ABORDAGEM DE REFINAMENTO ITERATIVO PARA O PROBLEMA DA ÁRVORE GERADORA COM NÚMERO MÍNIMO DE VÉRTICES BRANCH

DIEGO MELLO DA SILVA

Orientador: Geraldo Robson Mateus

Coorientador: Ricardo Martins Abreu Silva

ABORDAGEM DE REFINAMENTO ITERATIVO PARA O PROBLEMA DA ÁRVORE GERADORA COM NÚMERO MÍNIMO DE VÉRTICES BRANCH

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Minas Gerais. Departamento de Ciência da Computação como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação.

Belo Horizonte

Março de 2011

Silva, Diego Mello da
S586a Abordagem de refinamento iterativo para o problema da árvore geradora com número mínimo de vértices Branch / Diego Mello da Silva. — Belo Horizonte, 2011
xxii, 108 f. : il. ; 29cm

Dissertação (mestrado) — Universidade Federal de Minas Gerais. Departamento de Ciência da Computação
Orientador: Geraldo Robson Mateus
Coorientador: Ricardo Martins Abreu Silva

1. Computação - Teses. 2. Teoria dos Grafos - Teses. I. Orientador. II. Título.

CDU 519.6*62 (043)

UNIVERSIDADE FEDERAL DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

FOLHA DE APROVAÇÃO

Abordagem de refinamento iterativo para o problema da árvore geradora com número mínimo de vértices branch

DIEGO MELLO DA SILVA

Dissertação defendida e aprovada pela banca examinadora constituída pelos Senhores:

PROF. GERALDO ROBSON MATEUS - Orientador
Departamento de Ciência da Computação - UFMG

PROF. RICARDO MARTINS DE ABREU SILVA - Co-orientador
Departamento de Ciência da Computação - UFLA

PROF. MAURÍCIO GUILHERME DE CARVALHO RESENDE
AT&T Labs Research

PROF. SEBASTIÁN ALBERTO URRUTIA
Departamento de Ciência da Computação - UFMG

Belo Horizonte, 02 de março de 2011.

Dedico este trabalho à Wanda e Otávio, meus verdadeiros tesouros.

Agradecimentos

Em primeiro lugar, agradeço a Deus por todas as portas que ele permitiu que se abrissem em minha vida, me dando forças para prosseguir à cada dificuldade que aparece.

Agradeço à meus pais Jair e Fátima por terem acreditado que eu era capaz e batalhado duro para que, anos atrás, eu realizasse o sonho da minha vida. Serei eternamente grato.

Agradeço igualmente à minha esposa Wanda, que muito me incentivou a continuar estudando quando o comodismo já tinha tomado conta da minha vida. Obrigado por relembrar meus sonhos e ideais, e me mostrar que com disciplina e vontade tudo é possível. Sem seu apoio durante todos esses anos, eu jamais teria chegado aqui.

Agradeço ao meu filho Otávio pela paciência de esperar, todos os finais de semana dos últimos dois anos, por alguns raros minutos de atenção, minutos estes que às vezes insistia em usar para me 'ajudar' a concluir minha pesquisa.

Agradeço aos amigos Robert Subtil e Júlio Alves pelo companheirismo durante esse último ano, pelas conversas sobre o presente e o futuro, pelas produtivas discussões sobre computação, mercado, inovação, pesquisa e otimização e, principalmente, pela capacidade de aturar meu comportamento inconstante. Agradeço o amigo Rafael Frinhani pelas palavras de incentivo, postura e filosofia de vida que, muitas vezes, foram exemplo a seguir.

Agradeço à Devex Tecnologia SA, em especial à Thomaz Nascimento e Guilherme Bastos, por terem me liberado para fazer as disciplinas do programa de mestrado, e aos amigos da Diretoria de Produtos e Inovação da Devex pelo excelente ambiente de trabalho propício ao desenvolvimento das soluções mais engenhosas.

Agradeço ao Programa de Pós Graduação em Ciência da Computação do DCC-UFMG, seus funcionários e professores, pela excelência do programa. Agradeço à CAPES pelo MINTER UFLA/UFMG, que me permitiu investir dois anos de minha vida para realizar mais um de meus sonhos. Agradeço os colegas do MINTER pela troca de experiências, em especial aos amigos Frank, Gabriel, Tiago e Erasmo. Agradeço os colegas do Laboratório de Pesquisa Operacional, pelo privilégio de compartilhar boas idéias no pouco contato que tivemos. Aos professores da banca examinadora, Sebastián

Urrutia e Maurício Resende, agradeço por terem aceitado o convite para participar de minha defesa e pelas análises e contribuições dadas para que esse trabalho pudesse atingir o patamar de qualidade que chegou.

Agradeço especialmente os meus orientadores Geraldo Robson e Ricardo Martins pela oportunidade de trabalhar com pesquisadores competentes, cujo zelo e acompanhamento constante puderam resultar neste trabalho. Que este primeiro contato com a pesquisa científica não seja o último!

"Os sonhos não envelhecem"
(Márcio Borges)

Resumo

O *Problema da Árvore Geradora com Número Mínimo de Vértices Branch* (do inglês, *Minimum Branch Vertices Problem* ou MBV) consiste em, dado um grafo $G = (V, E)$ conexo, não direcionado e não valorado, encontrar a árvore geradora T dentre todas as árvores geradoras de G que possui a menor quantidade de vértices com grau maior ou igual à 3, denominados vértices *branch*. Tal problema surge na tomada de decisão sobre onde alocar *switches* WDM especiais no projeto de redes ópticas *multicast*, e foi provado ser da classe $\mathcal{NP}$-Completo.

Neste trabalho o problema é investigado por meio de uma proposta de heurística que baseia-se na abordagem de Refinamento Iterativo (IR), onde uma árvore geradora irrestrita é modificada usando o artifício de substituição de arcos considerados infratores em T por arcos de G de forma que a sua topologia original seja ajustada para possuir a menor quantidade de vértices *branch* possível. Experimentos foram realizados sobre 6 diferentes conjuntos de instâncias, comparando-se os resultados pelo algoritmo IR proposto com os resultados de duas outras heurísticas existentes para o problema. A análise dos resultados experimentais sugere que o algoritmo IR pode encontrar soluções de melhor qualidade do que estas heurísticas conforme a densidade de G aumenta.

Palavras-chave: Árvore Geradora Restrita, MBV, Heurística, Refinamento Iterativo, Teoria dos Grafos

Abstract

Given a connected, undirected, unweighted graph $G = (V, E)$ the *Minimum Branch Vertices Problem* (MBV) consists in finding a spanning tree T of G that contains the minimum number of vertices with degree greater than or equal to 3, also called *branch* vertices. This problem arises in the design of optical multicast networks when it is necessary to decide where to allocate a special kind of WDM switch, called *light-splitting switches*. This problem is proved to be $\mathcal{NP}$-Complete.

In this work the MBV problem has been investigated with a new heuristic based on the Iterative Refinement Approach (IR), where an unconstrained spanning tree is changed using a edge-replacement strategy until the tree topology achieves a minimum number of branch vertices. Experiments were done applying the IR algorithm over six sets of instances, and the results were compared with two other heuristics for the MBV problem. The analysis of the results suggest that the IR algorithm can find better solutions than these heuristics as the graph density increases.

Keywords: Contrained Spanning Tree, MBV, Heuristic, Iterative Refinement, Graph Theory

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Diversas situações do mundo real podem ser descritas através de diagramas formados por um conjunto de pontos e linhas ligandos certos pares desses pontos, onde os pontos podem representar entidades, e as linhas podem representar algum tipo de relação entre essas entidades. Por exemplo, se os pontos representam pessoas, então as linhas podem juntar pessoas que são amigas; se os pontos representam centros de comunicação, as linhas podem representar *links* de comunicação. A abstração matemática de situações desse tipo originou o conceito matemático de grafo (Bondy e Murty [1976]).

Um grafo consiste em um modelo matemático formado por um conjunto de vértices e um conjunto de arestas que representam algum tipo de interação entre esses vértices. Formalmente, um grafo pode ser definido como um par $G = (V, E)$ de conjuntos que satisfazem $E \subseteq V \times V$, de forma que cada elemento de E é composto por dois elementos de V. Os elementos de V são denominados vértices ou nós, e os elementos de E são denominados arcos ou arestas (Diestel [2000]). Segundo Bondy e Murty [1976], grafos possuem essa denominação porque podem ser representados graficamente, e sua representação gráfica nos permite entender muitas de suas propriedades.

Diversos são os algoritmos existentes na literatura capazes de resolver problemas em grafos. Encontrar os componentes conexos de um grafo, realizar buscas em profundidade e em largura em grafos, determinar a menor distância entre os vértices do grafo, determinar o número cromático de um grafo, verificar se um dado grafo é planar ou não, e calcular árvores geradoras de um grafo são exemplos de problemas clássicos abordados pela Teoria dos Grafos em diversos livros da área. Em Bondy e Murty [1976] são apresentados diversos problemas e fundamentos teóricos e matemáticos envolvendo coloração de grafos, conectividade, caminhos, planaridade de grafos, redes e fluxos. Em Golumbic e Hartman [2005] são apresentados fundamentos teóricos e algoritmos para algumas aplicações do mundo real modeladas como grafos. Em Cormen et al. [2001] são descritos algoritmos e análise de complexidade para vários problemas de grafos.

Em especial, o problema de encontrar uma árvore geradora de um grafo G tem sido utilizado amplamente em uma série de aplicações, que variam desde seu uso em heurísticas para resolver problemas $\mathcal{NP}$-Completo, como é o caso do método aproximado baseado na desigualdade de triângulos para o problema do caixeiro viajante (Cormen et al. [2001]), até modelagem de redes de comunicação (Cerulli et al. [2009]), uma vez que para muitos problemas envolvendo redes de computadores o *backbone* da rede pode ser aproximado por uma árvore geradora (Golumbic e Hartman [2005]).

No campo da otimização em grafos, essa importância se mantém. Ahuja et al. [1993] destacam que árvores geradoras possuem papel fundamental no campo de fluxos em rede, pois aparecem em diversas ocasiões quando solucionando problemas desse tipo. Como exemplo, problemas de fluxo de custo mínimo sempre possuem árvores geradoras como solução; o algoritmo *network-simplex* para resolver problemas de fluxo de custo mínimo é de certa forma um algoritmo de manipulação de árvores geradoras que iterativamente move-se de uma árvore geradora para outra, incluindo um novo arco no lugar de outro. Bazaraa et al. [1990] correlacionam uma árvore geradora com cada base quando caracterizam um *tableau* para o problema clássico de transporte. Essa mesma correlação é apresentada em Bertsimas e Tsitsiklis [1997] quando caracterizam matrizes básicas.

De acordo com Golumbic e Hartman [2005], o problema mais simples de otimização em grafos consiste no problema da árvore geradora mínima (do inglês, *minimum spanning trees* ou MST), enunciado como: dado um grafo $G = (V, E)$, não direcionado e valorado nas arestas, encontrar uma árvore geradora cuja soma dos custos de seus arcos é mínima. Em Graham e Hell [1985] encontramos um pouco sobre a história desse problema, onde os autores comparam os trabalhos desenvolvidos independentemente na França, Polônia e antiga Tchecoslováquia antes da publicação dos trabalhos clássicos de Borůvka (1926), Kruskal (1956) e Prim (1957) e os relacionam com os avanços recentes sobre o problema MST. Em Bazlamaçci e Hindi [2001] é apresentada uma comparação entre os algoritmos clássicos conhecidos e os modernos algoritmos de Cheriton e Tarjan [1976], Fredman e Tarjan [1987] e Karger et al. [1995] e suas estruturas de dados complexas e eficientes, difíceis de implementar. Embora os métodos apresentados por esses autores variam em complexidade, todos são polinomiais. No contexto de modelagem de problemas inteiros em programação matemática, Bertsimas e Tsitsiklis [1997] apresentam duas formulações clássicas para esse problema, denominadas *subtour elimination formulation* e *cutset formulation*, ambas com número exponencial de restrições. Ahuja et al. [1993] comentam que é possível dar uma formulação polinomial do problema com a inclusão de novas variáveis de fluxo multiproduto na formulação de eliminação de subciclos.

Ainda relacionado ao tema existem problemas de encontrar árvores geradoras restritas com diversas aplicações no mundo real. Tais problemas geralmente envolvem encontrar uma árvore geradora de um grafo $G = (V, E)$ que acrescenta restrições às formulações acima comentadas. Exemplos desse tipo de problema incluem o problema *degree-constrained minimum spanning tree* (Narula e Ho [1980]) que consiste em encontrar uma árvore geradora de custo mínimo T em G tal que o grau de cada vértice em T seja de no máximo uma constante d; e o problema *diameter-constrained minimum spanning tree* que consiste em encontrar uma árvore geradora T de menor custo dentre todas as árvores geradoras de G tal que não existam caminhos em T com mais do que k arcos entre quaisquer par de vértices (Noronha et al. [2008]). Garey e Johnson [1979] publicaram uma lista de problemas de tempo não-polinomiais envolvendo árvores geradoras que cobrem grande parte dos problemas clássicos conhecidos na literatura.

O objetivo dessa dissertação é o estudo de um problema que também está relacionado com a construção de árvores geradoras restritas. Motivado por um problema de alocação de *switches* em redes ópticas, Gargano et al. [2002] enunciaram um novo problema denominado *minimum branch vertices problem* (MBV): dado um grafo $G = (V, E)$ não direcionado e não valorado, encontrar uma árvore geradora T que possui o menor número de vértices com grau maior ou igual à três (denominados vértices *branch*). Cerulli et al. [2009] estudaram esse problema e propuseram um modelo de programação inteira mista e heurísticas baseadas em técnicas de coloração de vértices e ponderação de arcos.

Como o problema é demonstrado ser da classe $\mathcal{NP}$-Completo (Gargano et al. [2002]), é pouco provável que existam algoritmos polinomiais capazes de resolvê-lo na otimalidade. Unindo isso ao fato do problema ser de ordem prática, recente e pouco explorado, torna-se necessário investir no projeto de novos algoritmos capazes de resolvê-lo para grandes instâncias com soluções de boa qualidade; daí a motivação para esse trabalho.

A dissertação está estruturada em capítulos para melhor organização de seu conteúdo. No Capítulo 2, o problema objeto de estudo desse trabalho será apresentado formalmente, assim como resultados que demonstram sua complexidade. No Capítulo 3 apresentaremos o estado da arte publicado na literatura para o problema da árvore geradora com número mínimo de vértices *branch* (MBV), que incluem modelo matemático e heurísticas (Cerulli et al. [2009]). No Capítulo 4 introduziremos a abordagem de refinamento iterativo (IR) para árvores geradoras restritas (Deo e Kumar [1997]). No Capítulo 5 um algoritmo inspirado na abordagem IR para o problema MBV será apresentado. Por fim, o Capítulo 6 detalhará os experimentos realizados para comparação de alguns métodos documentados para o MBV com o algoritmo IR

proposto, sendo a conclusão do trabalho apresentada no Capítulo 7.

Capítulo 2

O Problema da Árvore Geradora com Número Mínimo de Vértices *Branch*

Esse capítulo apresenta os fundamentos matemáticos do Problema da Árvore Geradora com Número Mínimo de Vértices *Branch*, sua descrição formal e complexidade.

2.1 Fundamentos

Essa seção tem por objetivo definir conceitos usados nesse trabalho e referenciados nas próximas seções do capítulo. A maioria deles são conceitos conhecidos de teoria dos grafos; entretando alguns novos conceitos sobre o problema estudado precisam ser esclarecidos para um melhor entendimento da seção referente à complexidade.

No decorrer do texto, $G = (V, E)$ refere-se a um grafo conexo, não direcionado e não valorado. Considere E' um sub-conjunto de arcos de E. Um *subgrafo induzido por arestas* $G' = (V', E')$ de G é o subgrafo G' cujo conjunto de vértices V' é o conjunto dos extremos dos arcos $e \in E'$ e cujo conjunto de arcos é o próprio conjunto E' (Bondy e Murty [1976]). O subgrafo G' é também denominado um *subgrafo gerador* de G quando $V' = V$ (Diestel [2000]).

Uma *árvore* T é um grafo minimamente conectado e maximalmente acíclico, isto é, T é conexo mas $T \setminus \{(i, j)\}$ é desconexo para todo arco $(i, j) \in T$, e T não contém ciclos mas $T \cup \{(i, j)\}$ contém para quaisquer vértices não adjacentes $i, j \in T$. Se T é acíclico e conecta todos os vértices de um grafo G, chamamos T de *árvore geradora*, uma vez que ela 'gera' o grafo G. Qualquer subgrafo gerador minimamente conectado será uma árvore e, portanto, todo grafo conexo contém uma árvore geradora. Um caminho em G que contém cada vértice de G é um *caminho hamiltoniano*. Um vértice que separa dois outros vértices do mesmo componente conexo é um *vértice de corte*. Se todos os vértices de um grafo G têm grau k, então G é chamado de *k-regular*, ou simplesmente

regular; um grafo 3-*regular* é também chamado de *cúbico* (Diestel [2000]).

Os próximos conceitos advém de Gargano et al. [2002], e são importantes para o entendimento do problema e sua complexidade.

Seja $\delta(v)$ o grau do vértice $v \in V$. Um vértice *branch* de G é um vértice de grau maior do que 2, isto é, $\delta(v) \geq 3$. Seja $s(G)$ o menor número de vértices *branch* em qualquer árvore geradora de G. Uma árvore geradora T de G não possuirá nenhum vértice *branch* (isto é, $s(G) = 0$) se e somente se G admitir um caminho hamiltoniano. Uma árvore geradora com no máximo 1 vértice *branch* é denominada uma *aranha* (do inglês, *spider*). A Figura 2.1 ilustra 3 árvores geradoras que são grafos *aranha*; a árvore geradora (b) é também um caminho hamiltoniano. Os vértices destacados são vértices *branch*.

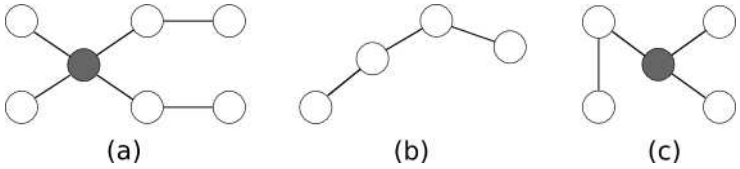

Figura 2.1. Exemplos de árvores geradoras do tipo *spider*

Um grafo G com $s(G) \leq 1$ admite um subgrafo gerador que é um grafo *aranha*, ou seja, existe uma árvore geradora T em G com no máximo um vértice *branch*; dizemos então que G admite uma *aranha geradora* (do inglês, *spanning spider*). A Figura 2.2 apresenta um grafo G em (a) com 9 vértices e 14 arcos. As árvores geradoras destacadas em (b) e (c) são exemplos de subgrafos geradores de G com, respectivamente, 1 e 0 vértices *branch* e portanto são *aranhas geradoras*. Nesse caso, o grafo (a) admite claramente uma *aranha geradora*, pois $s(G) \leq 1$.

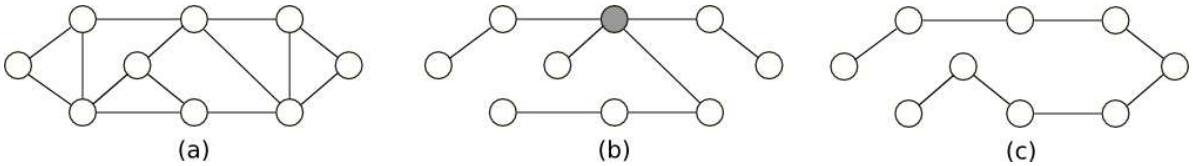

Figura 2.2. Grafos *aranha geradoras* de G

Se G possuir uma *aranha geradora* centralizada em cada vértice de G, então tal grafo é denominado uma *aracnóide* (do inglês, *arachnoid*). Por definição, um grafo *aranha* com 1 vértice *branch* é dito estar centralizado nesse vértice *branch*; um grafo *aranha* sem vértices *branch* é visto como centralizado em algum vértice da árvore. Dessa definições, temos que um grafo G com $s(G) = 0$ é *aracnóide* e que todo grafo *aracnóide* têm $s(G) \leq 1$.

Observe que $s(G) = 0$ se e somente se G possui um caminho hamiltoniano. Decidir se um grafo G possui um caminho hamiltoniano é um problema $\mathcal{NP}$-Completo; com isso, não existem algoritmos conhecidos que resolvem o problema em tempo polinomial.

2.2 Histórico

O problema da árvore geradora com número mínimo de vértices *branch* (do inglês, *minimum branch vertices problem*) surgiu a partir do interesse de Gargano et al. [2002] em um problema de alocação de *switches* em redes ópticas, descrito a seguir.

A tecnologia de multiplexação por divisão do comprimento de ondas (WDM) utilizada em redes ópticas suporta a propagação de múltiplos feixes de luz em um mesmo cabo de fibra óptica, desde que cada feixe de luz utilize um comprimento de onda diferente. Entende-se por *lightpath* a sequência de enlaces de fibra óptica que conecta dois nós de uma rede usando um comprimento de onda fixo. Assim, dois *lightpaths* que utilizam o mesmo enlace óptico devem usar diferentes comprimentos de onda.

Considere uma nova situação, onde uma nova tecnologia torne os *switches* ópticos capazes de replicar o sinal através da divisão do feixe de luz (em inglês, *light-splitting switches*). Estenderemos o conceito de *lightpath* em um novo conceito, denominado *light-tree*. Uma *light-tree* incorpora a capacidade de comunicação *multicast* em uma rede óptica, permitindo transmitir informação de um único nó origem para múltiplos nós destinos. Muitas aplicações que utilizam intensivamente a largura de banda requerem a capacidade de *multicast* para serem eficientes: *Web-browsing*, vídeo conferência, serviços de vídeo sob demanda, dentre outras aplicações.

A capacidade de realizar comunicação *multicast* em redes ópticas pode ser alcançada ao permitir que os *switches* WDM copiem os dados diretamente no domínio óptico através da divisão do feixe de luz. Assim o *multicast* óptico, implementado sob a forma de *light-trees*, possui vantagens sobre o *multicast* eletrônico uma vez que dividir o feixe de luz é 'mais fácil' do que copiar pacotes de dados em *buffers* eletrônicos (Gargano et al. [2002]).

Uma *light-tree* permite uma comunicação completamente óptica de um nó de origem para um conjunto de nós de destino, que pode incluir todos os demais nós da rede. Assuma que o nó de origem do *multicast* pode transmitir o sinal óptico para qualquer número de vizinhos. Os aparelhos *switch* alocados à nós da rede com grau superior à 2 devem possuir essa capacidade especial de dividir o feixe de luz, repassando adiante um número de cópias do sinal igual ao número de seus vizinhos, enquanto que os demais nós precisam suportar apenas a recepção e repasse de apenas uma cópia dos dados recebidos.

Uma rede óptica típica terá um número limitado desses *switches* sofisticados; alguém tem que posicioná-los de tal forma que eles possam realizar todas as possibilidades de *multicast*. Essa necessidade leva ao problema de encontrar árvores geradoras com a menor quantidade de vértices *branch* possível. Tal problema prático foi denominado pelos autores de *minimum branch vertices problem* (MBV), e será formalmente apresentado na próxima seção.

2.3 Problema da Árvore Geradora com Número Mínimo de Vértices *Branch*

Seja $G = (V, E)$ um grafo conexo, não direcionado e não valorado cujos vértices representam os *switches* da rede e os arcos correspondem aos enlaces de fibra óptica. Com uma quantidade $s(G)$ de *light-splitting switches* é possível prover todas as possibilidades de comunicação *multicasts* em uma rede óptica. No decorrer do texto denominaremos tais *switches* de *switches* 'especiais'.

Se $s(G) = 1$, então G possui uma *aranha geradora* e todas as possibilidades de comunicação *multicast* podem ser realizadas com o uso de apenas um *switch* especial. Se G for um grafo *aracnóide*, então nenhum *switch* especial será necessário, uma vez que o nó de origem do *multicast* pode transmitir a informação para qualquer número de vizinhos (ver Seção 2.2). Se $s(G) > 0$, o menor número de *switches* especiais necessários será exatamente $s(G)$.

O problema MBV pode ser enunciado formalmente da seguinte maneira: dado um grafo $G = (V, E)$ conexo, não direcionado e não valorado que representa uma rede óptica, determine o número mínimo de *switches* 'especiais' necessários para garantir todas as possibilidades de comunicação *multicast* na rede. Em outras palavras, isso corresponde à procurar pela árvore geradora T de G que possui o menor número de vértices *branch* (Cerulli et al. [2009]).

Esse problema foi estudado por Gargano et al. [2002] com foco na investigação do parâmetro $s(G)$ de grafos G conexos que admitem *aranhas geradoras* ou que são grafos *aracnóides*, onde foi demonstrado que tais problemas são $\mathcal{NP}$-Completos e que o valor de $s(G)$ é igualmente difícil de determinar por aproximação. A próxima seção irá apresentar os principais resultados de complexidade obtidos pelos autores nessa investigação.

2.4 Complexidade

Essa seção apresenta os principais resultados de complexidade obtidos acerca do problema MBV por Gargano et al. [2002] e Gargano et al. [2004]. Todos serão apresentados na forma de proposições e de provas que foram registradas por esses autores. A consideração inicial é que decidir se um grafo G possui um caminho hamiltoniano é um problema $\mathcal{NP}$-Completo (Garey e Johnson [1979]).

A principal técnica usada para mostrar que dois problemas estão relacionados é a *redução* de um problema em outro, fornecendo uma transformação construtiva que 'mapeia' qualquer instância do primeiro problema em uma instância equivalente do segundo. Essa transformação fornece um meio para converter qualquer algoritmo que resolve o segundo problema no algoritmo correspondente para resolver o primeiro problema (Garey e Johnson [1979]).

De acordo com Cormen et al. [2001], quando tentamos mostrar que um problema é $\mathcal{NP}$-Completo, fazemos declarações sobre o qual difícil de resolver ele é; não tentamos provar a existência de um algoritmo eficiente para ele, mas sim que nenhum algoritmo eficiente é provável de existir. Para tal usamos essa relação de 'facilidade' ou 'dificuldade' com outros problemas cuja 'dificuldade' é conhecida. Os próximos parágrafos fornecem os fundamentos para a demonstração de problemas $\mathcal{NP}$-Completo.

Seja A um problema de decisão que desejamos resolver em tempo polinomial. Chamamos de α uma instância particular desse problema. Suponha existir um outro problema de decisão, B, onde um algoritmo de tempo polinomial é conhecido existir. Um *algoritmo de redução em tempo polinomial* é qualquer procedimento capaz de transformar uma instância α do problema A em uma instância β do problema B em tempo polinomial e cujas respostas para os problemas de decisão são as mesmas, ou seja, a resposta de α é 'sim' se e somente se a resposta de β for 'sim'. Dessa maneira, um algoritmo de redução em tempo polinomial é capaz de resolver o problema A também em tempo polinomial, já que todos os passos envolvidos na transformação e resolução de α tomam tempo polinomial.

Para mostrar que um problema é $\mathcal{NP}$-Completo, a *redução* é usada no sentido contrário: ao resolver o problema A usando o problema B, usamos a 'dificuldade' de B para provar a 'dificuldade' de A. A redução em tempo polinomial é usada para mostrar que nenhum algoritmo de tempo polinomial pode existir para um dado problema B.

Suponha existir um problema de decisão A para o qual sabemos que nenhum algoritmo polinomial existe. Suponha existir um algoritmo de tempo polinomial capaz de transformar instâncias do problema A em instâncias do problema B. Suponha que B pode ser resolvido em tempo polinomial. Ao aplicar a redução, estamos afirmando que o problema A pode então ser resolvido em tempo polinomial através da transfor-

mação polinomial das instâncias de A em instâncias de B, seguido da sua resolução pelo algoritmo polinomial de B. Isso contradiz a consideração inicial de que não existe algoritmo capaz de resolver A em tempo polinomial, de onde temos que o problema B é *tão difícil* de resolver quanto o problema A e, portanto, não existe nenhum algoritmo de tempo polinomial para resolvê-lo.

Se é possível reduzir em tempo polinomial um problema intratável do ponto de vista computacional em outro problema, então esse outro problema é também intratável. As provas das proposições de Gargano et al. [2002] e Gargano et al. [2004] transcritas a seguir partem desse mesmo princípio. Mais detalhes sobre teoria de complexidade podem ser encontrados em Garey e Johnson [1979]; Cormen et al. [2001]; Ziviani [2004].

Proposição 1. *É $\mathcal{NP}$-Completo decidir se um grafo G admite uma aranha geradora.*

Demonstração. Suponha G um dado grafo, e v é um vértice de G. Construa um novo grafo G' que consiste de 3 cópias de G e um vértice adicional adjacente ao vértice v de todas as cópias de G. Assim, G' possui uma *aranha geradora* necessariamente centralizada no vértice adicional w, se e somente se G admitir um caminho hamiltoniano iniciando em v (Gargano et al. [2002]). □

A Figura 2.3 ilustra a redução de uma instância arbitrária (a) de acordo com o procedimento descrito na prova da Proposição 1. Em (a) temos um grafo G hipotético, com vértice v em destaque. Em (c), o grafo G' é construído de acordo com o procedimento descrito na prova da Proposição 1.

