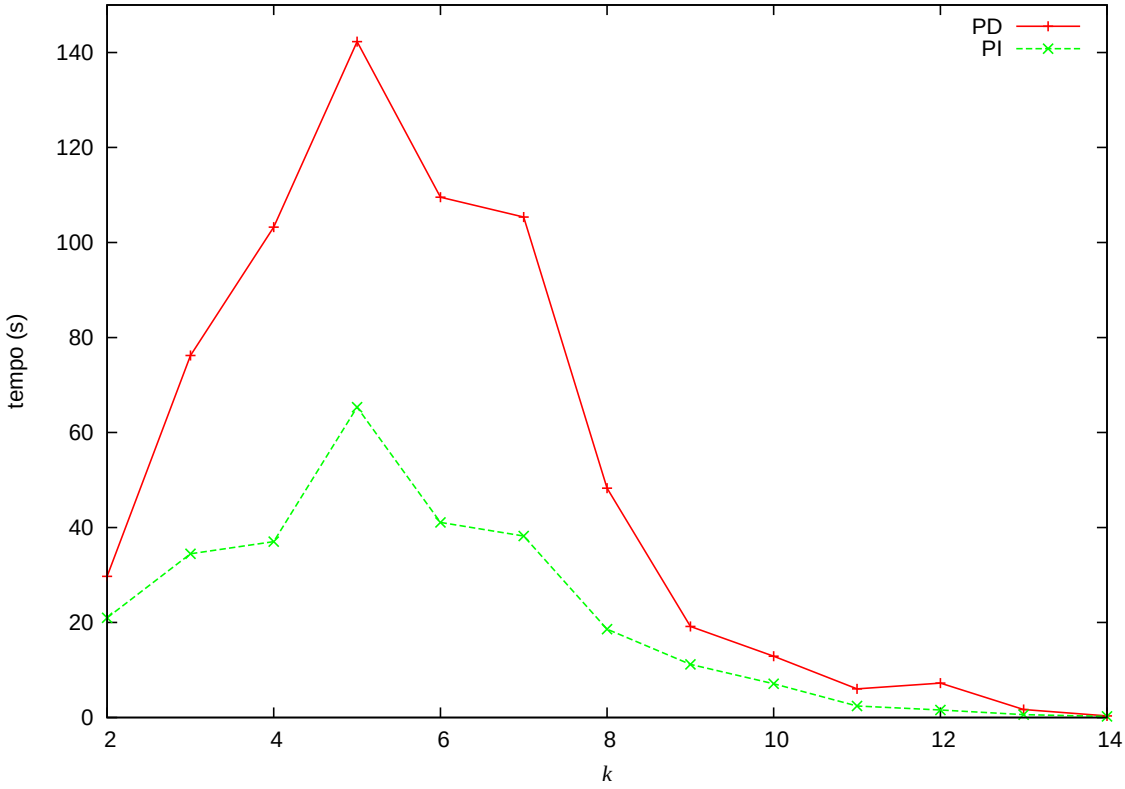
A formulação PD falhou ao tentar encontrar soluções ótimas dentro do tempo limite de três horas para $k = 9$ e $k = 11$ no caso de instâncias com $|V| = 20$ e topologia unidirecional com entrada assimétrica (ver Tabela 2.3) e para $k = 11$ no caso de topologia unidirecional com entrada simétrica (ver Tabela 2.4). As formulações PI e PI-B sempre encontraram a solução ótima dentro do tempo limite de três horas de processamento. A formulação PC não aparece nas Tabelas 2.1 a 2.4 pois ela não encontra as soluções ótimas para $|V| = 20$.

As Tabelas 2.1 a 2.4 mostram que todas as formulações tornam-se progressivamente mais difíceis de serem resolvidas a medida que k cresce. As Figuras 2.6 e 2.7 apresentam o tempo computacional médio em segundos em função de valores crescentes de k, de acordo com os números das Tabelas 2.1 e 2.2. Esse comportamento é explicado pelo crescimento do número de variáveis, já que o número de comodidades é determinado por $|C| = k \cdot O(|V|)$ e o número de restrições e variáveis é determinado por $|C| \cdot O(|V|^2)$.

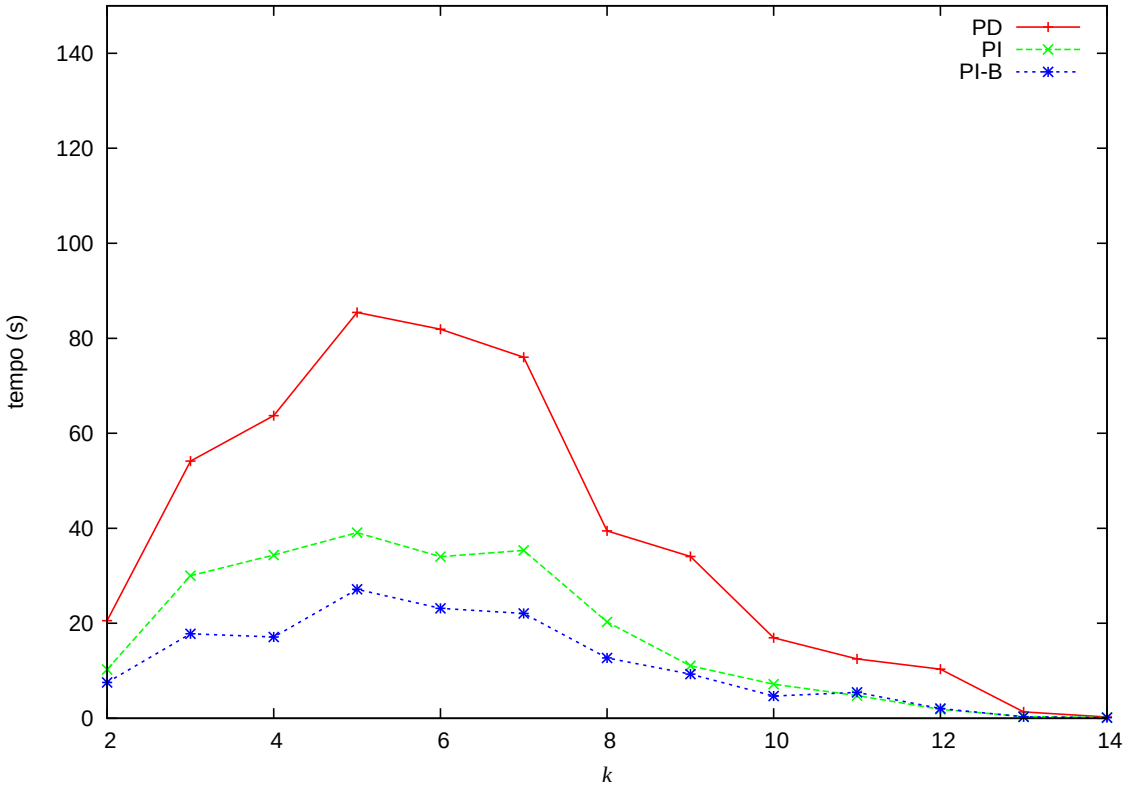
Entretanto, observa-se que o tempo de processamento decai à medida que k se aproxima de $|V| - 1$. Em redes tradicionais, a potência atribuída ao nó i deve ser maior ou igual a $e(i, u) + e(i, v)$ se ambos os nós u e v devem ser alcançados pelas transmissões do nó i. Porém, em redes sem fio, a potência atribuída ao nó i deve ser maior ou igual a $\max\{e(i, u), e(i, v)\}$ na mesma situação, que é um valor inferior a $e(i, u) + e(i, v)$. Essa redução de potência a ser atribuída a cada nó em relação às redes tradicionais pode ser vista como uma vantagem [54] alcançada pela transmissão *multicast* das antenas omini-direcionais usadas em redes ad hoc sem fio..

Quando o parâmetro k aumenta, cada nó requisita uma maior potência atribuída para ser capaz de transmitir através de pelo menos k arcos, garantindo a existência de k caminhos vértice-disjuntos. Portanto, a transmissão *multicast* obriga um número crescente de arcos a entrar na solução a medida que k cresce, reduzindo o número de variáveis de decisão livres.

A Figura 2.8 apresenta o crescimento, em função de k, dos valores das soluções ótimas para todos os tipos de instâncias euclideanas com $|V| = 15$, de acordo com as Tabelas 2.1 e 2.2. O valor médio das potências também cresce a medida que k aumenta. Na próxima seção, a análise dos experimentos é focada no caso biconexo ($k = 2$), pois é o caso com menor valor de potência atribuída total.

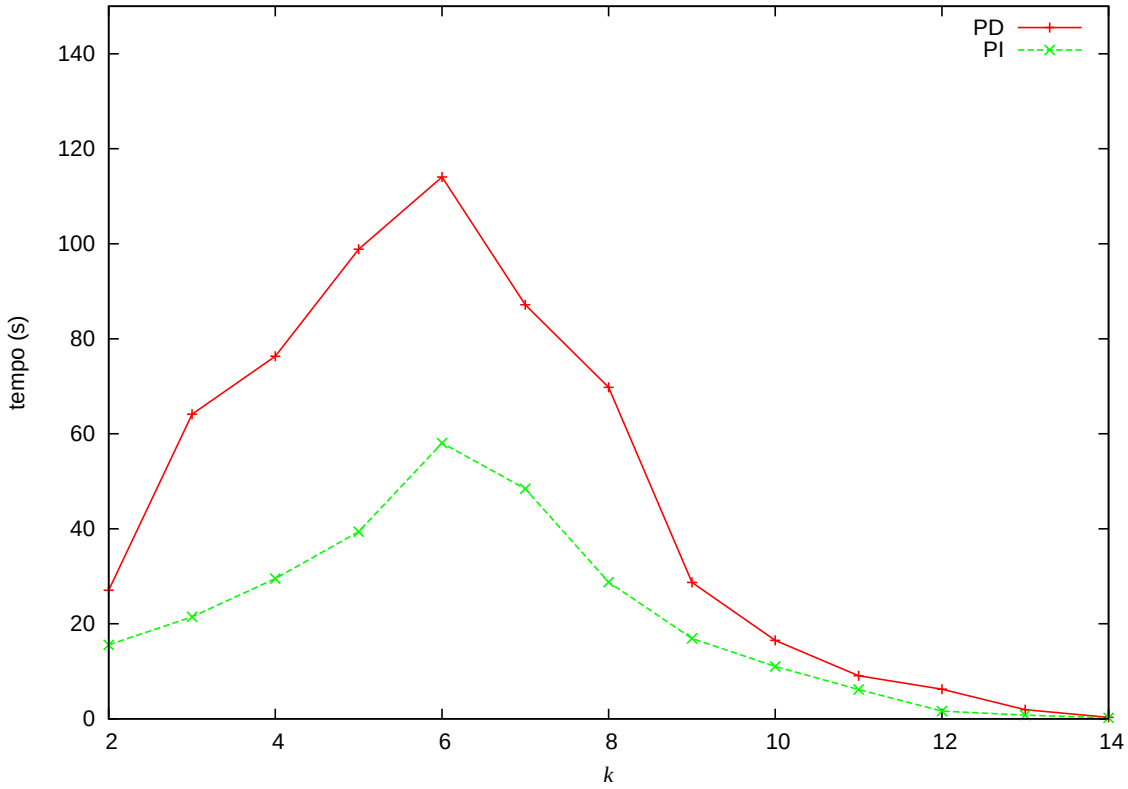


(a) Topologia unidirecional

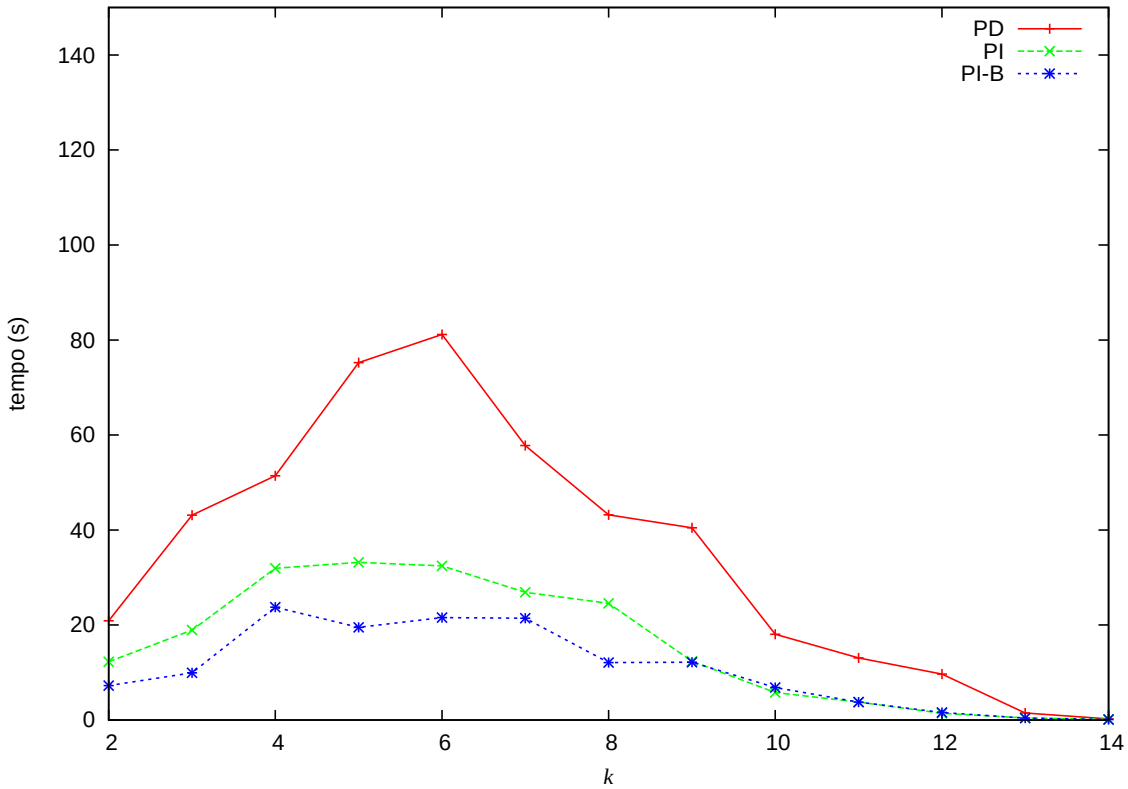


(b) Topologia bidirecional

Figura 2.6: Tempos de processamento para $|V| = 15$ e entrada assimétrica.



(a) Topologia unidirecional



(b) Topologia bidirecional

Figura 2.7: Tempos de processamento para $|V| = 15$ e entrada simétrica.

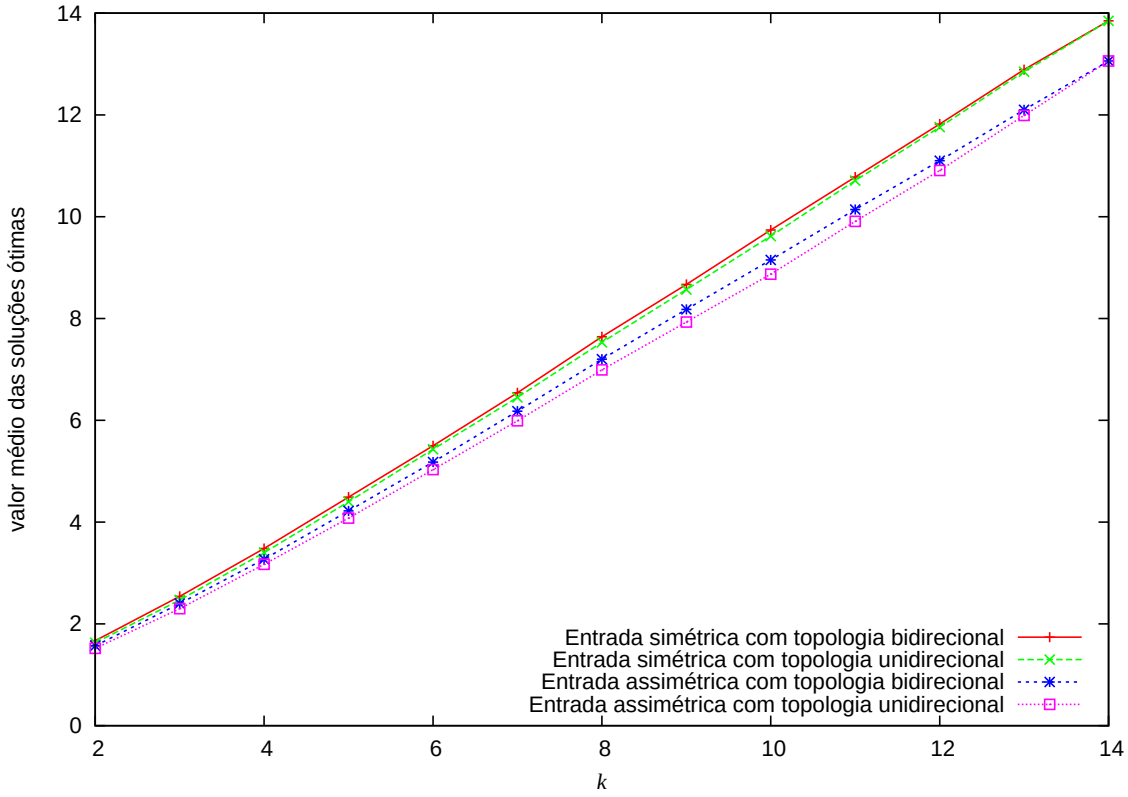


Figura 2.8: Valores das soluções ótimas para $|V| = 15$ para $k = 2, \ldots, 14$.

2.4.2 Segundo Experimento: Restrição de Biconexidade

A principal métrica utilizada para avaliação de desempenho no segundo experimento foi o tempo de processamento gasto por cada formulação para encontrar a solução ótima do problema de consumo mínimo de potência sob restrição de biconexidade (CMP2C) para instâncias euclideanas e randômicas.

A Tabela 2.5 mostra a quantidade de instâncias solucionadas pelo CPLEX com limite de tempo de até três horas de processamento para cada modelo e instância. Células com valor zero como conteúdo mostram que nenhuma instância foi solucionada pelo CPLEX dentro do tempo limite. Já as células em branco (marcadas com "–") correspondem aos tipos e tamanhos de instâncias para os quais uma determinada formulação não pode ser aplicada pois os tempos de processamento necessários para encontrar a solução ótima foram maiores do que três horas.

Como algumas formulações não solucionaram todas as instâncias em três horas, os valores das Tabelas 2.6 e 2.7 são a média dos resultados das instâncias cuja solução ótima foi encontrada por todas as formulações. Para cada dimensão do problema $|V| =$

		Euclideana							Randômica						
		Unidiretional			Bidiretional				Unidiretional			Bidiretional			
	$\|V\|$	PC	PD	PI	PC	PD	PI	PI-B	PC	PD	PI	PC	PD	PI	PI-B
Assimétrica	10	15	15	15	15	15	15	15	15	15	15	15	15	15	15
	15	12	15	15	15	15	15	15	9	15	15	1	15	15	15
	20	0	15	15	12	15	15	15	0	15	15	0	15	15	15
	25	–	15	15	–	15	15	15	–	15	15	–	1	4	15
	30	–	5	11	–	9	12	12	–	15	15	–	–	–	1
	50	–	–	–	–	–	–	–	–	0	0	–	–	–	0
Simétrica	10	15	15	15	15	15	15	15	10	15	15	15	15	15	15
	15	12	15	15	15	15	15	15	0	15	15	1	15	15	15
	20	0	15	15	12	15	15	15	–	15	15	–	15	15	15
	25	–	15	15	–	15	15	15	–	15	15	–	15	15	15
	30	–	6	10	–	9	13	12	–	15	15	–	15	15	15
	50	–	–	–	–	–	–	–	–	0	0	–	–	–	11

Tabela 2.5: Quantidade de instâncias solucionadas em três horas de processamento por cada formulação PC, PD, PI e PI-B, do problema CMP2C.

$10, 15, 20, 25, 30$ e para cada formulação, as Tabelas 2.6 e 2.7 exibem o tempo médio de processamento em segundos gasto pelo CPLEX e as diferenças médias percentuais relativas (M) entre a primeira solução inteira encontrada e o valor da relaxação linear naquele momento e entre o valor da relaxação linear e o valor da solução ótima inteira (salto de dualidade, D).

Os gráficos em escala logarítmica das Figuras 2.9 e 2.10 resumem os resultados das Tabelas 2.6 e 2.7 em relação ao comportamento das formulações exatas em termos do tempo computacional médio quando o número de nós cresce de 10 a 30, Os resultados mostram que a formulação PC consome mais tempo computacional e é muito difícil de ser resolvida. As formulações PD e PI se mostram mais fortes, levando a tempos de processamento menores e valores de relaxação linear mais próximos das soluções ótimas inteiras, para todos os tamanhos e tipos de instância. A formulação PI alcança tempos de processamento menores que PD.

A formulação PI-B para topologias bidirecionais é obtida pelo reforço da formulação PI através das inequações (2.26) e (2.27). O tempo de processamento e o salto de dualidade D observados com PI-B são reduzidos quando comparados com o modelo PI. Já a diferença relativa M obtida pelo CPLEX sofre pequeno aumento pois, apesar do limite da relaxação linear ser aprimorado, as restrições adicionais levam ao aumento do valor da função objetivo da primeira solução inteira encontrada.

		Instâncias euclideanas								
		Formulação PC			Formulação PD			Formulação PI		
	$\|V\|$	Tempo (s)	M (%)	D (%)	Tempo (s)	M (%)	D (%)	Tempo (s)	M (%)	D (%)
Assimétrica	10	51,73	84,67	61,69	0,75	24,82	11,06	0,89	27,53	11,06
	15	2909,30	92,98	68,87	23,72	35,49	13,75	16,20	43,60	13,75
	20	–	–	–	268,79	61,67	13,40	177,59	57,07	13,40
	25	–	–	–	3011,59	72,75	11,96	1563,94	66,93	11,96
	30	–	–	–	7186,82	77,04	7,47	2837,09	64,58	7,47
Simétrica	10	41,02	83,88	61,73	0,79	20,13	10,90	0,78	28,64	10,90
	15	2298,42	92,09	68,72	23,48	33,21	14,23	16,03	40,62	14,23
	20	–	–	–	277,16	61,20	12,80	179,02	58,21	12,80
	25	–	–	–	2405,44	74,57	12,15	1600,28	76,05	12,15
	30	–	–	–	7009,86	89,04	11,51	4875,97	82,02	11,51
		Instâncias randômicas								
		Formulação PC			Formulação PD			Formulação PI		
	$\|V\|$	Tempo (s)	M (%)	D (%)	Tempo (s)	M (%)	D (%)	Tempo (s)	M (%)	D (%)
Assimétrica	10	164,13	84,96	62,72	0,06	3,55	1,19	0,07	5,32	1,19
	15	2166,93	91,04	63,34	0,24	0,00	0,00	0,16	0,00	0,00
	20	–	–	–	1,98	0,01	0,01	0,87	0,27	0,01
	25	–	–	–	10,55	2,00	0,01	2,36	0,22	0,01
	30	–	–	–	36,40	0,09	0,02	5,69	0,01	0,02
Simétrica	10	3697,63	80,42	66,65	0,10	5,55	0,85	0,11	6,31	0,85
	15	–	–	–	1,11	6,70	0,40	0,74	3,67	0,40
	20	–	–	–	15,65	4,53	0,29	6,78	2,33	0,29
	25	–	–	–	55,76	13,77	0,32	20,43	6,99	0,32
	30	–	–	–	570,86	16,97	0,22	102,12	5,99	0,22

Tabela 2.6: Resultados computacionais obtidos por cada formulação (PC, PD e PI) para as variantes unidirecionais do CMP2C

		Instâncias euclideanas											
		Formulação PC			Formulação PD			Formulação PI			Formulação PI-B		
	$\|V\|$	Tempo (s)	M (%)	D (%)	Tempo (s)	M (%)	D (%)	Tempo (s)	M (%)	D (%)	Tempo (s)	M (%)	D (%)
Assimétrica	10	10,83	80,07	58,16	0,83	25,45	8,93	0,87	30,22	8,93	0,47	25,48	7,51
	15	400,55	91,67	66,75	20,54	32,11	11,51	10,28	31,27	11,51	7,55	28,07	10,34
	20	6878,64	94,61	68,74	141,12	40,45	9,40	62,69	45,37	9,40	66,61	72,26	8,10
	25	–	–	–	2357,44	55,05	9,20	543,38	41,16	9,20	298,53	63,73	7,71
	30	–	–	–	5393,10	53,16	5,42	1983,42	38,05	5,42	1351,98	56,92	4,56
Simétrica	10	9,52	80,18	58,61	0,63	24,48	8,54	0,73	19,44	8,54	0,48	23,61	7,25
	15	410,63	91,02	66,65	20,88	44,35	11,42	12,25	26,64	11,42	7,24	42,58	10,14
	20	5837,51	94,75	94,75	226,05	56,60	9,62	54,82	33,69	9,62	47,26	52,80	8,27
	25	–	–	–	1679,72	49,74	9,05	703,74	37,10	9,05	509,83	57,71	7,70
	30	–	–	–	5263,26	71,83	5,41	2436,90	52,81	5,41	1373,72	82,79	4,20
		Instâncias randômicas											
		Formulação PC			Formulação PD			Formulação PI			Formulação PI-B		
	$\|V\|$	Tempo (s)	M (%)	D (%)	Tempo (s)	M (%)	D (%)	Tempo (s)	M (%)	D (%)	Tempo (s)	M (%)	D (%)
Assimétrica	10	51,29	79,86	61,91	1,16	22,22	9,43	1,12	23,36	9,43	0,48	21,96	5,98
	15	–	–	–	95,00	29,35	20,48	20,65	26,96	20,48	6,99	31,92	10,83
	20	–	–	–	2306,43	47,88	25,90	576,04	36,02	25,90	117,36	35,32	10,87
	25	–	–	–	–	–	–	6438,73	57,32	28,70	872,44	63,49	13,48
	30	–	–	–	–	–	–	–	–	–	5559,86	81,42	13,55
Simétrica	10	63,49	78,67	66,45	0,31	7,79	1,31	0,25	4,74	1,31	0,15	8,93	0,82
	15	–	–	–	1,35	10,73	0,79	0,72	4,79	0,79	0,23	6,37	0,22
	20	–	–	–	18,10	6,45	1,31	12,13	5,18	1,31	2,69	12,73	0,28
	25	–	–	–	108,61	7,48	1,09	61,69	0,49	1,09	10,95	18,48	0,12
	30	–	–	–	479,79	7,68	1,22	334,03	0,55	1,22	73,71	22,26	0,24
	50	–	–	–	–	–	–	–	–	–	562,42	6,57	0,06

Tabela 2.7: Resultados computacionais obtidos por cada formulação (PC, PD, PI e PI-B) para as variantes bidirecionais do CMP2C.

Os resultados nas Tabelas 2.5, 2.6 e 2.7 mostram que o problema de consumo mínimo de potência é de difícil solução. Os saltos de dualidade D não são pequenos para instâncias randômicas da variante com entrada assimétrica e topologia bidirecional e para as instâncias euclideanas, o que torna muito difícil para o resolvedor encontrar as soluções ótimas dentro dos limites de tempo impostos. As outras variantes do problema em instâncias randômicas são mais fáceis de serem solucionadas, pois as soluções ótimas de suas relaxações lineares na raiz da árvore de busca estão mais próximas de suas soluções ótimas inteiras.

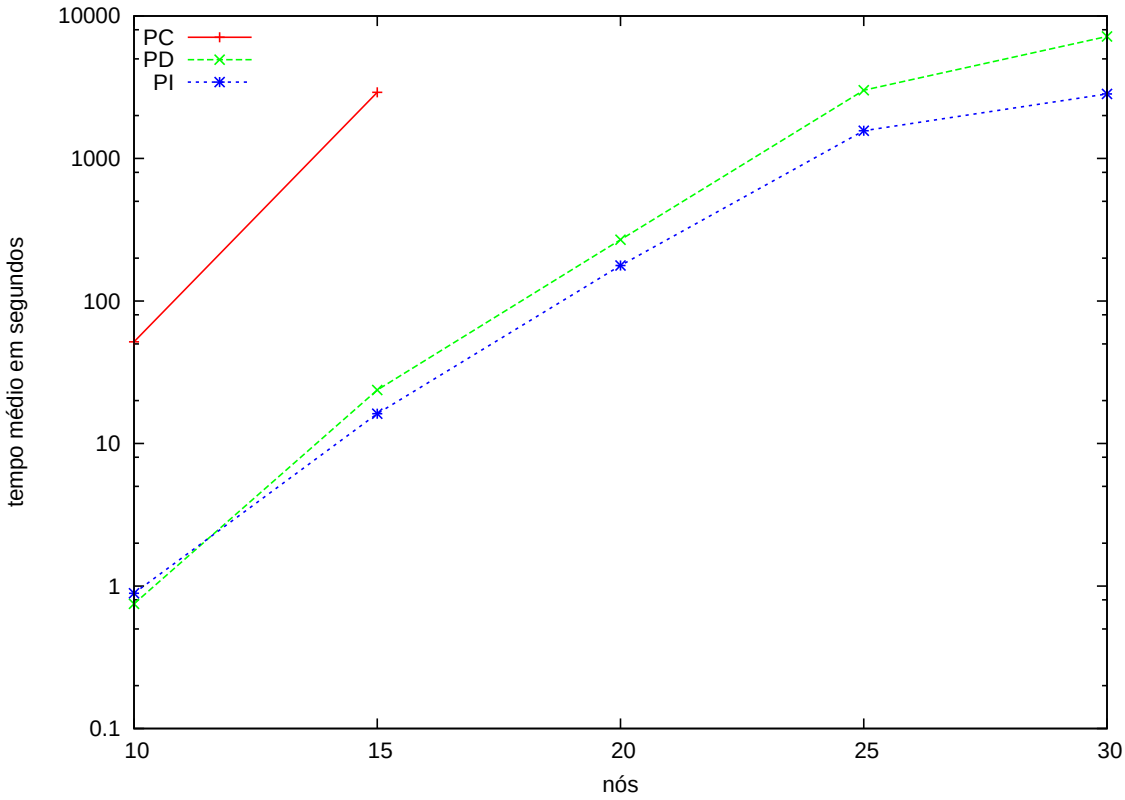
2.5 Conclusões

Neste capítulo foram propostos três modelos de programação inteira mista para o CMPkC. Todas as formulações foram aplicadas às quatro variantes do problema, dependendo da topologia do grafo de entrada (simétrica ou assimétrica) e da solução (unidirecional ou bidirecional).

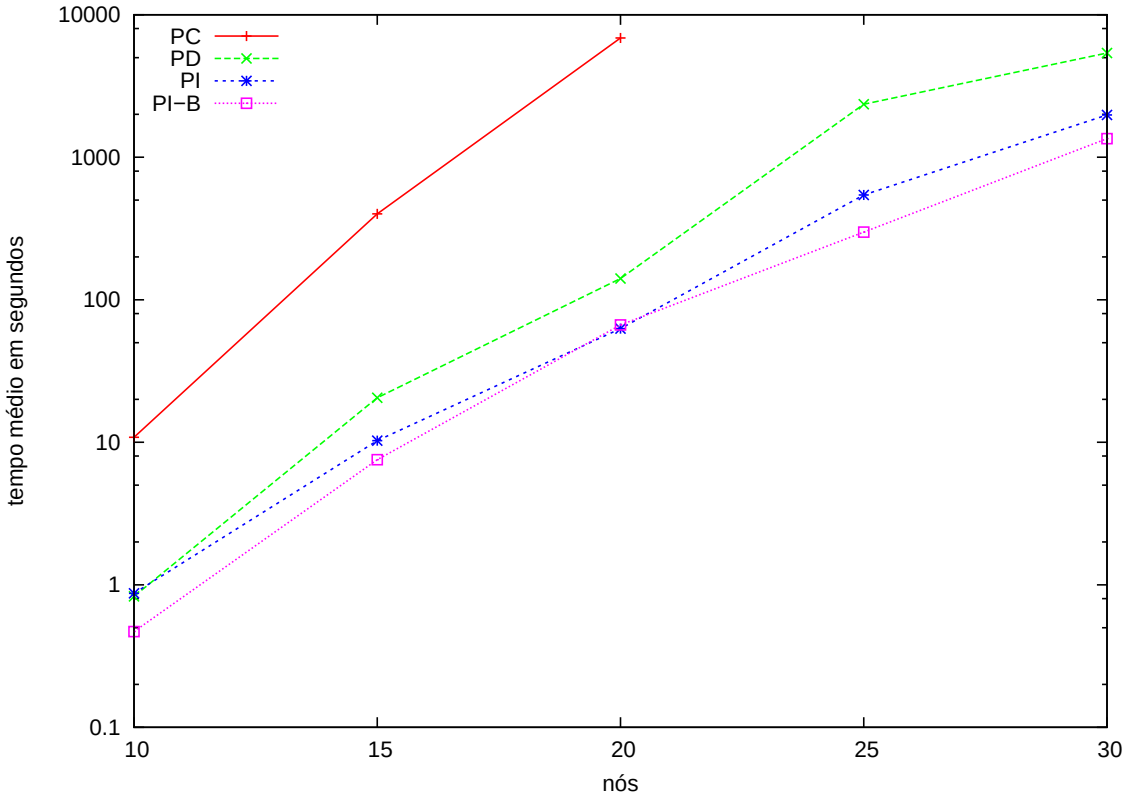
O primeiro modelo foi desenvolvido utilizando-se uma representação contínua da potência a ser atribuída a cada nó. Melhorias significativas nos tempos de processamento e nos limites da relaxação linear foram obtidas com formulações mais elaboradas utilizando-se uma representação discretizada das potências. Uma formulação ainda mais forte foi desenvolvida para soluções bidirecionais através da adição de desigualdades válidas às formulações com potências discretas.

Todas as conclusões foram apoiadas por experimentos computacionais abrangentes sobre duas classes de instâncias propostas nesta tese. Instâncias com até 50 nós foram resolvidas de forma exata.

Os resultados obtidos neste capítulo mostraram que a solução exata de instâncias de grande dimensão é inviável. Nos próximos capítulos os melhores resultados obtidos pelas formulações exatas serão utilizados na avaliação de heurísticas para o CMP2C com entrada assimétrica e topologia bidirecional (CMP2C-AB).