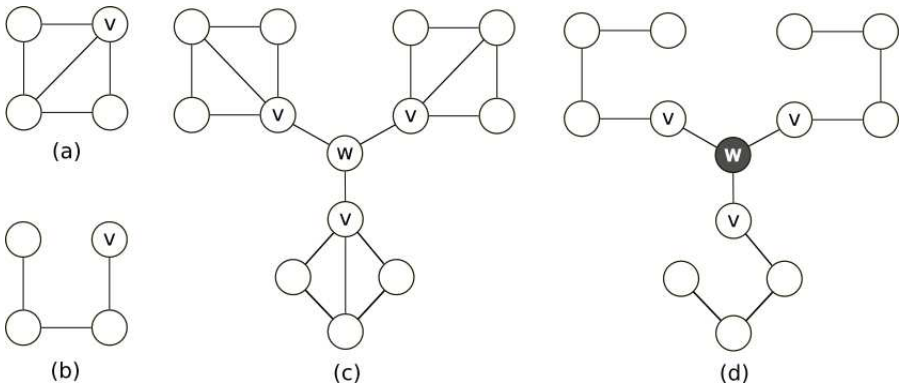

Figura 2.3. Redução do grafo G em G' para a prova da Proposição 1

O grafo G' admitirá uma *aranha geradora* centralizada em w se e somente se G possui um caminho hamiltoniano partindo de v. Embora decidir se existe um caminho hamiltoniano em um dado grafo seja um problema $\mathcal{NP}$-Completo, por inspeção visual sabemos que o grafo G possui um caminho hamiltoniano ilustrado em (b) e,

portanto, para essa instância existirá uma *aranha geradora* no grafo G' centralizada em w, destacada o grafo (d).

Proposição 2. *É $\mathcal{NP}$-Completo decidir se um dado grafo G é aracnóide.*

Demonstração. Suponha G um dado grafo, e v um dado vértice de G. Construa um novo grafo G' incluindo um novo vértice w em G e adicionando um arco entre v e w. O grafo G' é um *aracnóide* se e somente se G possui um caminho hamiltoniano partindo de v (Gargano et al. [2004]). □

A Figura 2.4 ilustra o procedimento de redução. Em (a), temos uma instância arbitrária que representa o grafo G. Em (b) o grafo G' é construído com a inclusão de um novo vértice, w, e sua ligação com o vértice v do grafo.

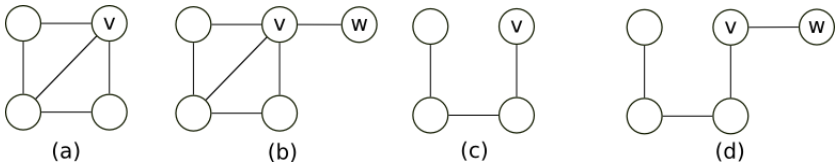

Figura 2.4. Redução do grafo G em G' para a prova da Proposição 2

Suponha um grafo G' (b) construído de acordo com o procedimento descrito na prova da Proposição 2. Se o grafo G em (a) possuir um caminho hamiltoniano (c) partindo de v, então $s(G) = 0$. A inclusão do novo arco (v, w) ao caminho hamiltoniano existente em G também resultará em um caminho hamiltoniano; logo G' também admite um caminho hamiltoniano (d). Por definição, se $s(G') = 0$ então existe uma *aranha geradora* centralizada em todos os seus vértices e portanto G' um *aracnóide*.

Proposição 3. *Seja k um inteiro não-negativo fixo. É $\mathcal{NP}$-Completo decidir se um dado grafo G satisfaz $s(G) \leq k$*

Demonstração. Em vista do resultado anterior, podemos assumir que $k \geq 2$. Seja G um dado grafo, e v um dado vértice de G. Construa um grafo G' a partir de $2k$ cópias disjuntas de G e um grafo completo com k vértices adicionais, fazendo com que cada vértice adicional seja adjacente ao vértice v de suas próprias duas cópias de G. O grafo G' admite uma árvore geradora com no máximo k vértices *branch* se e somente se G admite um caminho hamiltoniano partindo de v (Gargano et al. [2002]). □

A Figura 2.5 ilustra a redução do grafo G apresentado em (a) no grafo G' de acordo com o procedimento descrito na prova da Proposição 3. Supondo $k = 3$, O grafo G' apresentado em (c) é construído a partir de $2k$ cópias de G unindo-se o vértice v de cada cópia ao vértice k correspondente no grafo completo.

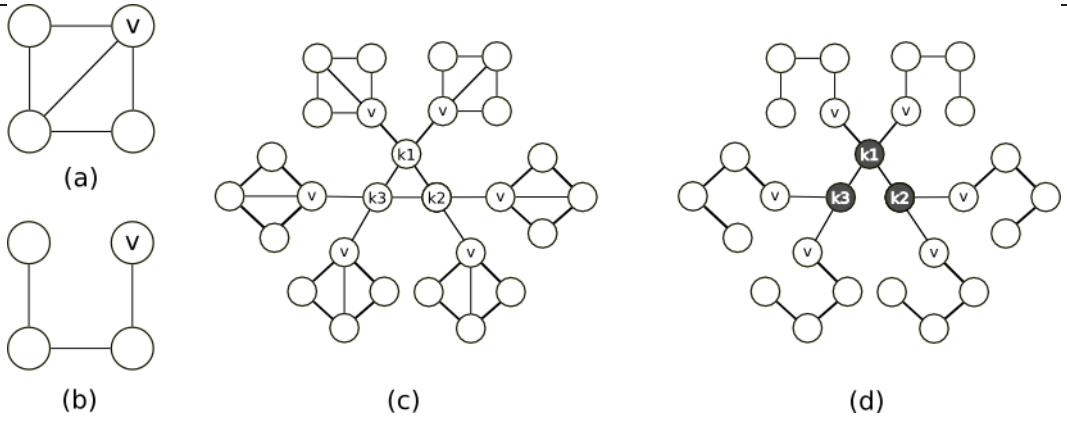

Figura 2.5. Redução do grafo G em G' para a prova da Proposição 3

O grafo completo usado na construção de G' terá o efeito de inserir *pelo menos* k vértices *branch* em G'. Como o grafo G desse exemplo possui um caminho hamiltoniano destacado em (b), então cada caminho estará ligado à apenas um desses k vértices *branch*. Com isso é possível construir uma árvore geradora em G' (d) que possua no máximo k vértices *branch* stisfazendo a condição de que $s(G) \leq k$. Entretanto, decidir se G admite um caminho hamiltoniano é um problema $\mathcal{NP}$-Completo.

Proposição 4. *Seja $k = O(n^{1-\epsilon})$, para ϵ fixo e $0 \leq \epsilon \leq 1$. Não existe algoritmo de tempo polinomial para verificar se $s(G) \leq k$, a menos que $\mathcal{P} = \mathcal{NP}$*

Demonstração. Seja G um dado grafo, e v um dado vértice de G. Construa um grafo G' com k cópias disjuntas de G e um vértice adicional v', fazendo com que o vértice v de cada cópia seja adjacente ao vértice v'. Se G admite um caminho hamiltoniano partindo de v, então G' contém uma árvore geradora com um vértice *branch* centralizado em v'. Por outro lado, se tal caminho hamiltoniano não existe em G então $s(G) \geq k+1$, uma vez que cada árvore geradora de G terá no mínimo um vértice *branch* para cada cópia de G (Gargano et al. [2004]). □

A Figura 2.6 ilustra a redução de uma instância (a) do grafo G em uma instância (c) do grafo G' usando o procedimento de redução descrito na prova da Proposição 4. O grafo G transforma-se em G' ao criar $k = 3$ cópias de G e conectar o vértice v de cada cópia ao vértice v' adicional.

Por inspeção, verifica-se que o grafo G em (a) possui um caminho hamiltoniano partindo de v que está ilustrado em (b). Com isso é possível verificar por inspeção que existe uma *aranha geradora* (d) em G' com um único vértice *branch* possível (nesse caso, v' com $\delta(v') = k$). Se G não admitisse um caminho hamiltoniano, então a árvore geradora de G' teria *pelo menos* 1 vértices *branch* em cada sub-árvore originada a partir

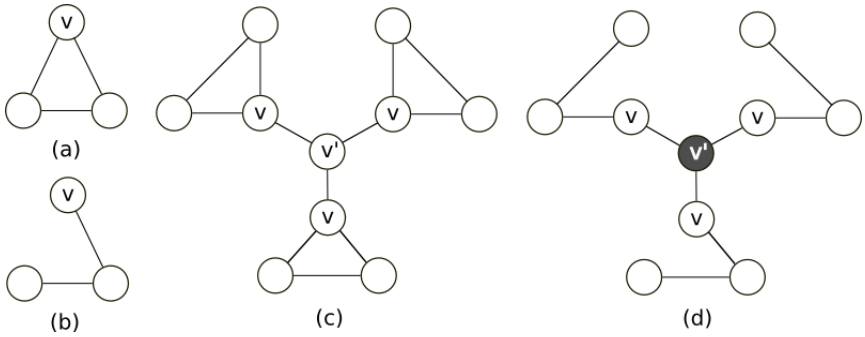

Figura 2.6. Redução do grafo G em G' para a prova da Proposição 4

de cada uma das k cópias, mais o vértice v' que é obrigatoriamente um vértice *branch*, totalizando pelo menos $k+1$ vértices *branch*.

Por fim, uma última proposição será enunciada mas sem descrever sua prova, uma vez que ela requer outros conceitos e resultados pesquisados por Gargano et al. [2002] que fogem ao contexto desse trabalho:

Proposição 5. *Seja k qualquer inteiro positivo fixo. A menos que $\mathcal{P} = \mathcal{NP}$, não existe algoritmo de tempo polinomial para verificar se $s(G) \leq k$, mesmo para grafos cúbicos com $s(G) = 0$.*

2.5 Outros tópicos relacionados

Ainda relacionado ao problema MBV, cabe destacar os trabalhos de Gargano et al. [2002] e Gargano et al. [2004] na investigação de condições de densidade em grafos que, quando satisfeitas, garantem a existência de um caminho hamiltoniano em grafos. Tais condições incluem a inexistência dos grafos $K_{1,3}$ e $K_{1,4}$ como subgrafos induzidos de G e que todos os vértices de G possuam grau de no mínimo $\frac{n-1}{2}$, dentre outras condições.

Gargano e Hammar [2003] estudaram o conceito de *maximalidade de caminhos* em grafos, mostrando como encontrar caminhos maximais em grafos conexos em tempo $O(n^3)$ e como transformá-los em *aranhas geradoras* para grafos gerais e bipartidos, desde que tais grafos satisfaçam algumas condições de densidade.

Por fim, Gargano et al. [2004] revisitam o problema de determinar árvores geradoras com a menor quantidade de vértices *branch*, acrescentando novas condições de densidade, corrigindo algumas provas de proposições publicadas em Gargano et al. [2002] sobre a complexidade do problema e apresentando o conceito de *maximalidade de caminhos* e o algoritmo para encontrar *aranhas geradoras* em grafos gerais proposto em

Gargano e Hammar [2003]. Ainda nesse trabalho relacionam o parâmetro $s(G)$ com outros parâmetros clássicos em teoria dos grafos.

Outro problema diretamente relacionado, que não será abordado nesse estudo, foi formalizado por Cerulli et al. [2009] onde de acordo com esses autores muitos dos *switches* ópticos reais conseguem apenas duplicar o feixe de luz incidente sendo, portanto, um problema mais adequado para modelar a tomada de decisão sobre alocação de *switches* ópticos em situações reais. Apesar dessa consideração o problema MBV ainda desperta interesse devido ao fato de ser $\mathcal{NP}$-Completo, e portanto intratável do ponto de vista computacional, o que motiva o desenvolvimento de heurísticas capazes de obter soluções sub-ótimas de boa qualidade para o MBV.

Capítulo 3

O Estado da Arte

Esse capítulo apresenta as técnicas conhecidas para solucionar o Problema da Árvore Geradora com Mínimo de Vértices *Branch*. Tais técnicas incluem uma formulação linear inteira, que permite resolver instâncias pequenas em tempo razoável, e algumas heurísticas baseadas em estratégias de ponderação de arcos e de coloração de vértices.

3.1 Formulação matemática para o problema MBV

Essa seção descreve a modelagem inteiro mista baseada em fluxo de um único produto proposta por Cerulli et al. [2009] para o problema MBV.

Seja $G = (V, E)$ um grafo conexo, não direcionado e não valorado. Seja $T = (V, E')$ uma árvore geradora em G definida pelo envio de uma unidade de fluxo de um vértice abitrário de origem $s \in V$ para cada vértice $v \in V \setminus \{s\}$ do grafo G.

Sejam f_{uv} e f_{vu} variáveis que definem, respectivamente, o fluxo que parte do vértice u em direção ao vértice v e o fluxo que parte do vértice v em direção ao vértice u ao longo do arco (u, v).

Seja x_e, $e \in E$, uma variável binária de decisão tal que $x_e = 1$ se o arco e pertence à árvore geradora T, e $x_e = 0$ caso contrário. Seja y_v, $v \in V$, uma variável binária de decisão tal que $y_v = 1$ se v é um vértice *branch*, e $y_v = 0$ caso contrário.

A notação $A(v)$ indica o sub-conjunto de arcos incidentes em um dado vértice $v \in V$ do grafo G. $A^+(v) = \{(v, w) \in V \times V \mid (v, w) \in E\}$ refere-se ao conjunto de arcos que *partem* de v e $A^-(v) = \{(w, v) \in V \times V \mid (w, v) \in E\}$ refere-se ao conjunto de arcos que *chegam* em v na versão direcionada de G.

A função objetivo deverá considerar a minimização do número de vértices *branch* de uma árvore geradora construída como solução, de acordo com o modelo que é dado

a seguir:

$$min \sum_{v \in V} y_v, \tag{3.1}$$

sujeito à

$$\sum_{e \in E} x_e = |V| - 1, \tag{3.2}$$

$$\sum_{(s,v) \in A^+(s)} f_{sv} - \sum_{(v,s) \in A^-(s)} f_{vs} = |V| - 1, \tag{3.3}$$

$$\sum_{(v,u) \in A^+(v)} f_{vu} - \sum_{(u,v) \in A^-(v)} f_{uv} = -1 \ \forall v \in V \setminus \{s\}, \tag{3.4}$$

$$f_{uv} \leq (|V| - 1)x_e \ \forall (u,v) \in E, \tag{3.5}$$

$$f_{vu} \leq (|V| - 1)x_e \ \forall (u,v) \in E, \tag{3.6}$$

$$\sum_{e \in A(v)} x_e - 2 \leq (|V| - 1)y_v \ \forall v \in V, \tag{3.7}$$

$$x_e \in \{0,1\} \ \forall e \in E, \tag{3.8}$$

$$y_v \in \{0,1\} \ \forall v \in V, \tag{3.9}$$

$$f_{uv} \geq 0 \ \forall (u,v) \in E, \tag{3.10}$$

$$f_{vu} \geq 0 \ \forall (u,v) \in E. \tag{3.11}$$

O modelo acima é capaz de retornar soluções exatas para o problema de encontrar árvores geradoras em um grafo G conexo, não direcionado e não valorado com quantidade mínima de vértices com grau maior ou igual a 3. Os próximos parágrafos detalham a função de cada uma de suas restrições.

A função objetivo (3.1) requer minimizar o número total de vértices *branch* na árvore.

A restrição (3.2) garante que cada solução factível terá $|V| - 1$ arcos ligados. As equações de conservação de fluxo modeladas pelas restrições (3.3) e (3.4) garantem que

$(|V|-1)$ unidades de fluxo partirão do vértice s em direção aos demais vértices do grafo, e cada vértice do grafo (exceto s) é capaz de consumir 1 unidade de fluxo. Em conjunto, essas restrições asseguram que cada solução factível será uma árvore geradora pois resultará em um grafo onde todos os vértice estão conectados por meio de $(|V|-1)$ arcos.

As restrições (3.5) e (3.6) garantem que, se um dado arco e é selecionado para participar da árvore geradora T, então haverá passagem de fluxo por ele. A quantidade de fluxo que pode passar por um arco é limitada à quantidade de fluxo que parte da origem, ou seja, $|V|-1$. Por outro lado, se o arco e não é selecionado para participar da árvore geradora T (isto é, $x_e = 0$), então não haverá passagem de fluxo por ele e portanto a única possibilidade para tornar essas desigualdades válidas é atribuir 0 às variáveis f_{uv} e f_{vu}.

A restrição (3.7) modela a relação entre o grau de um vértice v e a variável binária de decisão y_v que indica se v será um vértice *branch*: se $\delta(v) \leq 2$, então y_v deverá assumir valor 0 para que a desigualdade seja verdadeira; caso contrário, a variável de decisão y_v que seleciona o vértice v como *branch* deverá assumir valor 1 para manter a desigualdade válida. Assim, o vértice v é selecionado como *branch* caso existam mais do que 2 arcos incidentes à ele na solução.

Por fim, as restrições (3.8), (3.9), (3.10) e (3.11) apenas definem o domínio de cada variável de decisão x_e, y_v, f_{uv} e f_{vu}.

3.2 Heurísticas

Diversas proposições e provas sobre a complexidade do problema MBV foram apresentadas na Seção 2.4, demonstrando que o problema é 'difícil' de ser resolvido do ponto de vista computacional por não haver algoritmos polinomiais capazes de fazê-lo em tempo razoável. Embora a Seção 3.1 tenha apresentado a formulação matemática de Cerulli et al. [2009], que torna possível resolver o problema empregando métodos exatos, o problema é $\mathcal{NP}$-Completo e portanto a formulação só é capaz de resolver instâncias pequenas com um esforço computacional aceitável.

O problema MBV tem aplicação prática no projeto de redes ópticas, conforme mencionado na Seção 2.2. Para resolver instâncias 'interessantes' (isto é, instâncias reais) de tamanho razoável é preciso utilizar algum método de aproximação ou heurística que, embora não retorne soluções provadamente ótimas, podem resultar em soluções sub-ótimas de boa qualidade com um esforço computacional menor do que resolver o modelo.

Nesse contexto, Cerulli et al. [2009] propuseram três estratégias heurísticas para o

problema MBV, capazes de construir uma árvore T de G levando em conta a preocupação de minimizar o número de vértices *branch*. A primeira delas baseia-se em uma estratégia de atribuir pesos aos arcos do grafo G para marcar arcos que, se inseridos na árvore, transformam um de seus extremos em *branch*. A segunda utiliza um esquema de codificação de cores para os vértices de T para determinar quais estão na iminência de tornar-se *branch*. Por fim, a terceira utiliza partes de ambas estratégias em uma heurística mista de ponderação de arcos e coloração de vértices. As próximas sub-seções detalham cada método e apresentam o pseudo-código correspondente.

3.2.1 EWS: Estratégia de Ponderação de Arcos

Essa estratégia consiste basicamente em criar, a partir de um grafo $G = (V, E)$ conexo, não direcionado e não valorado, um novo grafo ponderado $G' = (V, E, w)$ onde $w : E \rightarrow N^+$ consiste em uma função positiva no conjunto de arcos E que indica a possibilidade de um dos vértices extremos de um dado arco (u, v) tornar-se *branch* caso (u, v) seja selecionado para participar da árvore geradora $T = (V, E')$.

No início da execução cada arco $(u, v) \in E$ recebe peso 1, enquanto o grafo T será composto por uma floresta formada por $|V|$ componentes conexos disjuntos. O algoritmo constrói uma árvore geradora tomando um arco (u^*, v^*) de G e inserindo-o em T à cada iteração, até que o grafo T resulte em um único componente conexo.

A escolha do arco (u^*, v^*) deve levar em conta algumas considerações. Uma delas é que o algoritmo tenta evitar a criação de vértices *branch* escolhendo sempre um arco (u^*, v^*) de menor peso em G que ainda não tenha sido selecionado para T. Outra consideração é que o arco escolhido não poderá gerar ciclos se inserido na árvore: um arco (u^*, v^*) participa da árvore geradora apenas se os vértices u^* e v^* estiverem em diferentes componentes conexos de T.

Quando um arco (u^*, v^*) é inserido em T ele afeta o peso w de todos os arcos incidentes aos vértices u^* e v^* em G, que têm seu peso incrementado em 1 unidade. Isso tem o efeito de torná-los uma escolha 'menos provável' na próxima iteração do algoritmo, uma vez que o arco (u^*, v^*) escolhido será o de menor peso dentre todos os arcos de G que ainda não fazem parte do grafo T.

A condição de parada do algoritmo será quando a árvore geradora T estiver construída, ou seja, quando todos os componentes conexos de T tornarem-se um único componente conexo com $|V|$ vértices interligados por $(|V| - 1)$ arcos sem formar ciclos.

O pseudo-código 1 foi extraído de Cerulli et al. [2009] e formaliza os passos discutidos textualmente. Nas próximas listagens, `adj(`v`)` refere-se aos arcos adjacentes à um dado vértice v em G. Os métodos `select()` e `cover()` serão detalhados adiante, com uma explicação sobre as situações especiais que eles atendem, a saber, desempate na

escolha dos arcos de L e aglutinação de novos arcos em vértices recém tornados *branch*.

Algoritmo 1 Pseudo-código da estratégia de ponderação de arcos do problema MBV

```
 1: procedure MBV-ESTRATEGIA-EWS(G = (V, E))
 2:     V' ← V
 3:     E' ← ∅
 4:     for all (u, v) ∈ E do
 5:         w(u, v) ← 1
 6:     end for
 7:     A ← E
 8:     while (|E'| ≠ |V'| − 1) do
 9:         L ← {(u', v') ∈ A | w(u', v') ≤ w(u, v), ∀ (u, v) ∈ A}
10:         (u*, v*) ← select(L)
11:         A ← A \ {(u*, v*)}
12:         if (u* e v* estão em diferentes componentes conexos do grafo T) then
13:             E' ← E' ∪ {(u*, v*)}
14:             for all v ∈ adj(u*) do
15:                 w(u*, v) ← w(u*, v) + 1
16:             end for
17:             for all v ∈ adj(v*) do
18:                 w(v*, v) ← w(v*, v) + 1
19:             end for
20:             cover((u*, v*), G, T)
21:         end if
22:     end while
        return T = (V', E')
23: end procedure
```

O conjunto A, inicialmente igual à E, guarda todos os arcos de G que não estão presentes em T. A cada iteração, um sub-conjunto L é construído a partir de A contendo todos os arcos cujo peso w é mínimo. O método `select()` (linha 10) escolhe dentre os arcos candidatos contidos em L qual será o o arco (u^*, v^*) a ser inserido em T.

O incremento do peso w aplica-se à todos os arcos incidentes aos vértices u^* e v^* (linhas 15 e 18). Devido à esse fato existirão casos onde o sub-conjunto L conterá múltiplos arcos com peso mínimo. Nesses casos é preciso aplicar algum critério de desempate, sendo que o critério estabelecido para o EWS consiste em escolher, dentre os candidatos, o arco cujos vértices extremos em T possuem grau máximo. Tal decisão favorece o uso de vértices *branch* já existentes e evita a criação de novos vértices *branch*.

A Figura 3.1 ilustra um caso com aplicação do critério de desempate. Sejam (u, w) e (u, v) arcos candidatos de peso mínimo contidos em L. O grafo T contém dois componentes conexos disjuntos. O vértice w é o único vértice *branch* de T. Dependendo de qual arco será selecionado, teremos duas árvores geradoras diferentes: T_1 contendo apenas um vértice *branch*, ou T_2 contendo dois vértices *branch*. Dentre os candidatos

de L, é mais vantajoso escolher o arco (u, w) que leva à árvore geradora T_1. Note que o arco (u, w) é justamente o candidato que possui vértices extremos com maior grau em T: $\delta(w) = 3$, enquanto que $\delta(v) = 2$, sendo (u, w) uma escolha melhor que (u, v).

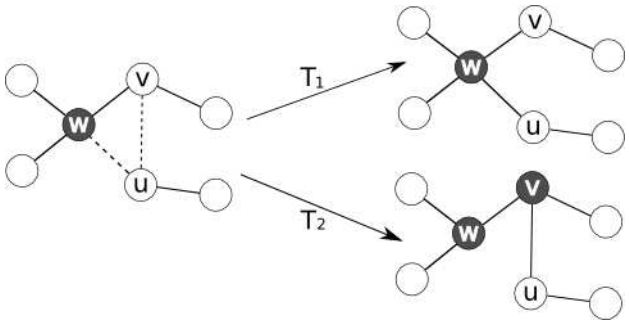

Figura 3.1. Possíveis árvores geradoras criadas a partir da seleção de um arco candidato de L (adaptado de Cerulli et al. [2009])

Conforme mencionado, o critério de seleção opera na lista de candidatos L sempre evitando a criação de novos vértices *branch*. Entretando, existem situações onde isso é inevitável e a escolha de um arco (u^*, v^*) irá transformar o vértice u^* ou v^* ou ambos em vértices *branch*.

Quando isso ocorre, é vantajoso 'aglutinar' o maior número de arcos de A em T ao redor de vértices que já são *branch*. Sempre que um arco (u^*, v^*) é selecionado de L e incluído no grafo T o método `cover()` (linha 20) é chamado com tal propósito. O pseudo-código 2 formaliza esse procedimento. A notação $A(v)$ refere-se ao sub-conjunto de arcos que incidem à um dado vértice v de G, e $\delta_T(v)$ é o grau do vértice v em T.

De acordo com o pseudo-código 2, se um dos vértices u^* ou v^* ou ambos tornaram-se *branch* após a inclusão do arco (u^*, v^*) em T (linhas 2 e 10), então os arcos contidos em A que são adjacentes ao novo *branch* são examinados. Dentre esses arcos, aqueles que unem dois componentes conexos disjuntos de T são acrescentados à árvore geradora em construção (linhas 5 e 13) e têm seu peso w atualizado pelo método `update-weights()` (linhas 6 e 14).

Em resumo, o método `cover()` têm a função de cobrir o maior número possível de arcos $e \in A, e \notin T$, concentrando-os em vértices que já são *branch* na tentativa de evitar a criação de novos vértices *branch* em T (critério de cobertura).

Por fim, a complexidade do método apresentado é da ordem de $O(|V| \times |E|)$ uma vez que, segundo os autores, o laço principal requer que $O(|E|)$ operações sejam executadas (linhas 9 à 20) por $O(|V|)$ vezes (linha 8).

Algoritmo 2 Pseudo-código para o procedimento de cobertura do método EWS

```
1: procedure COVER((u*, v*), G, T)
2:     if (δ_T(v*) = 3) then
3:         for all (u, v*) ∈ A(v*) ∩ A do
4:             if (δ_T(u) ≠ 2) e u e v* estão em diferentes componentes conexos de T then
5:                 T ← T ∪ {(u, v*)}
6:                 update-weights((u, v*), G, T)
7:             end if
8:         end for
9:     end if
10:    if (δ_T(u*) = 3) then
11:        for all (u*, v) ∈ A(u*) ∩ A do
12:            if (δ_T(v) ≠ 2) e u* e v estão em diferentes componentes conexos de T then
13:                T ← T ∪ {(u*, v)}
14:                update-weights((u*, v), G, T)
15:            end if
16:        end for
17:    end if
18: end procedure
```

3.2.2 NCH: Estratégia de Coloração de Vértices

Seja $G = (V, E)$ um grafo conexo, não direcionado e não valorado. Seja $T = (V, E')$ uma árvore geradora de G que será construída pela estratégia de coloração de vértices. Tal estratégia utiliza um rótulo $cor : V \rightarrow \{Verde, Azul, Amarelo, Vermelho\}$ atribuído à cada vértice $v \in V$ de T que indica se este torna-se *branch* caso um arco $e \in E$ incidente à v seja inserido em T. Seja $\delta_T(v)$ o grau do vértice v no grafo T. O rótulo de cor segue o mapeamento:

$$cor(v) = \begin{cases} Verde & \text{se } \delta_T(v) = 0 \\ Azul & \text{se } \delta_T(v) = 1 \\ Amarelo & \text{se } \delta_T(v) = 2 \\ Vermelho & \text{se } \delta_T(v) \geq 3 \end{cases} \tag{3.12}$$

O algoritmo inicia atribuindo a cor $Verde$ à todos os vértices de T, que é composto inicialmente por $|V|$ componentes conexos disjuntos. Uma árvore geradora será construída iterativamente em T através da inserção de arcos de G em T, um por vez, evitando a formação de ciclos. Inserir um arco (u^*, v^*) causa alterações no grau dos vértices u^* e v^* e, consequentemente, no seu rótulo associado. Note que, de acordo com (5.1), um vértice está na *iminência* de tornar-se *branch* caso esteja rotulado com a cor *Amarelo*.

O algoritmo NCH opera de maneira semelhante ao algoritmo EWS. O conjunto A contém todos os arcos de E que ainda não foram inseridos em T. Seja L um sub-

conjunto composto por todos os arcos de A que respeitam o critério de seleção adotado pelo algoritmo NCH. A cada iteração um arco (u^*, v^*) é sacado de L e, desde que não crie ciclos em T, participará da construção da árvore geradora.

O critério de seleção consiste em tomar, dentre os arcos contidos em L, aquele com a menor quantidade de vértices extremos rotulados de *Amarelo*. Tal critério orienta o algoritmo a escolher arcos que causam o menor impacto no número total de vértices *branch* de T, uma vez que um vértice com cor *Amarelo* torna-se *branch* caso algum arco que nele incida seja inserido em T.

Conforme a árvore geradora é construída pelo algoritmo, podem ocorrer situações em que existam mais de um candidato em L com quantidade mínima de extremos de cor *Amarelo*. Nesses casos o algoritmo utiliza um critério de desempate que tenta reduzir a quantidade de novos vértices *potencialmente* amarelos em T escolhendo, dentre os arcos empatados, aquele com a quantidade mínima de vértices de coloração *Azul*, isto é, com algum de seus vértices extremos de grau 1.

O algoritmo encerra quando $(|V| - 1)$ arcos de A forem escolhidos para T. Os passos da estratégia são formalizado no pseudo-código a seguir. As notações $n_Y(u, v)$ e $n_B(u, v)$ indicam a quantidade de vértices de cor *Amarelo* e *Azul* de um arco (u, v).

Algoritmo 3 Pseudo-código para a estratégia de coloração para o problema MBV

```
 1: procedure MBV-ESTRATEGIA-NCH(G = (V, E))
 2:     V' ← V
 3:     E' ← ∅
 4:     for all v ∈ V' do
 5:         cor[v] ← Verde
 6:     end for
 7:     A ← E
 8:     while |E'| ≠ |V'| − 1 do
 9:         L ← {(u', v') ∈ A | n_Y(u', v') ≤ n_Y(u, v), ∀ (u, v) ∈ A}
10:         (u*, v*) ← arg min_{(u,v)∈L} {n_B(u, v)}
11:         A ← A \ {(u*, v*)}
12:         if (u* e v* estão em diferentes componentes conexos do grafo T) then
13:             E' ← E' ∪ {(u*, v*)}
14:             Atualiza as cores de u* e v*
15:             cover((u*, v*), G, T)
16:         end if
17:     end while
         return (T = (V', E')
18: end procedure
```

O método `cover()` (linha 15) é chamado sempre que um arco (u^*, v^*) é inserido no grafo T. Ele verifica se os vértices u^* ou v^* ou ambos tornaram-se *branch* (linhas 2 e 10) e, em caso positivo, cobre diretamente todos os arcos que ainda não fazem parte de T, que incidem no novo vértice *branch* e que não formem ciclos em T, inserindo-os

na árvore geradora (linhas 5 e 13). Note que a inclusão desses arcos em T afetam o grau de seus vértices extremos e, consequentemente, seu rótulo de cor.

O pseudo-código para o método `cover()` é dado a seguir. A notação $A(v)$ refere-se ao sub-conjunto formado pelos arcos que incidem no vértice v no grafo G.

Algoritmo 4 Pseudo-código para a cobertura da estratégia de coloração dos vértices para o problema MBV

```
1:  procedure COVER((u*, v*), G, T)
2:      if (cor(v*) = Vermelho) then
3:          for all (u, v*) ∈ A(v*) ∩ A do
4:              if cor(u) ≠ Amarelo e u e v* estão em diferentes comp. conexos de T then
5:                  T ← T ∪ {(u, v*)}
6:                  Atualiza as cores dos vértices u e v*
7:              end if
8:          end for
9:      end if
10:     if (cor(u*) = Vermelho) then
11:         for all (u*, v) ∈ A(u*) ∩ A do
12:             if cor(v) ≠ Amarelo e u* e v estão em diferentes comp. conexos de T then
13:                 T ← T ∪ {(u*, v)}
14:                 Atualiza as cores dos vértices u* e v
15:             end if
16:         end for
17:     end if
18: end procedure
```

De acordo com os autores, a complexidade do método que implementa a estratégia de coloração de vértices é da ordem de $O(|V| \times |E|)$, uma vez que o laço principal repete os passos das linhas 9 à 15 no máximo $|V|$ vezes, requerendo $|E|$ operações.

3.2.3 MIX: Abordagem Combinada

A terceira estratégia proposta por Cerulli et al. [2009] atribui pesos aos arcos de G e rótulos aos vértices de T seguindo as mesmas regras utilizadas pelas estratégias EWS e NCH.

Sejam $G = (V, E)$ um grafo conexo, não direcionado e não valorado, e $T = (V, E')$ uma árvore geradora construída a partir de G. Considere transformar o grafo G em um novo grafo G' através da atribuição de peso w à cada um de seus vértices. No início, todos os vértices de T são rotulados de $Verde$, e todos os arcos de G' têm peso unitário. A árvore geradora T consiste inicialmente de $|V|$ componentes conexos disjuntos.

A cada iteração a estratégia empregada pela abordagem combinada seleciona, dentre os arcos de G' que ainda não fazem parte de T, um arco (u^*, v^*) de peso mínimo. Caso seus vértices extremos u^* e v^* pertençam à diferentes componentes conexos de

T, então a inserção desse arco não criará ciclos na árvore geradora e portanto o arco (u^*, v^*) deve ser incluído em T.

Tal como nas estratégias apresentadas, é possível que empates ocorram. Nesses casos decide-se qual será o arco selecionado empregando primeiro o critério de desempate que prioriza o arco com menor quantidade de vértices extremos com rótulo *Azul* (Seção 3.2.2); seguido pelo critério de desempate que adota o arco com vértices extremos de máximo grau (Seção 3.2.1).

Embora os autores tenham comentado resumidamente o funcionamento da abordagem combinada, nenhuma listagem em pseudo-código do algoritmo foi publicada. Considerando as informações disponibilizadas nas seções anteriores, *pressupõe-se* que os passos do algoritmo sejam os propostos no próximo pseudo-código. Sejam A o conjunto de arcos de G' que ainda não foram inseridos em T, L um sub-conjunto de A composto por arcos de peso mínimo e `adj(`v`)` os vértices adjacentes à um dado vértice v de G.

Algoritmo 5 Pseudo-código da estratégia mista para o problema MBV

```
1:  procedure MBV-ESTRATEGIA-MIX(G = (V, E))
2:      V' ← V
3:      E' ← ∅
4:      for all (u, v) ∈ E do
5:          w(u, v) ← 1
6:      end for
7:      for all v ∈ V do
8:          cor[v] ← Verde
9:      end for
10:     A ← E
11:     while (|E'| ≠ |V'| − 1) do
12:         L ← {(u', v') ∈ A | w(u', v') ≤ w(u, v), ∀ (u, v) ∈ A}
13:         (u*, v*) ← arg min_{(u,v)∈L}{n_B(u, v)}
14:         A ← A \ {(u*, v*)}
15:         if (u* e v* estão em diferentes componentes conexos do grafo T) then
16:             E' ← E' ∪ {(u*, v*)}
17:             for all v ∈ adj(u*) do
18:                 w(u*, v) ← w(u*, v) + 1
19:             end for
20:             for all v ∈ adj(v*) do
21:                 w(v*, v) ← w(v*, v) + 1
22:             end for
23:             Atualiza as cores de u* e v*
24:             cover((u*, v*), G, T)
25:         end if
26:     end while
        return T = (V', E')
27: end procedure
```

O corpo do algoritmo é semelhante ao corpo dos algoritmos correspondentes às estratégias EWS e NCH, sendo composto por um laço que repete-se iterativamente até que $(|V|-1)$ arcos de G' sejam inseridos na árvore geradora T. Na inicialização o peso w de cada arco de G' é definido como 1, e cada vértice de T é rotulado de $Verde$. A cada iteração um arco (u^*, v^*) é tomado de A sendo tal arco de peso mínimo, contendo a menor quantidade de extremos rotulados de $Azul$ e sendo incidente à vértices de T com grau máximo (linhas 12 e 13).

Uma vez selecionado, verifica-se se os vértices u^* e v^* do arco (u^*, v^*) estão em diferentes componentes conexos de T, isto é, $T \cup \{(u^*, v^*)\}$ é acíclico. Em caso positivo o arco (u^*, v^*) é inserido em T, o rótulo dos vértices u^* e v^* são atualizados de acordo com a equação (5.1) e os arcos de G' que incidem nos vértices u^* e v^* são incrementados em uma unidade.

Pressupõe-se a existência de algum método `cover()` para essa estratégia que seja capaz de cobrir arcos de G' que ainda não estão presentes em T mas que incidem em u^*, v^* ou ambos caso estes vértices tornem-se *branch*. Cerulli et al. [2009] não comentam nenhum procedimento de cobertura em seu trabalho; portanto uma possibilidade é utilizar um dos métodos de cobertura descritos nas sub-Seções 3.2.1 e 3.2.2.

Capítulo 4

Refinamento Iterativo e Árvores Geradoras Restritas

O presente capítulo introduz uma abordagem pouco explorada para resolver problemas envolvendo àrvores geradoras restritas proposta por Deo e Kumar [1997], denominada de *refinamento iterativo*. Tal abordagem foi utilizada para resolver alguns problemas clássicos envolvendo árvores, que são comentados no decorrer desse capítulo.

4.1 Algoritmo de Refinamento Iterativo Geral

Uma das abordagens da literatura de otimização para resolver problemas $\mathcal{NP}$-completo de árvores geradoras restritas consiste no projeto de algoritmos de refinamento iterativo (do inglês, *iterative refinement* ou IR) (Deo e Kumar [1997]). Essa seção apresentará a idéia central da abordagem, que é a base para a proposta de algoritmo desta dissertação.

Considere um problema de árvore geradora restrita especificado por um grafo ponderado G e duas restrições, $\mathcal{C}_1$ e $\mathcal{C}_2$, onde $\mathcal{C}_1$ consiste tipicamente no objetivo de minimizar a soma dos pesos da árvore geradora. O algoritmo IR inicia a partir de uma árvore geradora parcialmente restrita (que atende apenas $\mathcal{C}_1$) e move-se a cada iteração em direção à uma árvore completamente restrita (que atende $\mathcal{C}_2$), sacrificando a otimalidade com respeito à $\mathcal{C}_1$.

O funcionamento do método é dado pelo pseudo-código a seguir, extraído de Deo e Kumar [1997]. Ele refere-se à um algoritmo 'geral' para resolver problemas em árvores, onde detalhes sobre cada problema devem ser incorporados (ver Seção 4.2).

Primeiro, uma árvore geradora T que satisfaz a restrição $\mathcal{C}_1$ é construída a partir de G (linha 2), que num primeiro momento pode não satisfazer a restrição $\mathcal{C}_2$. Em seguida, os arcos de T que violam a restrição $\mathcal{C}_2$ são identificados e seus pesos associados são modificados em G (linha 4), dando origem ao grafo G'. Uma nova árvore geradora T é

Algoritmo 6 Pseudo-código do algoritmo de refinamento iterativo geral

```
1: procedure ALGORITMO-REFINAMENTO-ITERATIVO-GERAL(G, C1, C2)
2:     No grafo G encontre uma árvore geradora T que satisfaça C1
3:     while (árvore geradora T viola C2) do
4:         Usando C2 altere os pesos dos arcos de G para obter G' com novos pesos
5:         No grafo G' encontre uma árvore geradora T que satisfaça C1
6:         Redefina G ← G'
7:     end while
8: end procedure
```

construída a partir de G' (linha 5), sendo reavaliada pelo laço principal do método com relação à satisfação da restrição $\mathcal{C}_2$ (linha 3). Caso T não satisfaça $\mathcal{C}_2$, o método repete o laço modificando iterativamente o peso dos arcos de G até encontrar uma árvore geradora que satisfaça $\mathcal{C}_2$. Note que tal árvore é sub-ótima em relação à restrição $\mathcal{C}_1$.

A ação de modificar o peso dos arcos infratores de T é denominada *blacklisting*. A função do *blacklisting* é fazer com que arcos envolvidos em infrações da restrição $\mathcal{C}_2$ sejam 'desencorajados' a reaparecerem novamente nas árvores geradoras resultantes das próximas iterações do algoritmo de refinamento iterativo. Normalmente o artifício empregado para tal propósito consiste em incrementar o valor do peso dos arcos infratores.

Deo e Kumar [1997] comentam que a operação do método de refinamento iterativo é similar ao funcionamento do algoritmo *dual simplex*. De acordo com Bazaraa et al. [1990], o método *dual-simplex* move-se, à cada iteração, de uma solução básica factível do problema dual para uma solução básica factível melhorada, até que a otimalidade do dual (e também do primal) seja atingida ou que seja concluído que o problema dual é ilimitado e portanto, o problema primal é infactível. Ainda de acordo com Deo e Kumar [1997], o método *dual-simplex* é útil para problemas para o qual uma solução ótima para um problema irrestrito é conhecida ou pode ser facilmente computada e nós queremos resolver um problema completamente restrito. Ele parte de uma solução super-ótima e move-se em direção à uma solução ótima esforçando-se para atingir factibilidade.

De forma similar, o algoritmo de refinamento iterativo inicia com uma solução parcialmente restrita (geralmente uma árvore geradora mínima) e move-se à cada iteração em direção à uma solução completamente restrita. O número de iterações e a qualidade da solução final dependerão da função de *blacklisting* específica do problema.

De fato, o 'coração' do método de refinamento iterativo é a definição de uma função de *blacklisting* efetiva que seja capaz de fazer com que, em uma dada iteração do algoritmo, a árvore geradora atual mova-se em direção à uma árvore geradora mais restrita considerando-se $\mathcal{C}_2$. Para tal, ela deve mapear *quantos* arcos penalizar, *quais* arcos penalizar, e *qual a quantidade* de penalidade que deve ser aplicada à esses arcos.

Deo e Kumar [1997] também destacam a facilidade de implementar o algoritmo de refinamento iterativo paralelo pois, de acordo com os autores, a etapa de construção de árvores geradoras é paralelizável, e um processador pode verificar e penalizar um arco da árvore T de maneira independente dos demais processadores.

A próxima seção apresenta alguns trabalhos em que o método IR foi aplicado para resolver problemas $\mathcal{NP}$-Completos, descrevendo a função de *blacklisting* utilizada em cada caso.

4.2 IR aplicado à problemas $\mathcal{NP}$-Completos em Árvores

Essa seção apresentará alguns problemas de árvores geradoras restritas que foram resolvidos com o emprego de algoritmos de refinamento iterativo inspirados em Deo e Kumar [1997]. O objetivo da seção é mostrar como a decisão sobre o projeto de boas funções de *blacklisting* deve estar associada com as características do problema.

4.2.1 Árvore Geradora Mínima com Restrição de Grau

O problema da árvore geradora mínima com restrição de grau (em inglês, *degree-constrained minimum spanning tree*) consiste em, dado um grafo $G = (V, E)$ completo, não direcionado com custo c_{ij} associado à cada arco $e_{ij} \in E$, construir uma árvore geradora de custo mínimo tal que o grau $\delta(i)$ de cada vértice $i \in V$ seja menor ou igual à b_i (Narula e Ho [1980]).

Em seu artigo, Deo e Kumar [1997] definiram o algoritmo de refinamento iterativo geral e experimentaram o método no problema de encontrar uma árvore geradora mínima com restrição de grau em grafos. O algoritmo de refinamento iterativo implementado é paralelo e alterna o cômputo de uma árvore geradora mínima T com acréscimos no peso dos arcos incidentes aos vértices $i \in T$ cujo grau excedem o limite b_i.

No *blacklisting*, esses arcos são penalizados por uma quantidade proporcional à: (i) o número $f[e]$ de vértices que violam a restrição de grau em que um arco e incide; (ii) uma constante k definida pelo usuário; (iii) o peso $w[e]$ de um arco infrator e a faixa de pesos de arcos da árvore geradora atual, denotados por $w_{min} \leq w[e] \leq w_{max}$. O arco infrator de menor peso de cada vértice i com $\delta(i) > b_i$ não sofre a penalidade, pois cada vértice em uma árvore geradora deve ter no mínimo grau 1. Os demais arcos

infratores têm seu peso modificado de acordo com a equação (4.1):

$$w'[e] = w[e] + kf[e]\left(\frac{w[e] - w_{min}}{w_{max} - wmin}\right) w_{max}. \tag{4.1}$$

Em um trabalho similar, Boldon et al. [1995] usaram uma abordagem *dual-simplex* aplicado ao problema de árvore geradora mínima com restrição de grau que envolveu a execução de iterações em dois estágios até que o critério de convergência fosse atingido. O primeiro estágio consistiu em computar uma árvore geradora mínima usando o método de Prim, que nas primeiras iterações viola as restrições de grau de diversos vértices. O segundo estágio consistiu em ajustar o peso dos arcos infratores usando uma função de penalidade que modifica os seus pesos de acordo com a equação (4.2)

$$w'[e] = w[e] + \mathit{fault} \times w_{max} \times \left(\frac{w[e] - w_{min}}{w_{max} - w_{min}}\right). \tag{4.2}$$

Nessa função, *fault* é uma variável que assume os valores 0, 1 ou 2 de acordo com o número de vértices incidentes ao arco que atualmente violam as restrições de grau. Note que a abordagem de Boldon et al. [1995] é muito simular à empregada por Deo e Kumar [1997]. Mao et al. [1997] também fazem referência ao método de refinamento iterativo para resolver o problema da árvore geradora com restrição de grau.

Em Deo e Kumar [1997], alguns resultados experimentais foram apresentados comparando a abordagem descrita acima com outro algoritmo de refinamento iterativo proposto por esses mesmos autores, que utiliza uma atribuição de pesos aleatória para $w[e]$ na faixa $w_{max} + kf[e](w_{max} - w_{min})$. Na comparação dos resultados referenciaremos o algoritmo que usa a função de *blacklisting* apresentado nessa seção como *determinístico*, e o algoritmo que usa a atribuição de pesos aleatória como *aleatório*.

Um dos experimentos realizados pelos autores utilizou algumas instâncias do problema do caixeiro viajante obtidas do *benchmark* TSPLIB (Reinelt [2008]). Uma vez que uma solução para o problema do caixeiro viajante modificada com a remoção do último arco do circuito (que liga o viajante à cidade de origem do trajeto) é uma 2-MST, então um circuito TSP consiste em um limite superior no custo de uma d-MST.

A Tabela 4.1, adaptada de Deo e Kumar [1997], ilustra os resultados obtidos pelos autores. As colunas `TSP` e `MST` mostram os resultados do caixeiro viajante (limite superior) e da árvore geradora mínima (limite inferior) sobre cada instância, e as colunas `DET` e `RAND` mostram, respectivamente, os resultados encontrados no problema 3-MST para os algoritmos de refinamento iterativo que atribuem penalidades nos arcos usando a equação (4.2) e a atribuição de pesos aleatórios para $w[e]$. Valores em destaque indicam os melhores resultados para o problema.

Tabela 4.1. Comparação de resultados para dois algoritmos IR para o problema da árvore geradora mínima com restrição de grau nos vértices (adaptado de Deo e Kumar [1997])

	Limites		**3-MST**	
Problema	*TSP*	*MST*	*DET*	*RAND*
pr264.tsp	49135	41142	**41143**	**41143**
rat575.tsp	6773	6246	**6265**	**6265**
d657.tsp	48912	42487	**42545**	42563
rl1304.tsp	252948	222849	**222982**	**222982**
rl1323.tsp	270199	239986	**240139**	240236
d1655.tsp	62128	56541	**56657**	56663
vm1748.tsp	336556	294627	295786	**295739**
pr2392.tsp	378032	342269	**343657**	378032
pcb3038.tsp	137694	127302	127753	**127731**
rl5934.tsp	–	513952	**515139**	515235

4.2.2 Árvore Geradora Mínima com Restrição de Diâmetro

Outros autores também empregaram a abordagem de refinamento iterativo para outros problemas de árvores, como é o caso do problema de encontrar uma árvore geradora mínima com restrição de diâmetro (do inglês, *diameter-constrained minimum spanning tree*). Tal problema pode ser enunciado como segue: dado um grafo G valorado, não direcionado, e um inteiro positivo k, encontrar uma árvore geradora de peso mínimo dentre todas as árvores geradoras de G que não contenha caminhos com mais do que k arcos. Entende-se por *diâmetro da árvore* o comprimento do caminho mais longo da árvore. Denominaremos DCMST(k) o problema da árvore geradora mínima com restrição de diâmetro para um k específico.

Abdalla et al. [2000] apresentaram dois algoritmos de refinamento iterativo para o problema da árvore geradora com restrição de diâmetro, denominados IR1 e o IR2.

O algoritmo IR1 primeiro computa uma árvore geradora mínima irrestrita. Em seguida, essa árvore geradora é refinada por meio de substituição de arcos até que a restrição de diâmetro seja satisfeita. A função de penalidade é aplicada a um subconjunto de arcos da árvore tal que eles sejam desencorajados de aparecer na árvore geradora mínima resultante da próxima iteração. Arcos presentes no centro de um caminho longo são candidatos naturais à penalização, uma vez que sua remoção divide cada caminho em dois sub-caminhos mais curtos de igual comprimento.

Sejam l um arco do conjunto de arcos a serem penalizados; $w(l)$ o peso atual de l; w_{max} e w_{min} o maior e menor peso de um arco presente na árvore geradora atual; $dist_c(l)$ a distância do arco l ao nó central do caminho, acrescido de 1 unidade. Quando o centro é o arco l_c, temos $dist_c(l_c) = 1$, assim como um arco l incidente à somente um dos extremos do arco central l_c terá $dist_c(l) = 2$. A penalidade imposta à cada arco l

na árvore geradora atual será

$$max\Big\{\left(\frac{w(l)-w_{min}}{dist_c(l)(w_{max}-w_{min})}\right)w_{max},\epsilon\Big\}, \tag{4.3}$$

onde $\epsilon > 0$ é a penalidade mínima que garante que o refinamento iterativo não permanecerá na mesma árvore geradora quando impondo penalizade zero à soma dos arcos.

De acordo com a equação (4.3), a penalidade diminui conforme os arcos penalizados tornam-se mais distantes do centro da árvore geradora atual, de forma que um caminho longo seja quebrado em dois subcaminhos significativamente menores ao invés de um subcaminho curto e outro subcaminho longo.

Diferente do algoritmo IR1, o algoritmo IR2 não recomputa uma árvore geradora mínima à cada iteração. Uma nova árvore geradora é criada a partir da modificação da árvore geradora atual, com a remoção de um arco por vez. Deo e Abdalla [2000] implementaram uma versão paralela do algoritmo IR2 de Abdalla et al. [2000].

Seja $ecc_T(u)$ a excentricidade do vértice u com respeito à T, ou seja, a máxima distância do vértice u para qualquer outro vértice de T. O diâmetro da árvore é dado por $max\{ecc_T(u)\}$, $\forall\ u \in T$.

O algoritmo IR2 inicia computando uma árvore geradora mínima irrestrita no grafo $G = (V, E)$ pelo método de Prim. O valor de excentricidade de todos os vértices da árvore é computado através de um caminhamento *preorder* na árvore, que computa a distância de um dado vértice à todos os demais vértices na árvore geradora em uma etapa que requer $O(n^2)$ cômputos. Quando ocorrem mudanças na árvore geradora apenas os valores de excentricidade que mudaram são recalculados.

Uma vez determinada a árvore geradora T inicial com seus valores de excentricidade computados, o algoritmo IR2 opera iterativamente sobre a árvore alternando os passos entre duas etapas. Na primeira, o algoritmo identifica um arco de T para ser removido de forma a reduzir o diâmetro da árvore. Na segunda etapa, um arco de G é escolhido para substituir o arco removido de T sem aumentar o diâmetro da árvore. Tais etapas são executadas até que a restrição de diâmetro seja cumprida por T. A listagem em pseudo-código extraído de Abdalla et al. [2000] que formaliza o algoritmo IR2 é apresentada a seguir.

Os arcos candidatos à remoção são mantidos em uma lista C ordenada de acordo com o peso desses arcos. A lista é implementada como uma *max-heap* de tal forma que o arco com maior peso é a raíz da *heap*. O centro da árvore T é computado considerando os vértices $u \in T$ tais que $ecc_T(u) = \lceil \frac{diametro}{2} \rceil$, e C conterá todos os arcos incidentes ao conjunto de vértice u.

Remover um arco da árvore não garante que todos os maiores caminhos da árvore

Algoritmo 7 Pseudo-código do algoritmo de IR2 para o árvores de diâmetro restrito

```
1: procedure ARVORE-DIAMETRO-RESTRITO-IR2(G = (V, E), k)
2:     C ← ∅
3:     move ← false
4:     repeat
5:         diameter ← max_{z∈V}(ecc_T(z))
6:         if (C = ∅) then
7:             if (move = true) then
8:                 move ← false
9:                 C ← arcos (u, z) mais distantes 1 arco do centro de T que na iteração anterior
10:            else
11:                C ← arcos (u, z) no centro de T
12:            end if
13:        end if
14:        repeat
15:            (x, y) ← arco de maior peso de C          ▷ Isso dividirá T em duas árvores: T_1 e T_2
16:        until ((C = ∅) OR (max_{u∈T_1}(ecc_T(u)) = max_{z∈T_2}(ecc_T(z))))
17:        if (C = ∅) then
18:            move ← true
19:        else
20:            Remova (u, v) de T
21:            Tome um arco de substituição e insira-o em T
22:        end if
23:        Recompute os valores de ecc_T
24:    until ((diameter < k) OR (estamos removendo arcos distantes do centro de T))
25: end procedure
```

serão 'quebrados', assim é preciso verificar se a remoção de um dado arco divide a árvore T em duas sub-árvores, T_1 e T_2, tal que cada uma delas contenha um vértice v que seja um dos extremos do maior caminho de T. Se o arco de maior peso de C não satisfaz essa condição, então ele é removido de C e o segundo maior em peso de C é considerado. Esse processo segue até que um arco de C seja escolhido respeitando essa condição, ou C esteja vazia (linhas 14 a 16).

Por outro lado, se a lista C não contém bons candidatos para remoção, então é preciso considerar arcos que estejam mais distantes do centro identificando os vértices u com $ecc_T(u) = \lceil \frac{diametro}{2} \rceil + bias$, com $bias$ iniciando em zero e sendo incrementando toda vez que a lista C não possuir arcos apropriados para remoção. A lista C é computada com todos os arcos adjacentes ao conjunto de vértice u. Se um arco apropriado é encontrado, então $bias$ é redefinido como 0.

De acordo com o algoritmo apresentado para o método IR2, a remoção do arco (x, y) causa a divisão da árvore T em duas sub-árvores, T_1 e T_2. Um arco de G que não faz parte de T é escolhido para reconectar essas sub-árvores de forma a reduzir o tamanho de pelo menos um caminho longo de T sem aumentar o diâmetro da árvore.

Tal arco é denominado de arco de substituição. Abdalla et al. [2000] propuseram o método ERM para determinar qual é o arco de substituição; Deo e Abdalla [2000] propuseram uma variação desse método, resultando nos métodos ERM1 e ERM2 onde este último funciona melhor em grafos incompletos e evita a criação de vértices de grau alto no centro de T como ocorre com o ERM1.

De acordo com Abdalla et al. [2000], existem casos desse problema que podem ser resolvidos de maneira exata por algoritmos polinomiais para $k = 2$, $k = 3$ e $k = (|V|-1)$ ou quando o peso de todos os arcos é idêntico. Deo e Abdalla [2000] propuseram um outro algoritmo de refinamento iterativo para resolver um dos casos onde o problema é $\mathcal{NP}$-Completo, para $k = 4$. Essa heurística aproximada inicia com uma solução exata para o problema com $k = 3$, onde o refinamento é feito através da substituição de arcos de maior peso por arcos de menor peso. Detalhes sobre o emprego da abordagem de refinamento para esse problema podem ser consultados nos trabalhos referenciados.

Como resultados experimentais os autores aplicaram o algoritmo IR capaz de encontrar uma solução para $k = 10$ usando os algoritmos ERM1 e ERM2 como variações do método de escolha do arco de substituição para grafos com 100, 200, 300, 400 e 500 vértices. A heurística aproximada proposta por Deo e Abdalla [2000] para o DCMST(4) forneceu um limite superior 'mais firme' na qualidade da solução para o problema DCMST(10) do que o algoritmo polinomial para o problema DCMST(3). Considerando a razão dos pesos das soluções $\left(\frac{DCMST(k)}{MST}\right)$ como uma medida de qualidade da solução, a variação ERM2 apresentou resultados melhores do que a variação ERM1 para todos os grafos testados. A Figura 4.1, extraída de Deo e Abdalla [2000], apresenta o gráfico que comprova esses resultados.

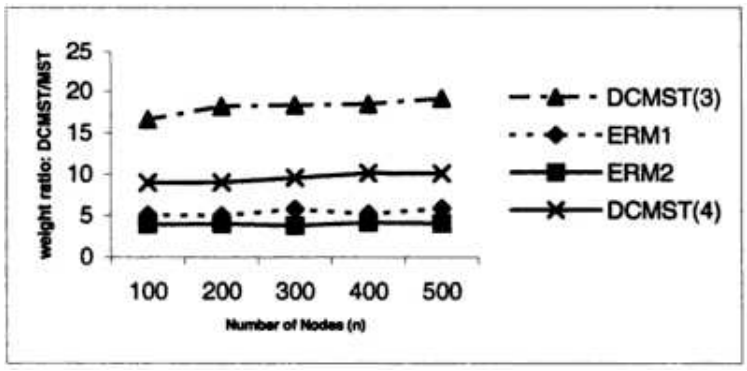

Figura 4.1. Gráfico da razão entre os pesos das soluções $\left(\frac{DCMST(k)}{MST}\right)$ para os algoritmos DCMST(3), DCMST(4) e DCMST(10) em função do tamanho do grafo de entrada, extraído de Deo e Abdalla [2000]

4.2.3 Árvore Geradora Mínima Capacitada

O problema da árvore geradora capacitada consiste em, dados um grafo $G = (V, E)$ não direcionado, um vértice raíz v_0, peso $w(e)$ nos arcos, capacidades $c(e)$ nos arcos e demanda $r(v)$ nos vértices, determinar uma árvore geradora de custo mínimo T dentre todas as árvores geradoras de G que satisfaça uma restrição de capacidade que requer, para cada arco $e \in T$, que $c(e) \geq \sum_{u \in U(e)} r(u)$. $U(e)$ denota o conjunto de vértices cujo caminho em T até a raíz v_0 contém o arco e (Deo e Kumar [1997]).

Seja $e = (p(v), v)$ um arco de T para o qual a capacidade é violada. A notação $p(v)$ refere-se ao 'pai' do vértice v na árvore. Se o arco e viola uma restrição de capacidade, então $c(e) < \sum_{u \in U(e)} r(u)$. Com base nessa observação Deo e Kumar [1997] sugeriram, mas não implementaram, uma função de *blacklisting* que tente *ou* reduzir a demanda para o conjunto de vértices $U(e)$ *ou* substituir um arco e por outro arco de maior capacidade na árvore geradora resultante da próxima iteração.

A redução da demanda de $U(e)$ é alcançada com o incremento do peso dos arcos $e' = (v, x)$ para o qual a demanda média por vértice em $U(e')$ seja muito alta. Essa penalidade encoraja a sub-árvore de raíz x a aparecer em outra parte da árvore geradora, e não *dentro* da sub-árvore de raíz v reduzindo, portanto, a demanda para os vértices de $U(e)$.