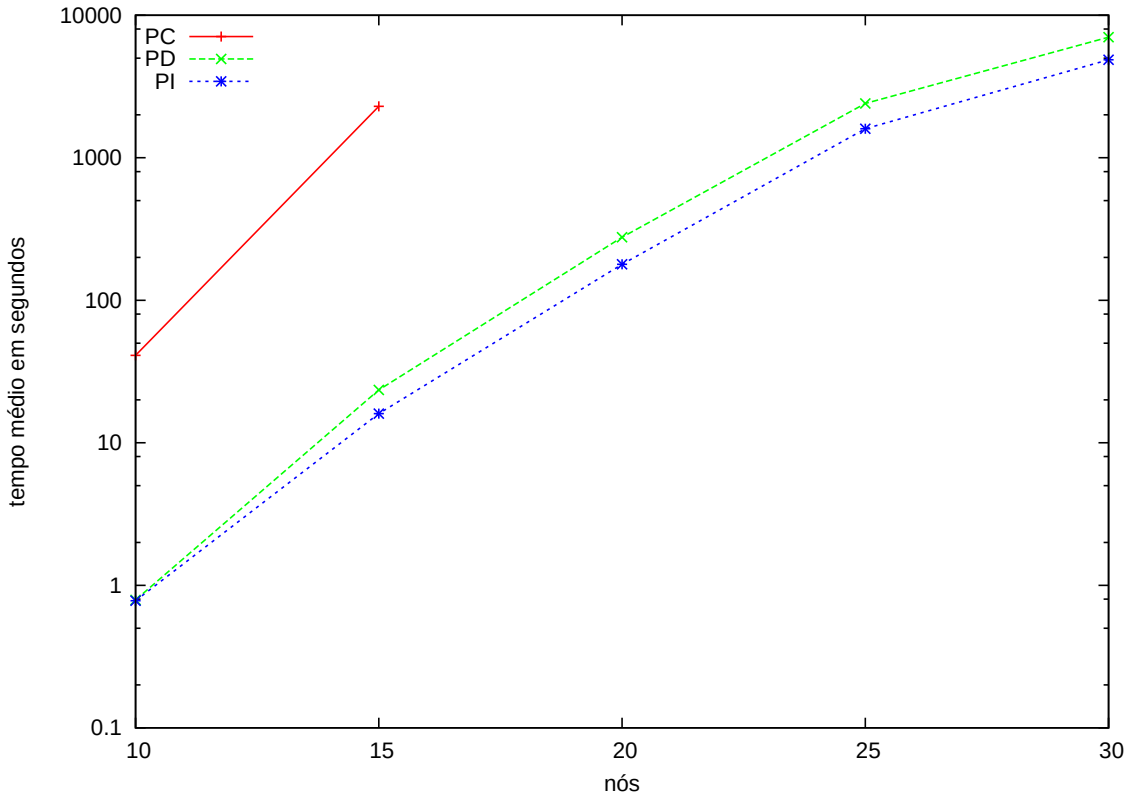


(a) Topologia unidirecional

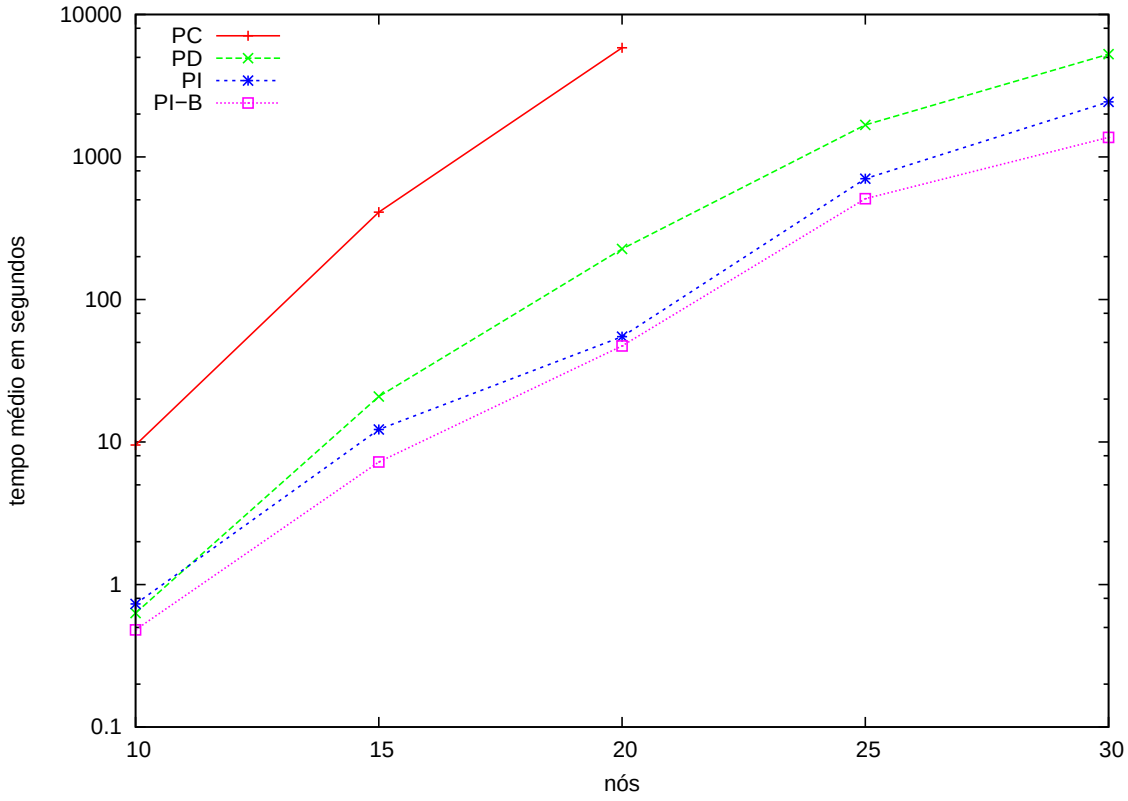


(b) Topologia bidirecional

Figura 2.9: Tempos de processamento para $|V| = 15$ e entrada assimétrica.



(a) Topologia unidirecional



(b) Topologia bidirecional

Figura 2.10: Tempos de processamento para $|V| = 15$ e entrada simétrica.

Capítulo 3

Algoritmos Gulosos para CMP2C-AB

No Capítulo 2 foi mostrada a dificuldade da resolução exata da variante do CMP2C-AB, para a qual só é possível encontrar resultados exatos em tempos computacionalmente aceitáveis para redes com até 30 nós. Entretanto, as redes ad hoc podem conter de dezenas a centenas de dispositivos interligados, o que torna bem limitados os resultados exatos para até 30 nós.

Na Seção 1.1 foi mostrado que os cenários típicos de utilização das redes ad hoc, aliados à complexidade dos modelos de transmissão sem fio, tornam mais realista a modelagem da rede através de grafos com entrada assimétrica. Viu-se ainda, na Seção 1.2.2, que, apesar de a solução unidirecional ser tecnicamente possível e consumir menos energia do que a solução bidirecional, a segunda facilita a utilização dos algoritmos que se encontram mais acima na pilha de protocolos de rede, como, por exemplo, os de roteamento. Além disso, a complexidade provocada por arcos unidirecionais torna a rede ineficiente [35].

A partir desse capítulo, serão desenvolvidos algoritmos para a variante do CMP2C-AB. Este problema reflete melhor as necessidades de uma rede ad hoc e gera a solução bidirecional com o menor consumo total de potência.

Considera-se que a rede ad hoc sem fio é representada pelo conjunto de nós V e por um custo não negativo $e(u, v)$ associado a cada par ordenado de nós (u, v) com $u, v \in V$.

Nesse capítulo, serão desenvolvidos algoritmos gulosos para o problema de minimização de potência em redes ad hoc com entrada assimétrica e topologia biconexa bidirecional. Em um algoritmo guloso, as soluções são construídas progressivamente desde o início, iteração a iteração. Em cada iteração, um novo elemento é incorporado à solução parcial em construção até que uma solução viável completa seja obtida.

A escolha do próximo elemento a ser incorporado à solução parcial é determinada pela

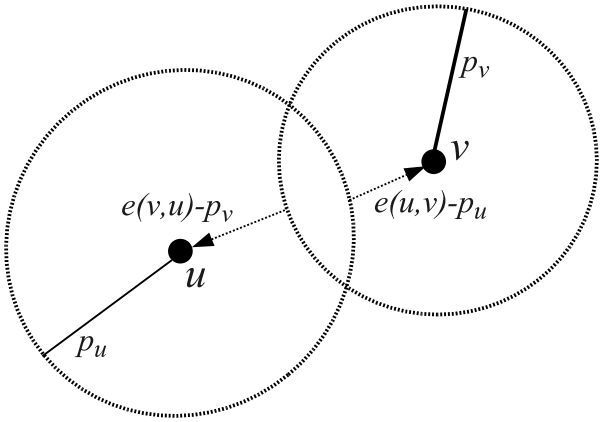


Figura 3.1: Exemplo de cálculo de função gulosa dinâmica.

avaliação de todos os elementos candidatos de acordo com uma função de avaliação gulosa $g : A \rightarrow R+$. Duas funções gulosas são definidas:

Definição 3.1 (função gulosa estática) *O custo estático da aresta* $[u, v]$ *é dado pela soma dos custos dos arcos* (u, v) *e* (v, u) *que a compõem:* $g(u, v) = e_e(u, v) = e(u, v) + e(v, u)$.

Definição 3.2 (função gulosa dinâmica) *O custo dinâmico da aresta* $[u, v]$ *é dado pela soma dos custos dos arcos* (u, v) *e* (v, u) *que a compõem, reduzidos da potência corrente dos nós u e v (ver Figura 3.1):* $g(u, v) = e_d(u, v) = \max\{0, e(u, v) - p_u\} + \max\{0, e(v, u) - p_v\}$.

Enquanto a função gulosa estática permite que o custo de cada aresta seja definido a priori, a função gulosa dinâmica exige sua atualização sempre que a potência de algum nó é alterada, o que ocorre sempre que uma nova aresta é incorporada à solução parcial. Observa-se que se $e_d(u, v) = 0$, então a aresta $[u, v]$ já está incorporada ao grafo de comunicação, ou seja, a potência dos nós u e v é tal que $p_u \geq e(u, v)$ e $p_v \geq e(v, u)$.

Os algoritmos apresentados a seguir utilizam as duas funções gulosas de maneira independente. Os algoritmos estão baseados na construção de grafos conexos e na determinação das componentes biconexas de um grafo.

3.1 Algoritmos Baseados na Construção de Grafos Conexos

Os algoritmos baseados na construção de grafos conexos são compostos de duas fases. Na primeira fase, encontra-se a atribuição de potências de transmissão $p_u : V \rightarrow R+$ para todo nó $u \in V$, tal que o grafo de comunicação bidirecional resultante $G(p) = (V, B(p))$ seja conexo, isto é, existe pelo menos um caminho nó-disjunto entre qualquer par de vértices. Na segunda fase, as potências são aumentadas (arestas são incorporadas à solução) até que o grafo de comunicação $G(p)$ satisfaça a restrição de biconexidade.

3.1.1 Primeira Fase

Nesta fase, encontra-se a atribuição de potências de transmissão $p_u : V \rightarrow R+$ para todo nó $u \in V$, tal que o grafo de comunicação bidirecional resultante $G(p) = (V, B(p))$, com $B(p) = \{(u, v) : u \in V, v \in V, p_u \geq e(u, v), p_v \geq e(v, u)\}$ (conforme definido na Seção 1.1) seja conexo. Os algoritmos desenvolvidos diferem pela função gulosa (estática ou dinâmica) utilizada.

3.1.1.1 Função Gulosa Estática

O valor da função gulosa estática é independente da potência corrente dos nós, mantendo-se inalterado durante toda a execução do algoritmo. Dessa forma, na primeira fase de construção, utiliza-se o algoritmo de Prim [40] para cálculo da Árvore Geradora Mínima (AGM) na construção do grafo de comunicação conexo $G(p) = (V, B(p))$ como apresentado no pseudocódigo do Algoritmo 3.1.

Algoritmo 3.1 Algoritmo para construção de grafo de comunicação conexo com função gulosa estática – gc(e_e)

Entrada: Conjuntos de nós V e custos $e(u, v), \forall u, v \in V$.
Saída: Potências de transmissão p e grafo de comunicação $G(p) = (V, B(p))$ conexo.

1: $p_u \leftarrow 0, \forall u \in V$;
2: Construir grafo não direcionado $H = (V, E)$;
3: Encontrar a árvore geradora mínima $T = (V, E')$ de H;
4: **para todo** $u \in V$ **faça**
5: $\quad p_u \leftarrow \max_{v \in V \setminus \{u\}} \{e(u, v) : [u, v] \in E'\}$;
6: **fim para**
7: **retorna** p;

O Algoritmo 3.1 inicializa na linha 1 todos os nós no nível de potência zero e, em

seguida, constrói na linha 2 um grafo não direcionado $H = (V, E)$, com $E = \{[u, v] : u, v \in V \text{ e } g(u, v) = e_e(u, v) = e(u, v) + e(v, u)\}$.

Em seguida, na linha 3, é encontrada a árvore geradora mínima $T(V, E')$, onde $E' \subseteq E$, aplicando-se o algoritmo de Prim ao grafo H. No laço da linha 4 à 6, para todo nó $u \in V$, é atribuída à potência p_u o maior custo dentre todos os custos dos arcos incidentes a u contidos na árvore geradora mínima T. O Algoritmo 3.1 possui a mesma complexidade $O(|E| \log |V|)$ do algoritmo de Prim e foi demonstrado ter fator de aproximação 2 em [9, 27] para o problema de minimização de potência em redes ad hoc com entrada simétrica e topologia 1-conexa para grafos euclideanos.

3.1.1.2 Função Gulosa Dinâmica

O valor da função gulosa dinâmica depende da potência corrente dos nós, exigindo sua atualização cada vez que a potência de algum nó é alterada. O pseudocódigo apresentado no Algoritmo 3.2 é similar ao algoritmo de Prim, onde as operações típicas de uma fila de prioridades são definidas como:

- `Insert`(i, a, F): insere o elemento i na fila de prioridades F, com prioridade a;
- `RetrieveKey`(i, F): retorna a prioridade do elemento i de F;
- `DecreaseKey`(i, a, F): altera em F a prioridade do elemento i para o valor a, exigindo que a nova prioridade seja menor do que a prioridade atual; e
- `ExtractMin`(F): remove e retorna o elemento com a menor prioridade de F.

A prioridade dos elementos é determinada pela função gulosa $g(u, v) = e_d(u, v)$. No Algoritmo 3.2 o campo $Pai(u)$ determina o vértice v pertencente à solução parcial ao qual o nó u conecta-se através da aresta $[u, v] = [u, Pai(u)]$ tal que `RetrieveKey`$(u, F) = e_d(u, v)$. A exceção é o nó r, que determina o início do grafo solução e possui $Pai(r) =$ **null**.

Na linha 1 todos os nós são inicializados no nível de potência zero. Na linha 2 a fila de prioridades F é inicializada como vazia. No laço da linha 3 à 6 a fila de prioridades é preenchida com uma entrada para cada nó $v \in V$ com valor de chave infinito e, como não há nenhuma aresta associada ao valor da chave, o campo $Pai(v)$ é inicializado como **null**. Na linha 7 o nó r é definido como o nó inicial ao ter sua chave reduzida a zero na fila de prioridades F.

A cada iteração do laço da linha 8 à 24 um nó é removido de F, finalizando quando F estiver vazia. Na linha 9 é escolhido o novo nó a ser removido de F. A ligação do novo nó à solução parcial é feita nas linhas 11 e 12 através da alteração das potências de u e $Pai(u)$ (e consequente incorporação da aresta $[u, Pai(u)]$ a $B(p)$).

Algoritmo 3.2 Algoritmo para construção de grafo de comunicação conexo com função gulosa dinâmica – gc(e_d)

Entrada: Conjuntos de nós V e custos $e(u, v), \forall u, v \in V$.
Saída: Potências de transmissão p e grafo de comunicação $G(p) = (V, B(p))$ conexo.

```
1:  p_u ← 0, ∀u ∈ V;
2:  F ← ∅;
3:  para todo v ∈ V faça
4:     Insert(v, ∞, F);
5:     Pai(v) ← null;
6:  fim para
7:  DecreaseKey(r, 0, F);
8:  enquanto F ≠ ∅ faça
9:     u ← ExtractMin(F);
10:    se Pai(u) ≠ null então
11:       p_u ← max{p_u, e(u, Pai(u))} ;
12:       p_Pai(u) ← max{p_Pai(u), e(Pai(u), u)} ;
13:    fim se
14:    para todo s ∈ {u, Pai(u)} faça
15:       se s ≠ null então
16:          para todo v ∈ F adjacente a s faça
17:             se e_d(s, v) < RetrieveKey(v, F) então
18:                DecreaseKey(v, e_d(s, v), F);
19:                Pai(v) ← s;
20:             fim se
21:          fim para
22:       fim se
23:    fim para
24: fim enquanto
25: retorna p;
```

O laço da linha 14 à 23 trata os dois nós que sofreram alteração de potência: u e $Pai(u)$. No laço da linha 16 à 21 atualiza-se a chave dos nós ainda pertencentes à fila de prioridades e adjacentes aos nós que tiveram suas potências alteradas. O Algoritmo 3.2 é similar ao algoritmo de Prim que, usando uma fila de prioridades, possui complexidade $O(|V|^2 \log |V|)$.

3.1.2 Segunda Fase

Nesta fase, encontra-se a atribuição de potências de transmissão $p_u : V \rightarrow R+$ para todo nó $u \in V$, tal que o grafo de comunicação bidirecional resultante $G(p) = (V, B(p))$ seja biconexo. O mesmo algoritmo é utilizado independentemente da função gulosa empregada, estática ou dinâmica, representada genericamente por $g(u, v)$.

O Algoritmo 3.3 inicializa a solução parcial $G(p)$ utilizando as potências de transmissão atribuídas na primeira fase e evolui aumentando as potências (incorporando arestas à solução) até que o grafo de comunicação $G(p)$ seja biconexo. Para testar a biconexidade de $G(p)$ é usado o algoritmo de Tarjan [51], que identifica as componentes biconexas e os pontos de articulação de um grafo não direcionado. O grafo solução é declarado biconexo quando o algoritmo de Tarjan retorna uma única componente biconexa.

Definição 3.3 (grafo biconexo) *Um grafo é biconexo se, para toda tripla de vértices distintos v, w e t, existe um caminho entre v e w que não utiliza t.*

Definição 3.4 (componentes biconexas ou bicomponentes) *As componentes biconexas de um grafo são os sub-grafos biconexos maximais desse grafo.*

Definição 3.5 (pontos de articulação) *Um vértice t é um ponto de articulação de um grafo se existem dois vértices v e w distintos tais que todo caminho entre v e w contém o vértice t.*

O Algoritmo 3.3 começa encontrando a atribuição de potências de transmissão p' na linha 1, por meio da chamada ao Algoritmo 3.1 (se a função gulosa estática for utilizada) ou ao Algoritmo 3.2 (caso contrário). No laço da linha 2 à 4, para todo nó $u \in V$, o custo do arco (u, v) é atribuído à potência p_u tal que, para todo $v \in V$, $e(u, v)$ é máximo e a aresta $[u, v] \in B(p')$.

A linha 5 do Algoritmo 3.3 corresponde à chamada ao algoritmo de Tarjan que identifica, em tempo $O(|B(p)| + |V|)$, se o grafo $G(p) = (V, B(p))$ é um grafo biconexo. A cada iteração do laço da linha 5 à 22 é identificada e incorporada a aresta com menor valor $g(u, v) > 0$ dentre aquelas que conectam dois nós que não são pontos de articulação pertencentes a diferentes bicomponentes. O laço da linha 5 à 22 executa enquanto $G(p)$ não é biconexo.

Na linha 6 do Algoritmo 3.3 o campo $minE$ é inicializado como **null** e será usado para armazenar o nó u cuja aresta $[u, Pai(u)]$ tem menor valor $g(u, Pai(u))$ dentre aque-

Algoritmo 3.3 Algoritmo para construção de grafo de comunicação biconexo baseado em grafo conexo – gbgc

Entrada: Conjuntos de nós V e custos $e(u,v), \forall u,v \in V$.
Saída: Potências de transmissão p e grafo de comunicação $G(p) = (V, B(p))$ biconexo.

```
1: Obter p' pela aplicação do Algoritmo 3.1 ou 3.2;
2: para todo u ∈ V faça
3:    p_u ← max_{v∈V\{u}}{e(u,v) : [u,v] ∈ B(p')} ;
4: fim para
5: enquanto IsNotBiconnected(G(p)) faça
6:    minE ← null;
7:    para todo u ∈ V e NotArticulationPoint(u) faça
8:       para todo v ∈ V adjacente a u tal que NotArticulationPoint(v) faça
9:          se NotSameBicomponent(u,v) então
10:            se minE = null então
11:               minE ← u;
12:               Pai(minE) ← v;
13:            senão se g(u,v) < g(minE, Pai(minE)) então
14:               minE ← u;
15:               Pai(minE) ← v;
16:            fim se
17:         fim se
18:      fim para
19:   fim para
20:   p_minE ← max{p_minE, e(minE, Pai(minE))} ;
21:   p_Pai(minE) ← max{p_Pai(minE), e(Pai(minE), minE)} ;
22: fim enquanto
23: retorna p;
```

las que conectam dois nós que não são pontos de articulação pertencentes a diferentes bicomponentes. O laços das linha 7 à 19 e da linha 8 à 18 percorrem todas as arestas que conectam dois nós que não são pontos de articulação pertencentes a diferentes bicomponentes e identificam a aresta $[u, Pai(u)]$ com menor custo $g(u, Pai(u))$. Nas linhas 20 e 21 é feita a alteração das potências de u e $Pai(u)$ (e a incorporação da aresta $[u, Pai(u)]$).

Seja $\mu \leq |V| - 1$ o número de bicomponentes do grafo conexo $G(p)$ construído pelo Algoritmo 3.1 ou pelo Algoritmo 3.2 (ver Figura 3.2). Como a cada iteração desses algoritmos dois nós que não são pontos de articulação pertencentes a diferentes bicomponentes são ligados por meio de uma aresta, o número de bicomponentes é decrementado de pelo menos uma unidade, assegurando a obtenção de uma solução biconexa em até μ iterações.

Cada chamada da função `isNotBiconnected(`$G(p)$`)` tem complexidade $O(|B(p)| + |V|)$. Ao final da execução da função `isNotBiconnected(`$G(p)$`)`, cada vértice possui uma classificação que indica se ele é ou não um ponto de articulação, fazendo com que o teste

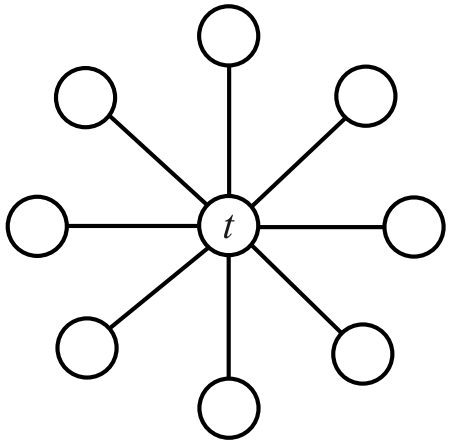


Figura 3.2: Exemplo de grafo conexo $G(p)$ com $\mu = |V| - 1$ bicomponentes. O único ponto de articulação é o nó t que pertence a $|V| - 1$ bicomponentes.

de ponto de articulação `NotArticulationPoint(`u`)` seja realizado em $O(1)$ para qualquer $u \in V$. A função `isNotBiconnected(`$G(p)$`)` também constrói, para cada vértice, um vetor com as bicomponentes às quais o vértice pertence. Como o teste de mesma bicomponente `NotSameBicomponent(`u, v`)` é realizado apenas para vértices que não são pontos de articulação e todo vértice que não é um ponto de articulação pertence a apenas uma bicomponente, a função `NotSameBicomponent(`u, v`)` faz uma única comparação e tem complexidade $O(1)$. Pontos de articulação podem pertencer a até $|V| - 1$ bicomponentes (ver Figura 3.2). Assim, quando aplicada a pontos de articulação, a função `NotSameBicomponent(`u, v`)` faz $O(|V|)$ comparações.

Como em cada iteração é feita uma chamada à função `isNotBiconnected(`$G(p)$`)` e, em seguida, são testados $O(|V|^2)$ arcos para encontrar a aresta de menor custo, o Algoritmo 3.3 tem complexidade $\mu \times O(|B(p)| + |V|) + \mu \times O(|V|^2)$. Como $\mu = O(|V|)$ e $O(|B(p)|) = O(|V|^2)$, então o Algoritmo 3.3 tem complexidade $O(|V|^3)$.

3.2 Algoritmo MST-Augmentation

O Algoritmo 3.4 denominado *MST-Augmentation* foi apresentado em [6]. Ele constrói um grafo de comunicação $G(p)$ biconexo em tempo $O(|V|^2 \log |V|)$ com fator de aproximação 8 quando a entrada são instâncias euclideanas. Instâncias randômicas não foram tratadas em [6], foram utilizadas apenas instâncias euclideanas. O algoritmo MST-Augmentation só foi descrito usando a função gulosa estática.

O Algoritmo 3.4 constrói na linha 1, o conjunto de arestas E cujos pesos são definidos

como $e_e(u,v) = e(u,v) + e(v,u)$, $E = \{[u,v] : u \in V, v \in V, e_e(u,v) = e(u,v) + e(v,u)\}$. Na linha 2, é encontrada a árvore geradora mínima $T(V,E')$, onde $E' \subseteq E$, aplicando-se o algoritmo de Prim. Para garantir uma solução biconexa, uma *árvore local* é construída para todo nó $u \in V$ não folha de T. Chama-se de árvore local de um nó $u \in V$, a uma árvore geradora do conjunto de nós adjacentes a u em T contendo apenas arestas de $E \setminus E'$, que também pode ser obtida pela aplicação do algoritmo de Prim.

Algoritmo 3.4 Algoritmo MST-Augmentation – ma(e_e)

Entrada: Conjuntos de nós V e custos $e(u,v), \forall u,v \in V$.
Saída: Potências de transmissão p e grafo de comunicação $G(p) = (V, B(p))$ biconexo.

1: Construir o conjunto de arestas $E = \{[u,v] : u,v \in V, e_e(u,v) = e(u,v) + e(v,u)\}$;
2: Encontrar a árvore geradora mínima $T = (V, E')$;
3: $p_u \leftarrow 0, \forall u \in V$;
4: **para todo** $u \in V$ **e** `NonLeaf(`u`,`T`)` **faça**
5: $\quad V(u) \leftarrow$ `Neighboors(`u`,`T`)`;
6: $\quad E(u) \leftarrow$ `NeighboorsArcs(`u`,`$V(u)$`,`E`,`E'`)`;
7: $\quad$ Encontrar árvore geradora mínima $T(u) = (V(u), E'(u))$ de $H(u) = (V(u), E(u))$;
8: $\quad E' \leftarrow E' \cup E'(u)$;
9: **fim para**
10: **para todo** $u \in V$ **faça**
11: $\quad p_u \leftarrow \max_{v \in V \setminus \{u\}} \{e(u,v) : [u,v] \in E'\}$;
12: **fim para**
13: **retorna** p;

Na linha 3 a potência de cada nó é inicializada no nível zero. A criação de uma árvore local a um nó não folha $u \in V$ é iniciada na linha 5 com a criação do grafo $H(u) = (V(u), E(u))$, onde $V(u) = \{v : v \in V \setminus \{u\}$ e $[u,v] \in E'\}$ é o conjunto dos nós adjacentes a u em $T = (V, E')$ e $E(u) = \{[s,v] : s,v \in V(u)$ e $[s,v] \in E \setminus E'\}$ é o conjunto de arestas que ligam os nós de $V(u)$ e que não estão presentes em T. A árvore local a u, $T(u) = (V(u), E'(u))$, é calculada aplicando-se o algoritmo de Prim ao grafo $H(u) = (V(u), E(u))$. Na linha 8, as arestas de cada árvore local são incorporadas à árvore original $T = (V, E')$, garantindo um grafo não direcionado biconexo ao final do laço da linha 4 à 9 [6].

No laço da linha 10 à 12, para todo nó $u \in V$ é atribuída à potência p_u o custo do arco (u,v) tal que, para todo $v \in V$, $e(u,v)$ é máximo e a aresta bidirecional $[u,v] \in E'$. Seja $\gamma \leq |V| - 2$ o número inicial de nós não folha (nós com grau maior ou igual a dois) de $T = (V, E')$, O Algoritmo 3.4 possui complexidade $\gamma \times O(|V|^2 \log |V|)$. Como $\gamma = O(|V|)$, então sua complexidade é $O(|V|^3 \log |V|)$.

3.3 Algoritmos Baseados em Componentes Biconexas

Os algoritmos baseados em componentes biconexas constroem o grafo de comunicação biconexo $G(p) = (V, B(p))$ utilizando diretamente as informações fornecidas pelo algoritmo de Tarjan a respeito da estrutura de componentes biconexas da solução parcial. A estrutura de componentes biconexas é computada a cada iteração do algoritmo guloso. A solução começa vazia e a incorporação de arestas é realizada até obter uma solução biconexa, sem a necessidade de uma solução conexa intermediária.

Como o mesmo algoritmo pode ser utilizado tanto para a função gulosa estática como para a função gulosa dinâmica, a função gulosa empregada é representada genericamente por $g(u, v)$, onde $g(u, v) = e_e(u, v)$ ou $g(u, v) = e_d(u, v)$.

Algoritmo 3.5 Algoritmo para construção de grafo de comunicação biconexo baseado em componentes biconexas – gbcb

Entrada: Conjuntos de nós V e custos $e(u, v), \forall u, v \in V$.
Saída: Potências de transmissão p e grafo de comunicação $G(p) = (V, B(p))$ biconexo.

```
1:  p_u ← 0, ∀u ∈ V;
2:  enquanto IsNotBiconnected(G(p)) faça
3:     minE ← null;
4:     para todo u ∈ V faça
5:        para todo v ∈ V adjacente a u faça
6:           se NotSameBicomponent(u,v) então
7:              se minE = null então
8:                 minE ← u;
9:                 Pai(minE) ← v;
10:             senão se g(u,v) < g(minE, Pai(minE)) então
11:                minE ← u;
12:                Pai(minE) ← v;
13:             fim se
14:          fim se
15:       fim para
16:    fim para
17:    p_minE ← max{p_minE, e(minE, Pai(minE))} ;
18:    p_Pai(minE) ← max{p_Pai(minE), e(Pai(minE), minE)} ;
19: fim enquanto
20: retorna p;
```

O Algoritmo 3.5 apresenta o pseudocódigo dessa abordagem. Seu funcionamento é similar ao do Algoritmo 3.3, diferindo pela inicialização de todos os nós no nível de potência zero na linha 1 e pelo conjunto de arestas candidatas a entrar na solução a cada iteração do laço da linha 2 à 19. Enquanto que no Algoritmo 3.3 a aresta de menor custo entre vértices de diferentes bicomponentes que não são pontos de articulação é escolhida a

cada iteração, no Algoritmo 3.5 a aresta de menor custo entre quaisquer vértices (pontos de articulação ou não) de diferentes bicomponentes é escolhida a cada iteração. Como não há um grafo conexo previamente construído como no Algoritmo 3.3, o fato de um vértice ser ou não um ponto de articulação não fornece informação suficiente (nem relevante) sobre a estrutura da solução parcial durante o processo de construção do Algoritmo 3.5.

O Algoritmo 3.5 tem complexidade $O(|V|^4)$, já que o teste realizado na linha 2 é executado $O(|V|^3)$ vezes com complexidade $O(|V|)$ quando considerados os pontos de articulação. Um ponto de articulação pode pertencer, no pior caso, a $|V|-1$ bicomponentes (ver Figura 3.2).

Algoritmo 3.6 Algoritmo para construção de grafo de comunicação biconexo baseado em componentes biconexas aprimorado – gbcba(e_d)

Entrada: Conjuntos de nós V e custos $e(u,v), \forall u,v \in V$.
Saída: Potências de transmissão p e grafo de comunicação $G(p) = (V, B(p))$ biconexo.

1: $p_u \leftarrow 0, \forall u \in V$;
2: **enquanto** `IsNotBiconnected(`$G(p)$`)` **faça**
3: **para todo** $u \in V$ **faça**
4: $Pai(u) \leftarrow$ **null**;
5: **para todo** $v \in V$ adjacente a u **faça**
6: **se** `NotSameBicomponent(`u`,`v`)` **então**
7: **se** $Pai(u) =$ **null então**
8: $Pai(u) \leftarrow v$;
9: **senão se** $e_d(u,v) < e_d(u, Pai(u))$ **então**
10: $Pai(u) \leftarrow v$;
11: **fim se**
12: **fim se**
13: **fim para**
14: **fim para**
15: **para todo** $u \in V$ em ordem crescente de custo $e_d(u, Pai(u))$ **faça**
16: **se** `NotSameBicomponent(`u`,`$Pai(u)$`)` **então**
17: $p_u \leftarrow \max\{p_u, e(u, Pai(u))\}$;
18: $p_{Pai(u)} \leftarrow \max\{p_{Pai(u)}, e(Pai(u), u)\}$;
19: **fim se**
20: **fim para**
21: **fim enquanto**
22: **retorna** $G(p)$;

Quando a função dinâmica é utilizada, a solução construída pelo Algoritmo 3.5 tende a concentrar arestas incidentes aos nós com maior nível de potência. Isto porque quando um nó u tem uma aresta incidente incorporada à solução parcial, todas as demais arestas incidentes a u têm o seu custo dinâmico reduzido. Para evitar que arestas incidentes a um mesmo nó sejam sucessivamente incorporadas, foi desenvolvido o Algoritmo 3.6, onde a aresta de menor custo incidente a cada nó é selecionada no laço da linha 3 à 14.

Em seguida, no laço da linha 15 à 20, as arestas previamente selecionadas no laço da linha 3 à 14 são incorporadas à solução parcial em ordem crescente de custo dinâmico. Quando uma aresta é incorporada à solução parcial, a estrutura de componentes biconexas é alterada, sendo necessário verificar na linha 16 se as arestas previamente selecionadas se mantêm conectando vértices em diferentes bicomponentes. Em caso afirmativo, a aresta é incorporada à solução pela alteração de potência dos nós incidentes nas linhas 17 e 18.

Enquanto que no Algoritmo 3.5 apenas a aresta de menor custo dentre aquelas que conectam nós em diferentes bicomponentes é incorporada à solução a cada iteração, no Algoritmo 3.6, $|V|$ arestas são incorporadas à solução a cada iteração. Ou seja, a cada iteração do Algoritmo 3.6, para todo nó $u \in V$, a aresta de u de menor custo que conecta u a um nó pertencente à bicomponentes diferentes às de u, é incorporada a cada iteração. Essa alteração evita soluções com nós de grau elevado.

Apesar da rotina de atualização das arestas selecionadas a entrar na solução (linha 3 à 14) ser executada um menor número de vezes, a complexidade do Algoritmo 3.6 é a mesma do anterior, ou seja, $O(|V|^4)$.

3.4 Resultados Experimentais dos Algoritmos Gulosos

Os experimentos foram realizados em uma máquina com processador PC Intel Pentium 4 HT com 3,2 GHz de relógio e 1 Gbyte de memória RAM com sistema operacional GNU/Linux 2.6.24. Os algoritmos foram codificados em C++ e o código executável gerado com o compilador GNU g++ versão 4.1, usando o parâmetro de otimização -O2.

Os testes computacionais foram realizados em três classes de instâncias:

1. Instâncias euclideanas EU: geradas como descrito na Seção 2.4, com $|V| \in [25, 400]$.
2. Instâncias euclideanas DE: geradas de maneira similar às instâncias da classe EU, com $|V| \in [25, 400]$ e $e(u, v) = F \cdot d_{uv}^2$. Entretanto, enquanto nas instâncias EU a grade onde se posicionam os nós, independente da sua quantidade, é constante e de lado igual a um (quanto maior a quantidade de nós maior a densidade), nas instâncias DE a dimensão da grade é ajustada para que a densidade seja mantida constante e igual a um nó por unidade quadrada.
3. Instâncias randômicas RD: geradas como descrito na Seção 2.4, com $|V| \in [25, 400]$.

Foram geradas quinze instâncias para cada valor de $|V|$. Em todas as instâncias

geradas, qualquer nó pode se comunicar com todos os demais. Os algoritmos foram executados uma única vez para cada instância e apresentam-se os resultados médios de tempo de execução e potência total obtidos sobre as quinze instâncias para cada valor de $|V|$. Nas tabelas a seguir, são apresentados os resultados obtidos pelos seguintes algoritmos:

- Algoritmo 3.1, que constrói um grafo de comunicação conexo utilizando a função gulosa estática, referenciado por gc(e_e).
- Algoritmo 3.2, que constrói um grafo de comunicação conexo utilizando a função gulosa dinâmica, referenciado por gc(e_d).
- Algoritmo 3.3, que constrói um grafo de comunicação biconexo a partir de um grafo conexo, referenciado por gbgc(e_e) quando utiliza a função gulosa estática e por gbgc(e_d) quando utiliza a função gulosa dinâmica.
- Algoritmo 3.4, que constrói um grafo de comunicação biconexo a partir da composição de árvores geradoras mínimas utilizando a função gulosa estática, referenciado por ma(e_e).
- Algoritmo 3.5, que constrói um grafo de comunicação biconexo baseado em componentes biconexas, sem a prévia construção de um grafo de comunicação conexo, referenciado por gbcb(e_e) quando utiliza a função gulosa estática e por gbcb(e_d) quando utiliza a função gulosa dinâmica.
- Algoritmo 3.6, que constrói um grafo de comunicação biconexo baseado em componentes biconexas e aprimorado para uso da função gulosa dinâmica, referenciado por gbcba(e_d).

Na Tabela 3.1 são apresentados os resultados médios referentes à potência total e ao tempo de execução em segundos para os algoritmos gc(e_e) e gc(e_d). Os resultados mostram que o uso da função gulosa dinâmica traz melhorias no valor da potência total para a construção de um grafo de comunicação conexo para todos os tipos e tamanhos de instâncias. Apesar de terem a mesma complexidade, o algoritmo gc(e_d) leva mais tempo devido à necessidade de atualização dos custos dinâmicos a cada iteração.