Já a substituição do arco e é feita incrementando-se apenas o seu peso. A quantidade de penalidade aplicada é computada somando-se as penalidades advindas de cada violação de capacidade que o arco participa.

Capítulo 5

Algoritmo IR para o Problema MBV

O objetivo deste trabalho foi apresentar um novo algoritmo para o problema de encontrar árvores geradoras com mínimo de vértices *branch*; tal algoritmo foi projetado utilizando o conceito de refinamento iterativo apresentado no Capítulo 4.

5.1 Considerações Iniciais

O Capítulo 4 apresentou casos onde a abordagem de refinamento iterativo foi utilizada para resolver problemas de árvores geradoras restritas. Embora os casos apresentados refiram-se à problemas em grafos com pesos, este trabalho propõe adaptar a abordagem IR para resolver o problema da árvore geradora com número mínimo de vértices *branch* (MBV).

O problema MBV pode ser enunciado como: dado um grafo $G = (V, E)$ conexo, não direcionado e não-valorado encontrar a árvore geradora T dentre todas as árvores geradoras de G que possui a menor quantidade de vértices com grau maior ou igual a 3.

Por definição o problema não é um problema de árvore geradora restrita no sentido conceitual, uma vez que não existe uma restrição explícita vinculada ao problema que restringe a quantidade de vértices *branch* que uma árvore geradora factível pode ter. Ao tratar a relação entre uma solução e sua quantidade de vértices *branch* na função objetivo do problema, permitimos que uma solução factível possua vértices *branch* em qualquer quantidade desde que seja uma árvore geradora; apenas as soluções ótimas possuirão a menor quantidade de vértices *branch*.

Apesar disso iremos considerar alguns elementos chaves da abordagem de refinamento iterativo no algoritmo proposto nesta dissertação. O primeiro deles é que o algoritmo IR proposto irá trabalhar considerando que o grafo G é valorado, embora no enunciado do problema ele não seja. Faremos essa consideração porque a construção

da árvore geradora inicial deve seguir algum tipo de orientação – tal orientação advém do uso de arcos ponderados usado por qualquer algoritmo de árvore geradora mínima.

A segunda consideração importante é sobre o critério de parada. No algoritmo geral proposto por Deo e Kumar [1997] (ver Seção 4.1), o algoritmo pára quando uma solução factível em relação à restrição $\mathcal{C}_2$ é encontrada. Conforme mencionado, para o problema MBV qualquer árvore geradora de G' é factível. Logo será preciso adaptar o critério de parada para considerar outra condição que não seja factibilidade. O algoritmo emprega um critério de parada baseado na melhoria da qualidade da solução: as iterações ocorrerão até que nenhuma melhoria seja realizada na topologia de duas soluções que diferem por apenas uma iteração.

A terceira consideração é sobre a quantidade de iterações necessárias para o algoritmo convergir. Isso dependerá principalmente da solução inicial que será iterativamente refinada pelo algoritmo. A quantidade de 'infrações' contidas na árvore inicial, assim como o grau dos vértices infratores irão ditar a quantidade de iterações executadas no refinamento já que o algoritmo remove um arco por iteração e em geral são necessárias $(\delta(v) - 2)$ substituições de arcos para transformar um vértice $v \in T$ que é *branch* em um vértice não *branch*.

5.2 Algoritmo de Refinamento Iterativo para o MBV

Conforme mencionado, o algoritmo de refinamento iterativo proposto para o problema MBV irá executar sobre uma versão ponderada do grafo G. Com isso o primeiro passo do algoritmo consiste em transformar o grafo G em um grafo com peso $G' = (V, E, w)$ associando-se à cada arco $e \in G$ uma função peso $w : E \rightarrow \mathcal{W}$, onde $\mathcal{W} = \{x \in \Re : 0 \leq x \leq 1\}$. O peso w de cada arco $e \in G'$ é definido inicialmente tomando-se um valor uniformemente aleatório no intervalo $[0, 1]$ e atribuindo-o diretamente à e. Os pesos w designados para cada arco $e \in G'$ constituem a matriz de custo M associada à G'.

Em seguida devemos sacar uma solução inicial para o problema, isto é, uma árvore geradora que seja simples de computar mas que possivelmente conterá uma grande quantidade de vértices *branch*. De fato, o algoritmo proposto utiliza o método de Kruskal para construir uma árvore geradora mínima a partir dos pesos da matriz de custo M. Como a atribuição de pesos de G' é aleatória, então existirão inumeras possibilidades de solução inicial para grafos com a mesma topologia.

É importante esclarecer que a matriz de custo M têm o propósito único de orientar a construção da árvore geradora inicial não tendo nenhum significado clássico associado à modelagem de redes por meio de grafos, tais como distância entre terminais, custo de transmissão ou latência do enlace. A definição do problema MBV feita por

Gargano et al. [2002] não exige que a árvore geradora T com mínimo de vértices *branch* possua também custo mínimo; portanto, qualquer que seja o custo da árvore final, a informação que interessa ao problema é a topologia dessa árvore geradora.

De posse da solução inicial, o algoritmo opera fazendo com que a árvore geradora atual mova-se em direção à uma árvore geradora cuja topologia *favoreça* a eliminação de vértices *branch*. Tal movimento é mapeado como uma modificação de topologia da árvore geradora realizada por meio da eliminação de um arco incidente à um vértice *branch* em T seguido pela substituição desse arco por outro arco de G' que cause o menor impacto possível na formação de vértices *branch* em T.

Isso implica que nem sempre um vértice *branch* é eliminado de T de uma iteração para a outra, pois o algoritmo apenas assegura que, entre uma iteração e outra, um arco seja eliminado da árvore sendo substituído por outro arco mais 'vantajoso' de acordo com a função de *blacklisting* que será apresentada na Seção 5.3. Dependendo do grau de um dado vértice *branch* v, várias iterações poderão ocorrer para reduzir o grau de v à $\delta(v) \leq 2$, considerando a existência de arcos candidatos 'vantajosos' para as substituições.

A aplicação de penalidades do algoritmo de refinamento iterativo proposto para o problema MBV difere das abordagens de penalização de arcos empregadas para o problema de encontrar árvores geradoras com restrição de grau (Seção 4.2.1) e assemelha-se com a abordagem de penalização empregada para o problema de encontrar árvores geradoras com restrição de diâmetro (Seção 4.2.2). Ao invés de penalizar um arco através do incremento de seu peso w para desencorajá-lo a reaparecer na solução da próxima iteração, o algoritmo de refinamento iterativo para o MBV permuta o peso w_c do arco $e_c \in T$ que será eliminado da árvore com o peso w_r do arco $e_r \notin T, e_r \in G$ que irá substituir e_c em T, realizando uma troca explícita de arcos na árvore geradora.

Com isso a árvore geradora que resulta da próxima iteração do algoritmo é construída a partir de uma substituição de arco na árvore geradora da iteração atual, de forma que uma solução não é completamente reconstruída a partir de G'. Denominaremos o arco $e_c \in T$ a ser removido da árvore de *arco de corte* (do inglês, *cutting edge*), uma vez que sua eliminação de T cria um corte na árvore que separa os vértices $v \in V$ em dois sub-conjuntos disjuntos de vértices (S e S') e a árvore geradora em dois componentes conexos disjuntos (T_1 e T_2). Por sua vez, o arco $e_r \notin T, e_r \in G$ que substitui e_c em T e reconecta T_1 e T_2 formando um único componente conexo T será denominado *arco de substituição* (do inglês, *replacement edge*).

Por fim, a estratégia de substituição de arcos em T usada pelo algoritmo de refinamento iterativo persiste até que não existam mais escolhas 'vantajosas' para substituição de arcos de corte por arcos de substituição. Nesse caso nenhuma substituição

melhora a topologia da árvore de forma que o algoritmo pára, retornando a árvore geradora computada na última iteração como solução para o problema.

O algoritmo de refinamento iterativo para o problema MBV é formalizado na listagem em pseudo-código apresentada no Algoritmo 8.

Algoritmo 8 Pseudo-código do algoritmo de refinamento iterativo para o MBV

```
 1: procedure MBV-ITERATIVE-REFINEMENT-ALGORITHM(G = (V, E))
 2:     G' ← AssignRandomWeights(G)
 3:     T ← CalculateMinimumSpanningTree(G');
 4:     T_best ← T
 5:     repeat
 6:         ThereWasExchange ← false
 7:         L_cut ← CreateCuttingList(T)
 8:         while ((ThereWasExchange ≠ true) AND (L_cut ≠ ∅)) do
 9:             (u*, v*) ← SelectArcFromCuttingList(L_cut)
10:             T ← T \ {(u*, v*)}
11:             L_cut ← L_cut \ { (u*, v*) }
12:             L_rep ← CreateReplacementListToCuttingArc(T, G', (u*, v*))
13:             (u', v') ← SelectArcFromReplacementList(L_rep, T, (u*, v*))
14:             if (∃ (u', v')) then
15:                 T ← T ∪ {(u', v')}
16:                 ThereWasExchange ← true
17:                 if (fitness(T) < fitness(T_best)) then
18:                     T_best ← T
19:                 end if
20:             else
21:                 T ← T ∪ {(u*, v*)}
22:             end if
23:         end while
24:     until (ThereWasExchange = false)
        return T_best
25: end procedure
```

Os conjuntos L_{cut} e L_{rep} denotam, respectivamente, o conjunto de arcos de corte da iteração atual e o conjunto de arcos de substituição potenciais para o arco de corte selecionado na iteração atual. A variável T_{best} guarda a topologia da árvore computada com a menor quantidade de vértices *branch*. Inicialmente T_{best} corresponde à árvore geradora mínima T; no decorrer do algoritmo ela é atualizada sempre que uma modificação na árvore T resultar em uma solução com menor quantidade de vértices *branch* (linhas 17 à 19).

O método `AssignRandomWeights()` (linha 2) transforma o grafo $G = (V, E)$ conexo, não direcionado e não valorado em um grafo G' valorado através da atribuição de peso w para cada um de seus arcos. A partir do grafo G' modificado, uma árvore geradora mínima é calculada pelo método `CalculateMinimumSpanningTree()` (linha

3) e será tomada como solução inicial do problema. Tal solução será refinada pelos passos contidos nas linhas 5 à 24.

O refinamento prossegue da seguinte maneira. Primeiro, um conjunto L_{cut} é populado com todos os arcos $e_{ij} \in T$ que incidem à um vértice *branch* em T pelo método `CreateCuttingList()` (linha 7). A variável `ThereWasExchange` marca que nenhuma substituição ocorreu na iteração atual. O método `SelectArcFromCuttingList()` escolherá, dentre os arcos $e_{ij} \in L_{cut}$, o arco (u^*, v^*) que causa mais infrações na árvore como arco de corte (linha 9). A regra usada para determinar tal arco será detalhada na Seção 5.3; por ora assuma que a remoção de (u^*, v^*) é 'vantajosa' para a árvore. O arco escolhido é removido de T (linha 10) e do conjunto L_{cut} (linha 11).

O método `CreateReplacementListToCuttingArc()`, que recebe como entrada o grafo G', a árvore T e o arco de corte (u^*, v^*), irá determinar o corte $[S, S']$ formado pela remoção de (u^*, v^*) em T e populará o conjunto L_{rep} com todos os arcos $e_r \in G'$, $e_r \notin T$ capazes de conectar as sub-árvores T_1 e T_2 originadas do corte sem formar ciclos na árvore T (linha 12). Para cada escolha de (u^*, v^*) haverá um conjunto L_{rep} correspondente.

O método `SelectArcFromReplacementList()` irá avaliar cada arco $e_{ij} \in L_{rep}$, verificando quais correspondem à escolhas vantajosas visando a substituição do arco de corte em T (linha 13). A melhor dessas escolhas corresponderá ao arco (u', v') selecionado. Existem duas situações de retorno para o método `SelectArcFromReplacementList()`: ou o método encontra arcos de substituição vantajosos e retorna o melhor deles, ou não existem arcos de substituição que agregam vantagem à T se trocados pelo arco de corte.

Se existir tal arco (u', v'), então o arco de substituição (u', v') tomará o lugar de (u^*, v^*) em T (linha 15). A variável `ThereWasExchange` sinalizará a substituição ocorrida, encerrando o laço da linha 8 e, consequentemente, a iteração atual.

Entretando, podem existir casos onde nenhum dos arcos $e_{ij} \in L_{rep}$ trazem vantagens à topologia da árvore T considerando uma troca hipotética entre eles e o arco de corte (u^*, v^*). Nesse caso, a variável `ThereWasExchange` irá sinalizar que os passos contidos no laço compreendido entre as linhas 8 e 23 deverão ser repetidos. O arco (u^*, v^*) atual retorna para T (linha 21), e um novo arco de corte (u^*, v^*) é sacado de L_{cut} formando um novo conjunto de candidatos L_{rep} que são potenciais candidatos para substituição de (u^*, v^*) em T.

O laço compreendido entre as linhas 8 e 23 se repete até que ocorra uma substituição de um arco $e_c \in L_{cut}$ por um arco $e_r \in L_{rep}$ ou não existam arcos de substituição 'vantajosos' em L_{rep} para todos os arcos $e_{ij} \in L_{cut}$. Se o último caso ocorre, então o algoritmo não consegue mais refinar a solução e retornará a árvore geradora T_{best}

computada como solução para o problema.

5.3 Política de Substituição de Arcos

O texto da Seção 5.2 cita por diversas vezes o termo arco 'vantajoso'. Em termos do problema MBV, consideraremos como arco 'vantajoso' um arco $e_r \in G', e_r \notin T$ tal que substituir um arco infrator e_c por e_r em T causa *ou* uma redução no número de vértices *branch* de T, *ou* uma modificação na topologia da árvore que reduz o grau de algum dos vértices *branch* de T.

As próximas sub-seções apresentam medidas de contribuição para a formação de infrações na árvore geradora, e o método empregado para selecionar arcos de corte e de substituição nos grafos T e G', respectivamente.

5.3.1 Contabilizando o Grau de Infração de um Arco

O primeiro passo para identificar arcos 'vantajosos' consiste em computar o nível de contribuição que a presença de um determinado arco e em T causa (caso seja um arco de corte) ou causaria (caso seja um arco de substituição) na formação de vértices infratores para a solução correspondente do problema MBV. Entende-se por infração a presença de vértices $v \in T, \delta(v) \geq 3$.

Seja $branch : V \rightarrow \{0, 1\}$ uma função definida por

$$branch(v) = \begin{cases} 1 & \text{se } \delta(v) \geq 3 \\ 0 & \text{se } \delta(v) < 3 \end{cases} \tag{5.1}$$

Dado um arco e_{ij} que incide nos vértices $i, j \in V$, duas medidas determinam a contribuição de e_{ij} para a formação de vértices infratores da árvore T, a saber:

- α_{ij}: quantidade de vértices v extremos do arco e_{ij} que possuem $\delta(v) \geq 3$, ou seja, o número de vértices *branch* que e_{ij} incide. Pode assumir os valores 0, 1 ou 2, e é calculado por

$$\alpha_{ij} = \sum_{v \in \{i,j\}} branch(v) \tag{5.2}$$

- σ_{ij}: soma dos graus dos vértices i e j que o arco e_{ij} incide, desconsiderando o próprio arco e_{ij}. Tal medida servirá para fornecer uma noção de densidade de

arcos desses vértices, e é dada por

$$\sigma_{ij} = \left(\sum_{v \in \{i,j\}} \delta(v) \right) - 2 \tag{5.3}$$

Tais medidas são a base de funcionamento dos algoritmos de seleção detalhados nas Seções 5.3.2 e 5.3.3 para a escolha de bons arcos candidatos à arco de corte e de bons arcos candidatos à arco de substituição.

A Figura 5.1 ilustra um grafo $G = (V, E)$ e uma possível árvore geradora $T = (V, E')$ de G. Os valores de α_{ij} e σ_{ij} de todos os arcos $(i, j) \in T$ são dados na Tabela 5.1.

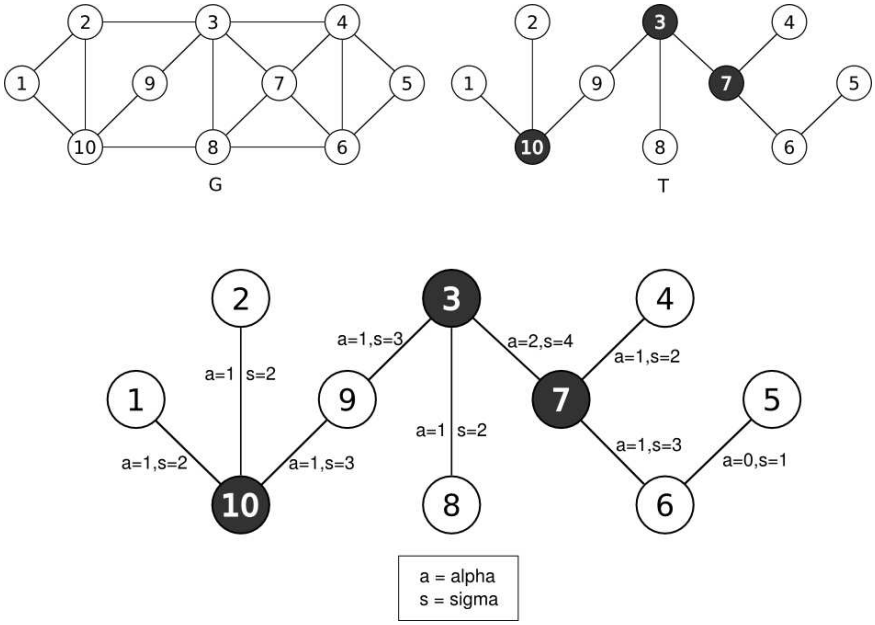

Figura 5.1. Medidas de infração de um arco (i, j) na árvore geradora T

Tabela 5.1. Valores de α_{ij} e $\sigma_{i,j}$ para os arcos $e_{ij} \in T$

i	j	$\delta(i)$	$\delta(j)$	$\delta(i) + \delta(j)$	α_{ij}	σ_{ij}
1	10	1	3	4	1	2
2	10	1	3	4	1	2
3	9	3	2	5	1	3
3	8	3	1	4	1	2
3	7	3	3	6	2	4
4	7	1	3	4	1	2
5	6	1	2	3	0	1
6	7	2	3	5	1	3
9	10	2	3	5	1	3

Nota-se claramente pela Tabela 5.1 que arcos (i, j) que possuem valores mais altos de α_{ij} e σ_{ij} são os arcos que mais contribuem para a formação de vértices *branch* em uma árvore geradora T. No caso específico da árvore T apresentada na Figura 5.1, esses arcos correspondem aos arcos $(3, 9)$, $(6, 7)$, $(9, 10)$ e $(3, 7)$, sendo $(3, 7)$ o arco mais prejudicial de todos pois incide em 2 vértices *branch* (de fato, $\alpha_{3,7} = 2$).

De maneira similar, arcos (i, j) que possuem os menores valores de α_{ij} e $\sigma_{i,j}$ são os que menos contribuem para a formação de vértices *branch* na árvore geradora T. No exemplo anterior, $(1, 10)$, $(2, 10)$, $(3, 8)$, $(4, 7)$ e $(5, 6)$ são arcos que menos influenciam a formação de vértices *branch*. O arco $(5, 6)$ é o arco mais desejável de ser mantido na árvore geradora, pois não incide em vértices *branch* e contribui pouco para que seus vértices extremos tornem-se *branch*: $\delta_T(5) = 1$ e $\delta_T(6) = 2$

Os resultados dessa observação foram aplicados no projeto dos algoritmos de seleção de 'bons' arcos de corte e de 'bons' arcos de substituição, apresentados nas próximas sub-seções. Como tais métodos consistem no coração do algoritmo de refinamento iterativo proposto para o problema MBV, podemos dizer que tal algoritmo é uma heurística que 'acredita' ser capaz de melhorar a qualidade de uma solução para o problema MBV orientando a substituição iterativa do arco $e_c \in T$ com maior valor de α e σ pelo arco $e_r \in G', e_r \notin T$ que, se inserido em T no lugar de e_c, apresentará o menor valor de α e σ.

5.3.2 Algoritmo de Seleção do Arco de Corte

A Sub-seção 5.3.1 apresentou a maneira que o algoritmo computa o grau de infração que um dado arco (i, j) exerce sobre uma árvore geradora T. Devido às observações feitas sobre as medidas α_{ij} e $\sigma_{i,j}$, sabemos que 'bons' arcos de corte são aqueles que incidem à muitos vértices infratores. Podemos formalizar que 'bons' candidatos à arco de corte são os arcos $e_{ij} \in T$ com os maiores valores de α_{ij} seguido pelos maiores valores de σ_{ij}.

O método `SelectArcFromCuttingList()` opera sobre o conjunto L_{cut}, formado por todos os arcos $e \in T$ que incidem à pelo menos um vértice *branch* de T. A construção de L_{cut} é formalizada no pseudo-código do Algoritmo 9. A notação $\delta_T(v)$ refere-se ao grau do vértice $v \in T$.

O algoritmo de seleção do arco de corte varre o conjunto L_{cut}, calculando os valores de α e σ de todos os arcos $e_{ij} \in L_{cut}$ e retornando o arco $e_c \in L_{cut}$ que possui o maior valor de α seguido pelo maior valor de σ

A listagem do Algoritmo 10 apresenta o pseudo-código para o método `SelectArcFromCuttingList()`. No código, e_c é uma variável que aponta para o melhor candidato à arco de corte. Tal variável é atualizada sempre que um arco com

Algoritmo 9 Pseudo-código para o método de construção de L_{cut}

```
1: procedure CreateCuttingList(T)
2:     L_cut ← ∅
3:     for (∀ e_ij ∈ T) do
4:         if ((δ_T(i) ≥ 3) OR δ_T(j) ≥ 3)) then
5:             L_cut ← L_cut ∪ {e_ij}
6:         end if
7:     end for
       return L_cut
8: end procedure
```

maior valor de α conhecido for encontrado, ou quando houverem empates em α com diferentes medidas de σ.

Algoritmo 10 Pseudo-código para o método de seleção do arco de corte

```
1: procedure SelectArcFromCuttingList(L_cut)
2:     α_c ← 0
3:     σ_c ← 0
4:     e_c ← NULL
5:     for (∀ e_ij ∈ L_cut) do
6:         α_ij ← CalculateAlpha(e_ij)
7:         σ_ij ← CalculateSigma(e_ij)
8:         if (α_ij > α_c) then
9:             α_c ← α_ij
10:            σ_c ← σ_ij
11:            e_c ← e_ij
12:        else
13:            if ((α_ij = α_c) AND (σ_ij > σ_c)) then
14:                α_c ← α_ij
15:                σ_c ← σ_ij
16:                e_c ← e_ij
17:            end if
18:        end if
19:    end for
       return e_c
20: end procedure
```

O valor de σ é importante para casos onde ocorrem empates na escolha do melhor arco de corte da iteração. Suponha existirem múltiplos arcos $e \in L_{cut}$ com o valor máximo admitido para α, isto é, 2. A remoção de qualquer um desses arcos causa impacto positivo na tentativa de redução do número total de vértices *branch* da solução e impacta diretamente na redução do grau de seus vértices extremos. Utilizar σ como critério de desempate tem o efeito de priorizar a eliminação de vértices *branch* com alta densidade de arcos incidentes. Assim, sempre que ocorrer empate entre dois arcos candidatos à corte com relação ao valor de α, o valor de σ será considerado,

prevalescendo como escolha o arco com maior σ, isto é, que incide em vértices 'mais densos'.

A Figura 5.2 ilustra a escolha do arco de corte para a árvore T da Figura 5.1. Nessa árvore geradora são candidatos à arco de corte os arcos $L_{cut} = \{ (1,10), (2,10), (3,7), (3,8), (3,9), (4,7), (6,7), (9,10) \}$. De acordo com a Tabela 5.1, a melhor escolha de arco de corte para T é o arco $(3,7)$, com $\alpha_{3,7} = 2$.

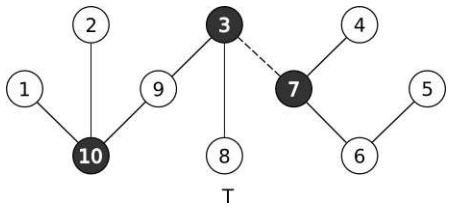

Figura 5.2. Árvore geradora T com o 'arco de corte' $(3,7)$ em destaque

Sua remoção de T causa a separação dos vértices da árvore em dois sub-conjuntos disjuntos $S = \{ 1, 2, 3, 8, 9, 10 \}$ e $S' = \{ 4, 5, 6, 7 \}$, assim como a divide em dois componentes conexos T_1 e T_2. Se houver algum arco de substituição em G' capaz de conectar as sub-árvores T_1 e T_2 em um único componente conexo sem formar ciclos, e que ao mesmo tempo seja 'vantajoso' (isto é, sua inclusão em T causa menos violações do que a árvore possui atualmente) então podemos reduzir o número de vértices *branch* da árvore. A próxima sub-seção detalhará como tal melhoria pode ser feita.

5.3.3 Algoritmo de Seleção do Arco de Substituição

A sub-seção anterior tratou sobre como fazer uma boa escolha de arco de corte na árvore geradora T. A presente sub-seção irá complementar o funcionamento do algoritmo de refinamento iterativo, explicando como ocorre a tomada de decisão sobre qual é o arco de substituição que melhor substitui o arco de corte, considerando os candidatos possíveis.

De acordo com a Seção 5.3.2, um bom arco de corte é decidido através da escolha do arco candidato com maior valor de α e σ, valores estes que orientam a detecção dos arcos que mais contribuem para a formação de vértices *branch*. De maneira similar, um bom arco de substituição é aquele capaz de ligar o corte $[S, S']$ formado pela remoção do arco e_c de T que ainda não tenha sido inserido em T em uma iteração passada, e que contribua para a redução do número de vértices *branch* de T.

Arcos candidatos à arco de substituição são guardados em um conjunto especial, denominado L_{rep}. O conjunto L_{rep} contém todos os arcos $e_{ij} \in G'$, $e_{ij} \notin T$, $e_{ij} \neq e_c$ tal que $i \in S$ e $j \in S'$, ou vice-versa. Tais condições fazem com que arcos candidatos

à substituição sejam arcos que, se inseridos em T, eliminam o corte $[S, S']$ sem formar ciclos na árvore. A listagem do Algoritmo 11 apresenta um pseudo-código para a construção de L_{rep}.

Algoritmo 11 Pseudo-código para o método de construção de L_{rep}

```
 1: procedure CreateReplacementListToCuttingArc(T, G', (u*, v*))
 2:     S ← {v ∈ T : v is reached from u*, v is not reached from v* }
 3:     S' ← {v ∈ T : v is reached from v*, v is not reached from u* }
 4:     L_rep ← ∅
 5:     for (∀ e_ij ∈ G') do
 6:         if (e_ij ≠ e_u*v*) then
 7:             if (e_ij ∉ T) then
 8:                 if (i ∈ S, j ∉ S) OR (i ∈ S', j ∉ S') then
 9:                     L_rep ← L_rep ∪ {e_ij}
10:                 end if
11:             end if
12:         end if
13:     end for
        return L_rep
14: end procedure
```

O arco de substituição é o arco $e_r \in L_{rep}$ que, após ser inserido em T no lugar de e_c, causa o menor impacto possível na formação de vértices *branch* em T. Assim, o algoritmo que seleciona do arco de substituição e_r computa os valores de α e σ de cada arco candidato em $e_{ij} \in L_{rep}$ supondo que ele estará substituindo e_c na árvore geradora T. O arco e_r escolhido será aquele que, se inserido na árvore geradora T em substituição ao arco e_c, admite o menor valor de α, seguido pelo menor valor de σ, respeitando $\alpha_r \leq \alpha_c$.

A listagem apresentada no Algoritmo 12 formaliza os passos para a de seleção do arco de substituição e_r a partir da lista de candidatos L_{rep}.

Embora o algoritmo de seleção do arco de substituição guarde semelhanças com o algoritmo de seleção do arco de corte, existem diferenças fundamentais na forma de escolher tal arco. Em primeiro lugar, as variáveis α_{ij} e σ_{ij} são computadas considerando-se o arco e_{ij} inserido em T (linhas 8 e 11). O laço compreendido entre as linhas 7 e 23 realiza a varredura sobre os arcos $e_{ij} \in L_{rep}$ para determinar qual deles é o arco e_r com menor valor de α e σ.

Mesmo que exista um arco $e_r \in L_{rep}$, nem sempre ele é retornado como arco de substituição para e_c. Isso acontece porque um arco de substituição tem que trazer alguma melhoria na topologia da árvore T. Entende-se por melhoria como a redução de um vértice *branch* de T ou a redução no grau de algum dos vértices *branch* da árvore. Se porventura o arco e_r encontrado de L_{rep} não possuir valores de α e σ 'melhores' do

Algoritmo 12 Pseudo-código para o método de seleção do arco de substituição

```
 1: procedure SELECTARCFROMREPLACEMENTLIST(L_rep, T, e_c)
 2:     α_c ← CalculateAlpha(e_c)
 3:     σ_c ← CalculateSigma(e_c)
 4:     α_r ← ∞
 5:     σ_r ← ∞
 6:     e_r ← NULL
 7:     for (∀ e_ij ∈ L_rep) do
 8:         T ← T ∪ {e_ij}
 9:         α_ij ← CalculateAlpha(e_ij)
10:         σ_ij ← CalculateSigma(e_ij)
11:         T ← T \ {e_ij}
12:         if (α_ij < α_r) then
13:             α_r ← α_ij
14:             σ_r ← σ_ij
15:             e_r ← e_ij
16:         else
17:             if ((α_ij = α_r) AND (σ_ij < σ_r)) then
18:                 α_r ← α_ij
19:                 σ_r ← σ_ij
20:                 e_r ← e_ij
21:             end if
22:         end if
23:     end for
24:     if (α_r < α_c) then
25:         return e_r
26:     else
27:         if ((α_r = α_c) AND (σ_r < σ_c)) then
28:             return e_r
29:         else
30:             return NULL
31:         end if
32:     end if
33: end procedure
```

que o arco e_c, então substituí-lo em T no lugar de e_c não traz vantagens para a topologia da solução. Tal verificação é feita no código compreendido entre as linhas 24 e 32.