Nas Tabelas 3.2 e 3.3 são apresentados, respectivamente, os resultados médios referentes à potência total e ao tempo de execução em segundos para os algoritmos gbgc(e_e), gbgc(e_d), ma(e_e), gbcb(e_e), gbcb(e_d) e gbcba(e_d). A Tabela 3.2 mostra que o algorimo

Instância		gc(e_e)		gc(e_d)	
tipo	|V|	potência	tempo (s)	potência	tempo (s)
RD	25	4,56265	0,00000	**4,23881**	0,00000
	50	7,32598	0,00000	**6,75835**	0,00027
	100	10,13910	0,00080	**9,50633**	0,00053
	200	14,56238	0,00107	**13,67885**	0,00400
	400	20,99737	0,00480	**19,74274**	0,01280
EU	25	0,92496	0,00000	**0,90479**	0,00000
	50	0,90968	0,00000	**0,89382**	0,00053
	100	1,07301	0,00027	**1,05948**	0,00053
	200	1,53879	0,00107	**1,52370**	0,00373
	400	2,59885	0,00533	**2,58128**	0,01280
DE	25	4,33243	0,00000	**4,19420**	0,00000
	50	8,41978	0,00000	**8,12429**	0,00027
	100	16,53229	0,00107	**15,84229**	0,00027
	200	32,45654	0,00107	**31,05434**	0,00320
	400	63,43463	0,00587	**60,77422**	0,01440

Tabela 3.1: Potências e tempos de execução (em segundos) médios obtidos pelos algoritmos gulosos para construção de um grafo de comunicação conexo.

gbgc(e_d) alcança os melhores valores de potência total para todas as classes e tamanhos de instâncias. Entre os algoritmos baseados na estrutura de componentes biconexas (gbcb(e_e), gbcb(e_d) e gbcba(e_d)), o algoritmo aprimorado gbcba(e_d) é aquele que consegue os melhores resultados.

O algorimo ma(e_e) obtém resultados de potência total ruins para a classe de instâncias randômicas, já que foi desenvolvido especificamente para instâncias euclideanas. Para essas instâncias, ele obtém resultados melhores, mas ainda piores do que os algoritmos desenvolvidos nessa tese, pois enquanto que os algoritmos gbgc(e_e), gbgc(e_d), gbcb(e_e), gbcb(e_d) e gbcba(e_d) utilizam as informações dadas pela estrutura de componentes biconexas a cada incorporação de uma nova aresta à solução parcial, o algoritmo ma(e_e) sempre considera todos os nós internos da árvore geradora inicial independente da estrutura de componentes biconexas.

A falta de atualização da estrutura de componentes biconexas pelo algoritmo ma(e_e), à medida em que novas arestas são incorporadas, permite que sejam incorporadas na solução arestas entre nós que passaram a pertencer a uma mesma componente biconexa, aumentando a potência consumida sem contribuir para a conexidade. Em compensação, os tempos de execução de ma(e_e) são menores do que os dos demais algoritmos devido à sua baixa complexidade. Entretanto, para instâncias puramente randômicas, os tempos aumentaram, já que o algorimo foi desenvolvido especificamente para grafos euclideanos.

Instância tipo	|V|	gbgc(e_e)	gbgc(e_d)	ma(e_e)	gbcb(e_e)	gbcb(e_d)	gbcba(e_d)
RD	25	6,33114	**6,15077**	12,84646	6,50817	6,50096	6,18674
	50	9,97445	**9,50228**	24,20324	10,39307	10,42908	9,56347
	100	13,98595	**13,47090**	44,97218	14,47433	14,67736	13,56481
	200	19,86876	**19,20624**	85,08285	20,67766	20,81530	19,31672
	400	28,72704	**27,70172**	158,26410	29,86980	30,09231	27,88118
EU	25	1,62354	**1,60533**	2,02696	1,65702	1,69079	1,65615
	50	1,55381	**1,47477**	2,04944	1,55967	1,59075	1,51612
	100	1,47937	**1,46062**	2,15343	1,53792	1,58895	1,50615
	200	1,87094	**1,84723**	2,84844	1,90962	2,04276	1,86861
	400	2,91245	**2,89559**	4,72278	2,94042	3,21407	2,90519
DE	25	6,01207	**5,87463**	6,83979	6,13895	6,32485	6,15604
	50	11,19187	**10,93232**	13,55728	11,90004	12,34812	11,57423
	100	21,26888	**21,05551**	26,81344	22,83687	23,98314	22,17833
	200	41,16340	**40,44505**	53,03260	44,31606	46,87778	42,59721
	400	80,75201	**78,94255**	102,90890	86,54003	92,18788	83,06382

Tabela 3.2: Potências médias obtidas pelos algoritmos gulosos para construção de um grafo de comunicação biconexo.

Instância tipo	|V|	gbgc(e_e)	gbgc(e_d)	ma(e_e)	gbcb(e_e)	gbcb(e_d)	gbcba(e_d)
RD	25	**0,00000**	0,00027	0,00053	0,00080	0,00080	0,00027
	50	**0,00106**	0,00107	0,00180	0,00213	0,00267	0,00107
	100	**0,00400**	0,00693	0,00907	0,01067	0,01467	0,00907
	200	**0,01840**	0,03787	0,09894	0,06240	0,07894	0,05467
	400	**0,09468**	0,25495	1,09100	0,39869	0,48750	0,34669
EU	25	0,00000	0,00027	**0,00000**	0,00027	0,00160	0,00027
	50	0,00133	0,00133	**0,00027**	0,00347	0,00373	0,00293
	100	0,00427	0,00800	**0,00053**	0,02240	0,02320	0,01893
	200	0,02987	0,05280	**0,00480**	0,13761	0,14934	0,11868
	400	0,18268	0,35815	**0,02987**	0,93873	1,05447	0,79925
DE	25	0,00053	0,00000	**0,00000**	0,00027	0,00027	0,00080
	50	0,00027	0,00053	**0,00053**	0,00320	0,00400	0,00347
	100	0,00427	0,00800	**0,00107**	0,02320	0,02400	0,02000
	200	0,02667	0,05120	**0,00427**	0,17814	0,17121	0,14854
	400	0,19414	0,39629	**0,01707**	1,55530	1,45876	1,30808

Tabela 3.3: Tempos de execução (em segundos) médios obtidos pelos algoritmos gulosos para construção de um grafo de comunicação biconexo.

Na Tabela 3.4 são apresentados os resultados obtidos na primeira e segunda fases do algoritmo baseado na construção de grafos conexos com função gulosa dinâmica (algoritmos gc(e_d) e gbgc(e_d), respectivamente). Esses algoritmos foram os que obtiveram os melhores valores de potência total para grafos de comunicação conexos (primeira fase) e para grafos de comunicação biconexos (segunda fase). Para as 15 instâncias de cada tipo e tamanho são mostradas as características médias das soluções obtidas por cada algoritmo: o número total de arcos unidirecionais, o número total de arestas bidirecionais, o grau (quantidade de arestas incidentes ao nó) médio dos nós, o grau máximo entre todos os nós, a quantidade de nós com grau máximo e o maior nível de potência sobre todos os nós do grafo.

Esses resultados mostram que, independentemente do tipo e tamanho da instância, grafos de comunicação conexos construídos pelo algoritmo gc(e_d) possuem uma quantidade de arestas muito próxima do número mínimo $|V|-1$ de arestas em grafos conexos, de onde se conclui que esses resultados estão muito próximos da solução ótima. Já as soluções biconexas dadas pelo algoritmo gbgc(e_d), possuem um número de arestas em torno de 50% maior do que o mínimo necessário ($|V|$ arestas formando um ciclo), concluindo-se que ainda é possível encontrar soluções com menor número de arestas e com menor valor de potência total. Essa conclusão é reforçada pelo valor do grau médio dos nós que se mostra em torno de $3,0$, enquanto que o mínimo é 2.

Os graus médio e máximo, e a quantidade de nós com grau máximo, também apresentados na Tabela 3.4, mostram que as arestas do grafo de comunicação estão bem distribuídas por todos os nós, sem vértices concentradores. Enquanto o grau médio e a quantidade de nós com grau máximo se mostram independentes do tamanho da instância, o mesmo não ocorre para o grau máximo, que aumenta à medida que o tamanho da instância aumenta, mostrando maior necessidade de concentração em torno de um nó na busca da solução.

		Primeira fase: Algoritmo gc(e_d)						Segunda fase: Algoritmo gbgc(e_d)					
Instância tipo	|V|	Número médio de arcos	Número médio de arestas	Grau médio	Valor médio do grau máximo	Número médio de vértices com grau máximo	Valor médio da potência máxima	Número médio de arcos	Número médio de arestas	Grau médio	Valor médio do grau máximo	Número médio de vértices com grau máximo	Valor médio da potência máxima
RD	25	119,86	24,86	1,98	5,46	1,53	0,44939	171,26	38,46	3,07	7,13	1,40	0,61257
	50	354,13	50,93	2,03	6,86	1,60	0,43074	495,40	77,40	3,09	8,53	1,53	0,48183
	100	963,13	103,73	2,07	8,80	1,33	0,34980	1364,46	157,33	3,14	9,93	1,26	0,41519
	200	2680,80	212,00	2,12	9,40	1,33	0,29063	3806,73	317,53	3,17	11,60	1,53	0,31057
	400	7444,13	423,66	2,11	9,66	1,33	0,19574	10686,26	635,86	3,17	11,73	1,40	0,23622
EU	25	61,13	25,06	2,00	4,00	2,06	0,10855	95,60	37,13	2,97	5,53	1,93	0,18928
	50	124,06	51,40	2,05	4,33	2,53	0,05486	194,86	74,13	2,96	6,00	1,66	0,10797
	100	253,73	102,06	2,04	5,13	2,80	0,03036	379,60	150,53	3,01	6,60	2,40	0,04646
	200	522,73	203,66	2,03	6,06	2,60	0,01587	758,06	292,60	2,92	7,06	2,00	0,02352
	400	1193,86	409,60	2,04	6,93	2,46	0,01121	1691,33	592,06	2,96	8,46	1,66	0,01418
DE	25	63,46	24,80	1,98	4,20	1,53	0,34631	99,20	38,20	3,05	5,46	1,93	0,44654
	50	127,93	50,66	2,02	4,33	2,93	0,36597	197,00	74,20	2,96	6,06	2,00	0,45261
	100	263,40	102,06	2,04	4,73	3,13	0,38379	393,33	149,86	2,99	6,20	2,40	0,47718
	200	528,33	204,93	2,04	5,00	4,13	0,38347	769,66	298,46	2,98	6,53	2,80	0,47956
	400	1057,73	410,73	2,05	5,33	2,60	0,38587	1530,93	589,06	2,94	6,93	3,06	0,49324

Tabela 3.4: Resultados médios obtidos pelos algoritmos gc(e_d) e gbgc(e_d) (respectivamente, a primeira e a segunda fase do algoritmo baseado na construção de grafos conexos com função gulosa dinâmica) sobre as 15 instâncias de cada tipo e tamanho.

Classe de instâncias	Algoritmo	Valor médio da potência total	Número médio de arcos	Número médio de arestas	Grau médio	Valor médio do grau máximo	Número médio de vértices com grau máximo	Valor médio da potência máxima
RD	Exato	5,46654	149,93	31,26	2,50	5,53	1,60	0,55168
	gbgc(e_d)	6,15077	171,26	38,46	3,07	7,13	1,40	0,61257
	Erro (%)	12,52	14,23	23,03	22,80	28,93	-12,50	11,04
EU	Exato	1,37880	83,00	33,00	2,64	4,86	2,20	0,16291
	gbgc(e_d)	1,60533	95,60	37,13	2,97	5,53	1,93	0,18928
	Erro (%)	16,44	15,18	12,52	12,50	13,79	-12,27	16,19
DE	Exato	5,40091	85,26	32,06	2,56	4,73	2,13	0,42737
	gbgc(e_d)	5,87463	99,20	38,20	3,05	5,46	1,93	0,44654
	Erro (%)	8,77	16,35	19,15	19,14	15,43	-9,39	4,49

Tabela 3.5: Comparação entre os resultados médios obtidos pelo algoritmo gbgc(e_d) e as soluções ótimas sobre as 15 instâncias de cada tipo e tamanho para $|V| = 25$.

O valor de potência máxima mostra a maior potência atribuída dentre todos os nós. Com o aumento da densidade dos nós nas instâncias da classe EU à medida que a quantidade de nós aumenta, a tendência é a diminuição da potência máxima devido à proximidade entre os nós. O mesmo não ocorre para a classe DE, pois a densidade dos nós é mantida constante. Para a instância da classe RD, os custos dos arcos permanecem no intervalo $(0, 1]$, independentemente da quantidade de nós. Quanto mais nós, maior a quantidade de arestas incidentes a um nó e maior a possibilidade de se encontrar arestas de baixo custo, fazendo com que a potência máxima diminua à medida que a quantidade de nós aumenta.

A Tabela 3.5 apresenta a comparação entre os resultados médios sobre 15 instâncias de cada tipo e tamanho, fornecendo valores referentes às soluções ótimas obtidas pelo algoritmo exato, valores referentes à melhor solução gulosa obtida pelo algoritmo gbgc(e_d) e o erro relativo percentual entre eles.

Para as instâncias com $|V| = 25$, o valor médio de potência total encontrado pelo algoritmo guloso gbgc(e_d) atinge até 16,44% para as instâncias da classe EU. Esse erro mostra que as soluções gulosas são de boa qualidade, mas que ainda podem ser melhoradas por procedimentos de busca local. Os demais indicadores apresentados na Tabela 3.5 levam à conclusão de que soluções de qualidade tendem a distribuir a potência pelos nós de maneira mais homogênea, evitando uma pequena quantidade de nós com alta potência e grau elevado.

3.5 Algoritmos Gulosos Randomizados

Algoritmos gulosos randomizados são baseados no mesmo princípio dos algoritmos puramente gulosos, diferenciando-se pelo uso de randomização na construção de diferentes soluções em diferentes execuções. Em cada iteração o conjunto de elementos candidatos é formado por todos aqueles que podem ser incorporados à solução parcial em construção. Os elementos candidatos não destroem a viabilidade da solução parcial. Como nos algoritmos puramente gulosos, a escolha do próximo elemento é determinada avaliando-se os candidatos de acordo com a função gulosa utilizada. A avaliação pela função gulosa leva à criação de uma Lista de Candidatos Restrita (LCR) formada pelos melhores elementos, isto é, aqueles cuja incorporação à solução parcial resulta no menor incremento de custo. O próximo elemento a entrar na solução parcial é escolhido aleatoriamente dentre aqueles que se encontram na LCR. Dessa forma, bons elementos, mas não necessariamente o melhor, são incluídos na solução parcial a cada iteração.

Os algoritmos gulosos randomizados propostos nesta seção são baseados nos Algoritmos 3.2, 3.3 e 3.5. Nestes algoritmos, uma *aresta válida* $[u, v]$ com o menor custo $g(u, v)$ é inserida na solução parcial. Uma aresta $[u, v]$ é considerada válida no Algoritmo 3.2 quando conecta um nó u fora da solução parcial a um nó v já pertencente à solução. Uma aresta $[u, v]$ é considerada válida no Algoritmo 3.3 quando u e v não são pontos de articulação e pertencem a diferentes bicomponentes. Já no Algoritmo 3.5, uma aresta $[u, v]$ é considerada válida quando u e v são vértices pertencentes a diferentes bicomponentes.

Os algoritmos gulosos randomizados foram desenvolvidos adicionando-se uma LCR aos Algoritmos 3.1, 3.2, 3.3, 3.5 e 3.6. A LCR é formada por todas as arestas $[u, v]$ válidas tais que

$$g(u, v) \leq g_{min} + \alpha(g_{max} - g_{min}), \tag{3.6}$$

onde $\alpha \in [0, 1]$, $g_{min} = \min\{g(u, v) : u \in V, v \in V \text{ e } [u, v] \text{ é uma aresta válida}\}$ e $g_{max} = \max\{g(u, v) : u \in V, v \in V \text{ e } [u, v] \text{ é uma aresta válida}\}$.

Para ilustrar o desenvolvimento desses algoritmos, apresenta-se a seguir o Algoritmo 3.7 que mostra o pseudocódigo da versão gulosa randomizada do Algoritmo 3.2 e o Algoritmo 3.8 que mostra o pseudocódigo da versão randomizada do Algoritmo 3.3.

O Algoritmo 3.7 corresponde à versão randomizada do Algoritmo 3.2, construindo de forma randomizada um grafo de comunicação conexo $G(p) = (V, B(p))$. Na linha 1, o conjunto de vértices V' pertencentes à solução parcial é inicializado como vazio. Na linha 2, todos os nós são inicializados no nível de potência zero. Na linha 3, a variável

$Pai(u), \forall u \in V$, é inicializada como **null**. O primeiro nó é escolhido aleatoriamente do conjunto V na linha 4 e inserido na solução parcial na linha 5. O laço da linha 6 à 31 é executado até que todos os nós de V sejam inseridos em V'. O laço da linha 9 à 25 atualiza g_{min}, g_{max} e, para todo nó $v \notin V'$, atualiza também a aresta válida de menor custo $[v, s]$ tal que $Pai(v) = s$ e $s \in V'$.

Algoritmo 3.7 Algoritmo guloso randomizado baseado em gc – rgc

Entrada: Conjuntos de nós V, custos $e(u, v), \forall u, v \in V$ e parâmetro α.
Saída: Potências de transmissão p e grafo de comunicação $G(p) = (V, B(p))$ conexo.

```
1: V' ← ∅;
2: p_u ← 0, ∀u ∈ V;
3: Pai(u) ← null, ∀u ∈ V;
4: u ← SelectNodeAtRandom(V);
5: V' ← V' ∪ {u};
6: repetir
7:    g_min ← 0;
8:    g_max ← ∞;
9:    para todo s ∈ {u, Pai(u)} faça
10:      se s ≠ null então
11:         para todo v ∈ V adjacente a s e v ∉ V' faça
12:            se Pai(v) = null então
13:               Pai(v) ← s;
14:            senão se g(v, s) < g(v, Pai(v)) então
15:               Pai(v) ← s;
16:            fim se
17:            se g(v, Pai(v)) < g_min então
18:               g_min ← g(v, Pai(v));
19:            fim se
20:            se g(v, Pai(v)) > g_max então
21:               g_max ← g(v, Pai(v));
22:            fim se
23:         fim para
24:      fim se
25:   fim para
26:   LCR ← {[u, Pai(u)] : u ∉ V' e g(u, Pai(u)) ≤ g_min + α(g_max − g_min)};
27:   [u, Pai(u)] ← SelectEdgeAtRandom(LCR);
28:   p_u ← max{p_u, e(u, Pai(u))} ;
29:   p_Pai(u) ← max{p_Pai(u), e(Pai(u), u)} ;
30:   V' ← V' ∪ {u};
31: até V' = V
32: retorna p;
```

Na linha 26, a LCR é construída como descrito na equação (3.6). A aresta $[u, Pai(u)]$ escolhida aleatoriamente da LCR na linha 27 é incorporada à solução parcial pela alteração das potências dos nós incidentes u e $Pai(u)$ nas linhas 28 e 29, respectivamente. Na

linha 30 o nó u é inserido na solução parcial. O Algoritmo 3.7 possui complexidade $O(|V|^2)$ já que o laço da linha 6 a 31 executa em $O(|V|)$, o laço da linha 9 a 25 executa em $O(1)$ e o laço da linha 11 a 23 em $O(|V|)$.

Algoritmo 3.8 Algoritmo guloso randomizado baseado em gbgc – rgbgc

Entrada: Conjuntos de nós V e custos $e(u,v), \forall u,v \in V$ e parâmetro α.
Saída: Potências de transmissão p e grafo de comunicação $G(p) = (V, B(p))$ biconexo.

1: Obter p' pela aplicação do Algoritmo 3.7;
2: **para todo** $u \in V$ **faça**
3: $p_u \leftarrow \max_{v \in V \setminus \{u\}}\{e(u,v) : [u,v] \in B(p')\}$;
4: **fim para**
5: **enquanto** `IsNotBiconnected`$(G(p))$ **faça**
6: $g_{min} \leftarrow 0$;
7: $g_{max} \leftarrow \infty$;
8: **para todo** $u \in V$ **e** `NotArticulationPoint`(u) **faça**
9: $Pai(u) \leftarrow$ **null**;
10: **para todo** $v \in V$ adjacente a u **e** `NotArticulationPoint`(v) **faça**
11: **se** `NotSameBicomponent`(u,v) **então**
12: **se** $Pai(u) =$ **null então**
13: $Pai(u) \leftarrow v$;
14: **senão se** $g(u,v) < g(u, Pai(u))$ **então**
15: $Pai(u) \leftarrow v$;
16: **fim se**
17: **se** $g(u, Pai(u)) < g_{min}$ **então**
18: $g_{min} \leftarrow g(u, Pai(u))$;
19: **fim se**
20: **se** $g(u, Pai(u)) > g_{max}$ **então**
21: $g_{max} \leftarrow g(u, Pai(u))$;
22: **fim se**
23: **fim se**
24: **fim para**
25: **fim para**
26: $LCR \leftarrow \{[u, Pai(u)] : u \notin V'$ e $g(u, Pai(u)) \leq g_{min} + \alpha(g_{max} - g_{min})\}$;
27: $[u, Pai(u)] \leftarrow$ `SelectEdgeAtRandom`(LCR);
28: $p_u \leftarrow \max\{p_u, e(u, Pai(u))\}$;
29: $p_{Pai(u)} \leftarrow \max\{p_{Pai(u)}, e(Pai(u), u)\}$;
30: **fim enquanto**
31: **retorna** p;

De forma análoga, o Algoritmo 3.8 é a versão randomizada do Algoritmo 3.3. O Algoritmo 3.8 possui complexidade $O(|V|^3)$ como o Algoritmo 3.3.

3.5.1 Ajuste do Parâmetro α

O parâmetro α da inequação (3.6) é um valor escolhido no intervalo $[0, 1]$. Se $\alpha = 0$, então o critério de seleção dos elementos da LCR é puramente guloso. Se $\alpha = 1$, a escolha dos elementos da LCR é estritamente aleatória.

Prais e Ribeiro [39] mostraram que a utilização de um único valor fixo para α frequentemente dificulta encontrar soluções de boa qualidade que podem ser encontradas mais facilmente quando outros valores são utilizados. Eles propuseram uma estratégia reativa para ajuste automático do parâmetro α.

Nessa estratégia reativa, o parâmetro α é selecionado dentro de um conjunto discreto de valores $\mathcal{F} = \{\alpha_1, \alpha_2, \ldots, \alpha_\tau\}$. Probabilidades P_i são associadas a cada α_i com valores iniciais iguais a $P_i = 1/\tau$ $(i = 1, \ldots, \tau)$. Em seguida, a distribuição de probabilidade é atualizada periodicamente a cada bloco de um certo número pré-fixado de B iterações realizadas, de acordo com a seguinte equação

$$P_i = q_i / \sum_{j=i}^{j=\tau} q_j \tag{3.7}$$

onde

$$q_i = (1/M_i)^\delta. \tag{3.8}$$

O valor M_i é a média das soluções encontradas utilizando o parâmetro α com o valor α_i. O expoente $\delta > 0$ é utilizado para atenuar a atualização dos valores das probabilidades P_i. Observa-se que quanto maior for o valor de q_i, maior será a probabilidade associada ao valor α_i. Consequentemente, mais chances este α_i terá de ser escolhido.

Os Algoritmos gulosos 3.1, 3.2, 3.3, 3.5 e 3.6, com função gulosa estática ou dinâmica apresentados nesse capítulo foram randomizados e testados utilizando vários tipos e tamanhos de instâncias. Como todos os algoritmos randomizados tiveram comportamento semelhante em termos das diferentes estratégias de atribuição do valor de α, apresenta-se na Tabela 3.6 apenas os resultados referentes ao Algoritmo 3.8 (Algoritmo 3.3 randomizado), que é aquele que obteve o melhor desempenho.

A Tabela 3.6 apresenta o histograma com os resultados de 100000 execuções do Algoritmo 3.8 para a instância RD100. Na primeira coluna dessa tabela são mostrados os intervalos de ocorrência dos valores das soluções. Em cada uma das colunas seguintes, apresenta-se para cada valor fixo de α, a quantidade de vezes em que o valor da solução

RD100	algoritmo rgbgc(e_d) – função gulosa dinâmica							
intervalo	$\alpha = 0$	$\alpha = 0,1$	$\alpha = 0,2$	$\alpha = 0,4$	$\alpha = 0,6$	$\alpha = 0,8$	$\alpha = 1,0$	reativo
(12, 56, 12, 63]							6	3
(12, 63, 12, 75]						1	27	14
(12, 75, 12, 86]				26	11	19	110	110
(12, 86, 12, 98]			6789	2325	633	170	383	1814
(12, 98, 13, 10]			5381	4346	4322	1205	1088	3939
(13, 10, 13, 22]			19506	12286	13677	4878	2753	10847
(13, 22, 13, 34]		16685	35055	28377	19003	11218	5572	19834
(13, 34, 13, 45]	100000	36902	9170	19775	24631	16565	9846	24151
(13, 45, 13, 57]		46413	17435	19879	21738	19552	13992	17512
(13, 57, 13, 69]			4787	8587	10977	18928	17149	9917
(13, 69, 13, 81]			1877	4054	3956	14168	17167	6111
(13, 81, 13, 92]				335	913	8200	13692	3124
(13, 92, 14, 04]				10	129	3533	9496	1593
(14, 04, 14, 16]					10	1248	5147	660
(14, 16, 14, 28]						264	2352	256
(14, 28, 14, 39]						46	855	86
(14, 39, 14, 51]						5	271	24
(14, 51, 14, 63]							75	4
(14, 63, 14, 75]							16	1
(14, 75, 14, 86]							3	
Diferença (%)	0,00	-1,52	-3,99	-4,68	-4,77	-5,39	-6,46	-6,28
Potência mínima	13,4508	13,2460	12,9134	12,8206	12,8084	12,7254	12,5807	12,6065
Tempo (s)	0,012	0,012	0,013	0,012	0,013	0,012	0,012	0,018

Tabela 3.6: Histograma com os resultados de 100000 execuções do algoritmo rgbgc(e_d) com função gulosa de avaliação dinâmica.

ocorre em cada intervalo. A última coluna apresenta a mesma informação para a estratégia reativa (parâmetros $B = 100$ e $\delta = 8$). As potências mínimas obtidas como o melhor resultado sobre todas as 100000 execuções para cada estratégia referente ao valor de α são apresentadas a seguir. Apresenta-se ainda, na linha diferença, as diferenças percentuais entre o valor do melhor resultado encontrado por cada estratégia de α em relação ao valor da solução gulosa encontrada para $\alpha = 0$, calculado como $100 \times (X - Y)/Y$, onde X é o valor da potência mínima obtida como o melhor resultado sobre todas as 100000 execuções para cada estratégia de α e Y o valor da potência mínima obtida quando α fixo em zero. Na última linha são mostrados os tempos médios de execução em segundos das 100000 execuções para cada estratégia.

O comportamento dos algoritmos randomizados para α fixo corresponde ao que foi descrito por Feo e Resende [15] sobre o efeito do valor de α na qualidade da solução e na diversidade das soluções geradas durante a fase de construção. Valores de α que levam a uma lista de candidatos restrita de tamanho muito limitado (ou seja, valores de α muito próximos 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 α próximo de 1, leva a uma grande diversidade de soluções geradas. Entretanto, muitas destas soluções terão qualidade inferior.

A utilização da estratégia reativa produz o mesmo comportamento descrito em [39],

onde os resultados mostram que o procedimento reativo pode ser usado para alcançar os melhores resultados em lugar de um processo de calibração exaustivo, com a vantagem adicional de gerar diversificação ao permitir a utilização de valores variados do parâmetro α. Assim sendo, o Algoritmo 3.8 com estratégia reativa será utilizado na fase construtiva do GRASP conforme descrito no próximo capítulo.

3.6 Conclusão

Neste capítulo foram implementados seis algoritmos gulosos para CMP2C-AB, sendo que cinco foram originalmente propostos nesta tese. Os algoritmos foram classificados conforme a função gulosa utilizada, estática ou dinâmica, e conforme sejam baseados na construção inicial de grafos conexos ou em componentes biconexas.

Devido à falta de instâncias disponíveis na literatura, todos os algoritmos foram aplicados a instâncias puramente randômicas e instâncias randômicas geométricas geradas para esta tese. Incluiu-se na comparação, um algoritmo da literatura originalmente desenvolvido para instâncias geométricas para melhor avaliar o desempenho dos algoritmos aqui propostos.

Os experimentos mostraram que os algoritmos gulosos baseados na construção de grafos conexos (Algoritmos 3.2 e 3.3) obtêm melhores resultados (menores valores de potências) quando é utilizada a função gulosa dinâmica. Para os algoritmos baseados em componentes biconexas (algoritmos 3.5 e 3.6), a utilização da função gulosa dinâmica obtêm melhores resultados quando a potência de vários nós é aumentada a cada iteração, evitando a construção de nós de grau elevado durante o processo de construção (ver Tabela 3.2). Quando a potência de apenas um nó é aumentada a cada iteração, o valor da função gulosa dinâmica para os demais arcos deste nó é reduzido, provocando aumentos sucessivos na potência de apenas um nó.

Os experimentos mostraram ainda que, mesmo restringindo-se a construção de nós de grau elevado nos algoritmos baseados em componentes biconexas, os algoritmos baseados na construção de grafos conexos, além de serem mais rápidos, constroem soluções biconexas com menores valores de potência total. Assim, esses algoritmos serão usados sempre que uma solução gulosa for exigida.

A análise das características das soluções construídas pelo melhor algoritmo mostrou que as soluções dos algoritmos gulosos são de boa qualidade, mas que ainda podem ser melhoradas por procedimentos de busca local, como será mostrado no capítulo seguinte.

Todos os algoritmos gulosos propostos neste capítulo superaram os resultados do algoritmo da literatura (Algoritmo 3.4) que utiliza a função gulosa estática e não leva em consideração a estrutura de componentes biconexas da solução parcial.

Também foram desenvolvidas e testadas neste capítulo as versões randomizadas dos algoritmos gulosos. As versões baseadas na construção de grafos conexos (Algoritmos 3.7 e 3.8) utilizando a estratégia reativa, obtiveram os melhores resultados entre todos os algoritmos randomizados. Esses algoritmos serão utilizados na fase construtiva do GRASP a ser desenvolvido no próximo capítulo.

Capítulo 4

Heurísticas GRASP para CMP2C-AB

Nesse capítulo, serão desenvolvidas heurísticas baseadas em metaheurísticas para CMP2C-AB, problema que reflete melhor as necessidades de uma rede ad hoc.

4.1 Vizinhanças e Buscas Locais

Dado o espaço de soluções S de um problema, a vizinhança $\mathcal{N}(s)$ de uma solução $s \in S$ é definida como um conjunto de soluções $N(s) \subseteq S$. Uma solução $s' \in N(s)$ pode ser alcançada pela aplicação de modificações elementares, chamadas de movimentos, em s. Os procedimentos que exploram sistematicamente uma vizinhança $\mathcal{N}(s)$ são denominados de buscal local. Começando de uma solução inicial, a busca local explora a vizinhança na procura de uma solução melhor do que a corrente. A solução corrente muda sucessivamente, até chegar-se a um ótimo local.

Uma busca local é do tipo *melhor aprimorante* se percorre sempre toda a vizinhança de uma solução e a solução corrente sempre muda para a melhor solução vizinha. Uma busca local é do tipo *primeiro aprimorante* se a solução corrente muda para a primeira solução vizinha melhor do que ela.

Três vizinhanças e diferentes estratégias de busca local nelas baseadas são propostas a seguir. Para a implementação das vizinhanças e das buscas locais, serão utilizadas duas operações básicas para o decremento e para o incremento da potência de um nó qualquer. Elas se caracterizam pela quantidade de nós afetados em uma única operação e pelo ajuste do nível de potência dos nós afetados.

Para a descrição das duas operações básicas, utiliza-se a lista ordenada $P_i = [p_i^1, \ldots, p_i^{\phi(i)}]$ e os conjuntos S_i^ℓ definidos na Seção 2.2 e os conjuntos T_i^ℓ definidos na Seção 2.3, para cada nó $i \in V$ e para cada nível $\ell = 1, \ldots, \phi(i)$. A lista P_i contém, em ordem crescente,

todos os níveis de potência que o nó i pode assumir. O conjunto S_i^ℓ contém todos os vértices alcançados com o nível de potência p_i^ℓ, enquanto que $T_i^\ell = S_i^\ell - S_i^{\ell-1}$ é o conjunto de vértices alcançados com o acréscimo de potência de $p_i^{\ell-1}$ para p_i^ℓ, para todo $\ell = 2, \ldots, \phi(i)$. Para $\ell = 1$ tem-se $T_i^1 = S_i^1$. Como toda busca local começa em uma solução viável do problema, assume-se que a potência p_i atribuída a cada nó $i \in V$ é tal que $p_i = p_i^\ell$, para algum $\ell \in \{1, \ldots, \phi(i)\}$.

Define-se também a lista $L_i = [l_i^1, \ldots, l_i^{\phi(i)}]$, para todo nó $i \in V$, com $l_i^\ell \in \{0, 1, 2\}$ para $\ell \in \{1, \ldots, \phi(i)\}$:

caso (a) $l_i^\ell = 0$ se o nível de potência p_i^ℓ é maior que a potência corrente p_i do nó i, $p_i^\ell > p_i$, ou seja, o nó i não utiliza o nível de potência p_i^ℓ para estabelecer comunicação,

caso (b) $l_i^\ell = 2$ se $p_i^\ell \leq p_i$ e existe um nó $j \in T_i^\ell$ e um nível $\kappa = 1, ..., \phi(j)$ tal que $p_j^\kappa \leq p_j$, ou seja, o nível de potência p_i^ℓ suporta a aresta $[i, j]$,

caso (c) $l_i^\ell = 1$ se o nível de potência p_i^ℓ é utilizado para estabelecer comunicação de i para um nó $j \in V$, mas não existe o arco de j para i, ou seja, o nível de potência p_i^ℓ suporta o arco (i, j).

Nos dois últimos casos, o nível de potência p_i^ℓ é utilizado por i no estabelecimento de comunicação. No caso $l_i^\ell = 1$ a ligação estabelecida é um arco enquanto que no caso $l_i^\ell = 2$, a ligação estabelecida é uma aresta como mostra a Figura 4.1.

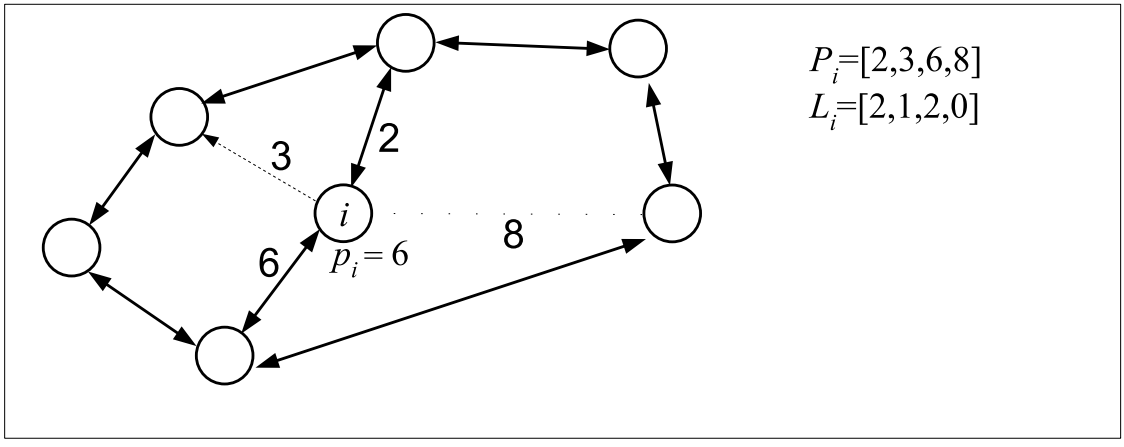


Figura 4.1: Exemplo dos conjuntos P_i e L_i para uma solução viável.

As duas operações básicas de decremento e incremento de potência são definidas como:

1. `decremento`(i, p): dadas as potências de transmissão p e o nó i, decrementar seu nível de potência de $p_i = p_i^\ell$ (com $\ell \geq 2$) para $p_i^{\ell'}$, onde ℓ' é o mais alto nível de

potência menor que ℓ a suportar uma aresta: $1 \leq \ell' < \ell$, com $l_i^{\ell'} = 2$ e $l_i^{\ell''} = 1$ para todo $\ell'' = \ell' + 1, \ldots, \ell - 1$. São removidas todos os arcos e arestas entre i e j para todo $j \in T_i^{\ell'+1} \cup \ldots \cup T_i^{\ell-1} \cup T_i^{\ell}$ e a potência total da solução é reduzida em $p_i^{\ell} - p_i^{\ell'}$ (ver Figura 4.2(b)).

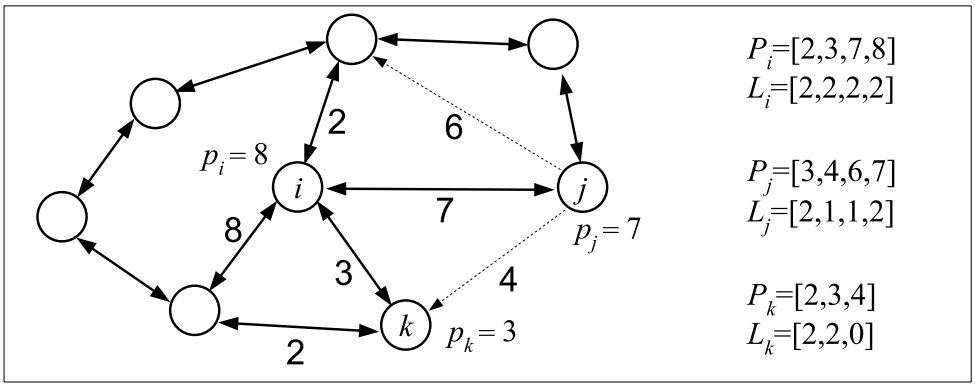
2. `incremento`(i, p): dadas as potências de transmissão p e o nó i, incrementar sua potência de $p_i = p_i^{\ell}$ para $p_i^{\ell+1}$, com $\ell + 1 \leq \phi(i)$. Para o nó $j \in T_i^{\ell+1}$, com $p_j^{\nu'} - p_j^{\nu} = \min_{j \in T_i^{\ell+1}} \{p_j^{\nu'} - p_j^{\nu} : i \in T_j^{\nu'}\}$, incrementar sua potência de $p_j = p_j^{\nu}$ para $p_j^{\nu'}$. A potência total é aumentada de $p_i^{\ell+1} - p_i^{\ell}$ mais $p_j^{\nu'} - p_j^{\nu}$ se $p_j^{\nu} < p_j^{\nu'}$, caso contrário não há acréscimo devido a j (ver Figura 4.2(c)).

A operação `decremento`(i, p) garante a remoção de pelo menos um arco a cada chamada. A operação `incremento`(i, p) garante que pelo menos uma aresta é incluída na solução corrente a cada chamada da operação. A inclusão de arestas se mostra mais eficaz que a inclusão de arcos, pois somente a inclusão de uma aresta pode alterar a estrutura de componentes biconexas de uma solução inviável contribuindo, assim, para a recuperação da sua viabilidade (biconexidade).

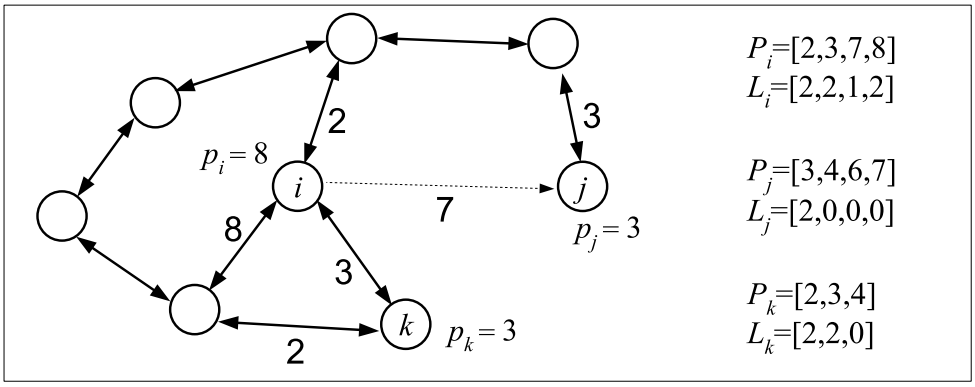
Um movimento completo (ver Figura 4.2) consiste em aplicar à solução corrente viável sucessivas operações de decremento até provocar a quebra da sua viabilidade, criando uma solução vizinha inviável. Em seguida, sucessivas operações de incremento são aplicadas à essa solução vizinha inviável até que ela se torne viável ou até que seu custo seja maior que o custo da solução corrente.

Movimentos onde a operação de incremento é realizada primeiramente são pouco atraentes, pois possuem alto custo computacional. Fazendo-se primeiramente a operação de incremento, não há a quebra de viabilidade e todas as operações de decremento precisariam ser realizadas para diminuir a potência total e verificar se há ou não a perda da biconexidade. Por outro lado, fazendo-se primeiramente operações de decremento que destroem a viabilidade da solução corrente, tem-se a estrutura de componentes biconexas disponível utilizando o algoritmo de Tarjan. Essa informação pode ser utilizada para diminuir as operações de incremento candidatas que efetivamente reconstruirão a viabilidade e deverão ser testadas como, por exemplo, não testar a inserção de arestas entre nós de uma mesma bicomponente pois essas operações não recuperam a biconexidade. Diminuir as operações de incremento candidatas reduz substancialmente o espaço de busca.

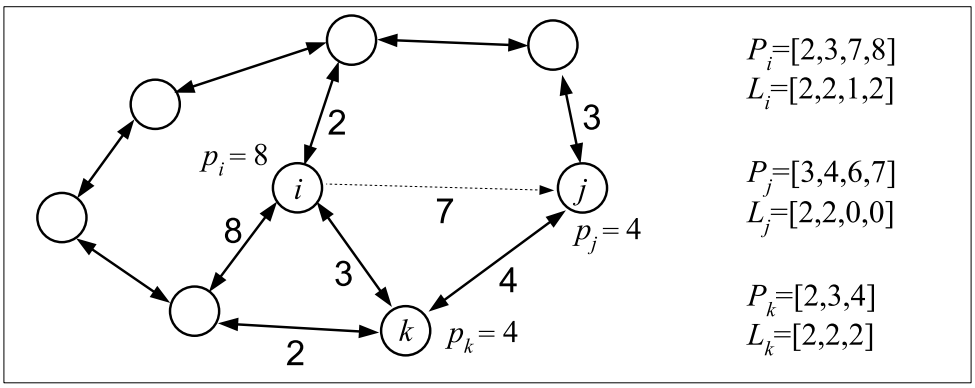
Três estratégias de exploração das vizinhanças são propostas de acordo com o conjunto de operações de incremento candidatas:



(a) Solução corrente bidirecional biconexa.



(b) Operação `decremento`(j, p) gera solução inviável.



(c) Operação `incremento`(k, p) gera solução vizinha aprimorante viável.

Figura 4.2: Exemplo de um movimento completo.

- **vizinhança reduzida**: após a quebra da biconexidade pela última operação de decremento, sucessivas operações de incremento são realizadas considerando apenas as arestas entre nós de diferentes bicomponentes, tal que o par de nós testado pertence às mesmas bicomponentes dos nós afetados pela última operação de decremento,
- **vizinhança estendida**: após a quebra da biconexidade pela última operação de decremento, sucessivas operações de incremento são realizadas considerando as arestas entre nós de diferentes bicomponentes,
- **vizinhança duplamente estendida**: após a quebra da biconexidade pela última operação de decremento, mais uma operação de decremento é aplicada à solução corrente e, em seguida, sucessivas operações de incremento são realizadas considerando as arestas entre nós de diferentes bicomponentes.

As vizinhanças reduzida, estendida e duplamente estendida, podem ser ordenadas dessa forma pelo tamanho do espaço de busca. Maiores espaços de busca, como por exemplo mais duas operações de decremento sucessivas após a quebra da biconexidade, tornariam a busca muito lenta e não foram implementados.

Os procedimentos de busca local começam em uma solução viável do CMP2C-AB e buscam uma solução vizinha viável utilizando a estratégia primeiro aprimorante. A busca local pára quanto é encontrado um ótimo local.

A busca local para as vizinhanças reduzida e estendida é descrita pelo Algoritmo 4.1. Uma atribuição de potências p, tal que o grafo de comunicação $G(p) = (V, B(p))$ é biconexo, é passada para o procedimento de busca local. Esse algoritmo retorna as atribuições de potência de transmissão p' que determinam uma solução melhor ou a mesma solução, caso não encontre uma melhor na vizinhança empregada.

O laço da linha 1 à 39 do Algoritmo 4.1 executa até que um ótimo local seja encontrado, isto é, enquanto houver melhoria da solução corrente. O laço da linha 3 à 18 efetua a diminuição das potências da solução corrente. A diminuição da potência da solução p é realizada através de chamadas à operação de decremento e, para cada decremento de potência, testa-se a viabilidade do grafo $G(p)$. O fim do laço da linha 3 à 18 é controlado pela variável *todas_inviaveis*. O laço pára quando, em uma iteração, todas as possíveis operações de decremento sobre p resultarem em um grafo de comunicação $G(p)$ inviável. Na linha 4, a variável *todas_inviaveis* recebe um valor verdadeiro que será alterado para falso quando alguma solução viável aprimorante for encontrada.

Na linha 5 é feito o cálculo das operações de decremento $C_{dec}(u) \geq 0$, para todo $u \in V$.

Algoritmo 4.1 Algoritmo de busca local para vizinhanças reduzida e estendida

Entrada: Potências de transmissão p tal que o grafo de comunicação $G(p) = (V, B(p))$ é biconexo.

Saída: Potências de transmissão p' e grafo de comunicação $G(p') = (V, B(p'))$ biconexo tais que $\sum_{u \in V} p'_u \leq \sum_{u \in V} p_u$.

1: **repetir**
2: $otimo\_local \leftarrow verdadeiro$;
3: **repetir**
4: $todas\_inviaveis \leftarrow verdadeiro$;
5: calcula valor $C_{dec}(u)$ da operação `decremento(`u, p`)`, $\forall u \in V$;
6: gera lista $C_{dec} = [C_{dec}(u) : u \in V]$ em ordem decrescente;
7: **enquanto** $C_{dec} \neq \emptyset$ **faça**
8: $p' \leftarrow p$;
9: altera potências p de acordo com a primeira operação `decremento(`u, p`)` da lista C_{dec};
10: remove primeira operação `decremento(`u, p`)` da lista C_{dec};
11: **se** $G(p)$ não é biconexo **então**
12: $p \leftarrow p'$;
13: **senão**
14: $otimo\_local \leftarrow falso$;
15: $todas\_inviaveis \leftarrow falso$;
16: **fim se**
17: **fim enquanto**
18: **até** $todas\_inviaveis$
19: $p' \leftarrow p$;
20: calcula valor $C_{dec}(u)$ da operação `decremento(`u, p`)`, $\forall u \in V$;
21: gera lista $C_{dec} = [C_{dec}(u) : u \in V]$ em ordem decrescente;
22: **enquanto** $C_{dec} \neq \emptyset$ **faça**
23: altera potências p de acordo com a primeira operação `decremento(`u, p`)` da lista C_{dec};
24: remove primeira operação `decremento(`u, p`)` da lista C_{dec};
25: calcula valor $C_{inc}(u)$ da operação `incremento(`u, p`)`, $\forall u \in V$;
26: gera lista $C_{inc} = [C_{inc}(u) : u \in V]$ em ordem crescente;
27: computa estrutura de componentes biconexas e elimina da lista C_{inc} as operações de incremento de acordo com a estratégia de exploração da vizinhança;
28: **repetir**
29: altera potências p de acordo com a primeira operação `incremento(`u, p`)` da lista C_{inc};
30: remove primeira operação `incremento(`u, p`)` da lista C_{inc};
31: **até** $\sum_{u \in V} p_u > \sum_{u \in V} p'_u$ **ou** viabilidade é re-estabelecida
32: **se** $\sum_{u \in V} p_u \leq \sum_{u \in V} p'_u$ **então**
33: $p' \leftarrow p$;
34: $otimo\_local \leftarrow falso$;
35: **senão**
36: $p \leftarrow p'$;
37: **fim se**
38: **fim enquanto**
39: **até** $otimo\_local$
40: **retorna** p';

O cálculo de uma operação de decremento (incremento) gera apenas o valor de potência a ser subtraído (somado) à potência total da solução sem que a solução seja alterada. Na linha 6, cria-se uma lista ordenada $C_{dec} = [C_{dec}(u) : u \in V]$ decrescentemente (maior valor a ser subtraído primeiro).

No laço da linha 7 à 17, as potências dos nós serão diminuídas de acordo com as operações de decremento da lista C_{dec} ordenada. Como as potências são alteradas, armazena-se a atribuição de potências p em p' na linha 8. Na linha 9, a atribuição de potências p é alterada de acordo com a primeira operação de decremento da lista C_{dec}. A primeira operação é então removida da lista na linha 10. Se o grafo de comunicação $G(p)$ for biconexo, uma solução aprimorante viável foi encontrada e as variáveis *otimo_local* e *todas_inviaveis* são modificadas, respectivamente, nas linhas 14 e 15, para que a busca local e o laço da linha 3 à 18 continuem. Caso contrário, na linha 12, atribui-se a p a solução viável armazenada em p'. Em ambos os casos, novo decremento é realizado sobre p até o fim da lista C_{dec}.

A linha 5 tem complexidade $O(|V|)$ e a lista C_{dec} de tamanho $|V|$ é ordenada em $O(|V| \log |V|)$ na linha 6. Como o teste de viabilidade é feito em $O(|V| + |B(p)|)$, a linha 9 tem complexidade $O(|V|) \times O(|V| + |B(p)|) = O(|V|^2)$.

Após o término do laço da linha 3 à 18, na linha 19, p' é utilizado mais uma vez para armazenar a melhor solução viável p. Agora, qualquer operação de decremento sobre a solução dada por p, destruirá a viabilidade do grafo de comunicação $G(p)$. Na linha 20 as operações de decremento são calculadas em tempo $O(|V|)$ e ordenadas na lista C_{dec} na linha 21 em $O(|V| \log |V|)$.

O laço da linha 22 à 38 percorre toda a lista ordenada C_{dec} de operações de decremento começando da mais aprimorante. Na linha 23, altera-se a atribuição de potências p de acordo com a primeira operação de decremento da lista C_{dec}, removida em seguida. Na linha 25, é feito o cálculo das operações de incremento $C_{inc}(u) \geq 0$, para todo $u \in V$. Na linha 26 cria-se uma lista ordenada $C_{inc} = [C_{inc}(u) : u \in V]$ crescentemente (menor valor a ser somado primeiro).

A estrutura de componentes biconexas é construída na linha 27 e as operações de incremento são eliminadas da lista C_{inc} de acordo com a estratégia de exploração da vizinhança sendo utilizada: vizinhança reduzida, que considera operações de incremento realizadas entre nós de diferentes bicomponentes e pertencentes às mesmas bicomponentes dos nós afetados pela operação de decremento prévia, ou vizinhança estendida, que considera operações de incremento realizadas entre nós de diferentes bicomponentes.

Algoritmo 4.2 Algoritmo de busca local para vizinhança duplamente estendida

Entrada: Potências de transmissão p tal que o grafo de comunicação $G(p) = (V, B(p))$ é biconexo.

Saída: Potências de transmissão p' e grafo de comunicação $G(p') = (V, B(p'))$ biconexo tais que $\sum_{u \in V} p'_u \leq \sum_{u \in V} p_u$.

repetir
  $otimo\_local \leftarrow verdadeiro$;
  repetir
    $todas\_inviaveis \leftarrow verdadeiro$;
    calcula valor $C_{dec}(u)$ da operação `decremento(`u, p`)`, $\forall u \in V$;
    gera lista $C_{dec} = [C_{dec}(u) : u \in V]$ em ordem decrescente;
    enquanto $C_{dec} \neq \emptyset$ **faça**
      $p' \leftarrow p$;
      altera potências p de acordo com a primeira operação `decremento(`u, p`)` da lista C_{dec};
      remove primeira operação `decremento(`u, p`)` da lista C_{dec};
      se $G(p)$ não é biconexo **então**
        $p \leftarrow p'$;
      senão
        $otimo\_local \leftarrow falso$;
        $todas\_inviaveis \leftarrow falso$;
      fim se
    fim enquanto
  até $todas\_inviaveis$
  $p' \leftarrow p$;
  calcula valor $C_{dec}(u)$ da operação `decremento(`u, p`)`, $\forall u \in V$;
  gera lista $C_{dec} = [C_{dec}(u) + C_{dec}(v) : u, v \in V, u \neq v]$ em ordem decrescente;
  enquanto $C_{dec} \neq \emptyset$ **faça**
    altera potências p de acordo com o primeiro par de operações `decremento(`u, p`)` e `decremento(`v, p`)` da lista C_{dec};
    remove o primeiro par de operações `decremento(`u, p`)` e `decremento(`v, p`)` da lista C_{dec};
    calcula valor $C_{inc}(u)$ da operação `incremento(`u, p`)`, $\forall u \in V$;
    gera lista $C_{inc} = [C_{inc}(u) : u \in V]$ em ordem crescente;
    computa estrutura de componentes biconexas e elimina da lista C_{inc} as operações de incremento de acordo com a estratégia de exploração da vizinhança;
    repetir
      altera potências p de acordo com a primeira operação `incremento(`u, p`)` da lista C_{inc};
      remove primeira operação `incremento(`u, p`)` da lista C_{inc};
    até $\sum_{u \in V} p_u > \sum_{u \in V} p'_u$ **ou** viabilidade é re-estabelecida
    se $\sum_{u \in V} p_u \leq \sum_{u \in V} p'_u$ **então**
      $p' \leftarrow p$;
      $otimo\_local \leftarrow falso$;
    senão
      $p \leftarrow p'$;
    fim se
  fim enquanto
até $otimo\_local$
retorna p';

No laço da linha 28 à 31, as operações de incremento são efetuadas na ordem da lista C_{inc}. As operações de incremento se sucedem no laço da linha 28 a 31 até que a solução p tenha valor da função objetivo maior que a melhor solução viável p' ou até que a viabilidade seja re-estabelecida gerando uma solução vizinha viável com valor da função objetivo menor. Nesse caso, atualiza-se a melhor solução viável p' na linha 33, caso contrário, recarrega-se a solução auxiliar com a solução corrente na linha 36. Em ambos os casos, uma nova iteração do laço da linha 22 à 38 é realizada até o esvaziamento da lista C_{dec}.

A linha 23 tem complexidade $O(1)$. A linha 25 tem complexidade $O(|V|)$ já que o número máximo de incrementos é $|V|$. A lista C_{inc} de tamanho $|C_{inc}| = O(|V|)$ é ordenada em $O(|V| \log |V|)$ na linha 26. Na linha 27, a estrutura de componentes biconexas é construída em $O(|V| + |B(p)|)$ e a verificação dos critérios da estratégia de exploração da vizinhança empregada é realizada em $O(1)$ A efetivação dos movimentos é feita em $|C_{inc}| \times O(|V| + B(p)) = O(|V|^2)$ na linha 29.

O Algoritmo 4.2 apresenta as mudanças no Algoritmo 4.1 para implementação da vizinhança duplamente estendida. Nesse caso, o procedimento de destruição da viabilidade utiliza duas operações de decremento sucessivas (linha 23) sobre a solução p. Em seguida, entra no laço de restauração da viabilidade da solução auxiliar através de sucessivas operações de incremento em busca de uma solução vizinha melhor que a solução corrente. Enquanto que o tempo de processamento para percorrer uma vizinhança reduzida ou estendida é $O(|V|^2)$, para a vizinhança duplamente estendida tem-se complexidade $O(|V|^3)$, pois para cada par de operações de decremento, $O(|V|)$ operações de incremento são efetivadas. Todos os $O(|V||V|-1)$ pares de operações de decremento são testados.

Cada procedimento de busca local proposto utiliza uma única estratégia de exploração: vizinhança reduzida, vizinhança estendida ou vizinhança duplamente estendida.

4.2 Heurísticas GRASP

A metaheurística GRASP (*Greedy Randomized Adaptative Search Procedure*) foi proposta em [14]. Trata-se de um processo iterativo, onde cada iteração é independente das demais e consiste de duas fases: a construtiva, na qual uma solução viável é gerada, e a de busca local, na qual procura-se um ótimo local a partir da solução construída. A melhor solução global é guardada como resultado. A metaheurística GRASP tem sido usada para obter soluções de qualidade para muitos problemas de otimização combinatória [16].

O Algoritmo 4.3 ilustra um procedimento GRASP padrão para a resolução do problema de minimização da função $f(p) = \sum_{u \in V} p_u$. Na linha 1 o melhor valor corrente da função objetivo é inicializado. O laço da linha 2 à 9 é repetido até que *Criterio_Parada* seja satisfeito. Uma solução gulosa randomizada é construída na linha 3 e em seguida uma busca local é aplicada à ela na linha 4. A melhor solução corrente e o respectivo valor da função objetivo são atualizados nas linhas 6 e 7, respectivamente. A melhor solução global p^* é retornada na linha 10.

Algoritmo 4.3 Pseudocódigo de um procedimento GRASP padrão para um problema de minimização

Entrada: Grafo direcionado $D = (V, A)$ e *Criterio_Parada*.
Saída: p^*.

```
1:  f* ← ∞;
2:  enquanto Criterio_Parada não satisfeito faça
3:     usar um algoritmo guloso randomizado para obter potências p;
4:     aplicar busca local à solução definida pelas potências p, obtendo potências p';
5:     se f(p') < f* então
6:        p* ← p';
7:        f* ← f(p*);
8:     fim se
9:  fim enquanto
10: retorna p*;
```

Nessa seção, os algoritmos gulosos randomizados apresentados na Seção 3.5 e as vizinhanças e buscas locais apresentadas na Seção 4.1 são combinadas e utilizadas na construção de procedimentos GRASP para CMP2C-AB.

4.2.1 Diferentes Versões de Procedimentos GRASP

Pode-se combinar os algoritmos gulosos randomizados apresentados na Seção 3.5 e as buscas locais apresentadas na Seção 4.1 de várias formas para montar versões de procedimentos GRASP.

Após análise dos resultados com os algoritmos gulosos randomizados apresentados na Seção 3.5, fixou-se, para todas as versões GRASP, o algoritmo guloso randomizado rgbgc com função gulosa dinâmica e estratégia reativa (ver Algoritmo 3.8). Esse algoritmo obteve os melhores resultados de tempo, de qualidade e diversidade de soluções. Fixada a parte construtiva do GRASP, varia-se as vizinhanças e buscas locais. Três versões do GRASP são geradas:

- GRASPr: GRASP com busca local reduzida,

- GRASPe: GRASP com busca local estendida,
- GRASPd: GRASP com busca local duplamente estendida.

4.2.1.1 Testes Computacionais com as Versões do GRASP

Os experimentos foram realizados em uma máquina com processador PC Intel Pentium 4 HT com 3,2 GHz de relógio e 1 Gbyte de memória RAM com sistema operacional GNU/Linux 2.6.24. Os algoritmos foram codificados em C++ e o código executável gerado com o compilador GNU g++ versão 4.1, usando o parâmetro de otimização -O2.

Os resultados e conclusões apresentados desta seção em diante, referem-se a um conjunto de oito instâncias identificadas pelos nomes: RD025, RD050, RD100, RD200, EU025, EU050, EU100 e EU200. Há uma instância de cada tipo (randômica e euclideana) para cada tamanho $|V| \in \{25, 50, 100, 200\}$. Essas instâncias foram escolhidas pois ilustram o comportamento padrão verificado para todas as instâncias usadas durante os testes (até 15 instâncias para cada tipo e tamanho). Para apresentar e ilustrar a discussão dos resultados foi escolhida a instância RD100, que também foi utilizada na apresentação dos resultados da Seção 3.5.1. Os resultados completos para todas as instâncias podem ser verificados nos Anexos A, B, C, D e E.

No primeiro experimento, cada algoritmo executou por um tempo fixo de 3125 segundos. Foram feitas cinco execuções independentes de cada algoritmo para cada instância. A Tabela 4.1 apresenta os resultados para a instância RD100 e as tabelas do Anexo A apresentam os resultados paras as demais instâncias. Todas essas tabelas mostram os resultados médios de cinco execuções para cada versão do GRASP. As linhas apresentam, para cada versão do GRASP, o valor médio da solução obtida no tempo em segundos definido em cada coluna. Esses resultados demonstram a capacidade dos algoritmos de alcançarem soluções cada vez melhores a medida que mais tempo é fornecido e permitem comparar, entre os algoritmos, aqueles que possuem a capacidade de alcançar melhores soluções em menos tempo.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	11,19388	11,13038	**11,04206**	10,98531	10,94703
GRASPe	**11,15398**	**11,10707**	11,05080	**10,92308**	**10,88231**
GRASPd	11,49712	11,27728	11,17588	11,13003	11,03263

Tabela 4.1: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP para a instância RD100.

A Tabela 4.1 e as demais do Anexo A mostram em negrito o melhor resultado al-

cançado dentro de cada limite de tempo estabelecido. Essas tabelas mostram que, independentemente do tipo da instância (randômica ou euclideana), as vizinhanças reduzida e estendida possuem comportamento similar e melhor do que o observado para a vizinhança duplamente estendida. A figura 4.3 mostra o valor do melhor resultado alcançado no decorrer de uma única execução de cada algoritmo para a instância RD100a. Esses resultados confirmam a maior eficácia dos algoritmos com vizinhanças reduzida e estendida, pois alcançam melhores soluções em menos tempo e permanecem melhorando a solução a medida que mais tempo é fornecido.

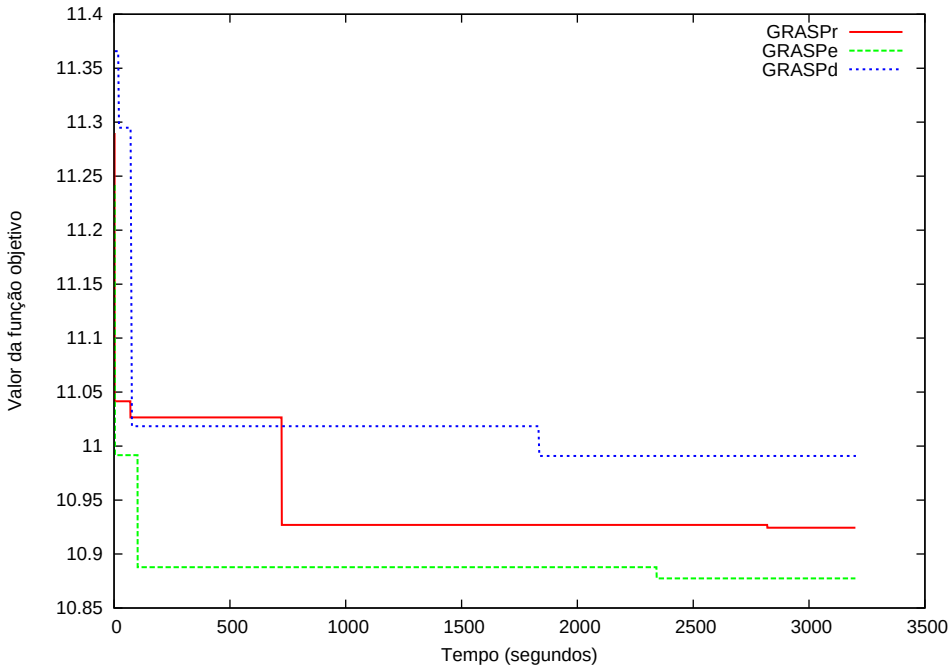


Figura 4.3: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP para a instância RD100.

O pior comportamento da vizinhança duplamente estendida pode ser explicado pela maior perturbação causada à solução corrente ao ser aplicada mais uma operação de decremento a uma solução já inviável e ao maior tempo de processamento para percorrer toda a vizinhança. Isso pode ser constatado observando-se que, para $|V| > 25$, os melhores resultados (marcados em negrito) se alternam entre as vizinhanças reduzida e estendida, o que mostra que a vizinhança duplamente estendida tem maior dificuldade de encontrar os melhores resultados. Pode-se concluir também dessas tabelas que todos os algoritmos permanecem melhorando a solução a medida que o tempo de processamento avança, até que se obtenha a solução ótima.

No segundo experimento, duzentas execuções independentes para cada algoritmo foram realizadas para cada instância. Cada execução foi encerrada assim que o valor da

solução encontrada tornou-se menor ou igual a um valor alvo estabelecido ou até que um tempo limite fosse alcançado. Os valores alvo foram escolhidos de forma a permitirem a comparação entre os algoritmos mais e menos efetivos.

Em seguida, os gráficos de distribuição de tempos de execução (GDTE) foram traçados da seguinte maneira: cada ponto da curva associado a um algoritmo representa o par (t_i, P_i) $(i = 1, \ldots, 200)$, onde t_i é o tempo de processamento da i-ésima execução mais rápida do algoritmo considerado e $P_i = (i - \frac{1}{2}/200)$ aproxima a probabilidade de que o algoritmo encontre o alvo no tempo de processamento t_i. Estes gráficos fornecem uma aproximação da distribuição de probabilidade da variável aleatória tempo para alcançar uma solução de custo menor ou igual ao valor alvo. Quanto mais à esquerda está a curva de um algoritmo, mais rapidamente ele converge para o valor alvo. Quanto menor for a diferença de tempo entre a execução mais rápida e a mais lenta, mais robusto é o algoritmo.

Para a comparação dos algoritmos GRASPr, GRASPe e GRASPd, foi definido um alvo fácil e estabelecido o tempo limite de 600 segundos para cada algoritmo encontrar o alvo. O alvo fácil foi determinado como um valor próximo ao valor da solução encontrada em 25 segundos de processamento pelo algoritmo GRASPd (algoritmo menos efetivo), conforme mostra a Tabela 4.1. A Figura 4.4 e as figuras do Anexo B apresentam a distribuição de tempos de execução dos algoritmos GRASPr, GRASPe e GRASPd.

As conclusões advindas da Figura 4.4 e das demais do Anexo B são consistentes com relação as observações dos resultados da Tabela 4.1 e das demais do Anexo A. Para todas as instâncias, as vizinhanças reduzida e estendida levam a resultados próximos e melhores do que a vizinhança duplamente estendida. Porém, apesar dos resultados muitos próximos, as vizinhanças reduzida e estendida alternam os melhores resultados dependendo da instância utilizada como entrada.

Na próxima seção, serão propostas e testadas combinações das vizinhanças em uma mesma busca local na tentativa de se construir algoritmos ainda mais robustos.

4.2.2 GRASP com VND

Nessa seção, as vizinhanças e buscas locais da Seção 4.2.1.1 são combinadas com uma estratégia de busca local do tipo *Variable Neighborhood Descent* (VND). Procedimentos do tipo VND consistem em explorar completamente uma vizinhança até que um ótimo local seja encontrado e atribuído à solução corrente. Em seguida, passa-se para uma outra

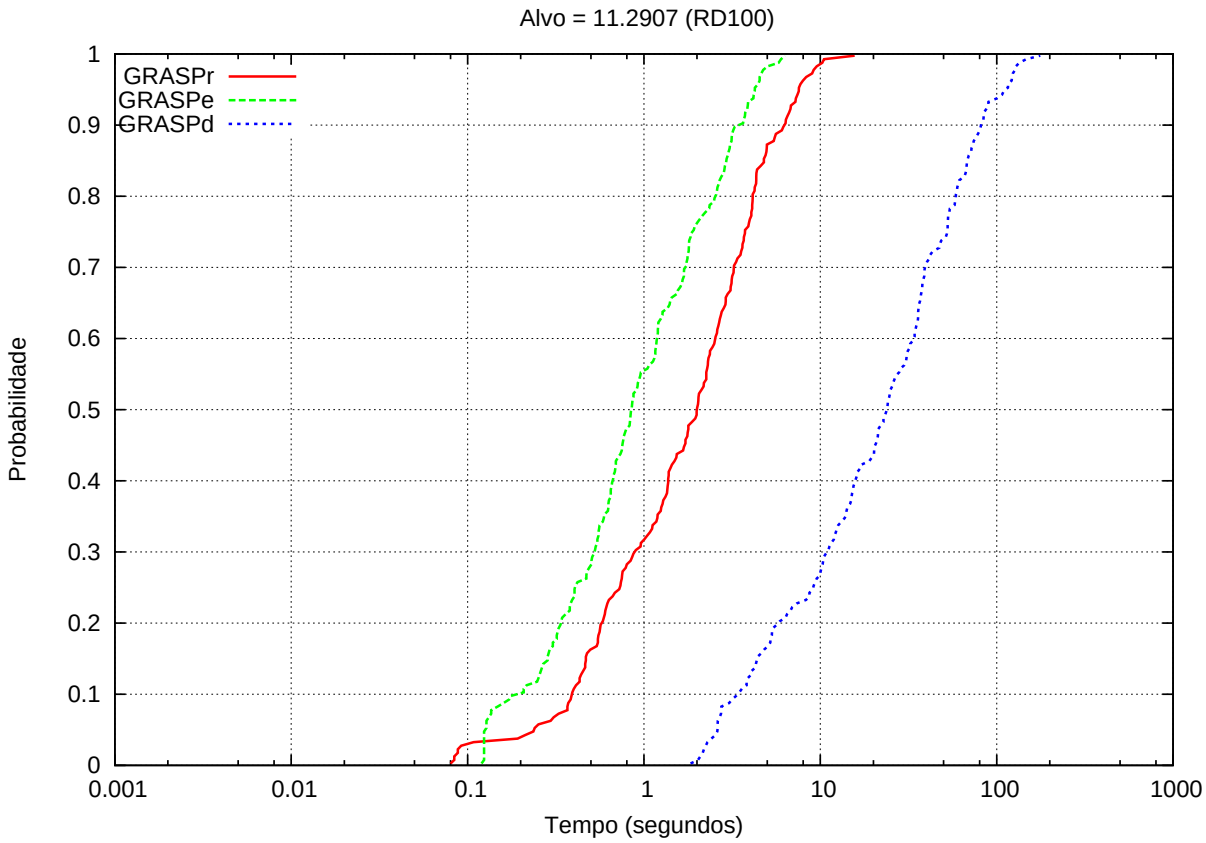


Figura 4.4: Distribuição de tempos de execução dos três algoritmos GRASP para instância RD100 e alvo fácil.

vizinhança. Se uma solução melhor do que a solução corrente for encontrada, em alguma das vizinhanças, então retorna-se para a primeira vizinhança [22, 23] conforme descrito no Algoritmo 4.4.

O algoritmo GRASP com VND, denominado GVND, é construído utilizando o Algoritmo 4.4 como fase de busca local na linha 4 do Algorimo 4.3. Considerando que o Algoritmo 4.4 recebe como entrada um conjunto de vizinhanças $\mathcal{N}_\kappa, \kappa = 1, \ldots, \kappa_{max}$, três algoritmos foram implementados:

- GVNDre: GVND com algoritmo guloso randomizado rgbgc, com vizinhança reduzida para $\kappa = 1$ ($\mathcal{N}_1$) e vizinhança estendida para $\kappa = 2$ ($\mathcal{N}_2$),
- GVNDer: GVND com algoritmo guloso randomizado rgbgc, com vizinhança estendida para $\kappa = 1$ ($\mathcal{N}_1$) e vizinhança reduzida para $\kappa = 2$ ($\mathcal{N}_2$),
- GVNDred: GVND com algoritmo guloso randomizado rgbgc, com vizinhança reduzida para $\kappa = 1$ ($\mathcal{N}_1$), vizinhança estendida para $\kappa = 2$ ($\mathcal{N}_2$) e vizinhança duplamente estendida para $\kappa = 3$ ($\mathcal{N}_3$).