Note que se nenhum arco e_r é retornado, é porque para o arco e_c atual não existe substituição vantajosa. Nesse caso, o algoritmo de refinamento iterativo deverá sacar outro arco de corte de L_{cut}, repetindo o processo de criar L_{rep} e determinar um arco e_r vantajoso para a substituição do novo arco e_c. Se $L_{cut} = \emptyset$, então não existe nenhuma substituição vantajosa na árvore, e o algoritmo encerra conforme descrito na Seção 5.2.

Segue um exemplo ilustrativo de funcionamento do algoritmo de seleção do arco de substituição. Considere a Figura 5.3, com o arco de corte $e_c = (3, 7)$ destacado em (a). A eliminação do arco $(3, 7)$ de T causa o efeito de criar o corte $[S, S']$ e separar T em

T_1 e T_2, conforme ilustra (c), com $S = \{ 1, 2, 3, 8, 9, 10 \}$ e $S' = \{ 4, 5, 6, 7 \}$. Para unir novamente T_1 e T_2 em um único componente conexo T, devemos construir a lista de candidatos L_{rep} capazes de unir os componentes disjuntos sem criar ciclos em T.

Respeitando as condições que permitem um dado arco e_{ij} fazer parte dessa lista, e considerando todos arcos de G' ilustrados em (b), teremos $L_{rep} = \{ (3,4), (6,8), (7,8) \}$. A Figura (d) mostra claramente que esses arcos são a única possibilidade de reconectar T_1 e T_2.

O próximo passos é varrer L_{rep} a fim de determinar se existe algum arco de substituição 'vantajoso' e_r tal que ocorra alguma melhoria na topologia de T se inserirmos o arco e_r no lugar do arco e_r. As Figuras (e), (f) e (g) ilustram cada possibilidade de conexão dos arcos $e_{ij} \in L_{rep}$ em T. Usando-as, obtemos as seguintes informações sobre os valores de α e σ dos arcos candidatos, dados pela Tabela 5.2:

Tabela 5.2. Valores de α_{ij} e $\sigma_{i,j}$ para os arcos $e_{ij} \in L_{rep}$

i	j	$\delta(i)$	$\delta(j)$	$\delta(i) + \delta(j)$	α_{ij}	σ_{ij}
3	4	3	2	5	1	3
6	8	3	2	5	1	3
7	8	3	2	5	1	3

Pela tabela, todos os arcos de substituição possuem os mesmos valores de α e σ. Tomaremos $e_r = (3,4)$. Uma vez determinado e_r, verifica-se se sua substituição por e_c traz vantagens à T, comparando os valores de α e σ. Como $\alpha_{3,4} = 1 < \alpha_{3,7} = 2$, então e_r é vantajoso e será inserido em T, resultando na árvore geradora ilustrada em (e).

A situação ilustrada resultou na redução do número de vértices *branch* de T, logo está claro que a substituição do arco $(3,7)$ pelo arco $(3,4)$ trouxe vantagens para T. Entretando, existem casos onde nem sempre é possível reduzir o número de vértices *branch* em uma única iteração. Na prática, o algoritmo pode empregar um esforço de muitas iterações para reduzir o número de vértices infratores, já que cada iteração pode eliminar, no melhor caso, 2 infrações.

5.4 Exemplo de execução passo-à-passo

O objetivo dessa seção é mostrar a execução passo à passo de uma instância simples do problema da árvore geradora com número mínimo de vértices *branch*, que pode ser resolvido em poucas iterações.

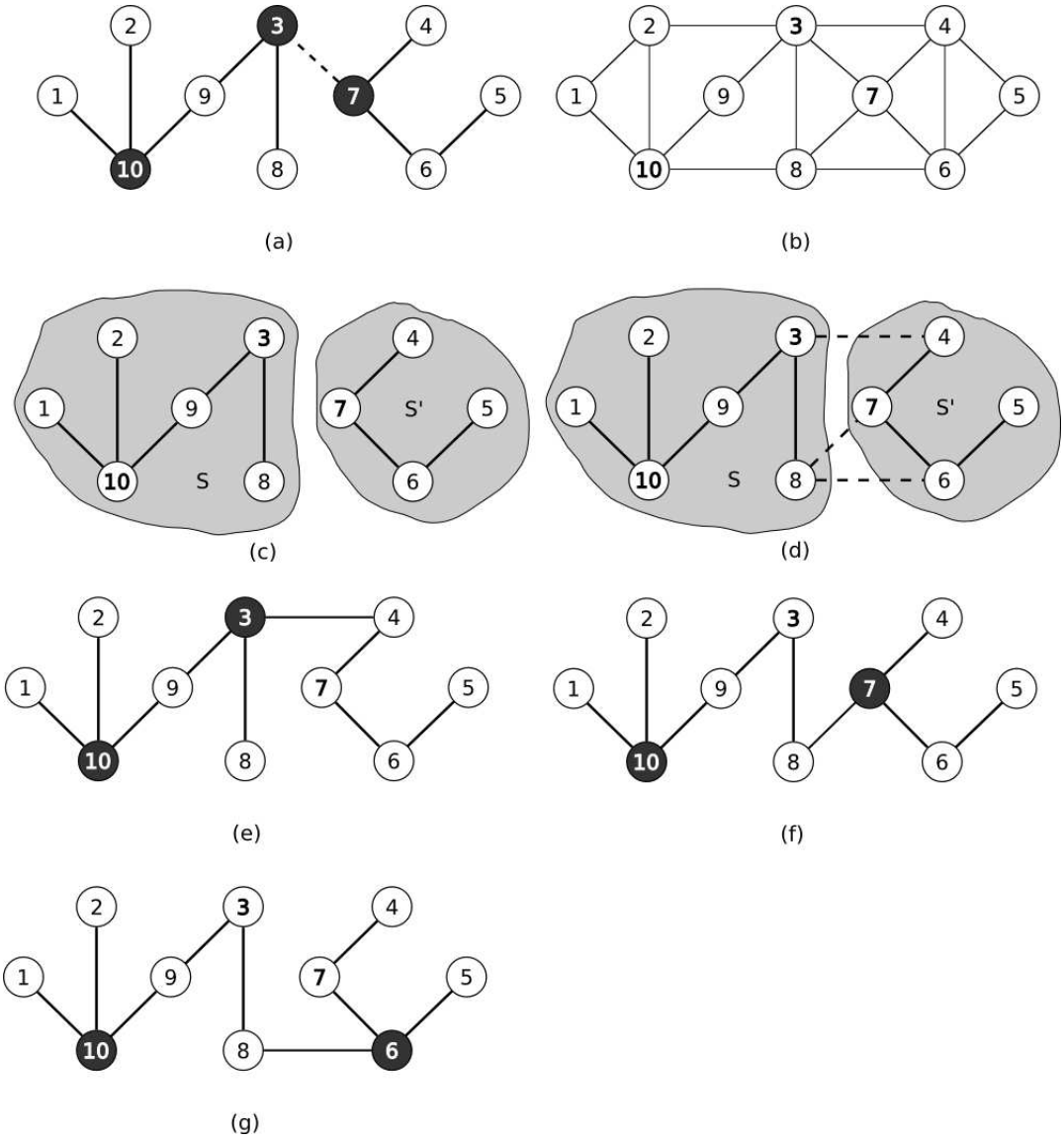

Figura 5.3. Processo de determinação de um 'arco de substituição' em T

5.4.0.1 Iteração 1

Seja G um grafo conexo, não direcionado e não valorado apresentado na Figura 5.4 (a). Suponha que a atribuição aleatória de pesos aos arcos de G transformaram-no em um grafo valorado G', cuja árvore geradora mínima T corresponde à Figura (b).

Note que T possui 2 vértices *branch*: o vértice 6 e o vértice 9. À cada iteração, o algoritmo percorre os arcos de T que incidem em vértices *branch* e insere-os na lista L_{cut}. Para a árvore apresentada em (b), temos $L_{cut} = \{ (6,4), (6,5), (6,7), (9,2), (9,7), (9,8) \}$, com $[\alpha_{6,4} = 1, \sigma_{6,4} = 3]$; $[\alpha_{6,5} = 1, \sigma_{6,5} = 2]$; $[\alpha_{6,7} = 1, \sigma_{6,7} = 3]$; $[\alpha_{9,2} = 1, \sigma_{9,2} = 3]$; $[\alpha_{9,7} = 1, \sigma_{9,7} = 3]$; $[\alpha_{9,8} = 1, \sigma_{9,8} = 2]$. Como ocorreu empate nos valores

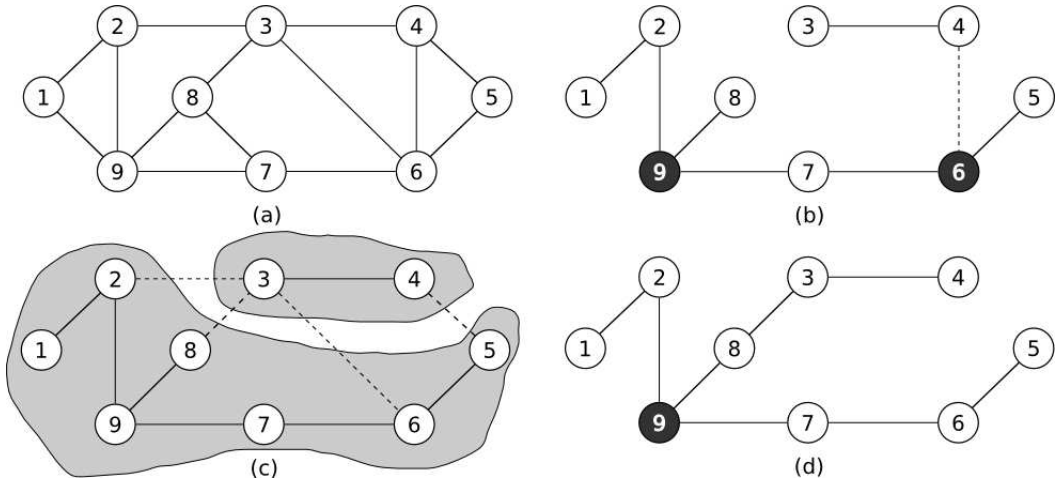

Figura 5.4. Resolvendo G pelo algoritmo IR: iteração 1

de α e σ, escolhemos o primeiro que apareceu com os maiores valores de α e σ, isto é $e_c = (6, 4)$.

O arco de corte $(6, 4)$ é então removido de T (c), causando um corte que separa o conjunto de vértices de T em $S = \{\ 1,\ 2,\ 5,\ 6,\ 7,\ 8,\ 9\ \}$ e $S' = \{\ 3,\ 4\ \}$. O próximo passo consiste em escolher um arco de substituição para e_c; construíremos o conjunto de candidatos L_{rep}. Levando em conta que L_{rep} deve conter os arcos que conectam os arcos do corte $[S, S']$ sem formar ciclos, então teremos $L_{rep} = \{\ (2, 3),\ (3, 6),\ (3, 8),\ (4, 5)\ \}$.

O algoritmo escolhe $e_r \in L_{rep}$ considerando o arco com menores valores de α e σ. Temos que $[\alpha_{2,3} = 1,\ \sigma_{2,3} = 3]$; $[\alpha_{3,6} = 1,\ \sigma_{3,6} = 3]$; $[\alpha_{3,8} = 0,\ \sigma_{3,8} = 2]$; $[\alpha_{4,5} = 0,\ \sigma_{4,5} = 2]$, de onde os arcos $(3, 8)$ e $(4, 5)$ substituem o arco $(6, 4)$ reduzindo o número de infrações na árvore T. Como ocorreu empate, tomaremos o primeiro, ou seja, $e_r = (3, 8)$. A substituição de e_r por e_c transforma T na árvore ilustrada em (d).

5.4.0.2 Iteração 2

A próxima iteração é ilustrada na Figura 5.5. Partindo da árvore geradora apresentada em (b), temos que ainda resta o vértice 9 como vértice *branch* em T, o recálculo de L_{cut} resulta em $L_{cut} = \{(9, 2), (9, 7), (9, 8)\}$, com $[\alpha_{9,2} = 1,\ \sigma_{9,2} = 3]$; $[\alpha_{9,7} = 1,\ \sigma_{9,7} = 3]$; $[\alpha_{9,8} = 1,\ \sigma_{9,8} = 3]$.

Como ocorreu empate em α e σ entre os arcos de L_{cut}, tomaremos o primeiro, $e_c = (9, 2)$. Eliminar e_c de T cria o corte $S = \{\ 1,\ 2\ \}$ e $S' = \{\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9\ \}$, ilustrado em (c). Os arcos capazes de ligar o arco $[S, S']$ sem formar ciclos são $L_{rep} = \{\ (1, 9),\ (2, 3)\ \}$ com $[\alpha_{1,9} = 1,\ \sigma_{1,9} = 3]$; $[\alpha_{2,3} = 1,\ \sigma_{2,3} = 3]$. Pela ocorrência do empate tomaremos o primeiro, $(1, 9)$.

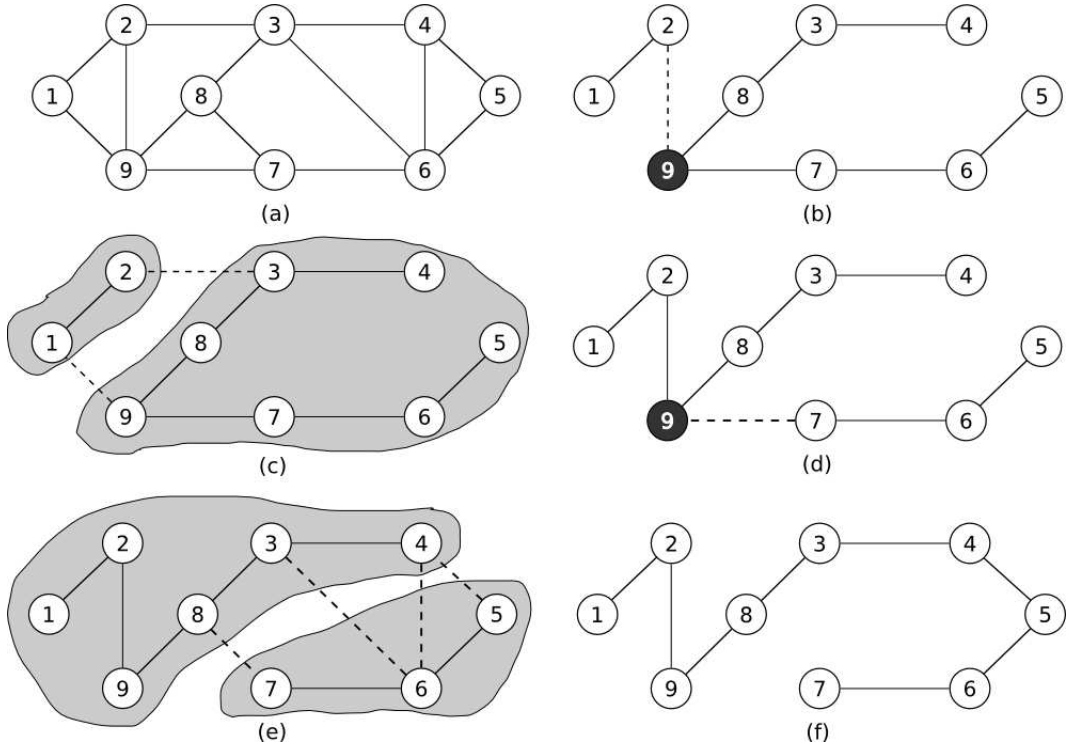

Figura 5.5. Resolvendo G pelo algoritmo IR: iteração 2

Note, entretanto, que $\alpha_{1,9} = 1$, $\alpha_{9,2} = 1$, $\sigma_{1,9} = 3$ e $\sigma_{9,2} = 3$; logo não existe vantagem alguma substituir o arco $e_c = (9, 2)$ pelo arco $e_r = (1, 9)$ pois continuaremos tendo 1 vértice *branch* na árvore, e a substituição não reduz a densidade de arcos do vértice 9. Para esse arco de corte, não existem arcos de substituição vantajosos.

O algoritmo aborta a substituição e tenta novamente, dessa vez com outro arco de corte tomado de $L_{cut} = \{(9, 7), (9, 8)\}$. Note que o arco $(9, 2) \notin L_{cut}$, pois foi retirado desse conjunto no início da iteração 2. Dos arcos que restam em L_{cut}, tomaremos o arco $(9, 7)$ devido ao empate em α e σ. Assim, o novo arco de corte da iteração será $e_c = (9, 7)$ ilustrado na Figura (d).

Remover o arco $(9, 7)$ da árvore resulta no corte formado pelos conjuntos $S = \{$ 1, 2, 3, 4, 8, 9 $\}$ e $S' = \{$ 5, 6, 7 $\}$, portanto $L_{rep} = \{$ $(3, 6)$, $(4, 6)$, $(4, 5)$, $(7, 8)$ $\}$ com $[\alpha_{3,6} = 2, \sigma_{3,6} = 4]$; $[\alpha_{4,5} = 0, \sigma_{4,5} = 2]$; $[\alpha_{4,6} = 1, \sigma_{4,6} = 3]$; $[\alpha_{7,8} = 1, \sigma_{7,8} = 3]$ conforme ilustra a Figura (e). Portanto, o melhor candidato para arco de substituição é o arco $e_r = (4, 5)$. Inserir o arco $(4, 5)$ no lugar do arco $(9, 7)$ elimina um vértice infrator de T, resultando no caminho hamiltoniano apresentado na Figura (f).

5.4.0.3 Iteração 3

A terceira iteração começa com a árvore geradora sem nenhum vértice infrator. Com isso, $L_{cut} = \emptyset$ e não existem mais arcos de corte para substituição na árvore. O

algoritmo encerra, retornando a árvore ilustrada na Figura 5.5 (c) como solução para essa instância.

Embora no caso ilustrado tenha resultado em uma solução ótima para o problema, não existe garantia de otimalidade no método, pois a qualidade da solução refinada depende da solução inicial obtida através do uso do método de Kruskal no início do algoritmo. Espera-se, com isso, obter soluções sub-ótimas para a maioria dos casos.

5.4.0.4 Um exemplo mais complexo

A seguir, um exemplo de execução do algoritmo em um grafo com $n = 50$, $m = 188$ cuja solução foi refinada pelo algoritmo implementado. Vértices em destaque são vértices $v \in T : \delta(v) \geq 3$.

A Figura 5.6 mostra 12 iterações do algoritmo de refinamento iterativo que, partindo de uma solução inicial com 13 vértices, conseguiu chegar à uma solução final que é um caminho hamiltoniano.

Exceto da iteração 5 para 6, que eliminou 2 vértices *branch* de uma só vez, as demais iterações conseguiram reduzir 1 infração por iteração.

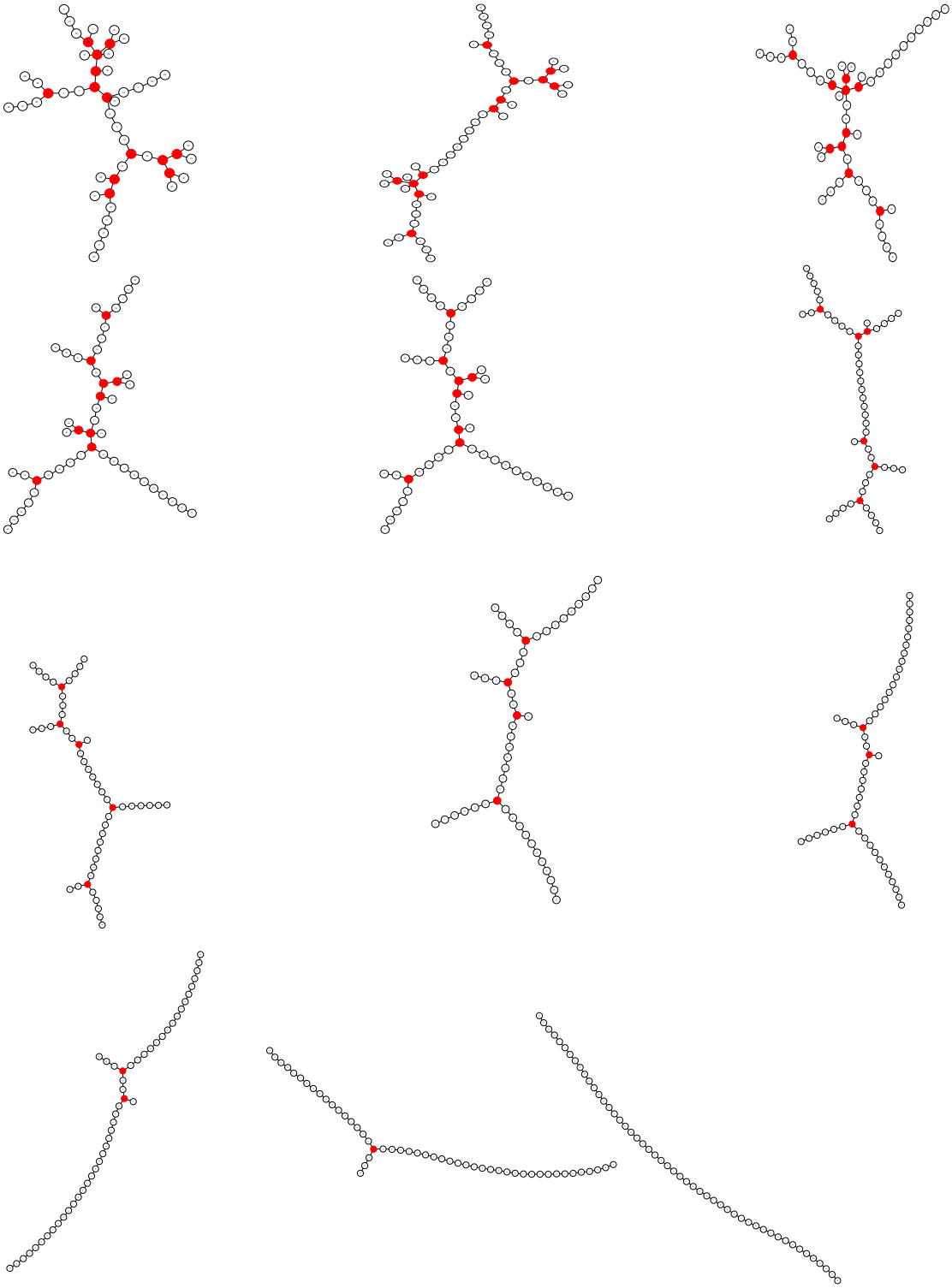

Figura 5.6. Resolvendo uma instância $n = 50$, $m = 188$ pelo IR em 12 iterações

Capítulo 6

Resultados Experimentais

Os experimentos descritos nesse capítulo visam criar um cenário de comparação de resultados para diferentes heurísticas capazes de resolver o problema da árvore geradora com número mínimo de vértices *branch*. Para tal são utilizados seis conjuntos de instâncias onde os métodos EWS e NCH de Cerulli et al. [2009] e o método IR desenvolvido nesta dissertação serão aplicados. As próximas seções descrevem o ambiente onde os algoritmos implementados serão executados, os conjuntos de instâncias usados, e os resultados experimentais que decorrem da aplicação dos três métodos comentados.

6.1 Ambiente

Os algoritmos descritos nas Seções 3.2.1, 3.2.2 e 5.2 foram implementados para permitir a comparação entre a qualidade dos resultados retornados por cada um deles sobre um conjunto de instâncias determinado. Todos esses algoritmos foram implementados em `ANSI C++` (Schildt [1998]) utilizando as bibliotecas `STL` (do inglês, *Standard Template Library*) (Weidl e Jazayeri [1996]) disponíveis para a linguagem, e executados em ambiente `Linux version 2.6.27 -14- generic (buildd@vernadsky) (gcc version 4.3.2 (Ubuntu 4.3.2-1ubuntu11) )` rodando em uma máquina `Intel(R) Core(TM)2 CPU T5500 1.66GHz` com `2048 KB` de memória cache e `1 GB` de memória RAM.

6.2 Ferramentas Computacionais

6.2.1 Aplicativo `mbv-solver.bin`

A implementação eficiente dos métodos citados, e do cálculo da árvore geradora mínima que representa a solução inicial para o método IR utilizaram uma estrutura de dados eficiente para representar conjuntos disjuntos denominada *union-find data structures*.

Tais estruturas permitem gerenciar eficientemente a operação de união entre conjuntos disjuntos, assim como verificar a pertinência de um dado vértice em um conjunto disjunto. Foi empregada principalmente para verificar arcos que ligam um corte $[S, S']$ no algoritmo IR, e para verificar se os vértices u^* e v^* pertencem à diferentes componentes conexos nos métodos EWS e NCH. Detalhes sobre a implementação eficiente e análise de complexidade de estruturas *union-find* podem ser encontradas em Cormen et al. [2001] e Ahuja et al. [1993].

Os algoritmos foram integrados em um aplicativo linha de comando que recebe grafos por meio de arquivos de entrada no formato DIMACS (ver adiante), e permite registrar os resultados de tempo e qualidade em arquivo de saída textual e árvore geradora solução no formato `.dot` (Koutsofios et al. [1991]). A sintaxe do aplicativo `mbv-solver.bin` é dada na Figura 6.1:

```
mbv-solver.bin <metodo> <seed> <entrada> <saida>
entrada: arquivo de entrada no formato NETGEN

saida  : arquivo de saida no formato DOT

metodos:
        0: Cerulli, Gentili, Iossa: estratégia de pesos nos arcos
        1: Cerulli, Gentili, Iossa: estratégia de coloração de vértices
        2: Silva                  : refinamento iterativo
```

Figura 6.1. Sintaxe do *solver* implementado para o problema MBV

Após a execução da aplicação, dois arquivos de saída são gerados contendo informações sobre o resultado. O primeiro deles, com extensão `.out`, contém um resumo da execução no formato textual ilustrado pela Figura 6.2.

```
alb1000.txt        1000        1998        24218        23.7975        73        NCH
```

Figura 6.2. Formato de saída de uma execução do aplicativo `mbv-solver.bin`

A primeira coluna contém o nome da instância executada, a segunda coluna o número de vértices do grafo correspondente, a terceira coluna contém o número de arcos desse grafo, a quarta coluna contém a semente informada pelo usuário, a quinta coluna contém o tempo de execução da instância em segundos, a sexta coluna contém o valor da função objetivo encontrada para essa instância em número de vértices *branch* e, por fim, a sétima coluna indica qual algoritmo foi utilizado para resolver a instância: EWS, NCH ou IR. O arquivo não contém cabeçalho para facilitar a concatenação de resultados obtidos sobre várias execuções em um único arquivo de resumo.

O outro arquivo contém o grafo resultante da execução, isto é, a árvore geradora com a menor quantidade de vértices *branch* que foi possível computar por cada um dos três algoritmos, armazenada no formato `.dot`. Tal formato permite descrever grafos textualmente, indicando os vértices e arcos do grafo e determinando propriedades tais como cor, tamanho e disposição dos cada vértice do grafo. A Figura 6.3 apresenta um exemplo de um arquivo no formato `.dot` com as estruturas mínimas descritas.

```
graph Grafo {
        1 -- 4
        1 -- 17
        1 -- 16
        2 -- 20
        2 -- 5
        2 -- 8
        3 -- 15
        4 -- 8
        4 -- 5
        4 -- 9
        5 -- 10
        6 -- 18
        6 -- 17
        6 -- 16
        7 -- 19
        8 -- 12
        8 -- 19
        8 -- 16
        9 -- 3
        10 -- 7
        10 -- 4
        10 -- 14
        11 -- 17
        12 -- 14
        12 -- 18
        13 -- 9
        13 -- 15
        14 -- 13
        14 -- 3
        14 -- 10
        15 -- 11
        16 -- 1
        16 -- 2
        17 -- 6
        17 -- 7
        17 -- 19
        18 -- 5
        19 -- 16
        19 -- 4
        19 -- 5
}
```

Figura 6.3. Exemplo de saída da aplicação `mbv-solver.bin` no formato `.dot`

A aplicação foi implementada de acordo com o paradigma de orientação à objetos. Segue uma breve explicação sobre cada uma das classes e arquivos envolvidos na implementação:

- `grafo.h, grafo.cpp`: implementam a classe abstrata que define as principais

interfaces que um objeto do tipo grafo devem fornecer para uso dessa estrutura de dados, tais como inserir e remover arcos, obter o número de vértices e arcos do grafo, retornar o peso de um arco, retornar o grau de um vértice e a lista de adjacência de um dado vértice informado.

- `mgrafo.h, mgrafo.cpp`: classe derivada da implementação abstrata de grafo de `grafo.h` que define um grafo através de uma matriz de adjacência. Implementa os métodos comentados usando uma matriz como estrutura de dados para armazenamento dos arcos e pesos associados.

- `union-find.h, union-find.cpp`: implementam a estrutura de dados de conjunto disjunto por meio de uma árvore, seguindo a descrição em Cormen et al. [2001].

- `mbv-io.h, mbv-io.cpp`: implementa a classe capaz de gerenciar a entrada e saída de dados da aplicação. Possui interfaces para ler grafos de entrada de arquivos nos formato `.txt` e NETGEN e transformar essa entrada em objetos grafos, assim como interfaces para transformar grafos em arquivos de saída nos formatos `.txt` e `.dot`.