Algoritmo 4.4 Busca Local com VND

Entrada: Potências de transmissão p tal que o grafo de comunicação $G(p) = (V, B(p))$ é biconexo e conjunto de vizinhanças $\mathcal{N}_\kappa, \kappa = 1, \ldots, \kappa_{max}$.

Saída: Potências de transmissão p' e grafo de comunicação $G(p') = (V, B(p'))$ biconexo tais que $\sum_{u \in V} p'_u \leq \sum_{u \in V} p_u$.

1: $p' \leftarrow p$;
2: $\kappa \leftarrow 1$;
3: $melhoria \leftarrow verdadeiro$;
4: **enquanto** $melhoria$ **faça**
5: $melhoria \leftarrow falso$;
6: **repetir**
7: aplicar busca local a p' usando a vizinhança $\mathcal{N}_\kappa$ obtendo potências p;
8: **se** $\sum_{u \in V} p_u < \sum_{u \in V} p'_u$ **então**
9: $p' \leftarrow p$;
10: $\kappa \leftarrow 1$;
11: $melhoria \leftarrow verdadeiro$;
12: **senão**
13: $\kappa \leftarrow \kappa + 1$;
14: **fim se**
15: **até** $\kappa = \kappa_{max}$;
16: **fim enquanto**
17: **retorna** p';

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	11,19388	11,13038	11,04206	10,98531	10,94703
GRASPe	11,15398	11,10707	11,05080	10,92308	10,88231
GRASPd	11,49712	11,27728	11,17588	11,13003	11,03263
GVNDre	**11,13763**	11,11716	11,04178	10,93142	**10,87491**
GVNDer	11,15328	**11,08949**	**10,98385**	**10,90096**	10,88207
GVNDred	11,33008	11,21244	11,12901	11,06520	11,02650

Tabela 4.2: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP e pelos três algoritmos GVND para instância RD100.

A Tabela 4.2 e as tabelas do Anexo C comparam os resultados médios de cinco execuções independentes de cada uma das três versões do algoritmo GVND com os algoritmos GRASP da seção anterior (GRASPr, GRASPe e GRASPd). Essas tabelas mostram em negrito o melhor resultado alcançado, dentro de cada limite de tempo estabelecido.

Verificando os resultados assinalados em negrito da Tabela 4.2 e aqueles nas tabelas do Anexo C, pode-se observar que as versões do algoritmo GVND obtêm resultados melhores que os algoritmos GRASP básicos. Esses mesmos indicadores evidenciam que as versões GVNDre e GVNDer sempre alcançam melhores resultados que a versão GVNDred, para o mesmo limite de tempo. O uso das vizinhanças reduzida e estendida se mostrou novamente mais eficaz que a vizinhança duplamente estendida.

A figura 4.5 mostra o valor do melhor resultado alcançado no decorrer de uma única execução de cada algoritmo para a instância RD100a. Esses resultados confirmam a maior capacidade dos algoritmos com vizinhanças reduzida e estendida alcançarem melhores soluções em menos tempo.

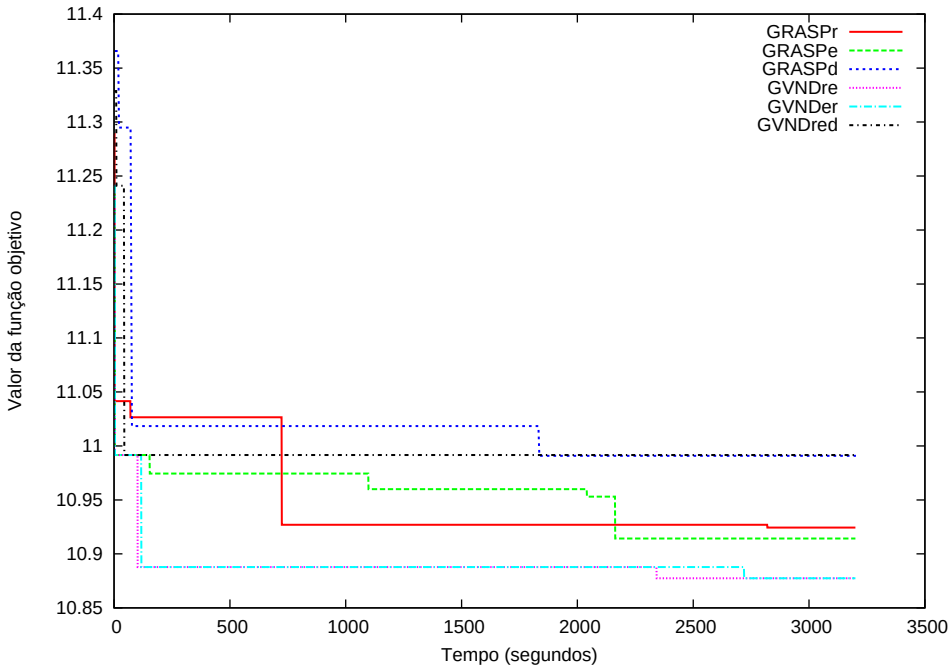


Figura 4.5: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP e pelos três algoritmos GVND para instância RD100.

A Figura 4.6 apresenta a distribuição de tempos de execução dos algoritmos da Tabela 4.2 para a instância RD100. O alvo fácil e o tempo limite utilizados (600 segundos) são os mesmos da Figura 4.4. Os mesmos resultados para as demais instâncias utilizadas nos testes podem ser vistos nas figuras do Anexo D.

A Figura 4.7 apresenta a distribuição de tempos de execução dos algoritmos da Tabela 4.2 para a instância RD100, utilizando como alvo um valor considerado de média dificuldade. O tempo limite para execução de cada algoritmo foi definido em 900 segundos. O alvo com média dificuldade foi determinado como um valor próximo ao valor médio da solução encontrada até os 625 segundos de processamento pelos algoritmos da Tabela 4.2. Os mesmos resultados para as demais instâncias utilizadas nos testes podem ser vistos nas figuras do Anexo E, observando que as instâncias com $|V| = 25$ não possuem alvo com média dificuldade pois a solução ótima é considerada um alvo fácil.

A partir da análise dos gráficos de distribuição de tempos de execução, percebe-se que os algoritmos GVNDre e GVNDer estão sempre bem próximos um do outro, mostrando

que a ordem de aplicação das vizinhanças é indiferente. Pode-se notar também que, na maioria das vezes, os algoritmos GVNDre e GVNDer possuem comportamento igual ou melhor que os algoritmos GRASPr e GRASPe.

Os resultados vem mostrando que, apesar de ter comportamentos semelhantes, o uso independente das vizinhanças reduzida e estendida pelo GRASP básico, faz com que os algoritmos GRASPr e GRASPe se alternam em termos de melhores resultados dependendo da instância sendo resolvida. Ao utilizar as duas vizinhanças reduzida e estendida no mesmo algoritmo, a busca local alcança os melhores resultados independentemente de instância.

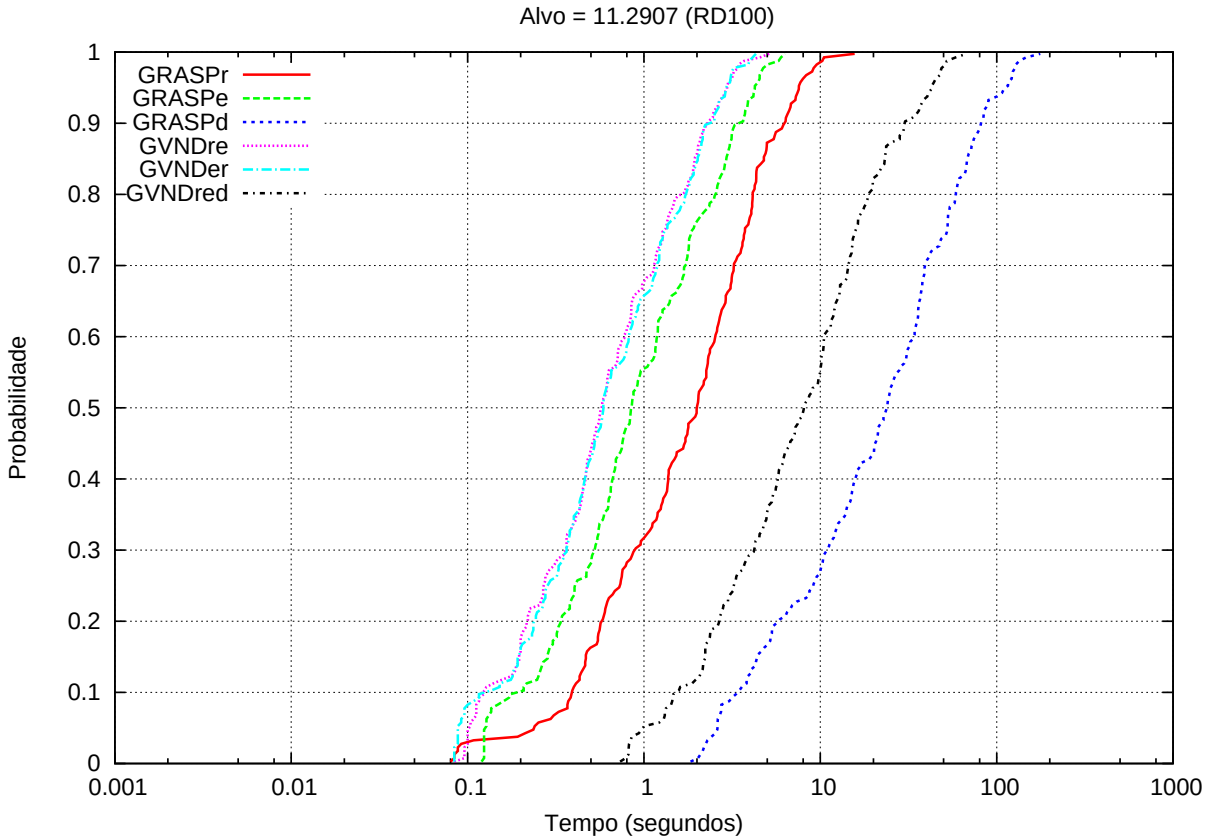


Figura 4.6: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para instância RD100 e alvo fácil.

Como a diferença entre os resultados dos algoritmos GVNDre e GVNDer é irrelevante, foi escolhido de forma arbitrária o algoritmo GVNDre para a incorporação da estratégia de reconexão por caminhos.

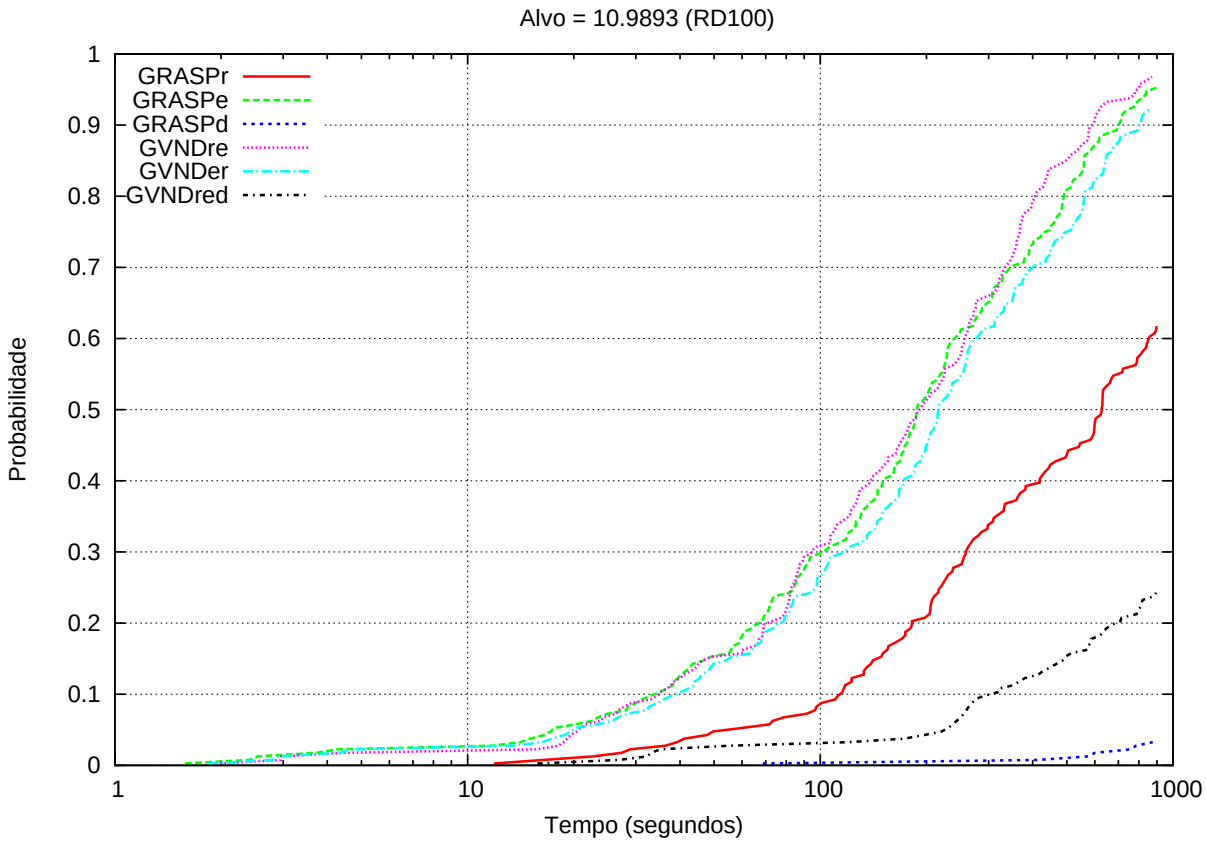


Figura 4.7: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para instância RD100 e alvo com média dificuldade.

4.2.3 GRASP com VND e Reconexão por Caminhos

O algoritmo de reconexão por caminhos (*path relinking*) foi originalmente proposto por Glover [18] como uma estratégia de intensificação, conectando soluções de elite obtidas por busca tabu ou busca por dispersão (*scatter search*) [19]. Considerando-se duas soluções de elite, identifica-se os movimentos que devem ser aplicados a uma solução para se chegar a outra, definindo-se uma trajetória. Uma trajetória inicia-se em uma das soluções de elite, chamada de solução inicial, e gera um caminho que se dirige à outra solução, chamada de solução guia. A exploração da trajetória que conecta soluções de alta qualidade gera novas soluções, que podem ser de melhor qualidade do que as soluções inicial e 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. Sejam $p^{(1)}$ e $p^{(2)}$ duas soluções para o problema CMP2C-AB, $\Delta(p^{(1)}, p^{(2)})$ o conjunto de movimentos que devem ser aplicados à solução inicial $p^{(1)}$ para alcançar a solução final $p^{(2)}$ e p^* a solução de menor custo f^* para o problema CMP2C-AB em

uma trajetória de $p^{(1)}$ a $p^{(2)}$. O Algoritmo 4.5 ilustra o pseudocódigo da reconexão por caminhos aplicada ao par de soluções $(p^{(1)}, p^{(2)})$.

Algoritmo 4.5 Reconexão por caminhos

Entrada: Par de soluções $p^{(1)}$ e $p^{(2)}$.
Saída: Solução p^*.

```
1: Calcular o conjunto de movimentos Δ(p^(1), p^(2));
2: f* ← min{f(p^(1)), f(p^(2))};
3: p* ← argmin(f(p^(1)), f(p^(2)));
4: p ← p^(1);
5: enquanto Δ(p^(1), p^(2)) ≠ ∅ faça
6:    η* ← argmin{(f(p ⊕ η) : η ∈ Δ(p, p^(2)))};
7:    p ← p ⊕ η*;
8:    se f(p) < f* então
9:       f* ← f(p);
10:      p* ← p;
11:   fim se
12: fim enquanto
13: retorna p*;
```

O algoritmo inicia na linha 1 calculando o conjunto de movimentos $\Delta(p^{(1)}, p^{(2)})$ que devem ser aplicados à solução inicial $p^{(1)}$ para alcançar a solução final $p^{(2)}$. A melhor solução p^* e seu custo f^* neste caminho são retornados pelo algoritmo na linha 13. Em cada passo, o Algoritmo 4.5 examina todos os movimentos $\eta \in \Delta(p, p^{(2)})$ a partir da solução atual p e seleciona aquele que resultar na solução de custo mínimo, isto é, aquele que minimiza $f(p \bigoplus \eta)$, onde $p \bigoplus \eta$ é a solução resultante da aplicação do movimento η à solução p (linha 6). Na linha 7 o melhor movimento η^* é realizado, produzindo a solução $p \bigoplus \eta^*$. Se necessário, a melhor solução p^* e o seu custo f^* são atualizados nas linhas 9 e 10.

Os movimentos em $\Delta(p^{(1)}, p^{(2)})$ são aqueles que permitem levar, para cada nó $i \in V$, do nível de potência do nó $p_i^{(1)}$ para o nível de potência $p_i^{(2)}$. Cria-se uma lista de movimentos ordenadas por seus custos, que são definidos pelo valor da diferença entre os níveis de potência dos nós cujas potências são afetadas pelo movimento.

Várias alternativas podem ser consideradas na reconexão por caminhos:

- *reconexão para frente*: a reconexão por caminhos é realizada usando a solução de pior custo entre $p^{(1)}$ e $p^{(2)}$ como solução inicial e a outra, como solução guia;
- *reconexão para trás*: usa-se a solução de melhor custo entre $p^{(1)}$ e $p^{(2)}$ como solução inicial e a outra, como solução guia;

- *reconexão para frente e para trás*: duas trajetórias diferentes são percorridas, uma partindo de $p^{(1)}$ e chegando a $p^{(2)}$ e a outra saindo de $p^{(2)}$ e chegando a $p^{(1)}$;
- *reconexão mista*: são explorados dois caminhos simultaneamente, o primeiro partindo de $p^{(1)}$ e o segundo de $p^{(2)}$, até que eles se encontrem em uma solução equidistante de $p^{(1)}$ e $p^{(2)}$;
- *reconexão aleatorizada*: a cada passo, selecionar aleatoriamente um movimento 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 $p^{(1)}$ e $p^{(2)}$ é analisada.

Todas as alternativas envolvem compromissos entre tempo computacional e qualidade da solução. Ribeiro et al. [44] observaram que explorar duas trajetórias diferentes para cada par $(p^{(1)}, p^{(2)})$ 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 $p^{(1)}$ e $p^{(2)}$. Como a vizinhança da solução inicial é muito mais 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 estão normalmente mais próximas da solução inicial do que da solução guia, permitindo finalizar a trajetória de religação antes da solução guia ser alcançada.

No contexto de uma heurística GRASP, a reconexão por caminhos é aplicada a pares $(p^{(1)}, p^{(2)})$ de soluções, onde $p^{(1)}$ é a solução localmente ótima obtida após a busca local e $p^{(2)}$ é uma solução de elite escolhida aleatoriamente de um conjunto com um número limitado Tam_Pool de soluções de elite encontradas ao longo das iterações anteriores.

Inicialmente, o conjunto de soluções de elite, chamado $Pool$, está vazio. Cada solução localmente ótima obtida pela busca local com VND é considerada como uma candidata a ser inserida no conjunto se ela é diferente de qualquer outra solução atualmente no $Pool$. 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.

Para a implementação da heurística GRASP com VND e reconexão por caminhos, foram investigadas as estratégias de reconexão para frente e para trás com escolha mais aprimorante de movimento e a estratégia de reconexão para trás com escolha randômica de movimento. Dada a lista de movimentos ordenada, as estratégias de reconexão para frente e para trás aprimorante examinam todos os movimentos na sequência da lista, começando pelo mais aprimorante. A estratégia para trás randômica escolhe um movimento aleatoriamente da lista segundo uma distribuição de probabilidades uniforme. Dessa forma, três versões do algoritmo GRASP com VND e reconexão por caminhos são construídas, baseadas no algoritmo GVNDre que obteve os melhores resultados pela análise dos experimentos computacionais da Seção 4.2.2:

- GVNDre(fa): usa-se o algoritmo guloso randomizado rgbgc, vizinhança reduzida para $\kappa = 1$ ($\mathcal{N}_1$), vizinhança estendida para $\kappa = 2$ ($\mathcal{N}_2$) e reconexão por caminhos para frente aprimorante,
- GVNDre(ta): usa-se o algoritmo guloso randomizado rgbgc, vizinhança reduzida para $\kappa = 1$ ($\mathcal{N}_1$), vizinhança estendida para $\kappa = 2$ ($\mathcal{N}_2$) e reconexão por caminhos para trás aprimorante,
- GVNDre(tr): usa-se o algoritmo guloso randomizado rgbgc, vizinhança reduzida para $\kappa = 1$ ($\mathcal{N}_1$), vizinhança estendida para $\kappa = 2$ ($\mathcal{N}_2$) e reconexão por caminhos para trás aleatória.

Todos os algoritmos utilizam $Tam\_Pool = 5$ e aplicaram a reconexão por caminhos a cada ótimo local obtido após a fase de busca local.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GVNDre	**11,13763**	**11,11716**	11,04178	**10,93142**	10,87491
GVNDre(fa)	11,19039	11,13870	**11,02355**	10,93226	10,87896
GVNDre(ta)	11,19039	11,13870	**11,02355**	10,93226	10,87896
GVNDre(tr)	11,17425	11,16089	11,03185	10,94830	**10,87203**

Tabela 4.3: Evolução dos resultados obtidos pelo algoritmo GVNDre e pelos três algoritmos GVND com reconexão por caminhos para a instância RD100.

A Tabela 4.3 e as tabelas do Anexo F comparam os resultados médios de cinco execuções independentes do algoritmo GVNDre da seção anterior com as três versões de GVND com reconexão por caminhos. Essas tabelas mostram em negrito o melhor resultado alcançado dentro de cada limite de tempo estabelecido. Essas tabelas mostram ainda que o algoritmo GVNDre chega a bons resultados em menores tempo de execução.

Já os algoritmos com reconexão por caminhos, usam um conjunto de soluções elite que vai se aprimorando ao longo da execução. Assim, o desempenho relativo de GVND com reconexão por caminhos frente à versão sem reconexão por caminhos melhora a medida que o tempo limite de processamento aumenta.

Assim, na maioria dos casos, esses algoritmos (principalmente o algoritmo GVNDre(tr)) superam em termos de qualidade de solução o algoritmo GVNDre a medida que o tempo de execução aumenta.

A figura 4.5 mostra o valor do melhor resultado alcançado no decorrer de uma única execução dos algoritmos GVNDre(tr) e GVNDre para a instância RD100a. Esses resultados mostram que o algoritmo GVNDre(tr) tem maior capacidade que o algoritmo GVNDre de alcançar melhores soluções a medida que o tempo de execução aumenta.

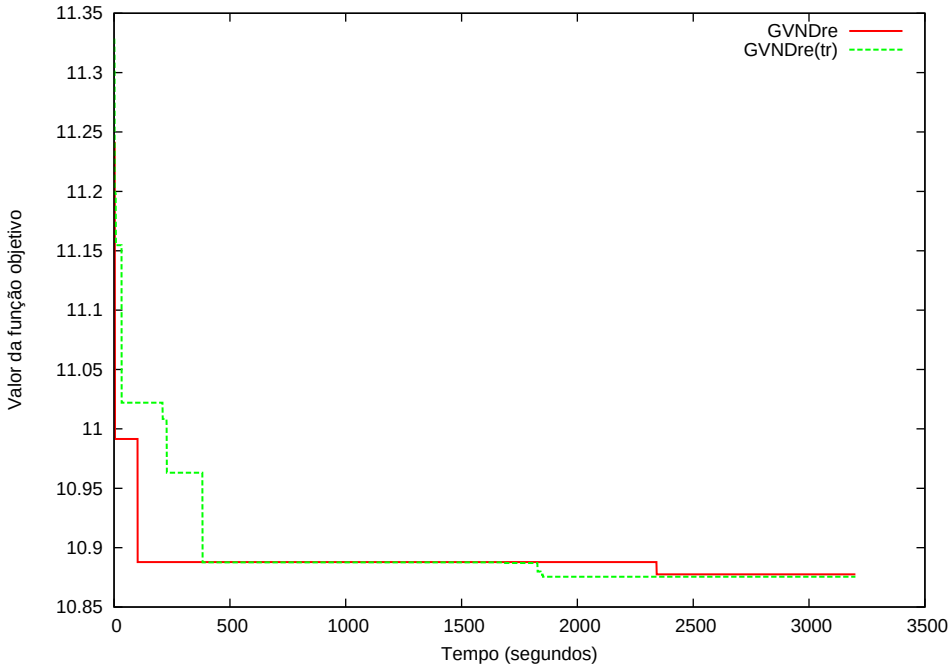


Figura 4.8: Evolução dos resultados obtidos pelos algoritmos GVNDre e GVNDre(tr) para instância RD100.

A Figura 4.9 e as figuras do Anexo G apresentam a distribuição de tempos de execução com alvos difíceis (mesmos alvos utilizados na Figura 4.7 e nas demais do Anexo E) e tempo limite de processamento de 600 segundos. Pode-se observar que há pouca diferença na utilização das três estratégias de reconexão por caminhos.

A Figura 4.10 e as figuras do Anexo H mostram, para as instâncias RD100, RD200, EU100 e EU200, a distribuição de tempos de execução com alvos mais difíceis e tempo limite de processamento de uma hora. Um alvo difícil foi determinado como um valor

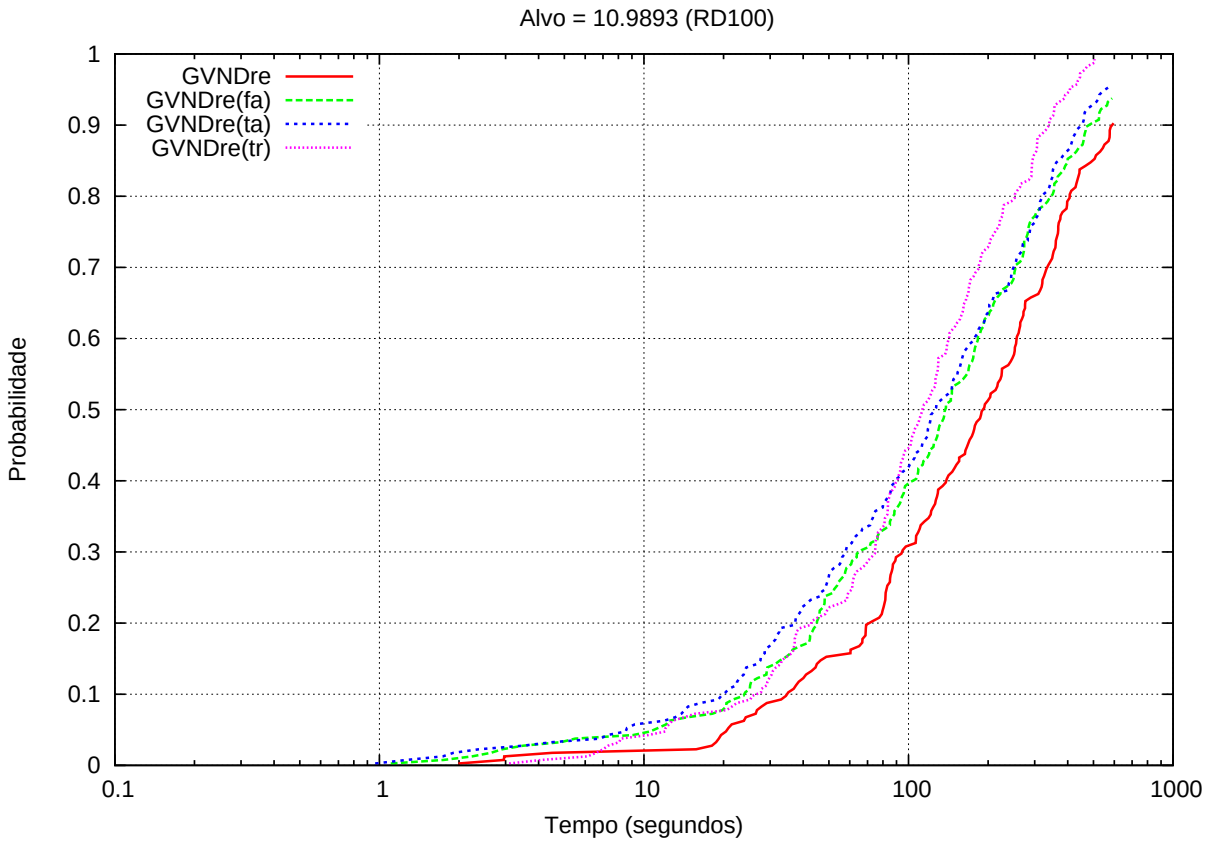


Figura 4.9: Distribuição de tempos de execução do algoritmo GVNDre e dos três algoritmos GVND com reconexão por caminhos para instância RD100 e alvo com média dificuldade.

próximo ao valor médio da soluções encontradas até 625 e 3125 segundos de processamento pelos algoritmos da Tabela 4.3. Esses experimentos visam melhor comparar os algoritmos GVNDre e GVNDre(tr). Os resultados confirmam que o algoritmo com reconexão por caminhos é mais robusto pois alcança mais rapidamente os alvos mais difíceis.

A próxima seção compara os melhores algoritmos propostos em cada capítulo desse trabalho.

4.3 Resultados Comparativos

Com o objetivo de comparar os algoritmos desenvolvidos em cada capítulo, foram construídas trinta instâncias para cada valor de $V \in \{25, 50, 100, 200, 400, 800\}$, sendo que dez são instâncias da classe DE, dez da classe EU e dez da classe RD, geradas conforme descrito na Seção 3.4.

Os algoritmos de cada capítulo que obtiveram os melhores resultados experimentais

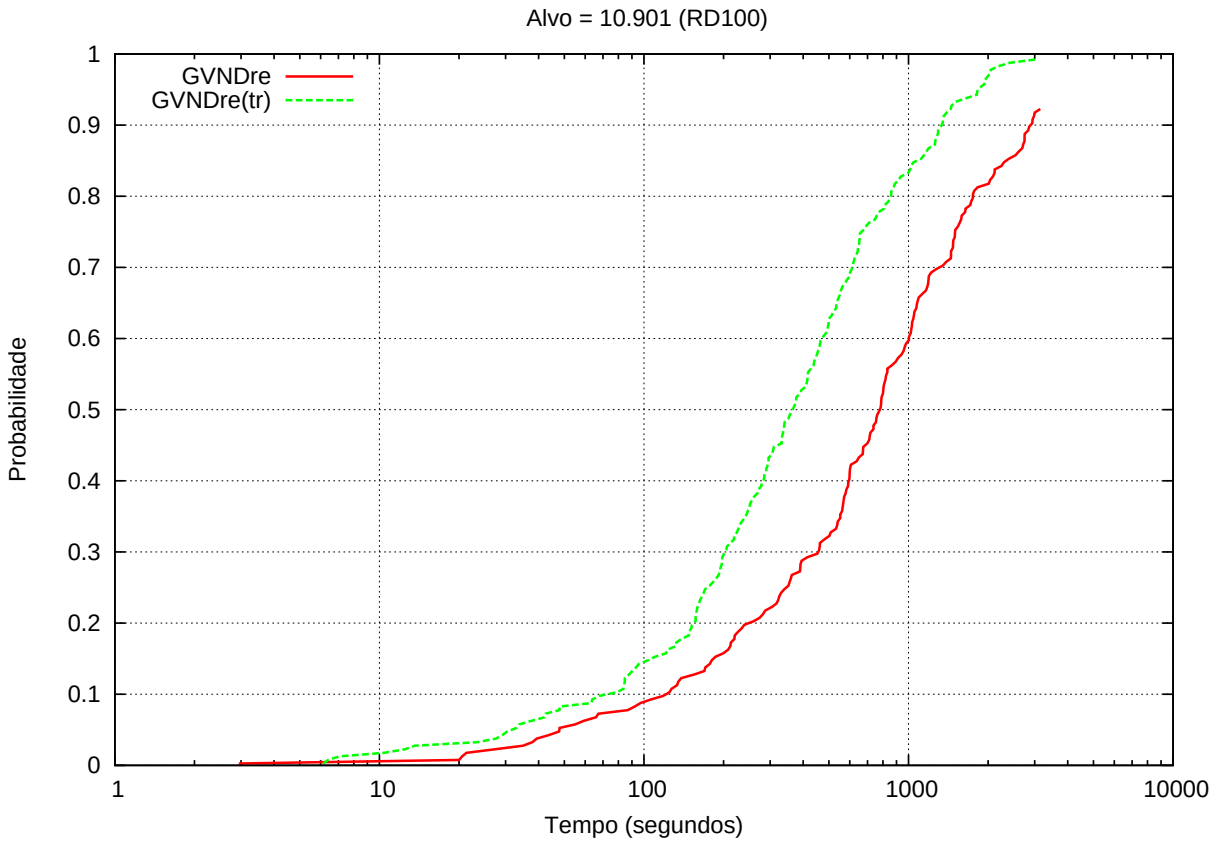


Figura 4.10: Distribuição de tempos de execução dos algoritmos GVNDre e GVNDre(tr) para instância RD100 e alvo difícil.

para a solução do CMP2C-AB foram selecionados para serem comparados:

- Algoritmo exato: formulação com potência incremental bidirecional apresentada na Figura 2.5 e estendida com o conjunto de inequações (2.26) e (2.27), referenciada por IP-B no Capítulo 2.
- Algoritmo gbgc(e_d): apresentado como Algoritmo 3.3, constrói um grafo de comunicação biconexo a partir de um grafo conexo utilizando a função gulosa dinâmica. É referenciado por gbgc(e_d) no Capítulo 3.
- Algoritmo GVNDre(tr): apresentado na Seção 4.2.3, corresponde à heurística GRASP usando algoritmo guloso randomizado rgbgc como fase construtiva, busca local do tipo VND com vizinhança reduzida e estendida e reconexão por caminhos para trás aleatória.

A Seção 4.3.1 mostra em detalhes a comparação entre os resultados exatos e aqueles obtidos pelas heurísticas. Na Seção 4.3.2, apresenta-se a comparação entre os resultados

aproximados para as instâncias cujos valores ótimos exatos não são conhecidos. Por último, na Seção 4.3.3, compara-se os melhores limites inferiores conhecidos com as melhores soluções obtidas.

4.3.1 Comparações com Resultados Exatos

Nessa seção apresenta-se a comparação entre as soluções exatas e as soluções heurísticas para as instâncias com $|V| = 25$. Todos as soluções ótimas foram encontradas aplicando-se o CPLEX à formulação IP-B sem limite de tempo. Foi feita uma única execução da heurística GVNDre(tr) para cada instância, utilizando como critério de parada o valor da solução ótima do problema.

A Tabela 4.4 mostra o valor da potência total obtida pelos algoritmos exato, GVNDre(tr) (soluções ótimas) e gbgc(e_d) e o respectivo tempo (em segundos) de processamento utilizado por cada algoritmo para encontrá-la. O algoritmo guloso gbgc(e_d) executa, para todas essas instâncias, em menos de um segundo de processamento. Também é apresentado o erro relativo percentual entre o valor da solução obtida pelo algoritmo guloso gbgc(e_d) e o valor da solução exata. Valores médios são apresentados para cada classe de instâncias.

Como esperado, dessa tabela observa-se que o algoritmo exato gasta mais tempo de processamento do que a heurística GVNDre(tr) para encontrar as soluções ótimas. A tabela mostra ainda, para instâncias da classe EU, que o algoritmo gbgc(e_d) pode gerar soluções com consumo de potência até 45,67% maiores que o valor ótimo. Entretanto, na média, o erro percentual fica abaixo dos 18,51%, mostrando esse algoritmo guloso pode gerar soluções de boa qualidade em milésimos de segundo, tornando-se uma boa opção para aplicações em tempo real.

4.3.2 Avaliações dos Resultados Obtidos pelas Heurísticas

Como visto no Capítulo 2, nenhuma solução ótima foi encontrada pelo CPLEX em três horas de processamento para instâncias com $|V| \geq 50$. Assim, os próximos resultados analisam as soluções obtidas pela heurística GVNDre(tr) e pelo algoritmo guloso gbgc(e_d).

Na Tabela 4.5 são apresentados os resultados comparativos referentes à potência média e ao grau médio dos nós dos grafos de comunicação estabelecidos pela heurística GVNDre(tr) e pelo algoritmo guloso gbgc(e_d). No caso do GVNDre(tr), apresentam-se os resultados da melhor solução encontrada após dez minutos de processamento. De modo

Instância		Exato		GVNDre(tr)		gbgc(e_d)	
Tipo	|V|	Potência	Tempo (s)	Potência	Tempo (s)	Potência	Erro (%)
		4,53587	1444,49	4,53587	0,18	5,04074	11,13
		5,61990	2613,70	5,61990	0,06	6,41634	14,17
		6,04886	2416,15	6,04886	0,18	6,83502	13,00
		5,75345	7403,70	5,75345	0,01	6,51117	13,17
RD	25	5,37601	998,83	5,37601	0,92	6,24658	16,19
		5,31916	272,47	5,31916	0,07	6,01090	13,00
		6,06659	4177,81	6,06659	44,68	7,03407	15,95
		6,09894	4782,42	6,09894	7,45	6,88422	12,88
		5,24336	6618,71	5,24336	0,01	5,81950	10,99
		4,88730	1345,95	4,88730	3,62	5,64903	15,59
média		5,49494	3207,42	5,49494	5,72	6,24476	13,65
		1,26713	4,30	1,26713	0,00	1,30920	3,32
		1,35492	180,68	1,35492	0,27	1,66993	23,25
		1,48489	111,47	1,48489	0,03	1,75514	18,20
		1,34786	195,78	1,34786	0,03	1,51189	12,17
EU	25	1,37864	1189,79	1,37864	0,34	2,00823	45,67
		1,38121	475,16	1,38121	0,01	1,53976	11,48
		1,26518	66,68	1,26518	0,00	1,38902	9,79
		1,30372	290,87	1,30372	0,02	1,56615	20,13
		1,79024	787,89	1,79024	0,04	2,17208	21,33
		1,22173	352,55	1,22173	0,02	1,42833	16,91
média		1,37955	365,52	1,37955	0,08	1,63497	18,51
		5,42020	557,84	5,42020	0,22	5,82523	7,47
		5,46512	912,07	5,46512	1,42	5,96974	9,23
		5,44700	2217,30	5,44700	0,22	5,71760	4,97
		5,50566	1922,80	5,50566	0,50	5,75377	4,51
DE	25	5,22509	26390,41	5,22509	0,02	5,77208	10,47
		5,39419	4397,92	5,39419	0,02	5,97752	10,81
		5,41784	961,58	5,41784	0,05	5,95489	9,91
		5,37637	2665,15	5,37637	0,01	5,45316	1,43
		6,03484	1230,24	6,03484	0,01	6,89224	14,21
		5,14622	2069,19	5,14622	0,02	5,76453	12,01
média		5,44325	4332,45	5,44325	0,25	5,90808	8,54

Tabela 4.4: Comparação das potências e dos tempos (em segundos) para obtenção da solução ótima pelos algoritmos exato e GVNDre(tr) e o erro relativo entre os valores ótimos e os obtidos pelo algoritmo guloso gbgc(e_d) para instâncias com $|V| = 25$.

Instância		Potência média			Grau médio		
Tipo	$\|V\|$	GVNDre(tr)	gbgc(e_d)	Diferença (%)	GVNDre(tr)	gbgc(e_d)	Diferença (%)
	25	5,49494	6,24476	12,01	2,48	3,04	18,42
	50	8,41205	9,55131	11,93	2,43	3,05	20,33
	100	11,69800	13,21970	11,51	2,63	3,11	15,43
RD	200	17,21848	19,20336	10,34	2,67	3,22	17,08
	400	24,93783	27,60327	9,66	2,65	3,16	16,14
	800	37,30339	40,92659	8,85	2,70	3,20	15,63
	25	1,37955	1,63497	15,62	2,64	2,94	10,20
	50	1,26619	1,47556	14,19	2,58	2,93	11,95
	100	1,31637	1,45481	9,52	2,60	2,99	13,04
EU	200	1,76927	1,85038	4,38	2,56	2,93	12,63
	400	2,82729	2,89912	2,48	2,54	2,94	13,61
	800	4,99544	5,07905	1,65	2,59	3,00	13,67
	25	5,44325	5,90808	7,87	2,55	3,06	16,67
	50	10,17110	10,95295	7,14	2,54	2,94	13,61
	100	19,34007	21,02384	8,01	2,53	3,00	15,67
DE	200	37,29308	40,35054	7,58	2,52	2,99	15,72
	400	73,72322	78,88391	6,54	2,55	2,96	13,85
	800	103,23984	109,68819	5,88	2,56	2,95	13,22

Tabela 4.5: Resultados comparativos referentes à potência média e ao grau médio para os algoritmos GVNDre(tr) e gbgc(e_d), para todas as classes de instâncias com $|V| \in \{25, 50, 100, 200, 400, 800\}$ (o algoritmo GVNDre(tr) executa durante 600 segundos).

geral, todas as execuções de gbgc(e_d) levaram menos de dois segundos.

Também são apresentadas as diferenças percentuais entre o valor da melhor solução obtida por GVNDre(tr) e o valor da solução gulosa obtida por gbgc(e_d) e entre o valor do grau médio dos nós do grafo de comunicação estabelecido pela melhor solução obtida por GVNDre(tr) e por gbgc(e_d). As diferenças são calculadas como $100 \times (X - Y)/X$, onde X é o valor (potência ou grau médio) obtido por gbgc(e_d) e Y o valor obtido por GVNDre(tr).

A Tabela 4.5 mostra que, para as classes RD e DE, os valores das diferenças percentuais são próximos independentemente do tamanho das instâncias. A diferença entre potências varia de $8,85\%$ a $12,01\%$ na classe RD e de $5,88\%$ a $8,01\%$ na classe DE. Quanto maior é o valor de $|V|$, mais difícil é para o GVNDre(tr) melhorar a solução obtida por gbgc(e_d), pois quanto maior é o valor de $|V|$ menos iterações são realizadas pelo GVNDre(tr) no mesmo período de tempo.

Para a classe EU, a diminuição da diferença relativa acontece de forma mais acentuada a medida que $|V|$ cresce, variando de $1,65\%$ a $15,62\%$. Entretanto, esse fato está relacionado à estrutura das instâncias que, com o aumento da densidade dos nós, possuem maior número de arestas com custo baixo e estas, ao serem removidas, causam baixa diminuição do valor absoluto da função objetivo. Esse fato pode ser confirmado pelos valores das

Instância		Potência média			Grau médio		
Tipo	$\|V\|$	GVNDre(tr)	gbgc(e_d)	Diferença (%)	GVNDre(tr)	gbgc(e_d)	Diferença (%)
	25	5,49494	6,24476	12,80	2,48	3,04	18,42
	50	8,40091	9,55131	12,20	2,42	3,05	20,66
	100	11,64082	13,21970	11,97	2,61	3,11	16,08
RD	200	17,14772	19,20336	11,01	2,64	3,22	18,01
	400	24,83953	27,60327	10,65	2,66	3,16	15,82
	800	37,16536	40,92659	10,24	2,68	3,20	16,25
	25	1,37955	1,63497	13,75	2,64	2,94	10,20
	50	1,26619	1,47556	14,92	2,58	2,93	11,95
	100	1,31584	1,45481	13,87	2,60	2,99	13,04
EU	200	1,76809	1,85038	9,88	2,54	2,93	13,31
	400	2,82558	2,89912	9,13	2,53	2,94	13,95
	800	4,99307	5,07905	8,51	2,58	3,00	14,00
	25	5,44325	5,90808	14,97	2,55	3,06	16,67
	50	10,17110	10,95295	14,14	2,54	2,94	13,61
	100	19,33094	21,02384	17,24	2,53	3,00	15,67
DE	200	37,20409	40,35054	15,78	2,51	2,99	16,05
	400	73,49927	78,88391	14,27	2,53	2,96	14,53
	800	102,78351	109,68819	12,22	2,54	2,95	13,90

Tabela 4.6: Resultados comparativos referentes à potência média e ao grau médio para os algoritmos GVNDre(tr) e gbgc(e_d), para todas as classes de instâncias com $|V| \in \{25, 50, 100, 200, 400, 800\}$ (o algoritmo GVNDre(tr) executa durante 10800 segundos).

diferenças percentuais no valor do grau médio. Esses valores mostram que a diferença no grau médio tem valores próximos independentemente do tamanho de $|V|$ para todas as classes. Em especial, para a classe EU, a diferença percentual aumenta a medida que $|V|$ cresce, demonstrando que mais arestas são removidas, entretanto, o baixo custo dessas arestas provoca pequena diminuição do valor da potência final. Assim, o valor da diferença percentual do valor da potência diminui a medida que $|V|$ cresce para a classe EU.

Os resultados apresentados na Tabela 4.5 para $|V| = 25$, correspondem aos valores ótimos já mostrados na Tabela 4.4. Com esses resultados é possível mostrar que as diferenças percentuais entre as soluções ótimas e as soluções obtidas pelo algoritmo gbgc(e_d) permanecem próximos às diferenças percentuais entre as soluções obtidas por GVNDre(tr) e as soluções obtidas pelo algoritmo gbgc(e_d), mesmo para instâncias grandes. De onde se conclui que as soluções obtidas por GVNDre(tr) permanecem de boa qualidade a medida que o tamanho das instâncias crescem.

A Tabela 4.6 mostra os mesmos resultados da Tabela 4.5 para GVNDre(tr) após três horas de processamento. Comparando os resultados dessas tabelas, percebe-se que GVNDre(tr), para qualquer tamanho e tipo de instância, obtém dentro do limite de tempo de 600 segundos de processamento um valor de solução de boa qualidade que é aprimorado quando o limite de tempo de processamento é aumentado para 10800 segundos.

	|V|			
Classe	25	50	100	200
RD	13,65	22,30	27,64	35,83
EU	4,41	5,26	4,22	3,25
DE	4,19	5,75	7,78	9,04

Tabela 4.7: Resultados médios da diferença percentual entre o valor do melhor limite superior e o valor do melhor limite inferior para $|V| \in \{25, 50, 100, 200\}$.

4.3.3 Comparações entre Limites Inferiores e Superiores

Para ter uma noção da qualidade das soluções encontradas pelo algoritmo GVNDre(tr) para $|V| \geq 50$, foram feitas comparações com os limites inferiores determinados pelo valor da solução da relaxação linear da formulação IP-B encontrado pelo algoritmo dual simplex implementado pelo CPLEX dentro do limite de tempo de três horas. O CPLEX obteve a solução ótima da relaxação linear dentro do limite de tempo para apenas algumas instâncias com $|V| = 50$. Para as demais instâncias, foi utilizada a solução parcial do dual simplex como limite inferior.

Os experimentos com o algoritmo GVNDre(tr) e com a relaxação linear para as instâncias com $|V| = 200$ foram realizados em uma máquina Intel Core 2 Quad com relógio de 2.40 GHz e 8 Gbytes de memória RAM com sistema operacional GNU/Linux 2.6.24. Nessa máquina, foi observado que o CPLEX demandou até 90% dos 8 Gbytes. Experimentos com instâncias maiores não puderam ser realizados nas máquinas disponíveis devido à limitações de memória. Todos os resultados foram obtidos utilizando o resolvedor CPLEX 11.0 em seu modo padrão.

Na Tabela 4.7 é mostrada, para $|V| \in \{25, 50, 100, 200\}$ e para cada classe de instância, a média da diferença relativa percentual entre o valor da melhor solução conhecida (valor ótimo para $|V| = 25$ ou melhor limite superior obtido por GVNDre(tr) para as instâncias maiores) e o valor do melhor limite inferior. Essa diferença tende a aumentar com o número de nós, exceto para a classe EU que tem o valor absoluto da potência afetado pelo aumento da densidade dos nós.

4.4 Conclusão

Neste capítulo foram desenvolvidos procedimentos GRASP para CMP2C-AB. Todos os algoritmos propostos foram avaliados e ajustados com o uso de resultados experimentais.

Primeiramente, foram propostas diferentes vizinhanças e procedimentos de busca lo-

cal utilizados na construção de um algoritmo GRASP padrão. Em seguida, algoritmos híbridos combinando GRASP com VND foram desenvolvidos. Analisando os resultados experimentais, foi possível determinar que o uso combinado das duas melhores vizinhanças tornou a heurística mais robusta e a qualidade das soluções produzidas independente de instância. Os resultados mostraram também que a sequência em que as vizinhanças são tratadas pela busca VND é indiferente.

Ao melhor algoritmo híbrido GRASP com VND foi acrescentada a estratégia de reconexão por caminhos. Três estratégias foram implementadas: reconexão para frente e para trás com escolha aprimorante e reconexão para trás com escolha randômica. Mais uma vez, foi possível determinar através de resultados experimentais que o acréscimo da estratégia de reconexão para trás com escolha randômica trouxe melhoria ao algoritmo híbrido GRASP com VND.

Por último, comparou-se os resultados dos melhores algoritmos desenvolvidos nos Capítulos 2, 3 e 4. Desta comparação, ficou evidenciada a capacidade do algoritmo guloso de encontrar boas soluções rapidamente, mesmo para grandes instâncias, o que o caracteriza como um bom candidato para aplicações em tempo real. A heurística GRASP demonstrou sua eficácia e robustez para grandes instâncias, encontrando boas soluções com pouco tempo de processamento e mantendo o mesmo comportamento independentemente do tipo e do tamanho das instâncias.

Capítulo 5

Conclusões

Da mesma maneira que as redes de computadores tradicionais se tornaram um campo de grande interesse científico e rapidamente ocuparam importante papel no desenvolvimento de aplicações computacionais, as redes ad hoc vêm crescentemente atraindo o interesse dos pesquisadores e prometem expandir ainda mais o uso das redes de computadores.

Novos desafios são introduzidos no projeto de redes ad hoc sem fio ao serem acrescentados objetivos como minimização de potência, minimização de interferência, maximização do tempo de vida e minimização do número de saltos, entre outros. O problema escolhido para estudo neste trabalho foi a minimização do consumo de potência sob restrições de conexidade. Quatro variantes do problema foram identificadas, dependendo da entrada (simétrica ou assimétrica) e da topologia da solução (bidirecional ou unidirecional).

A primeira proposta de solução desse problema foi desenvolvida através de modelos exatos para as quatro variantes do problema de consumo mínimo de potência com restrição de k-conexidade (CMPkC). Os três modelos propostos foram solucionados por um resolvedor de problemas de programação inteira. Os resultados alcançados pelos modelos exatos apresentados em [38] permitiram a comparação entre os modelos e entre as variantes do problema. Também forneceram os valores exatos utilizados como referência para a análise das técnicas aproximadas.

Para a variante mais relevante do problema de consumo mínimo de potência, onde são consideradas restrições de biconexidade com entrada assimétrica e topologia bidirecional (CMP2C-AB), foram propostas diferentes heurísticas. Primeiramente, foram desenvolvidos algoritmos gulosos baseados na construção de grafos conexos e no algoritmo de Tarjan para determinação das componentes biconexas. Também foi implementado um algoritmo da literatura. Todos os algoritmos foram comparados em termos da qualidade da solução obtida. O melhor algoritmo guloso se caracterizou como um bom candidato para aplica-

ções em tempo real, pois foi capaz de encontrar boas soluções rapidamente, mesmo para grandes instâncias.

Em seguida, a fim de construir heurísticas GRASP, os algoritmos gulosos foram adaptados para se tornarem algoritmos gulosos randomizados. Esse algoritmos foram ajustados através de experimentos computacionais que permitiram a identificação do melhor algoritmo e os respectivos valores dos parâmetros.

Completando as heurísticas GRASP, foram definidas diferentes vizinhanças e buscas locais. Em seguida, propôs-se a combinação das vizinhanças em uma implementação de GRASP com VND. Após a identificação da melhor combinação entre vizinhanças, testaram-se três estratégias de reconexão por caminhos. Todos os algoritmos foram testados e ajustados por experimentos computacionais. Esses experimentos demonstraram as características de cada algoritmo, permitindo a escolha do algoritmo com melhores resultados de tempo e qualidade da solução. Os resultados obtidos com o GRASP foram apresentados em [37].

A heurística GRASP obteve todas as soluções ótimas para pequenas instâncias e demonstrou eficácia e robustez para grandes instâncias, encontrando boas soluções com pouco tempo de processamento e mantendo o mesmo comportamento independentemente do tipo e do tamanho das instâncias. A heurística GRASP também obteve resultados melhores do que o algoritmo da literatura implementado.

Pode-se apontar várias extensões para desenvolvimento de trabalhos futuros. Com relação ao problema, poder-se-ia estender outras de suas variantes através da inclusão de interferência, limitação na quantidade de energia disponível, limitação da potência máxima, limite no número de saltos, mobilidade dos nós e suas combinações. As restrições de topologia poderiam ser modificadas para o estabelecimento de uma árvore de comunicação ligando todos os nós da rede (*broadcast*) ou tratar o estabelecimento de ligações entre um determinado par de nós origem e destino. Novas variantes do problema devem torná-lo cada vez mais próximo da sua aplicação prática. Diferentes técnicas e algoritmos de solução podem ser desenvolvidos para os problemas aqui propostos e suas variações.

Referências

[1] Althaus, E., Calinescu, G., Mandoiu, I. I., Prasad, S., Tchervenski, N., Zelikovsky, A. Power efficient range assignment for symmetric connectivity in static ad hoc wireless networks. *Wireless Networks 12* (2006), p. 287–299.

[2] Bahramgiri, M., Hajiaghayi, M., Mirrokni, V. S. Fault-tolerant and 3-dimensional distributed topology control algorithms in wireless multi-hop networks. Em *Proceedings of the Eleventh International Conference on Computer Communications and Networks* (Los Alamitos, 2002), IEEE Computer Society Press, p. 392–397.

[3] Blough, D. M., Leoncini, M., Resta, G., Santi, P. On the symmetric range assignment problem in wireless ad hoc networks. Em *Proceedings of the IFIP 17th World Computer Congress - TC1 Stream* (Montreal, 2002), p. 71–82.

[4] Calinescu, G., Kapoor, S., Olshevsky, A., Zelikovsky, A. Network lifetime and power assignment in ad hoc wireless networks. Em *Proceedings of the 11th Annual European Symposium on Algorithms* (Budapest, 2003), p. 114–126.

[5] Calinescu, G., Mandoiu, I. I., Zelikovsky, A. Symmetric connectivity with minimum power consumption in radio networks. Em *Proceedings of the IFIP 17th World Computer Congress - TC1 Stream* (Montreal, 2002), p. 119–130.

[6] Calinescu, G., Wan, P. J. Range assignment for biconnectivity and k-edge connectivity in wireless ad hoc networks. *Mobile Network and Applications 11* (2006), p. 121–128.

[7] Caragiannis, I., Kaklamanis, C., Kanellopoulos, P. Energy-efficient wireless network design. *Theory of Computing Systems 39* (2006), p. 593–617.

[8] Carmi, P., Segal, M., Katz, M. J., Shpungin, H. Fault-tolerant power assignment and backbone in wireless networks. Em *Proceedings of 4th Annual IEEE International Conference on Pervasive Computing and Communications Workshops* (Pisa, 2006), p. 80–84.

[9] Chen, W. T., Huang, N. F. The strongly connecting problem on multihop packet radio networks. *IEEE Transactions on Communications 37* (1989), p. 293–295.

[10] Cheng, X., Narahari, B., Simha, R., Cheng, M. X., Liu, D. Strong minimum energy topology in wireless sensor networks: NP-Completeness and heuristics. *IEEE Transactions on Mobile Computing 2* (2003), p. 248–256.

[11] Clementi, A. E. F., Penna, P., Silvestri, R. Hardness results for the power range assignmet problem in packet radio networks. Em *Proceedings of the 2nd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems* (Berkeley, 1999), p. 197–208.

[12] Das, A. K., Marks, R., El-Sharkawi, M., Arabshahi, P., Gray, A. Optimization methods for minimum power bidirectional topology construction in wireless networks with sectored antennas. Em *Proceedings of the IEEE Global Telecommunications Conference* (Dallas, 2004), vol. 6, p. 3962–3968.

[13] Das, A. K., Mesbahi, M. K-node connected power efficient topologies in wireless networks: A semidefinite programming approach. Em *Proceedings of the IEEE Global Telecommunications Conference* (Saint Louis, 2005), vol. 1, p. 468–473.

[14] Feo, T. A., Resende, M. G. C. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters 8* (1989), p. 67–71.

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

[16] Festa, P., Resende, M. G. C. GRASP: An annotated bibliography. Em *Ribeiro, C. C.; Hansen, P., editores, Essays and Surveys in Metaheuristics* (2002), Kluwer Academic Publishers, p. 325–367.

[17] Garey, M. R., Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP–Completeness*. W. H. Freeman and Company, San Francisco, 1979.

[18] Glover, F. Tabu search and adaptive memory programming: Advances, applications and challenges. Em *Interfaces in Computer Science and Operations Research* (1996), Kluwer, p. 1–75.

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

[20] Hajiaghayi, M., Immorlica, N., Mirrokni, V. S. Power optimization in fault-tolerant topology control algorithms for wireless multi-hop networks. Em *Proceedings of the 9th Annual International Conference on Mobile Computing and Networking* (San Diego, 2003), p. 300–312.

[21] Hajiaghayi, M., Kortsarz, G., Mirrokni, V. S., Nutov, Z. Approximating power optimization problems. Em *Proceedings of the 11th Conference on Integer Programming and Combinatorial Optimization* (Berlin, 2005), p. 349–361.

[22] Hansen, P., Mladenović, N. Variable neighborhood search: Principles and applications. *European Journal of Operational Research 130* (2001), p. 449–467.

[23] Hansen, P., Mladenović, N. Developments of variable neighborhood search. Em *Essays and surveys in metaheuristics*, C. Ribeiro e P. Hansen, Eds. Kluwer Academic Publishers, 2002, p. 415–439.

[24] Harary, F. The maximum connectivity of a graph. Em *Proceedings of the National Academy of Sciences of the United States of America* (1962), vol. 48, p. 1142–1146.

[25] Jia, X., Kim, D., Makki, S., Wan, P.-J., Yi, C.-W. Power assignment for k-connectivity in wireless ad hoc networks. *Journal of Combinatorial Optimization 9* (2005), p. 213–222.

[26] Kawadia, V., Kumar, P. R. Principles and protocols for power control in wireless ad hoc networks. *IEEE Journal on Selected Areas in Communications 23* (2005), p. 76–88.

[27] Kirousis, L. M., Kranakis, E., Krizanc, D., Pelc, A. Power consumption in packet radio networks. *Theoretical Computer Science 243* (2000), p. 289–305.

[28] Krumke, S. O., Liu, R., Lloyd, E. L., Marathe, M. V., Ramanathan, R., Ravi, S. Topology control problems under symmetric and asymmetric power thresholds. Em *Proceedings of the 2nd International Conference on Ad hoc and Wireless Networks* (Montreal, 2003), p. 187–198.

[29] Li, L., Halpern, J. Y., Bahl, P., Wang, Y.-M., Wattenhofer, R. Analysis of a cone-based distributed topology control algorithm for wireless multi-hop networks. Em *Proceedings of the 20th Annual ACM symposium on Principles of Distributed Computing* (Newport, 2001), p. 264–273.

[30] Li, N., Hou, J. C. Localized topology control algorithms for heterogeneous wireless networks. *IEEE/ACM Transactions on Networking 13* (2005), p. 1313–1324.

[31] Lloyd, E. L., Liu, R., Marathe, M. V., Ramanathan, R., Ravi, S. S. Algorithmic aspects of topology control problems for ad hoc networks. *Mobile Networks and Applications 10* (2005), p. 19–34.

[32] Madan, R., Lall, S. Distributed algorithms for maximum lifetime routing in wireless sensor networks. *IEEE Transactions on Wireless Communications 5* (2006), p. 2185–2193.

[33] Magnanti, T. L., Raghavan, S. Strong formulations for network design problems with connectivity requirements. *Networks 45* (2005), p. 61–79.

[34] Marina, M. K., Das, S. R. On-demand multipath distance vector routing in ad hoc networks. Em *Proceedings of the 9th International Conference on Network Protocols* (Riverside, 2001), p. 14–23.

[35] Marina, M. K., Das, S. R. Routing performance in the presence of unidirectional links in multihop wireless networks. Em *Proceedings of the 3rd ACM International Symposium on Mobile Ad Hoc Networking and Computing* (Lausanne, 2002), p. 12–23.

[36] Montemanni, R., Gambardella, L. M. Exact algorithms for the minimum power symmetric connectivity problem in wireless networks. *Computers and Operations Research 32* (2005), p. 2891–2904.

[37] Moraes, R. E. N., Ribeiro, C. C. Power optimization in ad hoc wireless network topology control with biconnectivity requirements. Em *Proceedings of Second Global Conference on Power Control and Optimization* (2009), ISBN 978-983-44483.

[38] Moraes, R. E. N., Ribeiro, C. C., Duhamel, C. Optimal solutions for fault-tolerant topology control in wireless ad hoc networks. *IEEE Transactions on Wireless Communications 8* (2009), p. 5970–5981.

[39] Prais, M., Ribeiro, C. Parameter variation in GRASP procedures. *Investigación Operativa 9* (2000), p. 1–20.

[40] Prim, R. Shortest connection networks and some generalizations. *Bell Systems Technology Journal 36* (1957), p. 1389.

[41] Raghavan, S. *Formulations and Algorithms for the netwotk design problems with connectivity requeriments*. PhD thesis, Massachussets Institute of Technology, Department of Electrical Engineering and Computer Science, Cambridge, 1995.

[42] Ramanathan, R., Rosales-Hain, R. Topology control of multihop wireless networks using transmit power adjustment. Em *Proceedings of the Ninetieth Annual Joint Conference of the IEEE Computer and Communications Societies* (Tel Aviv, 2000), p. 404–413.

[43] Rappaport, T. *Wireless Communications: Principles and Practice*. Prentice Hall, Upper Saddle River, 2001.

[44] Ribeiro, C. C., , Uchoa, E., Werneck, R. F. A hybrid grasp with perturbations for the steiner problem in graphs. *INFORMS Journal on Computing 14* (2002), p. 228–226.

[45] Rodoplu, V., Meng, T. H. Minimum energy mobile wireless networks. *IEEE Journal on Selected Areas in Communications 17* (1999), p. 1333–1344.

[46] Romer, K., Mattern, F. The design space of wireless sensor networks. *IEEE Wireless Communications 11*, 6 (2004), p. 54–61.

[47] Saha, I., Sambasivan, L. K., Ghoshand, S. K., Patro, R. K. Distributed fault tolerant topology control in wireless ad-hoc sensor networks. Em *Proceedings of the 2006 IFIP International Conference on Wireless and Optical Communications Networks* (2006), p. 5–13.

[48] Santi, P. Topology control in wireless ad hoc and sensor networks. *ACM Computing Surveys 37* (2005), p. 164–194.

[49] Santi, P. *Topology control in wireless ad hoc and sensor networks*. John Wiley & Sons, Ltd., West Sussex, 2005.

[50] Shpungin, H., Segal, M. k-fault resistance in wireless ad-hoc networks. Em *Proceedings of the 2005 Joint Workshop on Foundations of Mobile Computing* (Cologne, 2005), p. 89–96.

[51] Tarjan, R. Depth-first search and linear graph algorithms. *SIAM Journal on Computing 1* (1972), p. 146–160.

[52] Wang, F., Thai, M. T., Li, Y., Du, D.-Z. Fault-tolerant topology control for all-to-one and one-to-all communication in wireless networks. *IEEE Transactions on Mobile Computing 7* (2008), p. 322–331.

[53] Wattenhofer, R., Li, L., Bahl, P., Wang, Y. M. Distributed topology control for power efficient operation in multihop wireless ad hoc networks. Em *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies* (Anchorage, 2001), vol. 3, p. 1388–1397.

[54] Wieselthier, J. E., Nguyen, G. D., Ephremides, A. On the construction of energy-efficient broadcast and multicast trees in wireless networks. Em *Proceedings of the 9th Annual Joint Conference of the IEEE Computer and Communications Societies* (Tel-Aviv, 2000), vol. 2, p. 585–594.

[55] Xing, G., Lu, C., Zhang, Y., Huang, Q., Pless, R. Minimum power configuration in wireless sensor networks. Em *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing* (Urbana-Champaign, 2005), ACM Press, p. 390–401.

APÊNDICE A – Execuções Longas das Versões de GRASP

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	5.62314	**5.61990**	**5.61990**	**5.61990**	**5.61990**
GRASPe	**5.61990**	**5.61990**	**5.61990**	**5.61990**	**5.61990**
GRASPd	5.65468	5.64390	**5.61990**	**5.61990**	**5.61990**

Tabela A.1: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP para a instância RD025.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	**9.01795**	**8.95437**	**8.93464**	8.92874	8.92341
GRASPe	9.07520	8.97034	8.94156	**8.91931**	**8.90703**
GRASPd	9.16967	9.09062	8.94780	8.93688	8.92722

Tabela A.2: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP para a instância RD050.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	11.19388	11.13038	**11.04206**	10.98531	10.94703
GRASPe	**11.15398**	**11.10707**	11.05080	**10.92308**	**10.88231**
GRASPd	11.49712	11.27728	11.17588	11.13003	11.03263

Tabela A.3: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP para a instância RD100.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	**17.72396**	17.66853	17.62556	**17.51775**	17.44616
GRASPe	17.73358	**17.63502**	**17.60225**	17.56595	**17.41286**
GRASPd	18.00174	17.84695	17.80312	17.66411	17.64906

Tabela A.4: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP para a instância RD200.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	**1.35492**	**1.35492**	**1.35492**	**1.35492**	**1.35492**
GRASPe	1.35512	**1.35492**	**1.35492**	**1.35492**	**1.35492**
GRASPd	1.35835	**1.35492**	**1.35492**	**1.35492**	**1.35492**

Tabela A.5: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP para a instância EU025.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	1.26057	**1.25873**	**1.25866**	**1.25866**	**1.25866**
GRASPe	**1.25896**	1.25881	**1.25866**	**1.25866**	**1.25866**
GRASPd	1.27690	1.26672	1.26338	1.26159	1.26062

Tabela A.6: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP para a instância EU050.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	**1.28856**	**1.28774**	**1.28573**	**1.28512**	**1.28367**
GRASPe	1.29235	1.29040	1.28794	1.28624	1.28468
GRASPd	1.32926	1.29566	1.28940	1.28731	1.28552

Tabela A.7: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP para a instância EU100.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	1.73658	1.73364	1.72974	1.72841	1.72784
GRASPe	**1.73152**	**1.72941**	**1.72898**	**1.72813**	**1.72770**
GRASPd	1.73232	1.73215	1.73199	1.73068	1.72884

Tabela A.8: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP para a instância EU200.

APÊNDICE B – Distribuição dos Tempos de Execução das Versões de GRASP

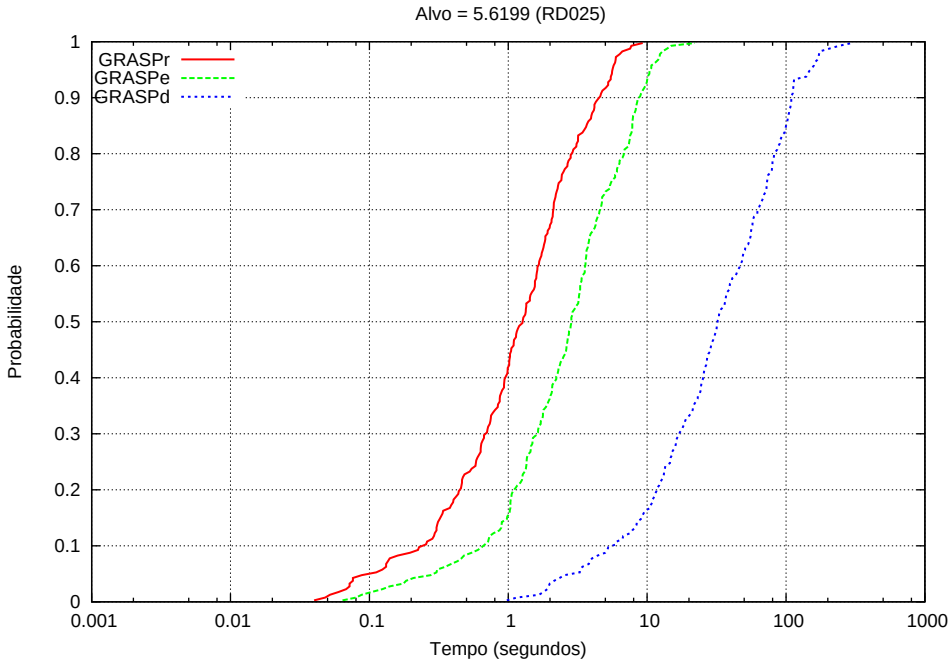


Figura B.1: Distribuição de tempos de execução dos três algoritmos GRASP para a instância RD025 e alvo fácil.

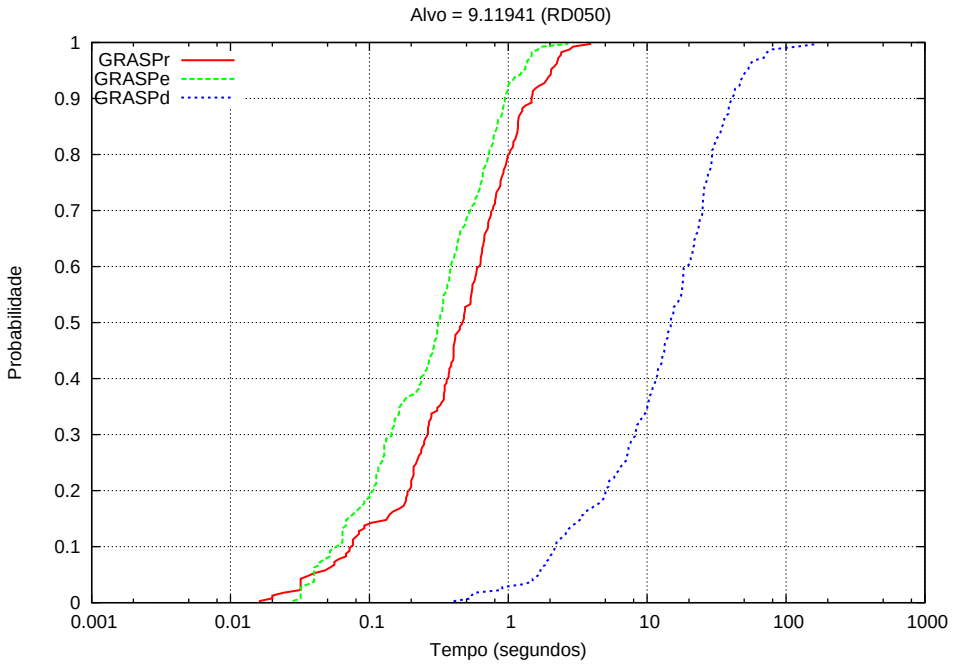


Figura B.2: Distribuição de tempos de execução dos três algoritmos GRASP para a instância RD050 e alvo fácil.

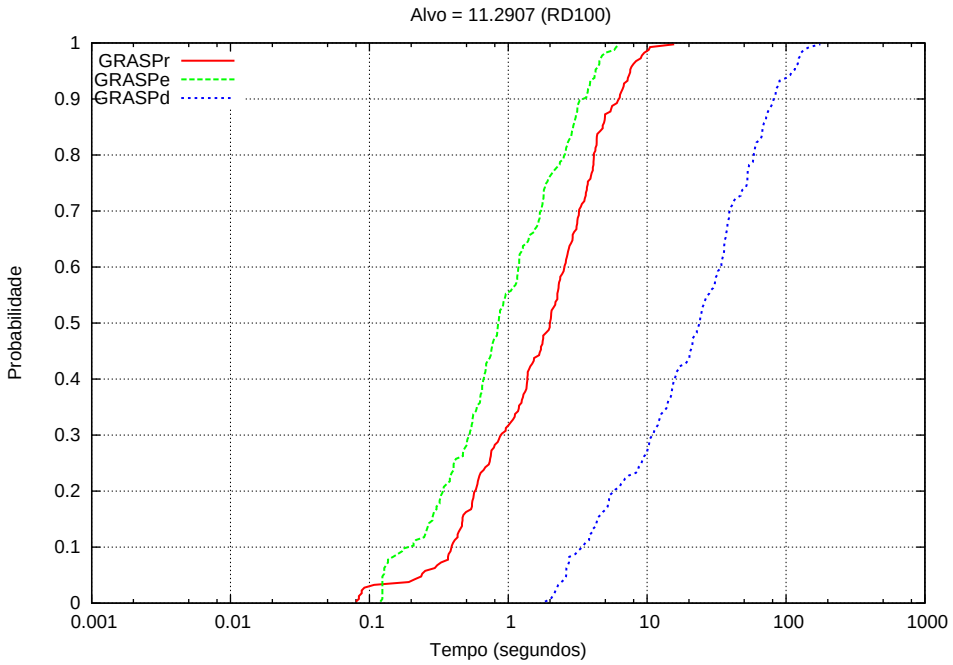


Figura B.3: Distribuição de tempos de execução dos três algoritmos GRASP para a instância RD100 e alvo fácil.