- `mbv-weight.h, mbv-weight.cpp`: implementa o algoritmo de ponderação de arcos de Cerulli et al. [2009].

- `mbv-color.h, mbv-color.cpp`: implementa o algoritmo de coloração de vértices de Cerulli et al. [2009].

- `mbv-iterative.h, mbv-iterative.cpp`: implementa o algoritmo de refinamento iterativo proposto nessa dissertação para o problema da árvore geradora com número mínimo de vértices *branch*.

- `mbv-solver.cpp`: implementa a aplicação que une os três algoritmos e os executa de acordo com a sintaxe de entrada.

6.2.2 *Scripts*

Além da aplicação, foram criados diversos *scripts* para auxiliar o processamento das diversas instâncias testadas nesse trabalho. Tais *scripts* foram escritos em *shell script* (Cooper [2002]) e linguagem *R* (R Development Core Team [2006]), uma linguagem que permite fazer análises estatísticas sobre dados. Seguem uma breve descrição desses *scripts*:

- `create_benchmark.sh`: cria instâncias pelo gerador NETGEN usando valores de referência para o número de vértices da rede e a densidade de arcos especificada.

- `run_cerulli.sh`: executa todas as instâncias empregando os algoritmos EWS e NCH de Cerulli et al. [2009]. Cada instância é executada uma única vez, e seus resultados são agrupados em um arquivo de resumo que facilita o processamento por planilhas eletrônicas e aplicativos estatísticos.

- `run_ir2.sh`: executa todas as instâncias do *benchmark* usando o algoritmo de refinamento iterativo. Como a atribuição de pesos do grafo G é aleatória, cada instância é executada por 100 vezes, agrupando os resultados obtidos em cada execução em um arquivo de resumo que facilita seu processamento pelo *script* escrito em `R`.

- `run_statistics.sh, run_time.sh`: criam automaticamente *scripts* em linguagem `R` para processar os arquivos de resumo gerados pelo algoritmo IR.

- `script.r`: Obtém a estatística descritiva da população (valores mínimos e máximos, média aritmética, mediana, primeiro e terceiro quartis, desvio padrão e variância) para os tempos de execução e função objetivo. Cada população corresponde à uma instância, sendo composta por 100 observações que advém de cada execução da mesma instância. Gera histogramas de frequência que indicam visualmente a probabilidade do algoritmo IR encontrar bons resultados para cada instância.

6.2.3 Gerador `NETGEN` de Klingman, Napier e Stutz

Alguns dos conjuntos de instâncias utilizados nessa dissertação para comparação de resultados entre as diferentes heurísticas implementadas foram criados aleatoriamente com uso do gerador de problemas de fluxo em rede NETGEN (Klingman et al. [1974]), obtido do `public ftp` do DIMACS [1]. Tal gerador é distribuído sobre a forma de duas implementações, em linguagem `C` e `Fortran`. O NETGEN constrói grafos para problemas de fluxo de rede usando como entrada um arquivo que especifica o valores e limites dos parâmetros de entrada, um por linha, de acordo com o formato da Tabela 6.1.

Se redirecionada para arquivo, a saída do NETGEN consistirá em um grafo valorado com capacidade nos arcos representado de maneira textual no formato utilizado nos *DIMACS Challenges* do *Center for Discrete Mathematics and Theoretical Computer Science*. Este consiste em um arquivo formato texto onde cada uma das linhas do

[1] `ftp://dimacs.rutgers.edu/pub/netflow/`

Tabela 6.1. Parâmetros do NETGEN para os arquivos de entrada, obtidos diretamente de seu código-fonte

#	*Parâmetro*	*Descrição*
1	SEED	Random numbers seed
2	PROBLEM	Problem number
3	NODES	Number of nodes
4	SOURCES	Number of sources (including transshipment)
5	SINKS	Number of sinks (including transshipment)
6	DENSITY	Number of (requested) arcs
7	MINCOST	Minimum cost of arcs
8	MAXCOST	Maximum cost of arcs
9	SUPPLY	Total supply
10	TSOURCES	Transshipment sources
11	TSINKS	Transshipment sinks
12	HICOST	Percent of skeleton arcs given maximum cost
13	CAPACITED	Percent of arcs to be capacitated
14	MINCAP	Minimum capacity for capacitated arcs
15	MAXCAP	Maximum capacity for capacitated arcs

arquivo de saída tem um propósito, que pode ser determinado pelo caracter marcador contido no início de cada linha. Os marcadores são:

- `c`: Linha de comentário. Usada para comentar informações do arquivo de saída;
- `p`: Linha que indica o problema (se é de minimização ou maximização), assim como o número de nós e arcos contidos no arquivo de saída;
- `n`: Marcam os vértices de oferta e demanda. O primeiro número consiste no vértice, e o segundo número corresponde ao valor de oferta/demanda correspondente;
- `a`: Linha que determina um arco no grafo. Os dois primeiros números indicam os vértices que formam o arco; o terceito número indica o peso/custo do arco, e os dois últimos números indicam a capacidade mínima e máxima do arco.

As Figuras 6.4 e 6.5 a seguir trazem um exemplo de saída para um dado arquivo de entrada do NETGEN:

```
c NETGEN flow network generator (C version)
c  Problem  1 input parameters
c  --------------------------
c   Random seed:               12345
c   Number of nodes:              10
c   Source nodes:                  1
c   Sink nodes:                    1
c   Number of arcs:               30
```

Figura 6.4. Formato de saída do aplicativo `NETGEN`

```
c   Minimum arc cost:              15
c   Maximum arc cost:              25
c   Total supply:                   1
c   Transshipment -
c     Sources:                      0
c     Sinks:                        0
c   Skeleton arcs -
c     With max cost:                1%
c     Capacitated:                  1%
c   Minimum arc capacity:           1
c   Maximum arc capacity:           3
c
c  *** Minimum cost flow ***
c
p min 10 30
n 1 1
n 10 -1
a 1 2 0 1 17
a 1 3 0 1 16
a 1 8 0 1 21
a 1 4 0 1 16
a 2 9 0 1 23
a 2 10 0 1 20
a 2 3 0 1 19
    .
    .
    .
a 7 3 0 1 17
a 8 5 0 1 22
a 8 4 0 1 19
a 8 7 0 1 18
a 9 6 0 1 16
a 9 5 0 1 23
a 9 8 0 1 25
```

Figura 6.5. Formato de saída do aplicativo `NETGEN` (Continuação)

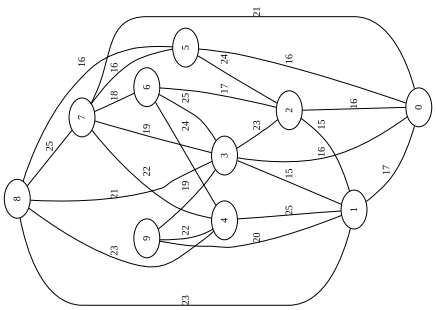

Figura 6.6. Exemplo de saída para uma instância gerada utilizando o `NETGEN`

O grafo que correspondente à descrição acima é dado na Figura 6.6.

Embora o gerador NETGEN seja capaz de criar instâncias com pesos e capacidades nos arcos assim como diferenciar os vértices que são de oferta, transbordo e demanda, para o problema MBV todas essas informações são ignoradas utilizando-se apenas a informação de topologia dos arcos da rede.

6.2.4 Gerador `crand` de Cherkassky e Goldberg

O gerador `crand` consiste em um dos geradores de instâncias usados por Boris V. Cherkassky e Andrew V. Goldberg no relatório técnico '*Negative-Cycle Detection Algorithms*' (Cherkassky e Goldberg [1996]).

É distribuido no pacote `SPC` em conjunto com diversos códigos em `C`, geradores e parâmetros de entrada para construção de instâncias para algoritmos de caminho mínimo com detecção de ciclos negativos. É disponibilizado no endereço `http://www.cs.sunysb.edu/~algorith/implement/goldberg/implement.shtml`. A sintaxe do `crand` é dada na Figura 6.7.

```
usage: ./crand  n  m  seed  [ -ll#i -lm#i -cl#i -p -pl#i -pm#i ... ]
help:  ./crand -h
```

Figura 6.7. Sintaxe do gerador de instâncias `crand`

Nos experimentos onde ele foi empregado, as instâncias foram construídas informando-se apenas os parâmetros obrigatórios n (número de vértices), m (número de arcos) e *seed* (semente do gerador de números aleatórios). A Figura 6.8 ilustra um exemplo de saída no `crand` no formato Extended DIMACS:

```
c random network for shortest paths problem
c extended DIMACS format
c
t rd_5_15_123_
c
c
p sp        5       15
c
n        1
c
a        1        2         9174
a        5        1          589
a        2        3         4037
a        3        4         7568
a        4        5         6870
a        2        5         9792
a        4        1         9366
a        2        3         8545
a        5        2         2000
a        2        1         1942
a        1        2         9113
a        2        4          805
a        3        1         4802
a        1        5         7177
a        2        4         8990
```

Figura 6.8. Saída do gerador `crand` no formato Extended DIMACS

6.3 Nomenclaturas e Convenções

Essa seção apresenta algumas nomenclaturas e convenções utilizadas no decorrer do capítulo para análise experimental dos algoritmos EWS, NCH e IR implementados para resolver o problema da árvore geradora com número mínimo de vértices *branch*.

Em primeiro lugar, os experimentos foram separados em seis diferentes conjuntos de instâncias. No início de cada sub-seção as instâncias que compõem cada conjunto são apresentadas, descrevendo o tamanho de cada grafo e densidade correspondente ou referenciando alguma instância conhecida de *benchmarks* disponíveis para problemas de pesquisa operacional. Não foi possível utilizar o conjunto de instâncias publicado por Cerulli et al. [2009] porque os autores não disponibilizaram as instâncias usadas; portanto a comparação dos resultados experimentais apresentados nesse artigo ficaram fora do escopo deste trabalho.

Em segundo lugar, cada um dos experimentos utiliza as implementações feitas dos métodos EWS, NCH e IR. O método MIX não foi implementado porque Cerulli et al. [2009] não disponibilizaram o pseudo-código do método no artigo em que as heurísticas foram apresentadas e, portanto, poderíamos estar usando uma implementação que difere da idéia original do método. Para cada execução foram registrados os resultados computados de 'número de vértices *branch*' e 'tempo de execução' de cada instância. Quanto houver alguma referência à palavra 'resultado' ou 'qualidade da solução', leia-se 'número de vértices *branch*'. Qualquer denominação diferente desta será explicitada quando for empregada.

A análise de tempo de execução também será apresentada para as instâncias presentes nos seis conjuntos testados. Existem algumas considerações a se fazer a respeito dos tempos de execução coletados nos experimentos. Em primeiro lugar, este trabalho não apresentou a análise de complexidade para o algoritmo IR proposto, de forma que para esse algoritmo os tempos coletados foram apenas registrados e analisados por meio de estatística descritiva, podendo variar de acordo com o número de iterações necessárias para refinar cada instância testada. Em segundo lugar, embora Cerulli et al. [2009] tenham apresentado ordem de complexidade de $O(mn)$ para os algoritmos EWS e NCH podemos considerar essa análise superficial sobre as operações executadas pelos Algoritmos 1 e 3, apresentados no Capítulo 3. Especificamente para o Algoritmo 1, linha 9, existe um passo de construção do conjunto L com os arcos de custo mínimo do grafo G' que é considerado ser da ordem de $O(1)$ na análise desses autores; entretanto esse passo pode exigir um custo de $O(mlogm)$ operações para ordenar os m arcos do grafo G' de acordo com seu peso, além de uma inspeção nos arcos ordenados para incluir em L aqueles que, além de possuírem o menor peso, não formem ciclos na árvore em construção. Com isso, teríamos uma ordem de complexidade sensívelmente maior do

que $O(mn)$ para o algoritmo EWS; de fato isso é mostrado ser verdade nos resultados experimentais apresentados no decorrer deste capítulo.

As heurísticas EWS e NCH retornam resultados determinísticos em relação à qualidade da solução; já o algoritmo IR pode resultar em diferentes soluções de acordo com a semente informada na sua sintaxe. Tal semente é usada no gerador de números aleatórios que determina a atribuição de pesos inicial do grafo de entrada, sendo que a saída é sensível aos pesos de entrada e qualidade da solução inicial. Para comparar os resultados obtidos pelo algoritmo IR com os resultados computados pelas heurísticas EWS e NCH, optou-se por construir uma amostra para cada instância testada, composta pelos resultados registrados de 100 execuções diferentes do algoritmo IR. A proposta é comparar os resultados médios da variável 'número de vértices *branch*' obtido nas repetições do algoritmo IR com os resultados determinísticos das heurísticas EWS e NCH, apresentando as medidas de posição e dispersão de cada amostra (estatística descritiva), assim como analisar a distribuição dos resultados por meio de histogramas de frequência.

Dessa forma, os resultados experimentais serão apresentados sob formato tabular e histograma de frequências. As tabelas serão separadas em dois blocos distintos, que referem-se aos resultados obtidos pelas heurísticas EWS e NCH (bloco I) e as medidas de posição e dispersão para os resultados obtidos pelo algoritmo IR (bloco II). Dentro do bloco II existe ainda separação das medidas que referem-se à variável 'número de vértices *branch*' (bloco `FUNÇÃO OBJETIVO`) e à variável 'tempo de execução' (bloco `RUNTIME`). Cada linha de resultados destacada em negrito ou que cuja coluna entitulada 'Melhor' esteja marcada com um caracter 'X' corresponde a um caso do *benchmark* onde o resultado médio do algoritmo IR foi melhor do que o melhor dentre os resultados obtidos pelas heurísticas EWS e NCH para essa mesma instância.

Na tabela, as colunas 'n', 'm' e 'd' correspondem ao número de vértices, de arcos e densidade dos grafos que cada instância representa. As colunas 'Val' e 'Tempo (s)' correspondem aos resultados obtidos de valor da função objetivo e 'tempo de execução' para as heurísticas EWS e NCH. As colunas 'Min', '1Q' (1^o quartil), 'Média', 'Mediana', '3Q' (3^o quartil), 'Max', 'Dev' e 'Var' dos blocos `FUNÇÃO OBJETIVO` e `RUNTIME` referem-se às medidas estatísticas das variáveis 'número de vértices *branch*' e 'tempo de execução' obtidos nas amostras contendo 100 observações advindas das execuções independentes do algoritmo IR.

Na legenda de cada histograma são apresentadas as grandezas IR_{min}, $IR_{\bar{x}}$, IR_{max}, IR_{med}, IR_{mod}, EWS_{val} e NCH_{val}, e referem-se aos valores amostrais mínimos, médios, máximos, mediana e moda para o algoritmo IR, e os resultados determinísticos calculados pelas heurísticas EWS e NCH.

Em terceiro lugar, os histogramas de frequência foram utilizados em alguns casos para estimar, com base nos resultados das 100 observações contidas em cada amostra, qual foi a probabilidade do algoritmo IR atingir um resultado compreendido entre $IR_{min} \leq IR_{val} \leq argmin\{EWS_{val}, NCH_{val}\}$, ou seja, de retornar resultados melhores ou iguais ao melhor resultado computado pelos algoritmos EWS e NCH.

6.4 Resultados Experimentais

Essa seção descreve os resultados experimentais obtidos com a execução dos algoritmos EWS, NCH e IR sobre alguns conjuntos de instâncias de diferentes *benchmarks*. Cada conjunto é descrito e os resultados experimentais são comentados.

6.4.1 Conjunto de Instâncias I (Klingman (1974))

6.4.1.1 *Benchmark*

Klingman et al. [1974] distribuíram junto com a implementação em `C` e `Fortran` do NETGEN um arquivo de entrada com os parâmetros para construir 40 problemas utilizando o gerador. Desses 40, foram tomados os dez primeiros para formar o Conjunto de Instâncias I. Para as primeiras dez instâncias apenas existe variação no número de vértices e de arcos do grafo de saída, sendo que os demais parâmetros são fixos e admitem os seguintes valores: `Random Seed:` 13502460, `Minimum Cost:` 1, `Maximum Cost:` 10000, `Total Supply:` 100000, `Trans. Source:` 0, `Trans. Sink:` 0, `Skel. Max.:` 0, `Skel. Capacity:` 0, `Min. Capacity:` 0, `Max. Capacity:` 0. Os demais parâmetros variáveis são apresentados a seguir, na Tabela 6.2.

Tabela 6.2. Parâmetros do NETGEN para as instâncias do Conjunto I

	p-1	**p-2**	**p-3**	**p-4**	**p-5**	**p-6**	**p-7**	**p-8**	**p-9**	**p-10**
# Nodes	200	200	200	200	200	300	300	300	300	300
# Source Nodes	100	100	100	100	100	150	150	150	150	150
# Sink Nodes	100	100	100	100	100	150	150	150	150	150
# Arcs	1300	1500	2000	2200	2900	3150	4500	5155	6075	6300
Density (%)	**7**	**8**	**10**	**11**	**15**	**7**	**10**	**11**	**14**	**14**

Nessa tabela, a linha 'Density (%)' corresponde à densidade de arcos das instâncias considerando o número de arcos que estas possuem em relação ao número de arcos do grafo completo correspondente, em porcentagem. Sejam n o número de vértices de G, m o número de arcos de G. A densidade é dada por:

$$Densidade(G) = \frac{m}{[\frac{n(n-1)}{2}]} \tag{6.1}$$

6.4.1.2 Análise dos Resultados

Os resultados experimentais obtidos com uso das instâncias detalhadas no Conjunto de Instâncias I são apresentados na Tabela 6.3. Nota-se que o algoritmo IR comportou-se bem nas instâncias do Conjunto I. Em aproximadamente 80% dos casos testados, o algoritmo IR apresentou resultado médio com qualidade superior aos resultados obtidos pelas heurísticas EWS e NCH.

Apenas em dois casos do *benchmark* (especificamente, nas instâncias `p-1` e `p-10`) o algoritmo IR não obteve um resultado médio melhor do que as heurísticas EWS e NCH; entretando, mesmo para esses casos os histogramas contidos nas Figuras 6.9 - 6.18 mostram que boa parte dos resultados são menores ou iguais aos resultados dos métodos EWS e NCH.

Em relação ao tempo de execução, o algoritmo IR consumiu menos tempo de execução que os algoritmos EWS e NCH em todas as 100 repetições amostradas de cada instância (ver coluna 'Max' do bloco `RUNTIME`). Para os grafos de entrada utilizados nesse conjunto não houveram diferenças significativas na ordem de grandeza para os tempos de execução dos algoritmos EWS e NCH.

Tabela 6.3. Comparação dos 'tempos de execução' e 'função objetivo' dos métodos EWS, NCH e IR para o Conjunto I

Benchmark				**Heurísticas Cerulli et al**				**Algoritmo Refinamento Iterativo**										
				EWS		**NCH**		**FUNÇÃO OBJETIVO**									**RUNTIME**	
Instância	*n*	*m*	*d(%)*	*Val*	*Tempo (s)*	*Val*	*Tempo (s)*	*Min*	*1Q*	*Média*	*Mediana*	*3Q*	*Max*	*Dev*	*Var*	*Melhor*	*Min*	*Méd*	*Max*
p-1	200	1300	7	7	1,13	5	1,12	4	6	7	7	8	12	1,69	2,85		0,24	0,46	0,7
p-2	**200**	**1500**	**8**	**7**	**1,29**	**7**	**1,3**	**2**	**5**	**5,87**	**6**	**7**	**11**	**1,9**	**3,61**	**X**	**0,24**	**0,5**	**0,78**
p-3	**200**	**2000**	**10**	**5**	**1,88**	**5**	**1,58**	**2**	**4**	**4,83**	**5**	**6**	**8**	**1,47**	**2,16**	**X**	**0,24**	**0,57**	**0,92**
p-4	**200**	**2200**	**11**	**7**	**2,19**	**5**	**1,94**	**1**	**3**	**4,23**	**4**	**5**	**8**	**1,61**	**2,6**	**X**	**0,22**	**0,54**	**1,11**
p-5	**200**	**2900**	**15**	**6**	**2,95**	**5**	**2,57**	**1**	**3**	**3,79**	**4**	**4**	**8**	**1,44**	**2,07**	**X**	**0,34**	**0,69**	**1,11**
p-6	**300**	**3150**	**7**	**8**	**4,51**	**6**	**4,17**	**1**	**5**	**5,61**	**6**	**7**	**9**	**1,69**	**2,85**	**X**	**0,68**	**1,58**	**2,71**
p-7	**300**	**4500**	**10**	**5**	**7,28**	**6**	**5,96**	**2**	**3**	**4,54**	**4**	**5**	**10**	**1,65**	**2,72**	**X**	**0,97**	**2,12**	**3,48**
p-8	**300**	**5155**	**11**	**7**	**8,61**	**6**	**6,84**	**1**	**2**	**3,49**	**3**	**4**	**7**	**1,5**	**2,25**	**X**	**0,82**	**2,04**	**4,4**
p-9	**300**	**6075**	**14**	**4**	**10,93**	**3**	**7,96**	**0**	**2**	**2,97**	**3**	**4**	**7**	**1,46**	**2,13**	**X**	**0,68**	**2,05**	**3,76**
p-10	300	6300	14	3	11,59	4	8,56	0	3	3,67	4	4,25	7	1,21	1,46		1,44	3,51	6,78

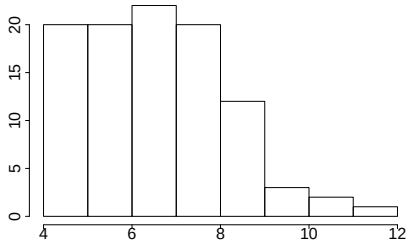

Figura 6.9. Histograma da instância `p-1` (Conjunto I). $IR_{min} = 4$, $IR_{\bar{x}} = 7$, $IR_{max} = 12$, $IR_{med} = 7$, $IR_{mod} = 6$; $EWS_{val} = 7$, $NCH_{val} = 5$

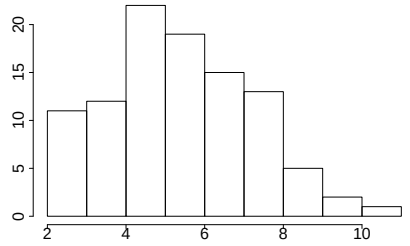

Figura 6.10. Histograma da instância `p-2` (Conjunto I). $IR_{min} = 2$, $IR_{\bar{x}} = 8.87$, $IR_{max} = 11$, $IR_{med} = 6$, $IR_{mod} = 4$; $EWS_{val} = 7$, $NCH_{val} = 7$

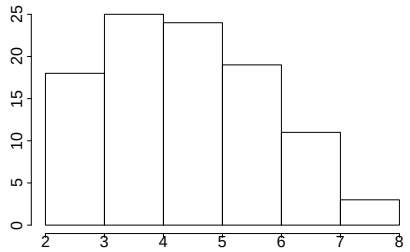

Figura 6.11. Histograma da instância `p-3` (Conjunto I). $IR_{min} = 2$, $IR_{\bar{x}} = 4.83$, $IR_{max} = 8$, $IR_{med} = 5$, $IR_{mod} = 3$; $EWS_{val} = 5$, $NCH_{val} = 5$

Em todos os casos testados, o algoritmo IR conseguiu encontrar soluções de melhor qualidade computadas pelo método IR do que pelos métodos EWS e NCH, na seguinte dimensão (aproximada): 20 casos para a instância `p-1`, 75 casos para a instância `p-2`, 70 casos para a instância `p-3`, 50 casos para a instância `p-4`, 75 casos para a instância `p-5`, 65 casos para a instância `p-6`, 75 casos para a instância `p-7`, 85 casos para a instância `p-8`, 65 casos para a instância `p-9` e 45 casos para a instância `p-10`.

Pelos histogramas, temos que a probabilidade estimada para o algoritmo IR atingir um resultado melhor ou igual aos métodos EWS e NCH foram, para cada instância, da ordem de 40% para a instância `p-1`, 90% para a instância `p-2`, 84% para a instância `p-3`, 74% para a instância `p-4`, 91% para a instância `p-5`, 87% para a instância `p-6`, 87% para a instância `p-7`, 95% para a instância `p-8`, 80% para a instância `p-9`, e, finalmente, 77% para a instância `p-10`.

Acredita-se que a vantagem mostrada pelos resultados do algoritmo IR sobre os algoritmos EWS e NCH ocorra porque os grafos de entrada são densos e oferecem

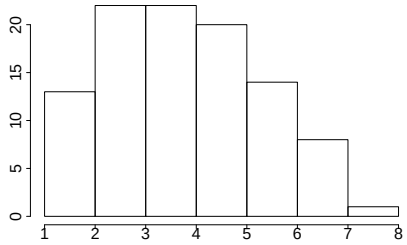

Figura 6.12. Histograma da instância `p-4` (Conjunto I). $IR_{min} = 1$, $IR_{\bar{x}} = 4.23$, $IR_{max} = 8$, $IR_{med} = 4$, $IR_{mod} = 2 - 3$; $EWS_{val} = 7$, $NCH_{val} = 5$

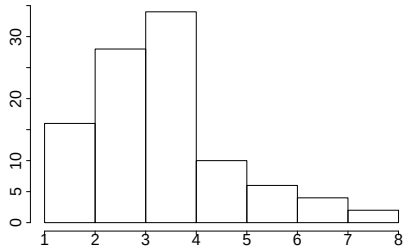

Figura 6.13. Histograma da instância `p-5` (Conjunto I). $IR_{min} = 1$, $IR_{\bar{x}} = 3.79$, $IR_{max} = 8$, $IR_{med} = 4$, $IR_{mod} = 3$; $EWS_{val} = 6$, $NCH_{val} = 5$

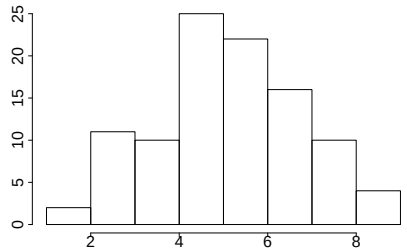

Figura 6.14. Histograma da instância `p-6` (Conjunto I). $IR_{min} = 1$, $IR_{\bar{x}} = 5.61$, $IR_{max} = 9$, $IR_{med} = 6$, $IR_{mod} = 5$; $EWS_{val} = 8$, $NCH_{val} = 6$

diversas oportunidades de substituição de arcos sem criar penalidades na árvore, uma vez que a razão do número de arcos pelo número de nós dos grafos é da ordem de, no mínimo, 6×. Assim, o refinamento consegue melhorar iterativamente as soluções sem prejudicar a árvore final, atuando melhor do que as heurísticas construtivas que decidem quais arcos participam da topologia final em uma única passada de maneira gulosa.

6.4.2 Conjunto de Instâncias II (NETGEN)

6.4.2.1 *Benchmark*

O Conjunto de Instâncias II foi composto por instâncias construídas no NETGEN com o objetivo de experimentar a execução dos algoritmos em diferentes cenários, cada qual contendo grafos com diferentes número de vértices e de arcos. Arcos repetidos nas

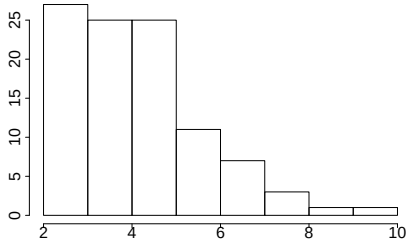

Figura 6.15. Histograma da instância `p-7` (Conjunto I). $IR_{min} = 2$, $IR_{\bar{x}} = 4.54$, $IR_{max} = 10$, $IR_{med} = 4$, $IR_{mod} = 2$; $EWS_{val} = 5$, $NCH_{val} = 6$

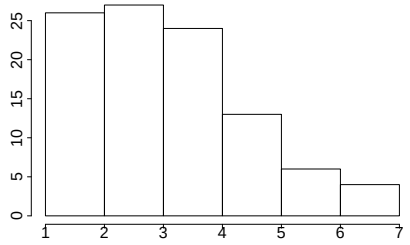

Figura 6.16. Histograma da instância `p-8` (Conjunto I). $IR_{min} = 1$, $IR_{\bar{x}} = 3.49$, $IR_{max} = 7$, $IR_{med} = 3$, $IR_{mod} = 2$; $EWS_{val} = 7$, $NCH_{val} = 6$

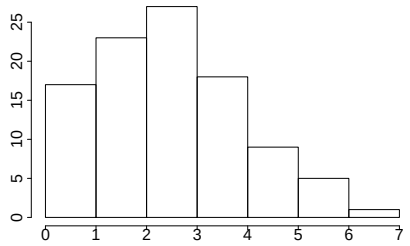

Figura 6.17. Histograma da instância `p-9` (Conjunto I). $IR_{min} = 0$, $IR_{\bar{x}} = 2.97$, $IR_{max} = 7$, $IR_{med} = 3$, $IR_{mod} = 2$; $EWS_{val} = 4$, $NCH_{val} = 3$

instâncias foram ignorados. Os arquivos de entrada usados para gerar as instâncias no NETGEN seguem o formato dado pela Tabela 6.4, onde alguns desses parâmetros foram fixados.

Pela tabela, os parâmetros que puderam variar correspondem à semente do gerador de números aleatórios usado pelo NETGEN, o número de vértices e de arcos do grafo de saída da instância correspondente. Na Tabela 6.5 esses valores são apresentados para cada instância na forma das colunas d (número de arcos), n (número de vertices) e s (semente do gerador). A coluna m apresenta a quantidade 'real' de arcos do grafo, descontando-se os arcos repetidos. Para esse conjunto foram geradas instâncias com 30, 50, 100, 150, 300 e 500 vértices, com densidade de arcos de 15% e 30%. Cada cenário testado foi composto por cinco grafos diferentes, porém compartilhando das mesmas características topológicas, ou seja, mesmo valor de n e d.