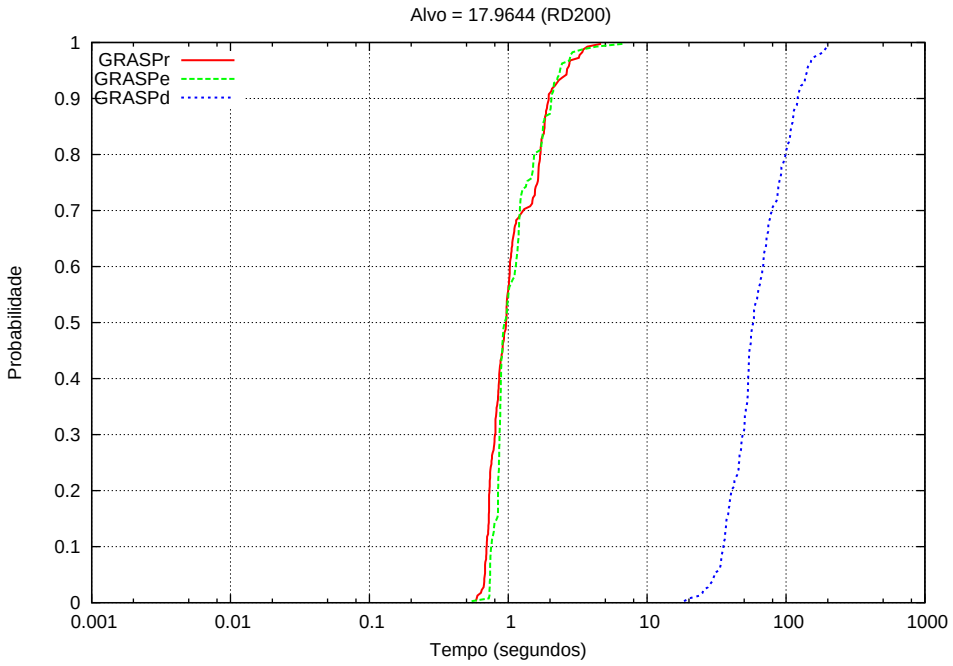


Figura B.4: Distribuição de tempos de execução dos três algoritmos GRASP para a instância RD200 e alvo fácil.

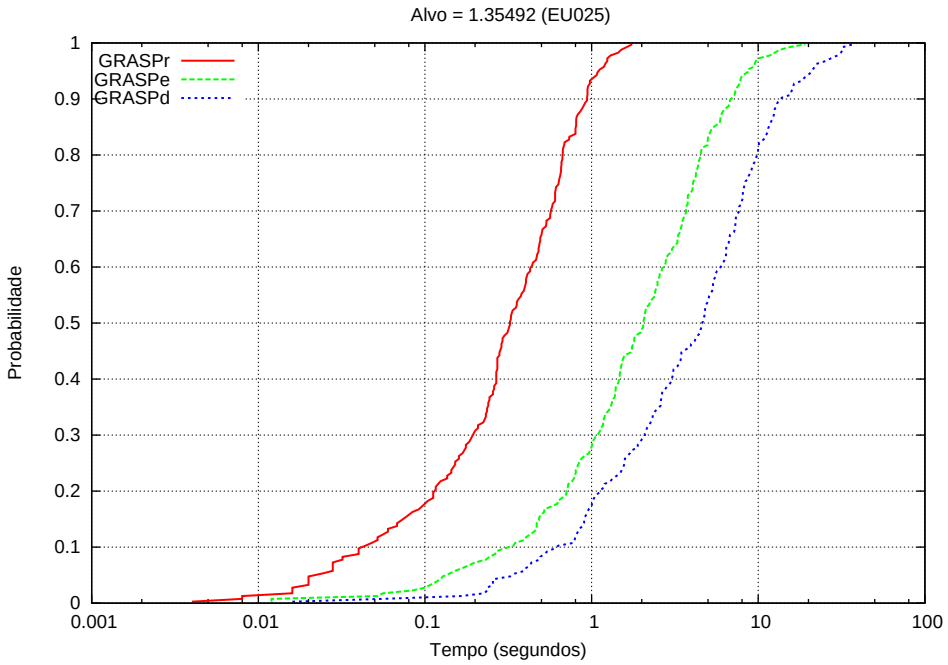


Figura B.5: Distribuição de tempos de execução dos três algoritmos GRASP para a instância EU025 e alvo fácil.

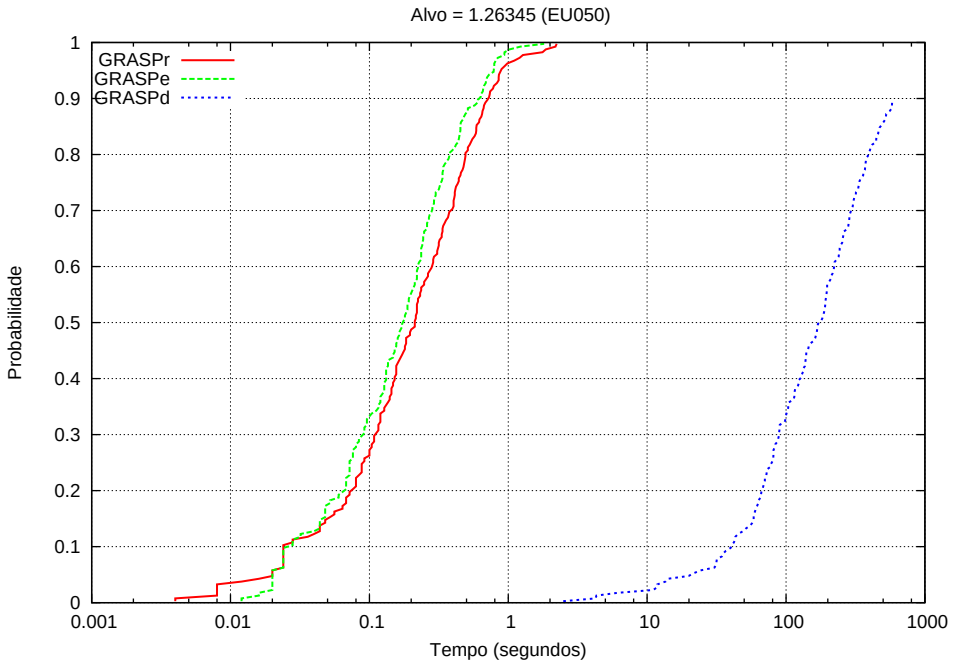


Figura B.6: Distribuição de tempos de execução dos três algoritmos GRASP para a instância EU050 e alvo fácil.

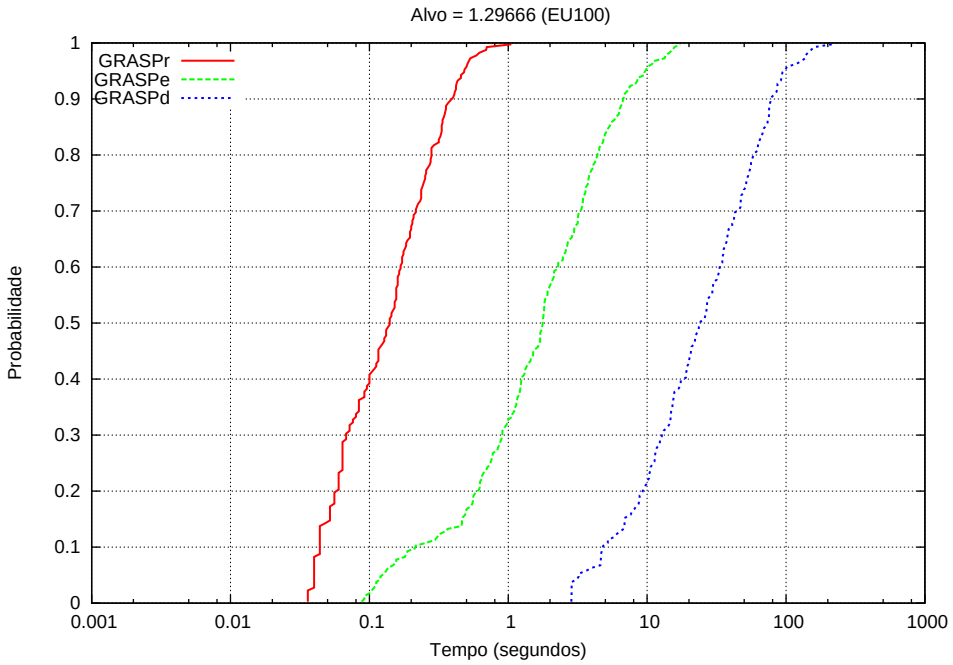


Figura B.7: Distribuição de tempos de execução dos três algoritmos GRASP para a instância EU100 e alvo fácil.

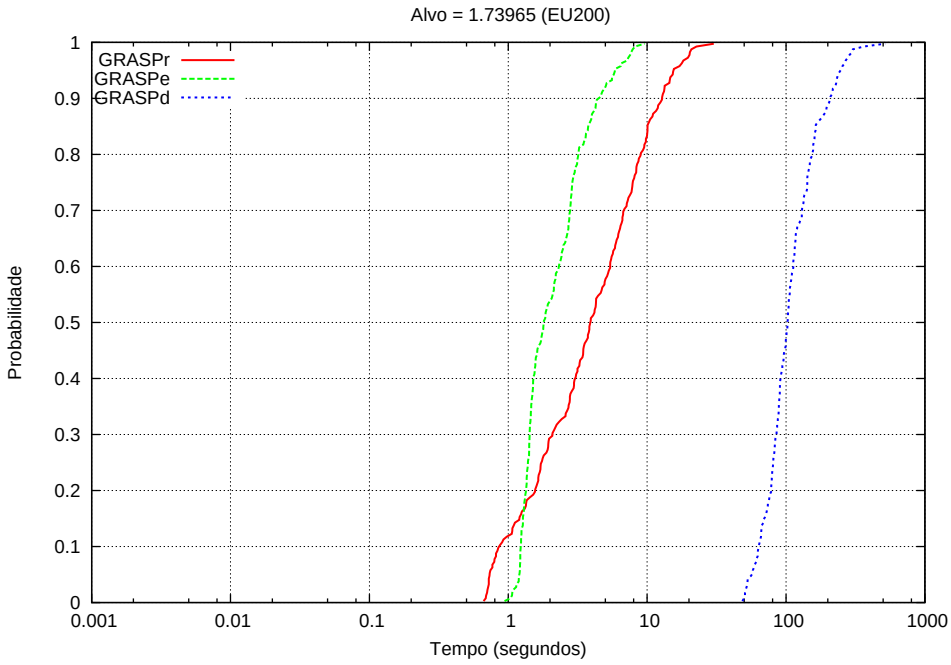


Figura B.8: Distribuição de tempos de execução dos três algoritmos GRASP para a instância EU200 e alvo fácil.

APÊNDICE C – Execuções Longas das Versões de GRASP e GVND.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	5.62314	**5.61990**	**5.61990**	**5.61990**	**5.61990**
GRASPe	**5.61990**	**5.61990**	**5.61990**	**5.61990**	**5.61990**
GRASPd	5.65468	5.64390	**5.61990**	**5.61990**	**5.61990**
GVNDre	5.62639	**5.61990**	**5.61990**	**5.61990**	**5.61990**
GVNDer	**5.61990**	**5.61990**	**5.61990**	**5.61990**	**5.61990**
GVNDred	5.64245	**5.61990**	**5.61990**	**5.61990**	**5.61990**

Tabela C.1: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP e pelos três algoritmos GVND para a instância RD025.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	9.01795	**8.95437**	8.93464	8.92874	8.92341
GRASPe	9.07520	8.97034	8.94156	8.91931	**8.90703**
GRASPd	9.16967	9.09062	8.94780	8.93688	8.92722
GVNDre	**9.01610**	8.99486	8.95196	8.92197	8.91088
GVNDer	9.04711	8.96849	**8.92696**	**8.91556**	**8.90703**
GVNDred	9.12866	9.00779	8.96173	8.94295	8.92245

Tabela C.2: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP e pelos três algoritmos GVND para a instância RD050.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	11.19388	11.13038	11.04206	10.98531	10.94703
GRASPe	1.15398	11.10707	11.05080	10.92308	10.88231
GRASPd	11.49712	11.27728	11.17588	11.13003	11.03263
GVNDre	**11.13763**	11.11716	11.04178	10.93142	**10.87491**
GVNDer	11.15328	**11.08949**	**10.98385**	**10.90096**	10.88207
GVNDred	11.33008	11.21244	11.12901	11.06520	11.02650

Tabela C.3: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP e pelos três algoritmos GVND para a instância RD100.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	17.72396	17.66853	17.62556	17.51775	17.44616
GRASPe	17.73358	**17.63502**	17.60225	17.56595	17.41286
GRASPd	18.00174	17.84695	17.80312	17.66411	17.64906
GVNDre	17.67473	17.63653	**17.58515**	**17.46543**	17.44284
GVNDer	**17.65712**	**17.63502**	17.58921	17.48145	**17.40375**
GVNDred	17.93202	17.77707	17.66735	17.65200	17.58789

Tabela C.4: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP e pelos três algoritmos GVND para a instância RD200.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	**1.35492**	**1.35492**	**1.35492**	**1.35492**	**1.35492**
GRASPe	1.35512	**1.35492**	**1.35492**	**1.35492**	**1.35492**
GRASPd	1.35835	**1.35492**	**1.35492**	**1.35492**	**1.35492**
GVNDre	1.35622	**1.35492**	**1.35492**	**1.35492**	**1.35492**
GVNDer	1.35622	**1.35492**	**1.35492**	**1.35492**	**1.35492**
GVNDred	1.35622	**1.35492**	**1.35492**	**1.35492**	**1.35492**

Tabela C.5: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP e pelos três algoritmos GVND para a instância EU025.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	1.26057	**1.25873**	**1.25866**	**1.25866**	**1.25866**
GRASPe	1.25896	1.25881	**1.25866**	**1.25866**	**1.25866**
GRASPd	1.27690	1.26672	1.26338	1.26159	1.26062
GVNDre	1.26065	1.25881	**1.25866**	**1.25866**	**1.25866**
GVNDer	**1.25888**	**1.25873**	**1.25866**	**1.25866**	**1.25866**
GVNDred	1.26146	1.26065	1.25881	**1.25866**	**1.25866**

Tabela C.6: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP e pelos três algoritmos GVND para a instância EU050.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	1.28856	1.28774	1.28573	1.28512	1.28367
GRASPe	1.29235	1.29040	1.28794	1.28624	1.28468
GRASPd	1.32926	1.29566	1.28940	1.28731	1.28552
GVNDre	1.28850	1.28744	**1.28536**	**1.28459**	**1.28367**
GVNDer	**1.28781**	**1.28712**	1.28588	1.28463	1.28399
GVNDred	1.29603	1.28877	1.28780	1.28607	1.28491

Tabela C.7: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP e pelos três algoritmos GVND para a instância EU100.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GRASPr	1.73658	1.73364	1.72974	1.72841	1.72784
GRASPe	1.73152	1.72941	1.72898	1.72813	1.72770
GRASPd	1.73232	1.73215	1.73199	1.73068	1.72884
GVNDre	1.73070	**1.72892**	**1.72870**	1.72816	1.72754
GVNDer	**1.73039**	1.72900	1.72874	**1.72796**	**1.72742**
GVNDred	1.73363	1.73212	1.73096	1.72923	1.72856

Tabela C.8: Evolução dos resultados obtidos pelas três versões do algoritmo GRASP e pelos três algoritmos GVND para a instância EU200.

APÊNDICE D – Distribuição dos Tempos de Execução das Versões de GRASP e GVND para Alvos Fáceis

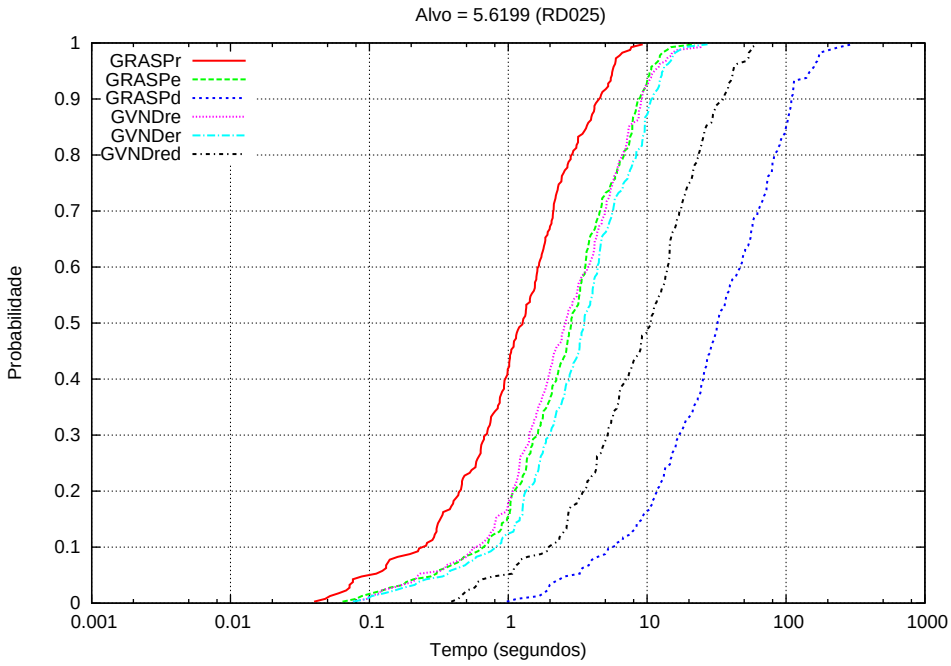


Figura D.1: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância RD025 e alvo fácil.

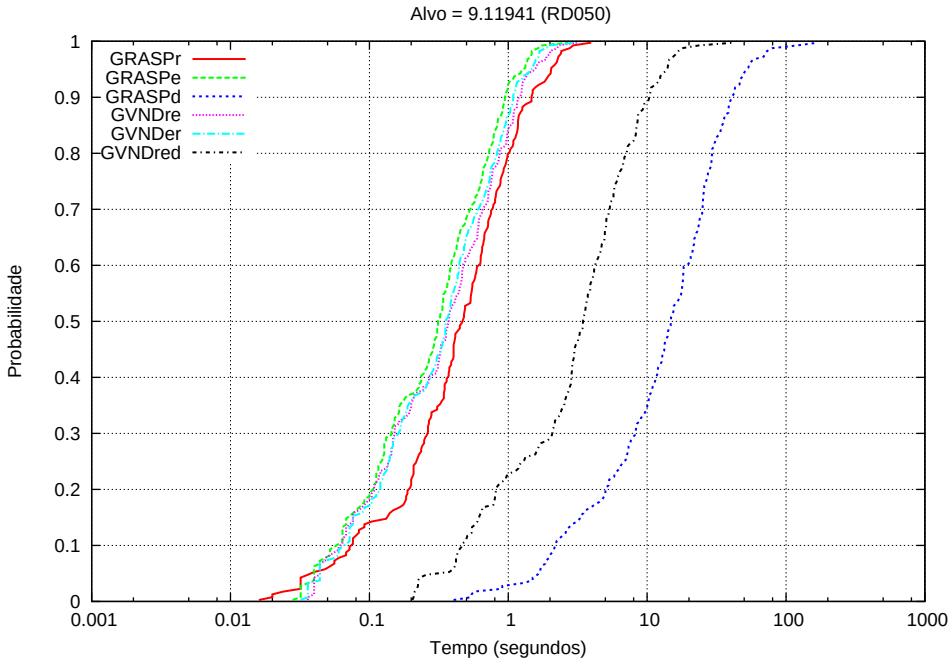


Figura D.2: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância RD050 e alvo fácil.

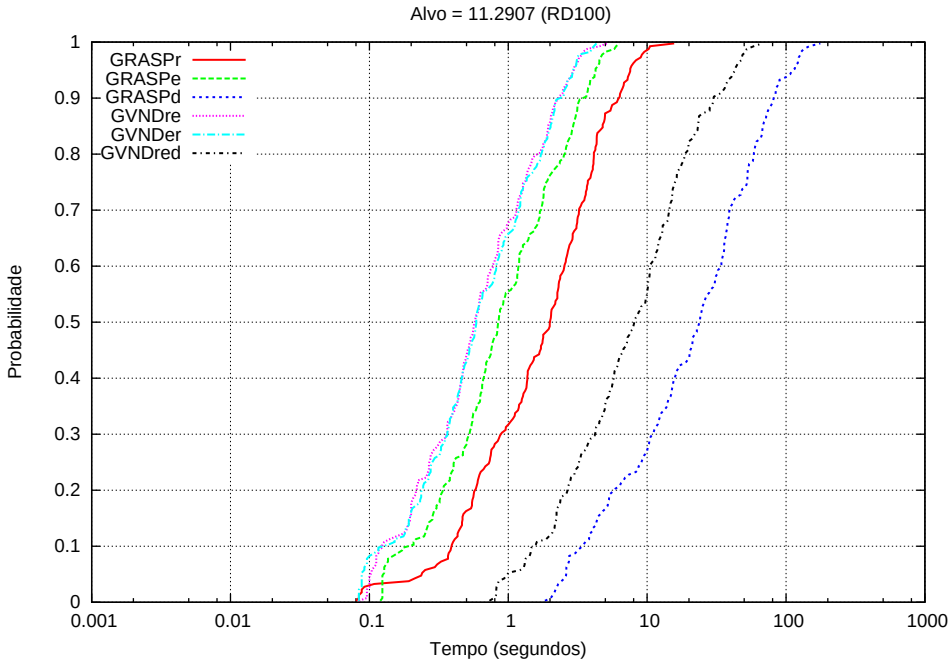


Figura D.3: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância RD100 e alvo fácil.

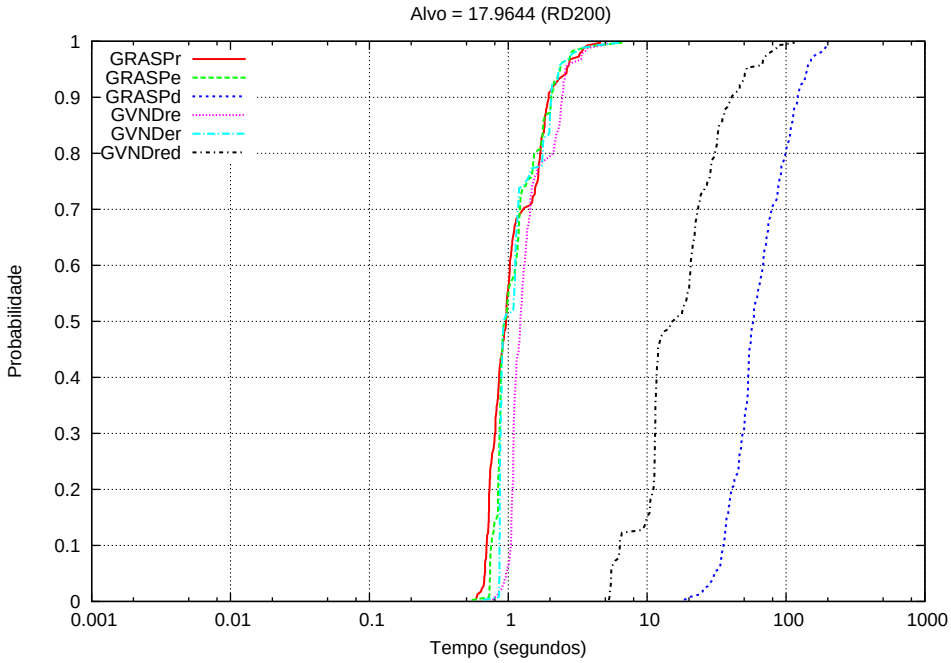


Figura D.4: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância RD200 e alvo fácil.

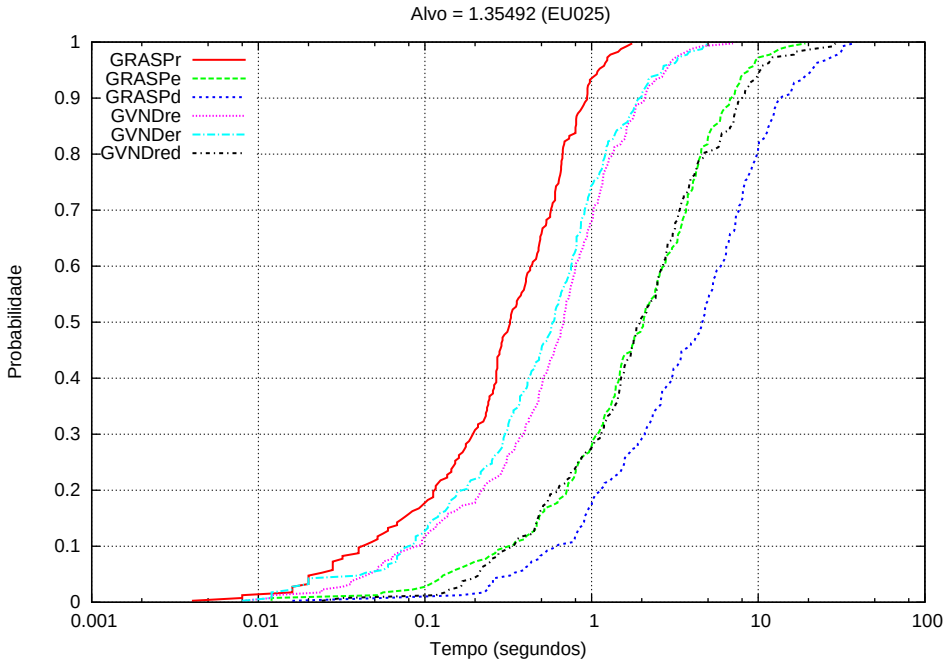


Figura D.5: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância EU025 e alvo fácil.

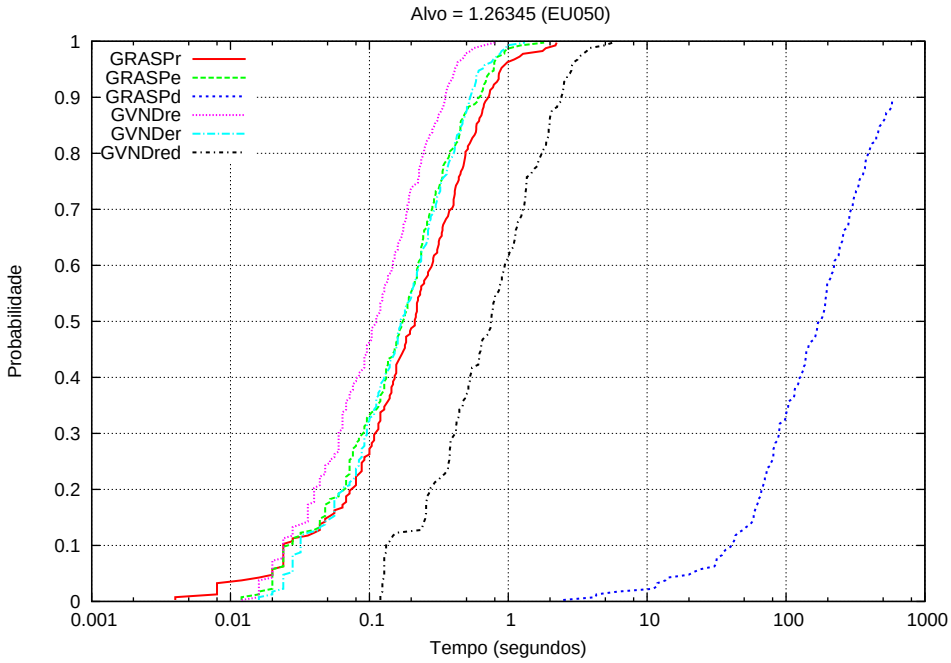


Figura D.6: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância EU050 e alvo fácil.

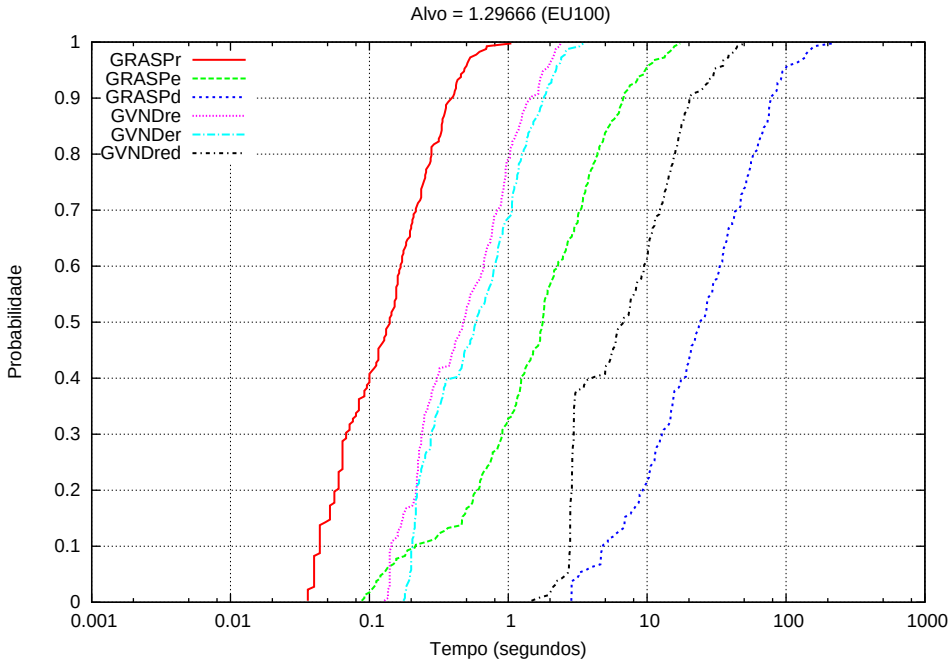


Figura D.7: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância EU100 e alvo fácil.

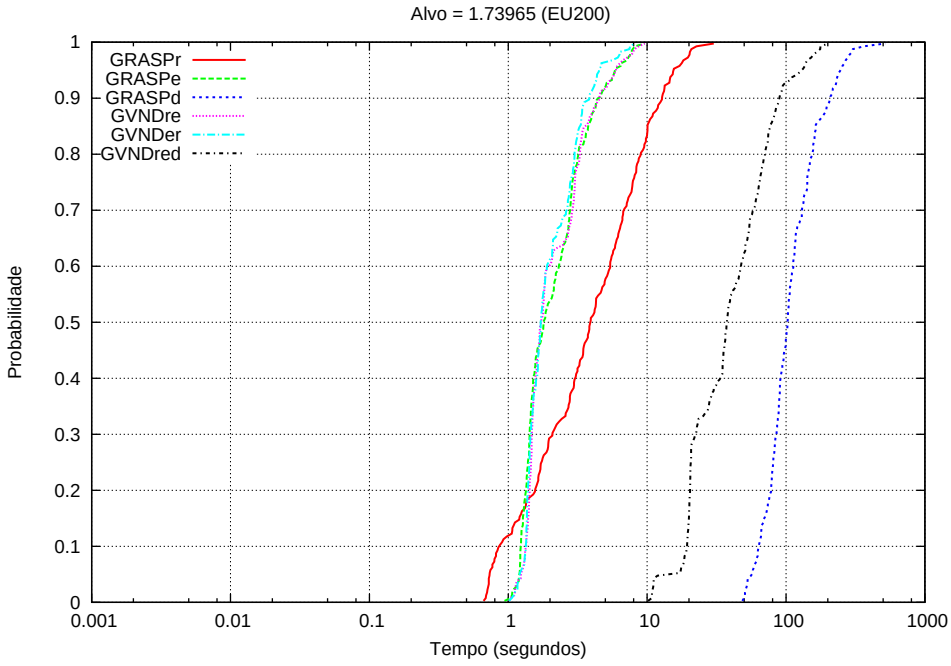


Figura D.8: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância EU200 e alvo fácil.