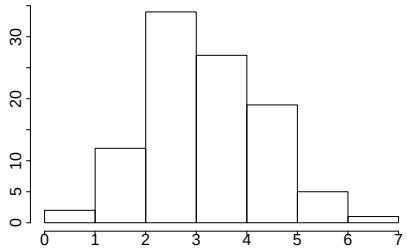

Figura 6.18. Histograma da instância `p-10` (Conjunto I). $IR_{min} = 0$, $IR_{\bar{x}} = 3.67$, $IR_{max} = 7$, $IR_{med} = 4$, $IR_{mod} = 2$; $EWS_{val} = 3$, $NCH_{val} = 4$

Tabela 6.4. Parâmetros do NETGEN para as instâncias do Conjunto II

#	*Parâmetro*	*Valor Usado*	*Descrição*
1	SEED	*Variável*	Random numbers seed
2	PROBLEM	1	Problem number
3	NODES	*Variável*	Number of nodes
4	SOURCES	1	Number of sources (including transshipment)
5	SINKS	1	Number of sinks (including transshipment)
6	DENSITY	*Variável*	Number of (requested) arcs
7	MINCOST	0	Minimum cost of arcs
8	MAXCOST	1000	Maximum cost of arcs
9	SUPPLY	1	Total supply
10	TSOURCES	0	Transshipment sources
11	TSINKS	0	Transshipment sinks
12	HICOST	1	Percent of skeleton arcs given maximum cost
13	CAPACITED	1	Percent of arcs to be capacitated
14	MINCAP	0	Minimum capacity for capacitated arcs
15	MAXCAP	3	Maximum capacity for capacitated arcs

6.4.2.2 Análise dos Resultados

Os resultados desse experimento estão documentados nas Tabelas 6.5 e 6.6. Pelas tabelas, o algoritmo IR atingiu resultados médios de melhor qualidade que os algoritmo EWS e NCH em 43 das 55 instâncias analisadas (78% dos casos). O valor da mediana de cada amostra também sugere que o algoritmo IR aplicou-se bem ao Conjunto de Instâncias II. Das 55 instâncias avaliadas, o algoritmo IR conseguiu valores de mediana melhores do que os resultados encontrados para as heurísticas EWS e NCH em 37 instâncias, e valores iguais aos resultados encontrados por essas mesmas heurísticas em 15 instâncias.

Mesmo para as instâncias em que o IR não atingiu um resultado médio de melhor qualidade do que as heurísticas EWS e NCH, verifica-se uma tendência de concentrar grande parte dos resultados em valores próximos aos encontrados por esses dois métodos. As Figuras 6.19, 6.20, 6.21, 6.22, 6.23, e 6.24 apresentam o histograma de frequências da amostra dos resultados de seis das doze instâncias do Conjunto II cujo número médio de vértices *branch* retornado pelo algoritmo IR não superou o resultados calculado pelas heurísticas EWS e NCH.

Tabela 6.5. Comparação dos 'tempos de execução' e 'função objetivo' dos métodos EWS, NCH e IR para o Conjunto II

Benchmark				Heurísticas Cerulli et al				Algoritmo Refinamento Iterativo											
				EWS		NCH		FUNÇÃO OBJETIVO							RUNTIME				
d	*n*	*m*	*s*	*Val*	*Tempo(s)*	*Val*	*Tempo(s)*	*Min*	*Média*	*Mediana*	*Max*	*Dev*	*Var*	*Melhor*	*Min*	*Média*	*Max*	*Dev*	*Var*
68	**30**	**67**	**1596**	**2**	**0,008**	**2**	**0,008**	**0**	**0,85**	**1**	**3**	**0,7**	**0,49**	**X**	**0,000**	**0,002**	**0,012**	**0,003**	**0,000**
68	**30**	**67**	**2429**	**2**	**0,008**	**2**	**0,012**	**0**	**0,68**	**1**	**3**	**0,71**	**0,5**	**X**	**0,000**	**0,002**	**0,008**	**0,002**	**0,000**
68	**30**	**66**	**7081**	**2**	**0,012**	**2**	**0,008**	**0**	**1,11**	**1**	**3**	**0,9**	**0,81**	**X**	**0,000**	**0,003**	**0,008**	**0,003**	**0,000**
68	30	66	7236	1	0,008	1	0,012	0	1,37	1	3	0,82	0,68		0,000	0,003	0,008	0,003	0,000
68	30	66	7880	1	0,008	1	0,012	0	1,37	1	3	0,77	0,6		0,000	0,002	0,008	0,002	0,000
135	**30**	**124**	**1172**	**1**	**0,008**	**1**	**0,016**	**0**	**0,84**	**1**	**2**	**0,65**	**0,42**	**X**	**0,000**	**0,004**	**0,016**	**0,003**	**0,000**
135	30	122	2488	0	0,016	0	0,020	0	0,4	0	2	0,55	0,3		0,000	0,004	0,012	0,003	0,000
135	**30**	**122**	**4970**	**1**	**0,016**	**1**	**0,016**	**0**	**0,45**	**0**	**2**	**0,54**	**0,29**	**X**	**0,000**	**0,004**	**0,012**	**0,003**	**0,000**
135	30	128	5081	0	0,016	0	0,016	0	0,24	0	2	0,47	0,22		0,000	0,003	0,012	0,003	0,000
135	**30**	**125**	**8788**	**1**	**0,016**	**1**	**0,016**	**0**	**0,28**	**0**	**1**	**0,45**	**0,2**	**X**	**0,000**	**0,004**	**0,016**	**0,003**	**0,000**
188	**50**	**182**	**1054**	**2**	**0,040**	**2**	**0,048**	**0**	**1,55**	**1**	**5**	**1,08**	**1,16**	**X**	**0,000**	**0,008**	**0,020**	**0,004**	**0,000**
188	**50**	**179**	**3335**	**2**	**0,040**	**2**	**0,040**	**0**	**1,16**	**1**	**4**	**0,73**	**0,54**	**X**	**0,000**	**0,009**	**0,024**	**0,004**	**0,000**
188	**50**	**180**	**4663**	**2**	**0,036**	**3**	**0,036**	**0**	**1,12**	**1**	**4**	**0,79**	**0,63**	**X**	**0,000**	**0,008**	**0,024**	**0,005**	**0,000**
188	**50**	**182**	**4985**	**2**	**0,040**	**2**	**0,040**	**0**	**1,5**	**1**	**4**	**0,92**	**0,84**	**X**	**0,000**	**0,008**	**0,020**	**0,004**	**0,000**
188	**50**	**186**	**7085**	**4**	**0,040**	**4**	**0,044**	**0**	**1,39**	**1**	**3**	**0,84**	**0,7**	**X**	**0,000**	**0,008**	**0,016**	**0,004**	**0,000**
375	50	341	1720	0	0,080	0	0,072	0	0,56	0	2	0,69	0,47		0,004	0,012	0,024	0,005	0,000
375	**50**	**345**	**6752**	**2**	**0,084**	**2**	**0,048**	**0**	**0,36**	**0**	**3**	**0,58**	**0,33**	**X**	**0,004**	**0,013**	**0,024**	**0,005**	**0,000**
375	**50**	**349**	**7009**	**2**	**0,052**	**2**	**0,076**	**0**	**0,42**	**0**	**2**	**0,59**	**0,35**	**X**	**0,004**	**0,012**	**0,020**	**0,004**	**0,000**
375	**50**	**343**	**7030**	**1**	**0,072**	**1**	**0,076**	**0**	**0,32**	**0**	**2**	**0,51**	**0,26**	**X**	**0,000**	**0,012**	**0,020**	**0,004**	**0,000**
375	50	344	9979	0	0,076	0	0,072	0	0,4	0	2	0,62	0,38		0,004	0,012	0,020	0,004	0,000
750	**100**	**723**	**2312**	**3**	**0,276**	**3**	**0,284**	**0**	**1,28**	**1**	**3**	**0,94**	**0,89**	**X**	**0,024**	**0,046**	**0,080**	**0,011**	**0,000**
750	**100**	**730**	**299**	**3**	**0,268**	**3**	**0,256**	**0**	**1,09**	**1**	**4**	**0,95**	**0,91**	**X**	**0,028**	**0,046**	**0,072**	**0,010**	**0,000**
750	**100**	**722**	**4414**	**2**	**0,236**	**2**	**0,316**	**0**	**1,41**	**1**	**4**	**0,98**	**0,95**	**X**	**0,024**	**0,044**	**0,068**	**0,010**	**0,000**
750	100	724	5885	1	0,212	1	0,292	0	1,5	1	4	0,99	0,98		0,024	0,046	0,084	0,010	0,000
750	**100**	**719**	**6570**	**3**	**0,296**	**3**	**0,228**	**0**	**1,69**	**2**	**5**	**1,12**	**1,25**	**X**	**0,028**	**0,046**	**0,084**	**0,011**	**0,000**
1500	**100**	**1399**	**5309**	**1**	**0,792**	**1**	**0,384**	**0**	**0,55**	**0**	**2**	**0,66**	**0,43**	**X**	**0,040**	**0,082**	**0,128**	**0,017**	**0,000**
1500	**100**	**1383**	**6105**	**1**	**0,764**	**1**	**0,416**	**0**	**0,43**	**0**	**2**	**0,59**	**0,35**	**X**	**0,040**	**0,076**	**0,128**	**0,015**	**0,000**
1500	**100**	**1386**	**6259**	**1**	**0,772**	**1**	**0,464**	**0**	**0,4**	**0**	**2**	**0,57**	**0,32**	**X**	**0,040**	**0,077**	**0,112**	**0,015**	**0,000**
1500	**100**	**1389**	**7695**	**1**	**0,628**	**1**	**0,480**	**0**	**0,34**	**0**	**2**	**0,54**	**0,29**	**X**	**0,036**	**0,074**	**0,112**	**0,015**	**0,000**
1500	100	1391	9414	0	0,656	0	0,612	0	0,66	1	3	0,71	0,51		0,056	0,083	0,132	0,017	0,000
1688	150	1624	199	3	1,200	2	0,996	0	2,06	2	6	1,25	1,55		0,092	0,146	0,208	0,024	0,001
1688	150	1619	3738	1	1,112	1	1,060	0	1,69	2	4	1,05	1,1		0,096	0,140	0,264	0,024	0,001
1688	**150**	**1624**	**5011**	**4**	**1,196**	**3**	**1,028**	**0**	**1,52**	**1,5**	**4**	**1,03**	**1,06**	**X**	**0,072**	**0,135**	**0,200**	**0,024**	**0,001**
1688	**150**	**1627**	**7390**	**2**	**1,084**	**2**	**1,032**	**0**	**1,62**	**1,5**	**5**	**1,1**	**1,21**	**X**	**0,068**	**0,146**	**0,244**	**0,035**	**0,001**
1688	**150**	**1624**	**878**	**3**	**0,988**	**2**	**1,048**	**0**	**1,82**	**2**	**5**	**1,11**	**1,24**	**X**	**0,076**	**0,147**	**0,272**	**0,040**	**0,002**
3375	**150**	**3120**	**2051**	**1**	**2,808**	**1**	**1,800**	**0**	**0,46**	**0**	**2**	**0,58**	**0,33**	**X**	**0,200**	**0,300**	**0,424**	**0,062**	**0,004**
3375	**150**	**3120**	**2833**	**1**	**2,756**	**1**	**1,704**	**0**	**0,5**	**0**	**2**	**0,58**	**0,33**	**X**	**0,204**	**0,303**	**0,432**	**0,062**	**0,004**
3375	**150**	**3141**	**3064**	**1**	**3,196**	**1**	**1,984**	**0**	**0,58**	**0**	**3**	**0,68**	**0,47**	**X**	**0,208**	**0,330**	**0,436**	**0,062**	**0,004**
3375	**150**	**3116**	**5357**	**1**	**2,648**	**1**	**1,564**	**0**	**0,29**	**0**	**2**	**0,48**	**0,23**	**X**	**0,192**	**0,292**	**0,416**	**0,054**	**0,003**
3375	**150**	**3117**	**5687**	**2**	**2,900**	**2**	**1,816**	**0**	**0,34**	**0**	**2**	**0,54**	**0,29**	**X**	**0,164**	**0,292**	**0,428**	**0,058**	**0,003**

Tabela 6.6. Comparação dos 'tempos de execução' e 'função objetivo' dos métodos EWS, NCH e IR para o Conjunto II (Continuação)

Benchmark				**Heurísticas Cerulli et al**				**Algoritmo Refinamento Iterativo**											
				EWS		**NCH**		**FUNÇÃO OBJETIVO**							**RUNTIME**				
d	*n*	*m*	*s*	*Val*	*Tempo(s)*	*Val*	*Tempo(s)*	*Min*	*Média*	*Mediana*	*Max*	*Dev*	*Var*	*Melhor*	*Min*	*Média*	*Max*	*Dev*	*Var*
6750	300	6502	1545	1	13,377	1	8,769	0	1,62	2	4	1,07	1,15		0,724	1,042	1,332	0,116	0,013
6750	**300**	**6471**	**365**	**3**	**13,429**	**3**	**8,869**	**0**	**1,81**	**2**	**5**	**1,01**	**1,02**	**X**	**0,884**	**1,054**	**1,444**	**0,108**	**0,012**
6750	**300**	**6481**	**4071**	**5**	**13,377**	**3**	**8,545**	**0**	**1,61**	**1,5**	**5**	**1,13**	**1,27**	**X**	**0,852**	**1,071**	**1,432**	**0,116**	**0,013**
6750	300	6513	4889	1	13,277	1	8,761	0	1,27	1	4	0,86	0,74		0,852	1,034	1,324	0,114	0,013
6750	**300**	**6505**	**681**	**4**	**13,249**	**4**	**8,837**	**0**	**1,88**	**2**	**5**	**0,99**	**0,98**	**X**	**0,868**	**1,056**	**1,444**	**0,116**	**0,013**
13500	**300**	**12539**	**1358**	**2**	**37,506**	**2**	**16,661**	**0**	**0,54**	**0**	**3**	**0,64**	**0,41**	**X**	**2,232**	**3,004**	**3,668**	**0,349**	**0,122**
13500	**300**	**12508**	**2067**	**3**	**37,478**	**2**	**17,257**	**0**	**0,34**	**0**	**3**	**0,55**	**0,31**	**X**	**2,460**	**3,074**	**3,748**	**0,263**	**0,069**
13500	**300**	**12447**	**4372**	**1**	**36,126**	**1**	**17,201**	**0**	**0,4**	**0**	**2**	**0,6**	**0,36**	**X**	**2,464**	**3,250**	**3,808**	**0,304**	**0,092**
13500	**300**	**12480**	**960**	**1**	**37,630**	**1**	**17,073**	**0**	**0,65**	**1**	**3**	**0,67**	**0,45**	**X**	**2,372**	**2,996**	**3,716**	**0,333**	**0,111**
13500	**300**	**12474**	**9886**	**1**	**36,994**	**1**	**16,401**	**0**	**0,49**	**0**	**3**	**0,69**	**0,47**	**X**	**1,740**	**2,939**	**3,772**	**0,453**	**0,206**
18750	**500**	**18034**	**1456**	**2**	**82,825**	**2**	**42,139**	**0**	**1,85**	**2**	**4**	**1,12**	**1,26**	**X**	**4,924**	**5,665**	**8,073**	**0,716**	**0,513**
18750	**500**	**18055**	**1653**	**3**	**82,913**	**3**	**42,415**	**0**	**1,4**	**1**	**4**	**1,03**	**1,07**	**X**	**4,860**	**6,188**	**8,129**	**0,853**	**0,727**
18750	**500**	**18009**	**4444**	**2**	**82,533**	**2**	**41,947**	**0**	**1,74**	**2**	**5**	**1,05**	**1,1**	**X**	**4,832**	**6,678**	**8,161**	**0,924**	**0,853**
18750	**500**	**18048**	**6849**	**2**	**82,925**	**2**	**42,275**	**0**	**1,81**	**2**	**5**	**1,06**	**1,13**	**X**	**4,912**	**6,833**	**8,181**	**0,913**	**0,833**
18750	**500**	**18037**	**8824**	**4**	**82,985**	**3**	**42,379**	**0**	**1,59**	**2**	**4**	**0,99**	**0,97**	**X**	**4,776**	**6,596**	**7,945**	**0,893**	**0,797**

Observe que a moda da distribuição para essas instâncias situa-se próxima aos resultados registrados pelas heurísticas de Cerulli et al. [2009] nesse experimento. Mesmo não se sobressaindo em termos de resultados médios para 12 das 55 instâncias, ainda assim o algoritmo IR foi capaz de obter, na maioria desses casos, soluções com número de vértices *branch* iguais ou até menores do que os encontrados pelas heurísticas EWS e NCH. De fato, os histogramas referentes à essas instâncias mostram que 55% dos resultados para a instância $n = 30$, $m = 68$, $s = 7236$, aproximadamente 83% dos resultados da instância $n = 100$, $d = 750$, $s = 5885$, aproximadamente 80% dos resultados da instância $n = 150$, $d = 1688$, $s = 3738$ e aproximadamente 88% dos resultados da instância $n = 300$, $d = 6750$, $s = 4889$ estão compreendidos no intervalo $IR_{min} \leq IR_{val} \leq \arg\, min\{EWS_{val}, NCH_{val}\}$, ou seja, alguns casos apresentam resultados até melhores do que os computados pelas heurísticas de Cerulli et al. [2009].

Considerando-se o cenário como um todo, em todos os casos ao menos três das cinco instâncias que compõem cada cenário foram melhores resolvidas pelo algoritmo IR do que pelas heurísticas de Cerulli et al. [2009] considerando a comparação do número médio de vértices *branch* resultante do IR com o resultado determinístico dos algoritmos EWS e NCH (ver Tabelas 6.5 e 6.6, coluna 'Melhor'). Isso mostra que para o *benchmark* composto pelos Conjunto de Instâncias II o algoritmo IR também se comportou bem, considerando-se o caso médio.

Cabe citar que, para cada uma das 55 instâncias que compõem o Conjunto II, ocorreu ao menos um caso dentre as 100 execuções realizadas em que o método de refinamento iterativo resultou como solução uma árvore geradora sem vértices *branch*, ou seja, um caminho hamiltoniano. As Figuras 6.25 e 6.26 ilustram a diferença entre a topologia de algumas das melhores soluções encontradas pelo método IR com a topologia da solução correspondente retornada pelo método NCH. Os vértices em destaque correspondem à vértices *branch*, isto é, $\delta(v) \geq 3$.

O Conjunto II é formado por grafos relativamente densos; isso implica novamente em uma disponibilidade de arcos para substituição que permite ao algoritmo IR realizar trocas sem prejudicar a árvore final.

Em relação ao tempo consumido para computar as soluções pelos três métodos testados no Conjunto de Instâncias II novamente observamos que o algoritmo IR conseguiu encontrar soluções em menos tempo que os algoritmos EWS e NCH. Considerando o *tradeoff* entre os resultados obtidos para esse conjunto em 'número de vértices *branch*' e o 'tempo de execução' temos que o algoritmo IR foi o que melhor se ajustou às instâncias do Conjunto II. Observa-se também que existe uma diferença na ordem de grandeza dos tempos de execução entre as heurísticas EWS e NCH para os grafos do conjunto com $m \geq 5000$; tais resultados vão de encontro com a consideração de que a

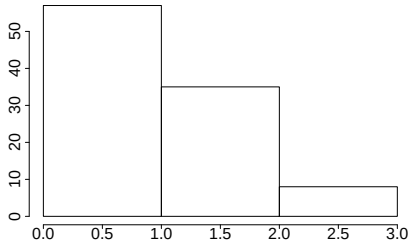

Figura 6.19. Histograma da instância $n = 30$, $m = 68$, $s = 7236$ (Conjunto II): $IR_{min} = 0$, $IR_{\bar{x}} = 1.37$, $IR_{max} = 3$, $IR_{med} = 1$, $IR_{mod} = 1$; $EWS_{val} = 1$, $NCH_{val} = 1$

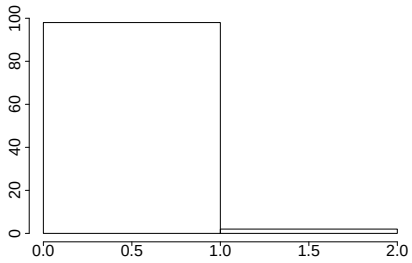

Figura 6.20. Histograma da instância $n = 30$, $m = 135$, $s = 5081$ (Conjunto II): $IR_{min} = 0$, $IR_{\bar{x}} = 0.24$, $IR_{max} = 2$, $IR_{med} = 0$, $IR_{mod} = 0$; $EWS_{val} = 0$, $NCH_{val} = 0$

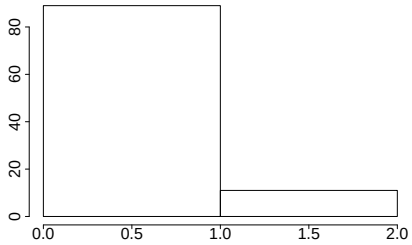

Figura 6.21. Histograma da instância $n = 50$, $m = 375$, $s = 1720$ (Conjunto II): $IR_{min} = 0$, $IR_{\bar{x}} = 0.56$, $IR_{max} = 2$, $IR_{med} = 0$, $IR_{mod} = 0$; $EWS_{val} = 0$, $NCH_{val} = 0$

ordem de complexidade dos métodos EWS e NCH são diferentes (ver Seção 6.3) já que o provável fator que diferencia a ordem de complexidade desses dois métodos é uma quantidade que varia em função de m, tornando-se mais evidente a medida que o valor de m aumenta (coluna 'Tempo(s)' das heurísticas EWS e NCH nas Tabelas 6.5 e 6.6).

6.4.3 Conjunto de Instâncias III (TSPLIB)

6.4.3.1 *Benchmark*

O TSPLIB consiste em um conjunto de instâncias publicado por Gerhard Reinelt em 1991 com o propósito de permitir que diversos grupos de pesquisa pudessem comparar

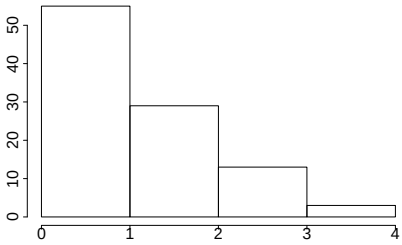

Figura 6.22. Histograma da instância $n = 100$, $m = 750$, $s = 5885$ (Conjunto II): $IR_{min} = 0$, $IR_{\bar{x}} = 1.5$, $IR_{max} = 4$, $IR_{med} = 1$, $IR_{mod} = 0$; $EWS_{val} = 1$, $NCH_{val} = 1$

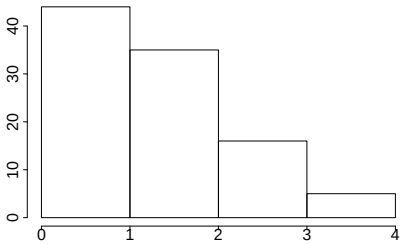

Figura 6.23. Histograma da instância $n = 150$, $m = 1688$, $s = 3738$ (Conjunto II): $IR_{min} = 0$, $IR_{\bar{x}} = 1.69$, $IR_{max} = 4$, $IR_{med} = 2$, $IR_{mod} = 0$; $EWS_{val} = 1$, $NCH_{val} = 1$

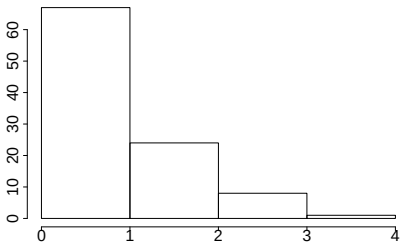

Figura 6.24. Histograma da instância $n = 300$, $m = 6750$, $s = 4889$ (Conjunto II): $IR_{min} = 0$, $IR_{\bar{x}} = 1.27$, $IR_{max} = 4$, $IR_{med} = 1$, $IR_{mod} = 0$; $EWS_{val} = 1$, $NCH_{val} = 1$

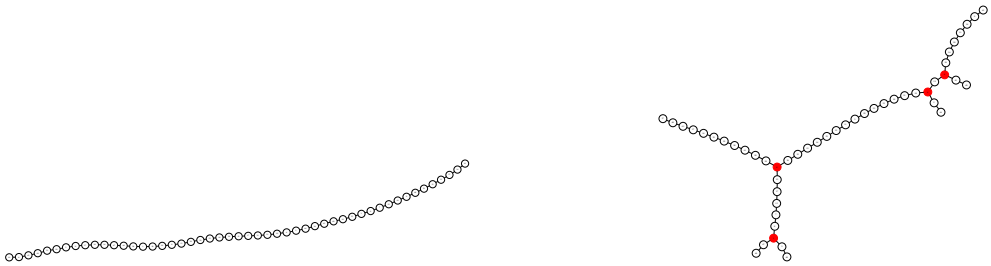

Figura 6.25. Soluções dos métodos IR (esquerda, 0 *branch*) e NCH (direita, quatro vértices *branch*) para a instância $n = 50$, $m = 186$, $s = 7085$

Figura 6.26. Soluções dos métodos IR (esquerda, 0 *branch*) e NCH (direita, três vértices *branch*) para a instância $n = 150$, $m = 1688$, $s = 5011$

seus resultados. O *benchmark* possui instâncias para diversas classes de problemas de otimização, a saber: problema do caixeiro viajante simétrico (TSP) e assimétrico (ATSP), problema dos ciclos hamiltonianos (HCP), problema de ordenação sequencial (SOP) e problema de roteamento de veículos capacitados (CVRP).

Tais instâncias seguem um formato específico do *benchmark* cujos detalhes podem ser consultados em Reinelt [2008], onde cada seção de dados do arquivo é descrita assim como os cálculos de distância (euclidiana, manhattan, geográfica, etc) usados como custo dos arcos do grafo correspondente à cada instância.

Para o propósito específico de comparar a qualidade dos resultados obtidos pelos algoritmos EWS, NCH e IR, tomamos algumas das instâncias contidos no *benchmark* TSPLIB, especificamente aquelas referentes ao problema dos ciclos hamiltonianos.

Conforme mencionado, os aplicativos implementados trabalham apenas com arquivos de entrada no formato `.txt` ou DIMACS. As instâncias do problema dos ciclos hamiltonianos foram então convertidas nesse formato, tomando a descrição da topologia dessas instâncias e convertendo-as para o formato DIMACS. Por exemplo, o trecho apresentado na Figura 6.27 que descreve a instância `alb1000.hcp` no formato do *benchmark* TSPLIB

```
NAME : alb1000
COMMENT : Hamiltonian cycle problem (Erbacci)
TYPE : HCP
DIMENSION : 1000
EDGE_DATA_FORMAT : EDGE_LIST
EDGE_DATA_SECTION
  1000    593
  1000    456
  1000    217
   999    577
   999    537
```

Figura 6.27. Descrição de trecho da instância `alb1000.hcp` no formato TSPLIB

foi convertido para o formato DIMACS no trecho ilustrado pela Figura 6.28

```
p  hcp           1000   2000
a  1000    593    1
a  1000    456    1
a  1000    217    1
a   999    577    1
a   999    537    1
```

Figura 6.28. Descrição de trecho da instância `alb1000.hcp` no formato DIMACS

Participam desse conjunto de instâncias algumas das instâncias para o problema de encontrar ciclos hamiltonianos em grafos da TSPLIB: `alb1000.hcp`, `alb2000.hcp`, `alb3000a.hcp` e `alb4000.hcp`. Detalhes sobre o número de vértices e densidade dos grafos que elas representam estão disponíveis na Tabela 6.7.

Tabela 6.7. Detalhes das instâncias que formam o Conjunto III, tomadas do *benchmark* TSPLIB para o problema dos ciclos hamiltonianos

Instância	***Vertices***	***Arcos***	$\|K_n\|$	***Densidade (%)***
`alb1000.hcp`	1000	1998	499500	0,4
`alb2000.hcp`	2000	3996	1999000	0,2
`alb3000a.hcp`	3000	5999	4498500	0,1
`alb4000.hcp`	4000	7997	7998000	0,1

Na descrição, a coluna '$|K_n|$' indica quantos arcos teria o grafo completo correspondente; a coluna 'Densidade (%)' informa a densidade de arcos do grafo, em %, considerando a quantidade de arcos atual do grafo (m) em relação à quantidade de arcos do grafo completo correspondente $[\frac{n(n-1)}{2}]$.

Note que esse conjunto é formado por instâncias de baixa densidade, todas com menos de 1% dos arcos que teria o grafo completo correspondente.

6.4.3.2 Análise dos Resultados

A Tabela 6.8 apresentam os resultados desse experimento e permite-nos levantar três observações interessantes a respeito desse conjunto de instâncias formado por grafos pouquíssimo densos quando solucionando o problema de encontrar árvores geradoras com mínimo de vértices *branch* pelas heurísticas implementadas.

Primeiramente, nota-se que a quantidade de vértices *branch* computadas pelos três métodos é maior do que os valores encontrados para os Conjuntos de Instâncias I e II. Isso deve-se, em parte, à densidade dos grafos de entrada. A coluna $d(\%)$ mostra que esses grafos possuem uma densidade muito baixa se comparados com os grafos utilizados nos experimentos anteriores e, portanto, uma menor quantidade de opções de arcos para interligar seus vértices. De fato, o número de arcos de cada uma das instâncias é aproximadamente $m = 2n$ (ver Tabela 6.7), que podem levar à árvores geradoras com poucas opções de escolha e com grande chance de formar vértices *branch* no processo de construção ou refinamento da árvore geradora.

Em segundo lugar, observa-se que o algoritmo IR apresentou resultado médio em número de vértices *branch* melhor do que as heurísticas EWS e NCH em três das quatro instâncias avaliadas. Destacam-se nesses resultados o *gap* formado entre o *menor valor* encontrado pelo IR para cada instância e o *menor valor* encontrado pelas heurísticas EWS e NCH para a mesma instância, que podem chegar em algumas dezenas de vértices: $gap_{(1000)} = (73 - 54) = 19$, $gap_{(2000)} = (121 - 129) = -8$, $gap_{(3000a)} = (226 - 191) = 35$ e $gap_{(4000)} = (277 - 247) = 30$.