APÊNDICE E – Distribuição dos Tempos de Execução das Versões de GRASP e GVND para Alvos Médios

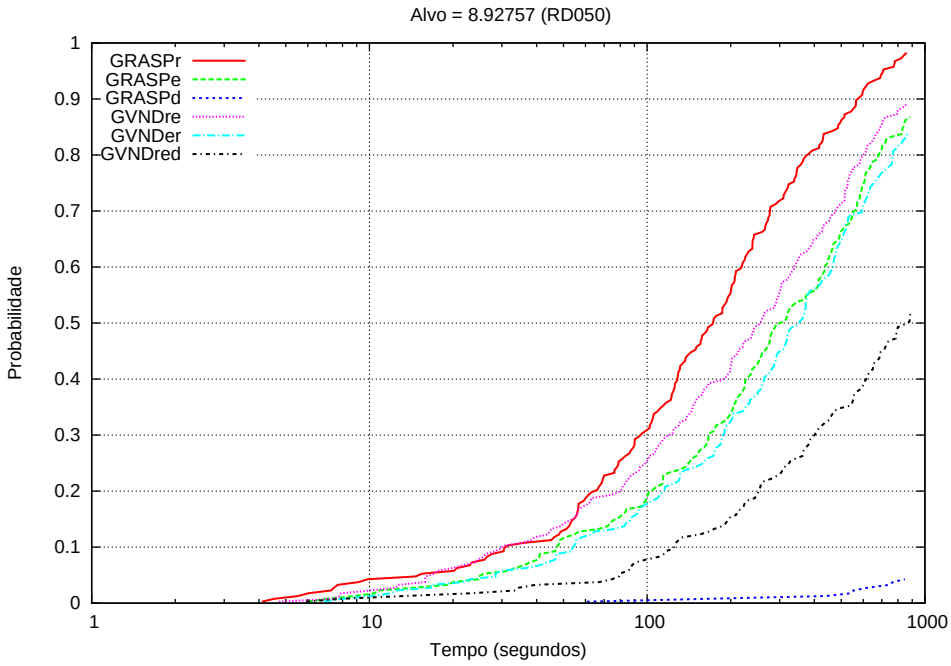


Figura E.1: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância RD050 e alvo com média dificuldade

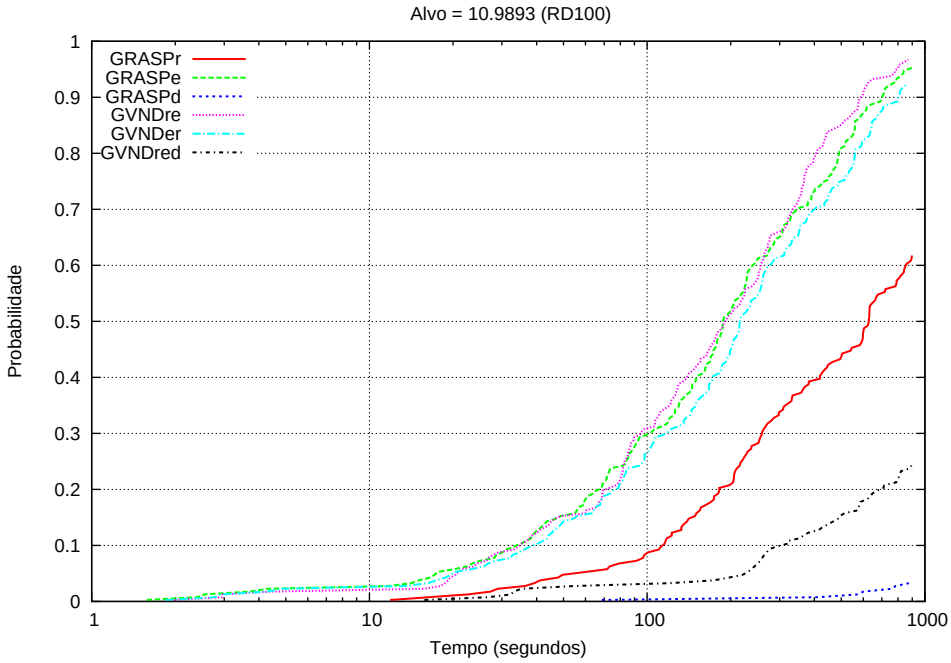


Figura E.2: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância RD100 e alvo com média dificuldade

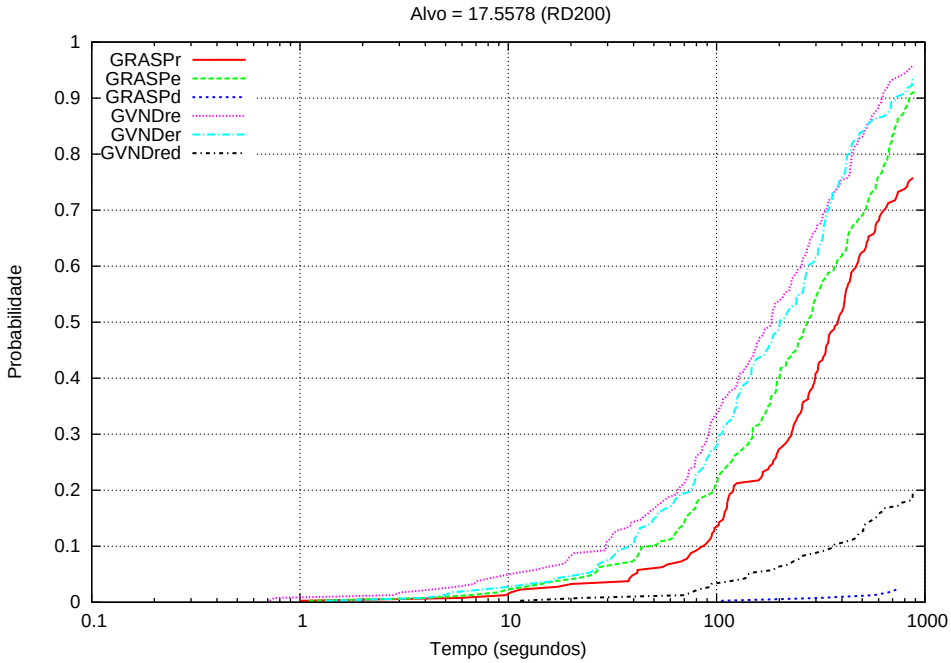


Figura E.3: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância RD200 e alvo com média dificuldade

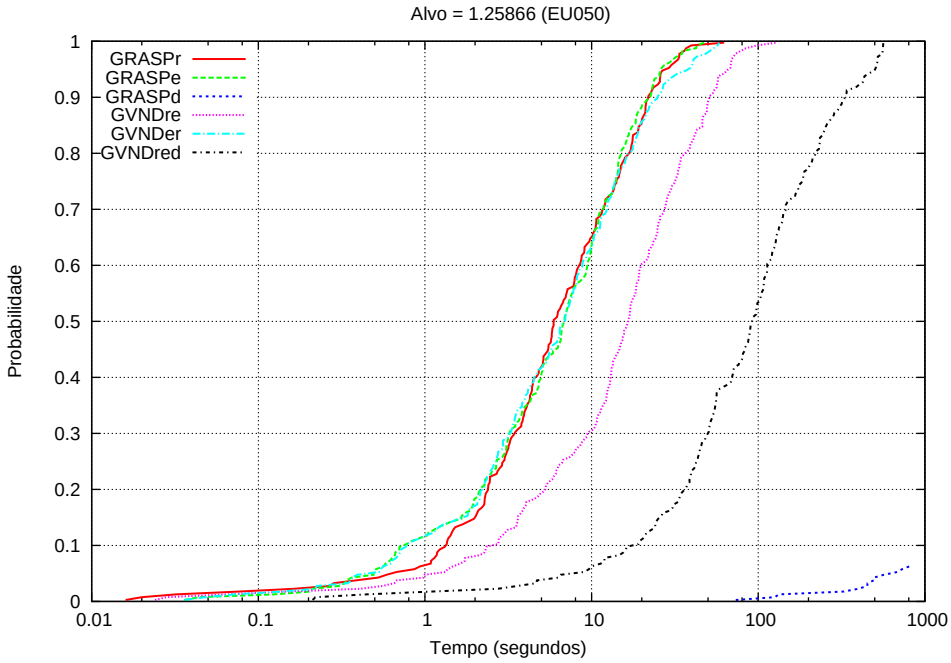


Figura E.4: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância EU050 e alvo com média dificuldade

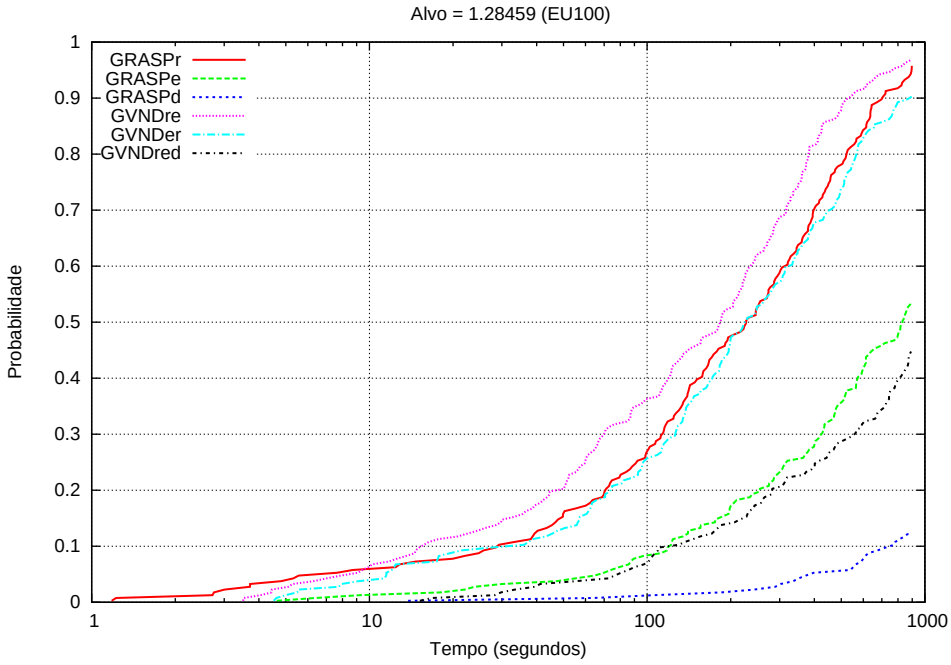


Figura E.5: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância EU100 e alvo com média dificuldade

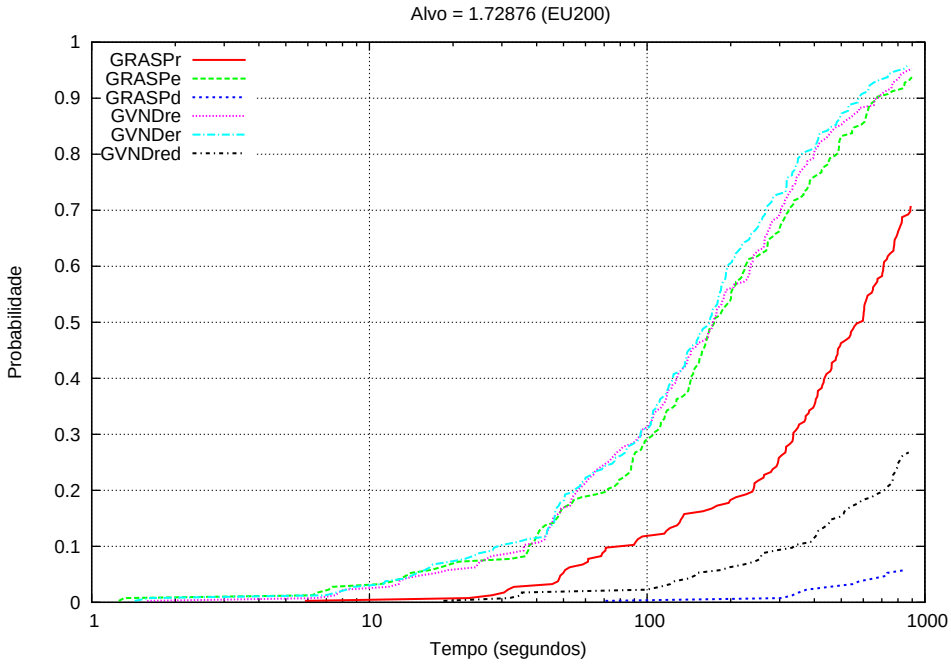


Figura E.6: Distribuição de tempos de execução dos três algoritmos GRASP básicos e dos três algoritmos GVND para a instância EU200 e alvo com média dificuldade

APÊNDICE F – Execuções Longas de GVND com Reconexão por Caminhos

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GVNDre	**9.01610**	8.99486	8.95196	8.92197	**8.91088**
GVNDre(fa)	9.05330	9.00340	8.93816	**8.92010**	8.91357
GVNDre(ta)	9.05330	9.00340	8.93816	**8.92010**	8.91357
GVNDre(tr)	9.05148	**8.97229**	**8.92829**	8.92127	**8.91088**

Tabela F.1: Evolução dos resultados obtidos pelo algoritmo GVNDre e pelos três algoritmos GVND com reconexão por caminhos para a instância RD050.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GVNDre	**11.13763**	**11.11716**	11.04178	**10.93142**	10.87491
GVNDre(fa)	11.19039	11.13870	**11.02355**	10.93226	10.87896
GVNDre(ta)	11.19039	11.13870	**11.02355**	10.93226	10.87896
GVNDre(tr)	11.17425	11.16089	11.03185	10.94830	**10.87203**

Tabela F.2: Evolução dos resultados obtidos pelo algoritmo GVNDre e pelos três algoritmos GVND com reconexão por caminhos para a instância RD100.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GVNDre	**17.67473**	**17.63653**	17.58515	**17.46543**	**17.44284**
GVNDre(fa)	17.72692	17.66103	**17.58429**	17.49754	17.48618
GVNDre(ta)	17.72692	17.66103	**17.58429**	17.49754	17.48618
GVNDre(tr)	17.72838	17.67964	17.61611	17.54567	17.49341

Tabela F.3: Evolução dos resultados obtidos pelo algoritmo GVNDre e pelos três algoritmos GVND com reconexão por caminhos para a instância RD200.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GVNDre	1.26065	1.25881	**1.25866**	**1.25866**	**1.25866**
GVNDre(fa)	**1.25969**	**1.25866**	**1.25866**	**1.25866**	**1.25866**
GVNDre(ta)	1.26058	**1.25866**	**1.25866**	**1.25866**	**1.25866**
GVNDre(tr)	1.26131	1.25954	**1.25866**	**1.25866**	**1.25866**

Tabela F.4: Evolução dos resultados obtidos pelo algoritmo GVNDre e pelos três algoritmos GVND com reconexão por caminhos para a instância EU050.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GVNDre	**1.28850**	1.28744	1.28536	1.28459	1.28367
GVNDre(fa)	1.28887	**1.28633**	**1.28464**	1.28459	1.28398
GVNDre(ta)	1.28887	1.28693	**1.28464**	1.28459	1.28398
GVNDre(tr)	1.28946	1.28712	1.28528	**1.28372**	**1.28303**

Tabela F.5: Evolução dos resultados obtidos pelo algoritmo GVNDre e pelos três algoritmos GVND com reconexão por caminhos para a instância EU100.

	Tempo (s)				
Algoritmo	5	25	125	625	3125
GVNDre	**1.73070**	**1.72892**	**1.72870**	1.72816	1.72754
GVNDre(fa)	1.73169	1.72985	1.72913	1.72827	1.72748
GVNDre(ta)	1.73169	1.72985	1.72913	1.72827	1.72748
GVNDre(tr)	1.73169	1.73048	1.72935	**1.72800**	**1.72742**

Tabela F.6: Evolução dos resultados obtidos pelo algoritmo GVNDre e pelos três algoritmos GVND com reconexão por caminhos para a instância EU200.

APÊNDICE G – Distribuição dos Tempos de Execução de GVND com Reconexão por Caminho para Alvos Médios

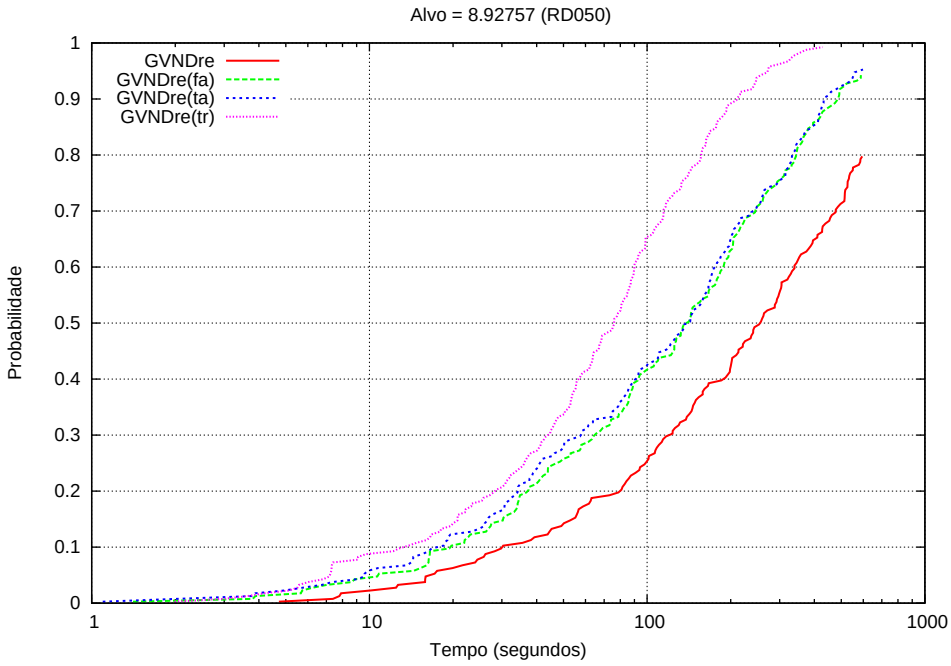


Figura G.1: Distribuição de tempos de execução do algoritmo GVNDre e dos três algoritmos GVND com reconexão por caminhos para a instância RD050 e alvo com média dificuldade.

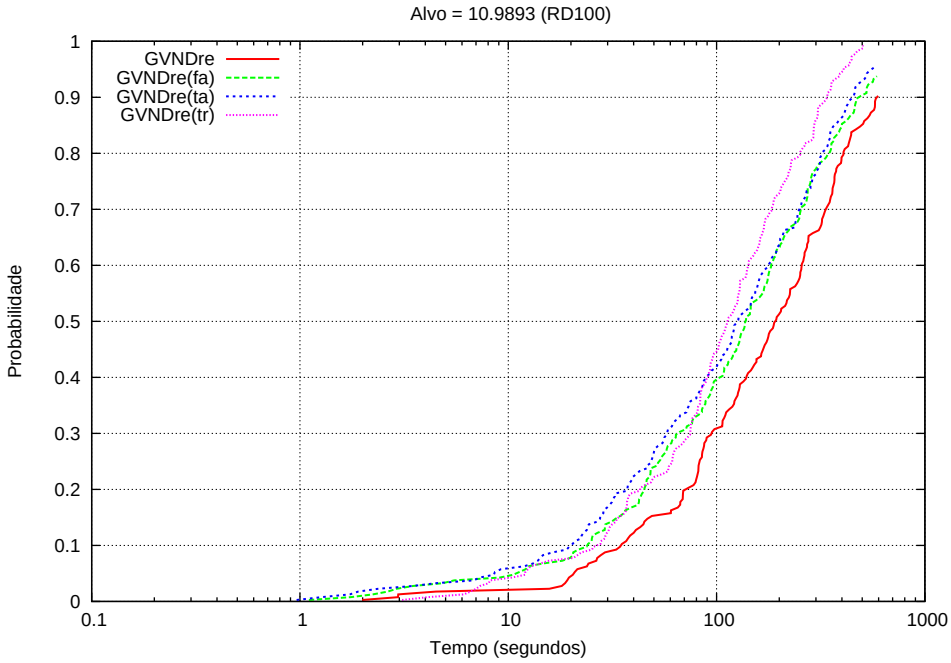


Figura G.2: Distribuição de tempos de execução do algoritmo GVNDre e dos três algoritmos GVND com reconexão por caminhos para a instância RD100 e alvo com média dificuldade.

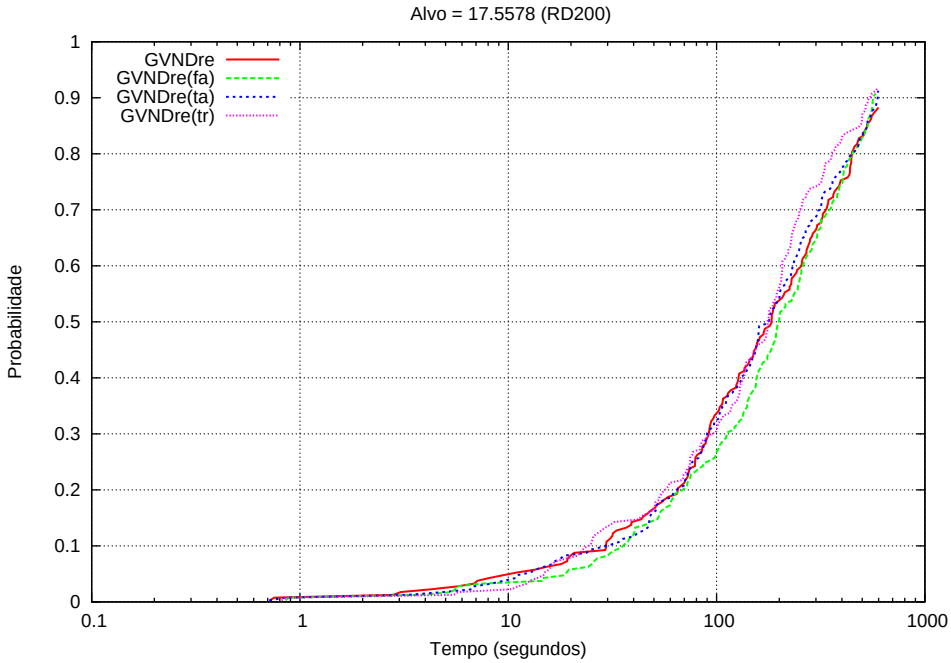


Figura G.3: Distribuição de tempos de execução do algoritmo GVNDre e dos três algoritmos GVND com reconexão por caminhos para a instância RD200 e alvo com média dificuldade.

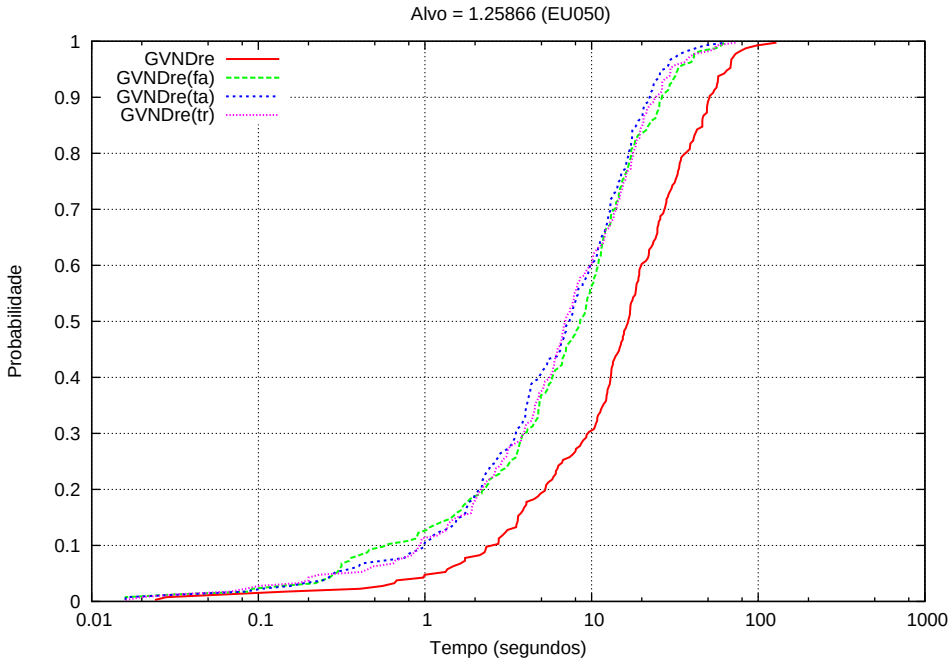


Figura G.4: Distribuição de tempos de execução do algoritmo GVNDre e dos três algoritmos GVND com reconexão por caminhos para a instância EU050 e alvo com média dificuldade.

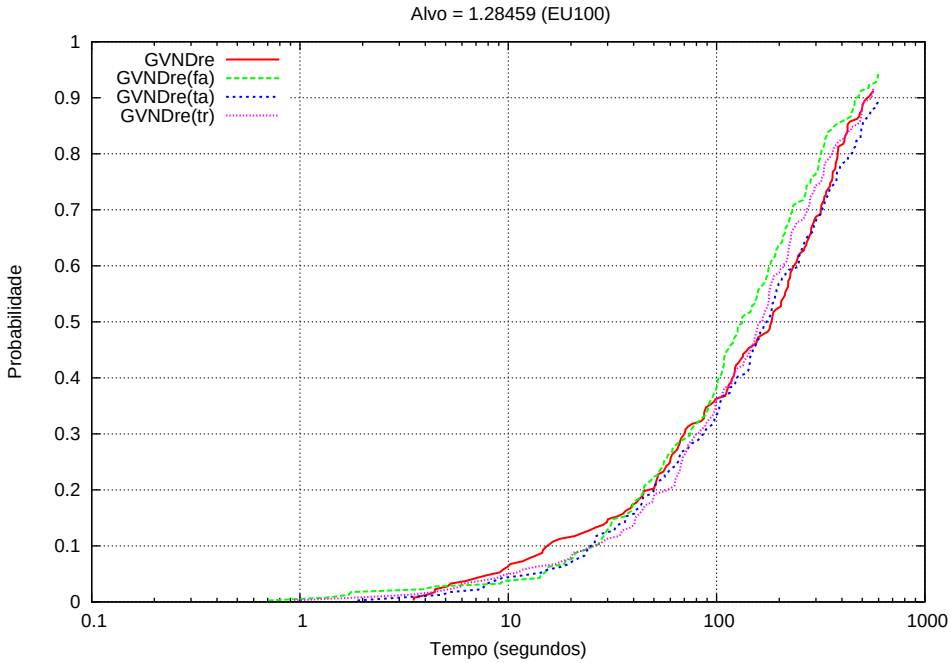


Figura G.5: Distribuição de tempos de execução do algoritmo GVNDre e dos três algoritmos GVND com reconexão por caminhos para a instância EU100 e alvo com média dificuldade.

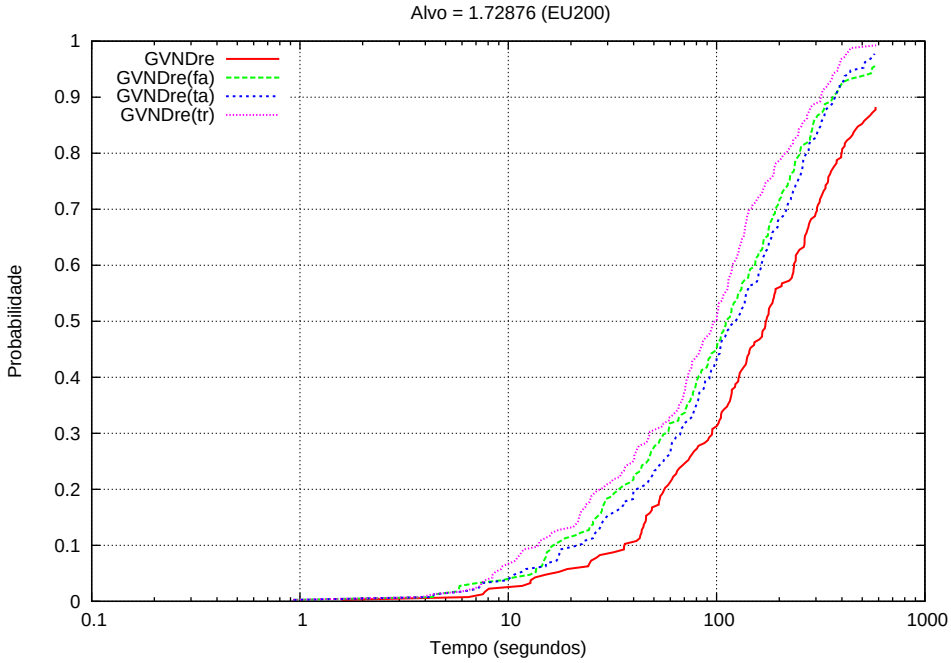


Figura G.6: Distribuição de tempos de execução do algoritmo GVNDre e dos três algoritmos GVND com reconexão por caminhos para a instância EU200 e alvo com média dificuldade.

APÊNDICE H – Distribuição dos Tempos de Execução de GVND com Reconexão por Caminho para Alvos Difíceis

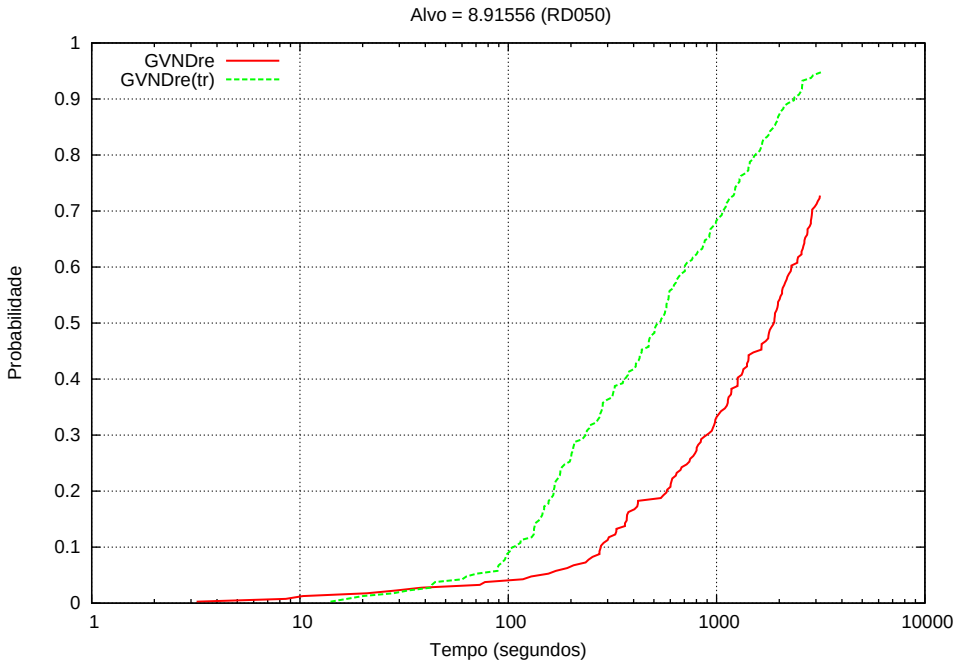


Figura H.1: Distribuição de tempos de execução dos algoritmos GVNDre e GVNDre(tr) para a instância RD050 e alvo difícil.

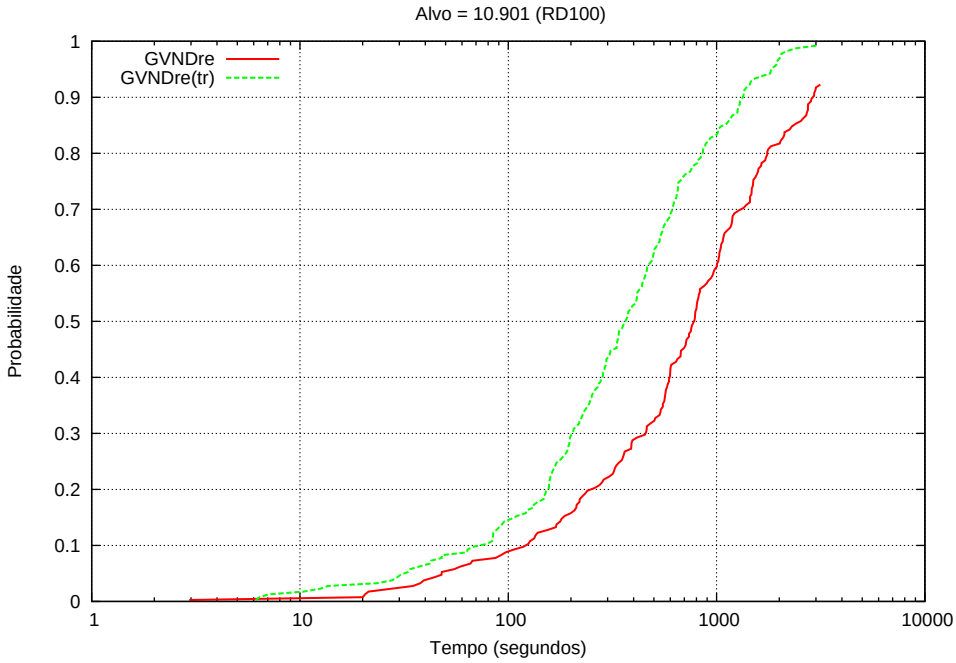


Figura H.2: Distribuição de tempos de execução dos algoritmos GVNDre e GVNDre(tr) para a instância RD100 e alvo difícil.

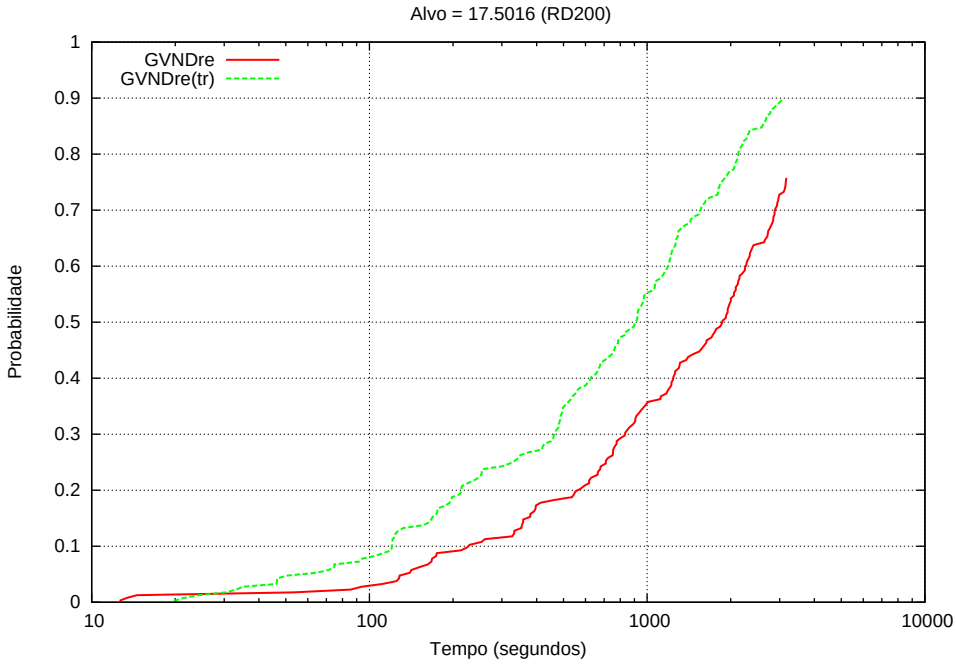


Figura H.3: Distribuição de tempos de execução dos algoritmos GVNDre e GVNDre(tr) para a instância RD200 e alvo difícil.

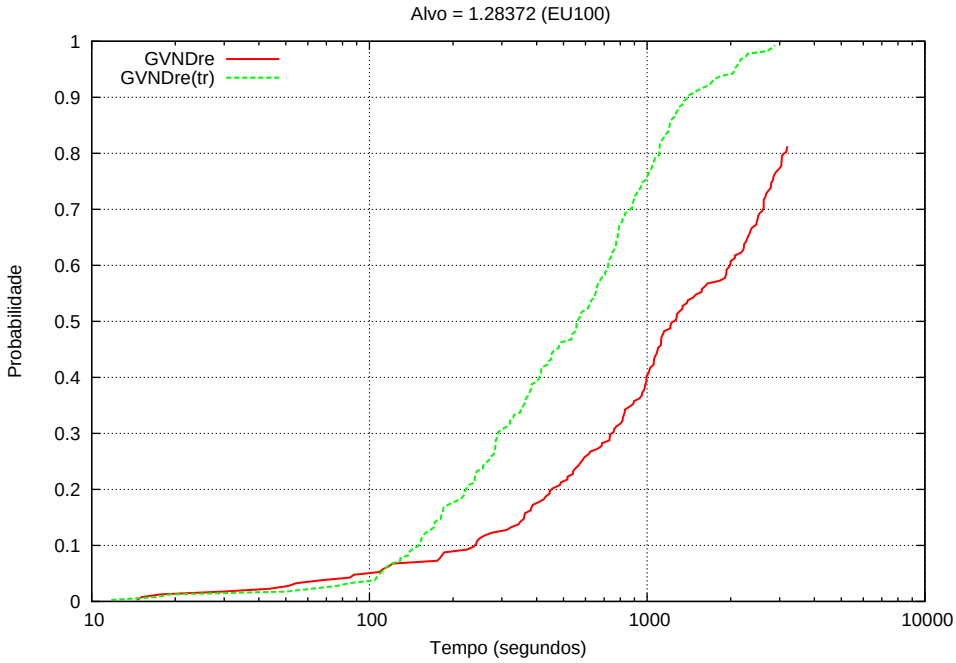


Figura H.4: Distribuição de tempos de execução dos algoritmos GVNDre e GVNDre(tr) para a instância EU100 e alvo difícil.

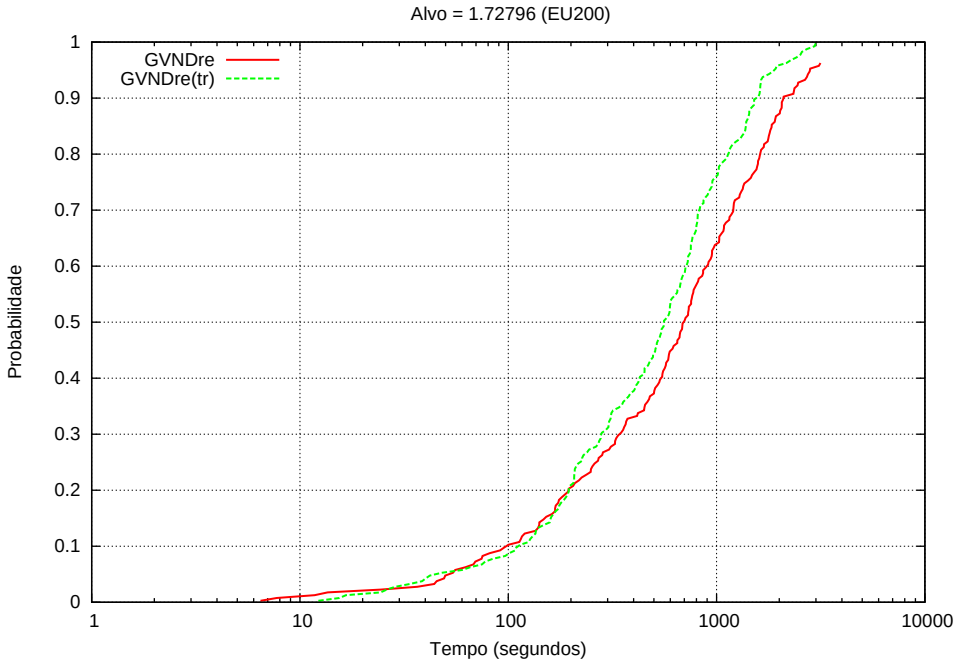


Figura H.5: Distribuição de tempos de execução dos algoritmos GVNDre e GVNDre(tr) para a instância RD200 e alvo difícil.