Por fim, nota-se que houve uma variação significativa no número de vértices *branch* computados mesmo entre os métodos EWS e NCH. As instâncias do Conjunto III testadas evidenciaram que a heurística EWS conseguiu obter resultados melhores do que a heurística NCH, em alguns casos com até algumas dezenas de unidades de diferença. Identificando que essa diferença também ocorreu entre os resultados do algoritmo IR e da heurística NCH, e levando em conta que o algoritmo IR opera considerando critérios de troca de arcos com base nas medidas de α e σ, podemos pensar em uma provável evidência de que para grafos pouco densos, com poucos arcos, colocar a tomada de decisão nos arcos (ex: critério de ponderação de pesos, critério de substituição de arcos em função de seu grau de penalidade) ao invés de colocá-la nos vértices do grafo (critério de coloração de vértices) pode ser uma escolha interessante no projeto de algoritmos para o problema MBV.

Tabela 6.8. Comparação dos 'tempos de execução' e 'função objetivo' dos métodos EWS, NCH e IR para o Conjunto III

Benchmark			**Heuristicas Cerulli et al**				**Algoritmo Refinamento Iterativo**											
			EWS		**NCH**		**FUNÇÃO OBJETIVO**									**RUNTIME**		
n	*m*	*d(%)*	*Val*	*Tempo(s)*	*Val*	*Tempo(s)*	*Min*	*1Q*	*Média*	*Mediana*	*3Q*	*Max*	*Dev*	*Var*	*Melhor*	*Min*	*Média*	*Max*
1000	**1998**	**0,4**	**73**	**6,640**	**73**	**7,937**	**54**	**65,75**	**69,07**	**69**	**73**	**80**	**5,18**	**26,83**	**X**	**12,86**	**19,62**	**27,91**
2000	3996	0,2	129	30,470	141	33,238	121	129,75	135,66	135,5	141	155	7,47	55,86		127,24	183,54	244,18
3000	**5999**	**0,1**	**226**	**69,504**	**244**	**76,949**	**191**	**203**	**208,55**	**208,5**	**213,25**	**233**	**8,06**	**65**	**X**	**536,46**	**713,27**	**927,87**
4000	**7997**	**0,1**	**277**	**126,080**	**308**	**136,497**	**247**	**265,75**	**271,93**	**271,5**	**279**	**298**	**10,02**	**100,49**	**X**	**1433,59**	**1783,39**	**2276,1**

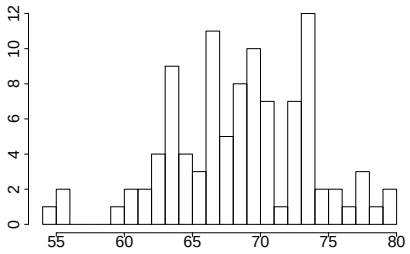

Figura 6.29. Histograma da instância `alb1000` (Conjunto III): $IR_{min} = 54$, $IR_{\bar{x}} = 69.07$, $IR_{max} = 80$, $IR_{med} = 69$, $IR_{mod} = 73$; $EWS_{val} = 73$, $NCH_{val} = 73$

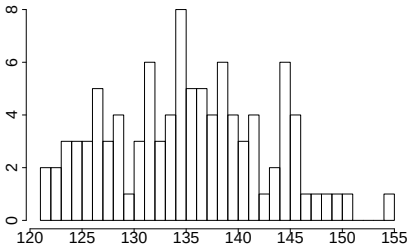

Figura 6.30. Histograma da instância `alb2000` (Conjunto III): $IR_{min} = 121$, $IR_{\bar{x}} = 135.66$, $IR_{max} = 155$, $IR_{med} = 135$, $IR_{mod} = 134$; $EWS_{val} = 129$, $NCH_{val} = 141$

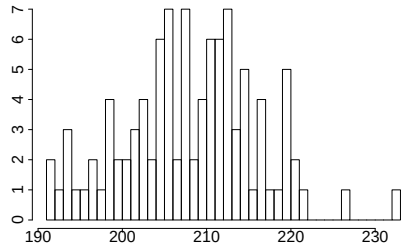

Figura 6.31. Histograma da instância `alb3000a` (Conjunto III): $IR_{min} = 191$, $IR_{\bar{x}} = 208.55$, $IR_{max} = 233$, $IR_{med} = 208.5$, $IR_{mod} = 205, 207, 208$; $EWS_{val} = 226$, $NCH_{val} = 244$

Os histogramas de frequência para a variável 'número de vértices *branch*' da amostra coletada de cada instância é apresentado nas Figuras 6.29, 6.30, 6.31 e 6.32. Por eles podemos observar que 89% das observações da instância `alb1000`, 25% das observações da instância `alb2000`, mais de 90% das observações da instância `alb3000a` e 72% das observações da instância `alb4000` obtiveram resultados menores ou iguais do que o 'melhor' resultado calculado pelas heurísticas EWS e NCH. Apenas para a instância `alb2000` o algoritmo IR apresentou uma distribuição cuja média, mediana e moda estão além dos valores calculados deterministicamente pelas heurísticas de Cerulli et al. [2009], mesmo assim apenas para os resultados da heurística EWS.

Os tempos de execução das heurísticas EWS e NCH e do algoritmo IR estão registrados na Tabela 6.8. Para esse conjunto de instância observa-se que o algoritmo IR

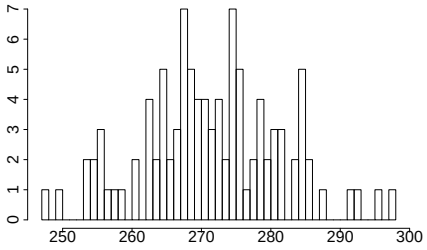

Figura 6.32. Histograma da instância `alb4000` (Conjunto III): $IR_{min} = 247$, $IR_{\bar{x}} = 271.93$, $IR_{max} = 298$, $IR_{med} = 271.5$, $IR_{mod} = 267, 274$; $EWS_{val} = 277$, $NCH_{val} = 308$

consumiu uma quantidade de tempo maior do que as heurísticas de Cerulli et al. [2009] para computar soluções para o problema MBV nos grafos do Conjunto III. Estas, por sua vez, empregaram um esforço computacional na mesma ordem de grandeza. Embora o algoritmo IR tenha despendido mais esforço que as heurísticas EWS e NCH, os resultados médios em 'número de vértices *branch*' encontrados por ele foram melhores para três das quatro instâncias que compõem o conjunto.

Sabe-se que cada uma das instâncias desse *benchmark* contém um ciclo hamiltoniano registrado por Reinelt [2008]. Se tomarmos tal ciclo e removermos um dos arcos que liga o vértice de partida aos demais vértices do ciclo teremos um caminho hamiltoniano. Levando em conta que nenhum dos métodos testados conseguiu encontrar tal caminho hamiltoniano nas instâncias `alb1000.hcp`, `alb2000.hcp`, `alb3000a.hcp` e `alb4000.hcp` temos que, para esse cenário, o método IR pode ser considerado a escolha mais adequada para resolver o problema MBV nos grafos desse conjunto em função da qualidade das soluções computadas (leia-se 'número de vértices *branch*') já que o melhor resultado encontrado nas 100 repetições independentes de cada instância (Tabela 6.8, coluna 'Min' do bloco `FUNÇÃO OBJETIVO`) apresenta uma árvore cuja topologia se aproxima mais de um caminho hamiltoniano do que as árvores equivalentes computadas pelos métodos EWS e NCH.

6.4.4 Conjunto de Instâncias IV (Goldberg (1996))

6.4.4.1 *Benchmark*

O Conjunto de Instâncias IV consiste em um grupo de instâncias geradas aleatoriamente com uso do gerador de instâncias de Goldberg (ver Seção 6.2.4) para grafos com número de vértices variando em $n = 500$, $n = 800$ e $n = 1000$ vértices, e número de arcos com densidade $d = 5\%$, $d = 10\%$ e $d = 15\%$. A Tabela 6.9 apresenta esses dados em formato tabular, com o número de arcos final. A coluna '$|K_n|$ indica o número de vértices do grafo completo correspondente.

Tabela 6.9. Parâmetros para o gerador `crand` para o Conjunto IV

instância	n	m	*seed*	$d(\%)$	$\lvert K_n \rvert$
`g-1`	500	6237	3572	5	124750
`g-2`	500	12475	1709	10	124750
`g-3`	500	18712	8562	15	124750
`g-4`	800	15980	1803	5	319600
`g-5`	800	31960	1388	10	319600
`g-6`	800	47940	5497	15	319600
`g-7`	1000	24975	9215	5	499500
`g-8`	1000	49950	2002	10	499500
`g-9`	1000	74925	1543	15	499500

6.4.4.2 Análise dos Resultados

A Tabela 6.10 apresenta os resultados obtidos na comparação das três heurísticas para as instâncias do Conjunto IV. Para esse caso, em apenas uma das nove instâncias o algoritmo IR se sobressaiu no número médio de vértices *branch*; as heurísticas EWS e NCH encontraram soluções para as instâncias do conjunto IV com poucos vértices *branch*.

Outra informação importante que pode ser obtida pela Tabela 6.10 consiste nos tempos de execução que os métodos EWS, NCH e IR consumiram para computar soluções para o problema MBV sobre as instâncias do Conjunto IV. Em primeiro lugar, o método IR foi o método que apresentou melhor desempenho em esforço computacional, embora não tenha obtido, na média, soluções com menor quantidade de vértices *branch* do que os métodos EWS e NCH. A análise de qualidade da solução com base na topologia da árvore resultante será feita nos próximos parágrafos dessa seção, de forma que a explicação corrente será focada na variável 'tempo de execução'. Em segundo lugar, observa-se novamente que o tempo de execução dos métodos EWS e NCH não cresce na mesma ordem de grandeza para os diversos grafos do Conjunto IV. Todas as instâncias possuem um número de arcos $m \geq 5000$, e a diferença entre os tempos de execução das duas heurísticas fica mais evidente conforme o valor de m aumenta. A hipótese levantada na Seção 6.3 pode ser uma possível explicação para essa diferença; entretando tal confirmação está fora do escopo deste trabalho.

Tabela 6.10. Comparação dos 'tempos de execução' e 'função objetivo' dos métodos EWS, NCH e IR para o Conjunto IV

Benchmark				**Heurísticas Cerulli et al**				**Algoritmo Refinamento Iterativo**											
				EWS		**NCH**		**FUNÇAO OBJETIVO**									**RUNTIME**		
inst	n	m	d	Val	Tempo(s)	Val	Tempo(s)	Min	1Q	Média	Mediana	3Q	Max	Melhor	Var	Dev	Min	Média	Max
g-1	500	6237	5	2	17,285	2	13,501	1	3	4,71	4	6	10		1,75	3,06	2,74	4,78	6,54
g-2	500	12475	10	0	45,400	0	26,641	0	1	1,77	2	2	5		1,05	1,11	5,72	8,63	11,91
g-3	500	18712	15	0	83,553	0	40,099	0	0	0,91	1	1	3		0,75	0,57	12,5	15,14	17,63
g-4	800	15980	5	2	88,094	2	57,400	2	3	4,52	5	6	8		1,51	2,27	15,09	22,41	30,57
g-5	800	31960	10	0	259,112	0	114,511	0	1	1,84	2	2	4		0,95	0,9	33,5	47,23	59,57
g-6	**800**	**47940**	**15**	**1**	**523,025**	**1**	**172,511**	**0**	**0**	**0,86**	**1**	**1**	**2**	**X**	**0,65**	**0,42**	**72,96**	**89,43**	**101,77**
g-7	1000	24975	5	0	191,832	0	107,235	2	3	4,45	4	5	9		1,5	2,25	33,89	51,68	64,66
g-8	1000	49950	10	1	670,982	1	234,123	0	1	1,8	2	2	4		0,93	0,87	92,57	115,96	136,36
g-9	1000	74925	15	0	1569,070	0	366,611	0	0	0,81	1	1	3		0,83	0,68	201,2	217,69	236,16

As Figuras 6.33 à 6.41 apresentam o histograma da variável 'número de vértices *branch*' para cada uma das amostras construídas com 100 observações advindas da execução do algoritmo IR sobre as instâncias do *benchmark*. Exceto para a instância `g-6`, verifica-se que o algoritmo IR retornou resultados em sua maior parte distribuídos além dos resultados determinísticos encontrados pelas heurísticas EWS e NCH; portanto para as instâncias geradas aleatoriamente para o Conjunto IV com densidades de 5%, 10% e 15% o algoritmo IR não apresentou bom desempenho.

Isso deve-se, provavelmente, ao fato de que as próprias heurísticas EWS e NCH conseguiram obter soluções com quantidade de vértices *branch* muito baixas, em alguns casos chegando até a encontrar caminhos hamiltonianos no grafo (ver resultados para as instâncias `g-2`, `g-3`, `g-5`, `g-7` e `g-9` na Tabela 6.10). É possível que a topologia dos grafos gerados pelo gerador `crand` facilite a construção da árvore; entretanto esse tipo de investigação demanda estudar o código fonte do gerador e é um tópico fora do escopo deste trabalho.

Usar a média aritmética nesse tipo de comparação tem o efeito de distorcer os resultados, já que para valores muito baixos, uma pequena variação numérica pode afetar abruptamente a média. Iremos então analisar os resultados pelos histogramas apresentados nas Figuras 6.33 - 6.41, tomando o valor mais frequente (moda) como referência. A Tabela 6.11 foi construída para facilitar essa análise.

Tabela 6.11. Resultados parciais para análise do Conjunto IV

Instância	EWS/NCH	IR_{min}	IR_{mod}	IR_{max}	$\lvert IR_{mod} - EWS/NCH \rvert$	$Min\%$	$Moda\%$	$Acum^*\%$
`g-1`	2	1	3	10	1	8	25	49
`g-2`	0	0	0	5	0	45	45	45
`g-3`	0	0	0	3	0	80	80	80
`g-4`	2	2	2	8	0	30	30	30
`g-5`	1	0	0	4	1	35	40	75
`g-6`	1	0	0	2	1	80	80	80
`g-7`	**0**	**2**	**2**	**9**	**2**	**30**	**30**	**30**
`g-8`	1	0	1	4	0	37	40	77
`g-9`	0	0	0	0	0	80	80	80

Nessa tabela, as colunas 'EWS/NCH' referem-se ao melhor valor encontrado para cada instância pelos métodos EWS e NCH, as colunas 'IR_{min}', 'IR_{mod}' e 'IR_{max}' referem-se aos resultados mínimo, moda e máximo da amostra que corresponde à cada instância, a coluna '$|IR_{mod} - EWS/NCH|$' refere-se ao módulo da 'distância' entre as soluções em número de vértices *branch*, e as colunas '$Min\%$', '$Moda\%$'e '$Acum^*\%$' referem-se à porcentagem de observações da amostra que estão situadas no valor mínimo, na moda e no intervalo (acumulado) entre $IR_{min} \leq IR_{val} \leq IR_{mod}$.

Observe que para a maioria dos casos, ou não houve distância entre o valor mais frequente e os resultados EWS/NCH ou se houve distância, esta ficou em até uma unidade do valor de referência. Apenas o caso destacado em negrito, da instância `g-7`

o algoritmo IR teve um desempenho ruim considerando os critérios da análise de moda. De fato, boa parte dos resultados concentram-se entre o menor valor e a moda (ver colunas '$Min\%$', '$Moda\%$'e '$Acum^*\%$') sendo que a média está sendo influenciada pela amplitude dos limites da amostra.

Outra consideração a ser feita consiste em determinar pelos histogramas qual é a probabilidade de se conseguir, em uma única execução do algoritmo IR, um resultado menor ou igual ao menor valor dentre os resultados das heurísticas EWS e NCH para uma mesma instância. Utilizando a análise de frequência, podemos estimar que tais probabilidades são da seguinte ordem: 25% para a instância `g-1`, 45% para a instância `g-2`, 80% para a instância `g-3`, 30% para a instância `g-4`, 35% para a instância `g-5`, 80% para a instância `g-6`, 0% para a instância `g-7`, 75% para a instância `g-8` e 80% para a instância `g-9`. Dessa forma, mesmo não se classificando bem de acordo com o critério de comparação da média do algoritmo IR com os resultados determinísticos das heurísticas EWS e NCH, ainda assim existem boas chances (mais de 70%) de conseguir um resultado no intervalo $IR_{min} \leq IR_{val} \leq argmin\{EWS_{val}, NCH_{val}\}$ com apenas uma execução para as instâncias `g-3`, `g-6`, `g-8` e `g-9` (isto é, 44% do conjunto IV).

Embora não tenha conseguido sobrepor os resultados dos métodos EWS e NCH segundo o critério de classificação por comparação da média da amostra, o algoritmo IR conseguiu obter resultados na proximidade dessas soluções, ficando a especulação sobre a influência do gerador sobre a topologia dos grafos testados e, consequentemente, seu impacto na topologia da solução para trabalhos futuros.

6.4.5 Conjunto de Instâncias V (Beasley (1989))

6.4.5.1 *Benchmark*

O Conjunto de Instâncias V foi construído tomando-se as instâncias `steind11`, `steind12`, `steind13`, `steind14` e `steind15` do *benchmark* OR-Library, de Beasley, utilizadas no artigo *An SST-based Algorithm for The Steiner Problem in Graphs* (Beasley [1989]).

Tais instâncias consistem em entradas para o problema de encontrar árvores de Steiner em grafos, e correspondem à grafos com $n = 1000$, $m = 5000$ e $d \sim 1\%$. O formato original do arquivo não corresponde ao formato DIMACS, portanto foi convertido para ser utilizado pelo aplicativo `mbv-solver.bin` aproveitando-se apenas as informações sobre topologia (arcos) da rede de entrada.

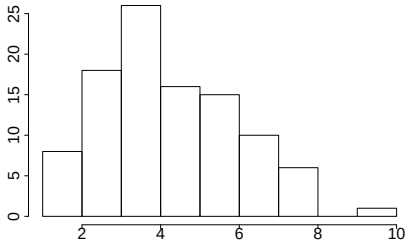

Figura 6.33. Histograma da instância `g-1` (Conjunto IV): $IR_{min} = 1$, $IR_{\bar{x}} = 4.71$, $IR_{max} = 10$, $IR_{med} = 4$, $IR_{mod} = 3$; $EWS_{val} = 2$, $NCH_{val} = 2$

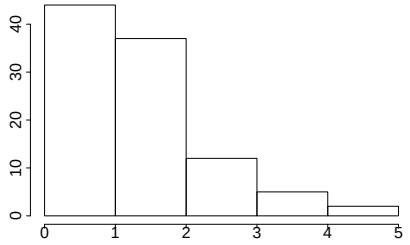

Figura 6.34. Histograma da instância `g-2` (Conjunto IV): $IR_{min} = 0$, $IR_{\bar{x}} = 1.77$, $IR_{max} = 5$, $IR_{med} = 2$, $IR_{mod} = 0$; $EWS_{val} = 0$, $NCH_{val} = 0$

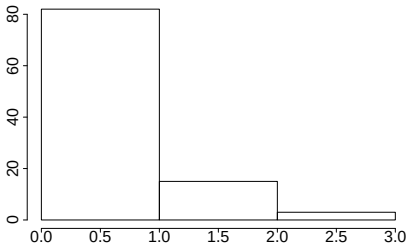

Figura 6.35. Histograma da instância `g-3`, (Conjunto IV): $IR_{min} = 0$, $IR_{\bar{x}} = 0.91$, $IR_{max} = 3$, $IR_{med} = 1$, $IR_{mod} = 0$; $EWS_{val} = 0$, $NCH_{val} = 0$

6.4.5.2 Análise dos Resultados

A Tabela 6.12 apresenta os resultados obtidos com a execução determinística das heurísticas EWS e NCH e com as 100 repetições do algoritmo IR para as instâncias do Conjunto V. Uma observação importante é que, dessa vez, o *benchmark* favoreceu a construção de árvores pelos métodos que usam o critério de escolha de arcos com base em informações nos vértices (NCH) do que pelos métodos que usam o critério de escolha de arcos baseado em informações nos arcos (EWS e IR). Apesar disso, o algoritmo IR conseguiu um resultado médio melhor em duas das cinco instâncias do conjunto se comparado com os resultados das heurísticas EWS e NCH, e resultados mínimos com até uma dezena de vértices *branch* de diferença em relação à essas heurísticas. Em relação ao tempo de execução, a Tabela 6.12 apresenta resultados experimentais que mostram que o algoritmo IR consumiu mais tempo de processamento do que as heurísticas EWS

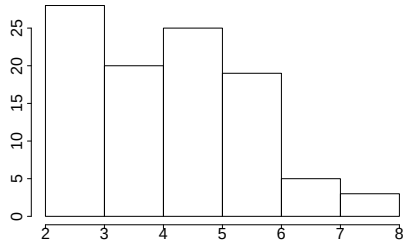

Figura 6.36. Histograma da instância `g-4` (Conjunto IV): $IR_{min} = 2$, $IR_{\bar{x}} = 4.52$, $IR_{max} = 8$, $IR_{med} = 5$, $IR_{mod} = 2$; $EWS_{val} = 2$, $NCH_{val} = 2$

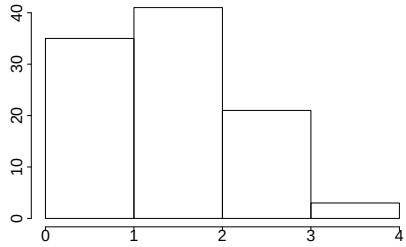

Figura 6.37. Histograma da instância `g-5` (Conjunto IV): $IR_{min} = 0$, $IR_{\bar{x}} = 1.84$, $IR_{max} = 4$, $IR_{med} = 2$, $IR_{mod} = 1$; $EWS_{val} = 0$, $NCH_{val} = 0$

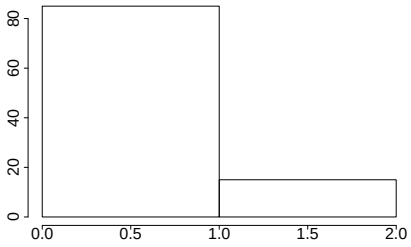

Figura 6.38. Histograma da instância `g-6` (Conjunto IV): $IR_{min} = 0$, $IR_{\bar{x}} = 0.86$, $IR_{max} = 2$, $IR_{med} = 1$, $IR_{mod} = 0$; $EWS_{val} = 1$, $NCH_{val} = 1$

e NCH quando aplicado sobre os grafos contidos no Conjunto V. Importante observar que os tempos de execução das heurísticas EWS e NCH apresentaram a mesma ordem de grandeza para esses grafos, que possuem $m = 5000$ arcos. Nos resultados anteriores, a diferenciação entre os tempos de execução desses dois métodos só ficou evidente em grafos com número de arcos a partir desse valor de m.

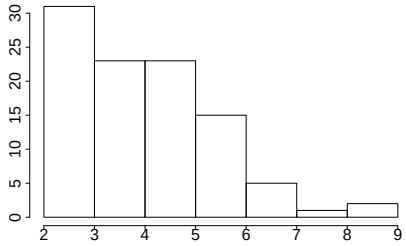

Figura 6.39. Histograma da instância `g-7` (Conjunto IV): $IR_{min} = 2$, $IR_{\bar{x}} = 4.45$, $IR_{max} = 9$, $IR_{med} = 4$, $IR_{mod} = 2$; $EWS_{val} = 0$, $NCH_{val} = 0$

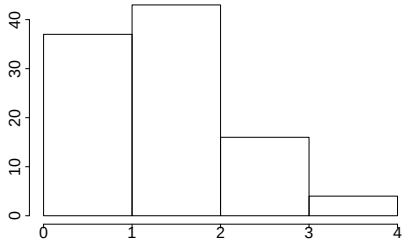

Figura 6.40. Histograma da instância `g-8` (Conjunto IV): $IR_{min} = 0$, $IR_{\bar{x}} = 1.8$, $IR_{max} = 4$, $IR_{med} = 2$, $IR_{mod} = 1$; $EWS_{val} = 1$, $NCH_{val} = 1$

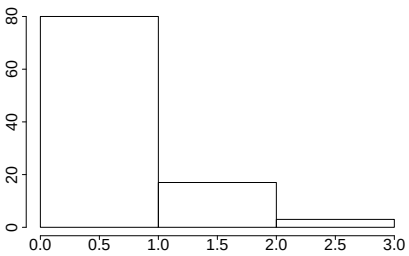

Figura 6.41. Histograma da instância `g-9` (Conjunto IV): $IR_{min} = 0$, $IR_{\bar{x}} = 0.81$, $IR_{max} = 3$, $IR_{med} = 1$, $IR_{mod} = 0$; $EWS_{val} = 0$, $NCH_{val} = 0$

Tabela 6.12. Comparação dos 'tempos de execução' e 'função objetivo' dos métodos EWS, NCH e IR para o Conjunto V

Benchmark	**Heurísticas Cerulli et al**				**Algoritmo Refinamento Iterativo**											
	EWS		**NCH**		**FUNÇÃO OBJETIVO**									**RUNTIME**		
Instâncias	*Val*	*Tempo(s)*	*Val*	*Tempo(s)*	*Min*	*1Q*	*Média*	*Mediana*	*3Q*	*Max*	*Melhor*	*Var*	*Dev*	*Min*	*Média*	*Max*
steind11	34	22,381	35	22,325	33	39	41,39	42	44	50		3,51	12,34	27,49	46,31	65,54
steind12	**40**	**22,321**	**36**	**22,445**	**26**	**32**	**35,46**	**35**	**39**	**46**	**X**	**4,51**	**20,33**	**24,99**	**40,44**	**63,53**
steind13	40	22,201	35	22,485	28	37	39,76	39	41,25	54		4,24	17,96	30,11	44,19	64,7
steind14	34	22,349	33	22,441	28	36	38,19	38	40,25	50		3,91	15,27	20,38	44,51	71,47
steind15	**45**	**22,413**	**40**	**22,529**	**27**	**36,75**	**38,87**	**38,5**	**42**	**48**	**X**	**3,6**	**12,98**	**27,84**	**45,59**	**70,34**

As Figuras 6.42 - 6.46 apresentam os histogramas de frequência para a variável 'número de vértices *branch*' das instâncias desse conjunto. Por eles é possível verificar que nesse caso a maioria das observações da amostra estão distribuídas além dos valores EWS_{val} e NCH_{val}, comprovando portanto que o algoritmo IR não foi bem sucedido nesse conjunto de instâncias.

Cabe notar, entretanto, que cada grafo desse *benchmark* possui $n = 1000$ e $m = 5000$, logo a densidade de tais arcos é da ordem de $d = 1\%$, ou seja, são grafos de baixa densidade e portanto possuem uma menor quantidade de arcos de substituição capazes de realizar uma troca sem prejudicar a qualidade da árvore final.

Some-se à este fato o conhecimento sobre o funcionamento dos algoritmos EWS/NCH e IR. Os métodos de Cerulli et al. [2009] criam uma árvore geradora usando os critérios de escolha de arcos que são específicos de cada algoritmo, sempre tentando evitar a criação de vértices *branch*. O resultado de uma única passada já é uma árvore geradora final, logo não há refinamento algum e toda a 'inteligência' possível foi utilizada no processo construtivo.

Por outro lado, o algoritmo IR inicia com uma árvore geradora irrestrita, com uma quantidade inicial de vértices consideravelmente maior do que as árvores-solução retornadas pelos métodos EWS e NCH. O refinamento é o artifício que permite transformar essa árvore geradora irrestrita em uma árvore geradora que respeita mais restrições; para tal é preciso haver possibilidades de reduzir as infrações ocorridas.

Conforme comentado no Capítulo 5 o algoritmo IR proposto para o problema MBV opera o refinamento por meio de substituições de arcos na árvore. A quantidade de iterações é limitada principalmente pela quantidade de infrações da solução inicial, mas também pela 'oferta' de arcos de substituição 'vantajosos', já que uma iteração 'bem sucedida' consegue eliminar no máximo dois vértices *branch*. Empiricamente, nota-se que no caso médio ocorrem mais substituições que reduzem o grau de um vértice infrator e que *no máximo* eliminam um vértice *branch*. Logo, para um grafo com grande quantidade de vértices, espera-se uma quantidade considerável de iterações no refinamento.

Para reforçar ainda mais a sugestão de que o algoritmo IR não se comporta muito bem nos casos onde existe escassez de arcos de substituição (isto é, em grafos com densidade muito baixa), é preciso lembrar do fato que nem todo arco que une os conjuntos $[S, S']$ é elegível para estar em L_{rep}. De fato, a situação que elimina a candidatura de um arco nesse conjunto é quando este forma ciclo na árvore.

Levando em conta essas informações, podemos suspeitar que em grafos pouco densos (i) existe pouca oferta de arcos de substituição; (ii) é possível que boa parte dos arcos disponíveis formem ciclos na árvore e portanto não sejam elegíveis. Para comprovar

essa suspeita é preciso realizar novos experimentos empregando o algoritmo IR cujos resultados sejam capazes de fundamentar essas hipóteses. Tais experimentos serão deixados como sugestão para trabalhos futuros.

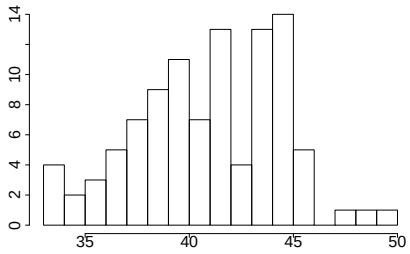

Figura 6.42. Histograma da instância `steind11` (Conjunto V): $IR_{min} = 33$, $IR_{\bar{x}} = 41.39$, $IR_{max} = 50$, $IR_{med} = 42$, $IR_{mod} = 44$; $EWS_{val} = 34$, $NCH_{val} = 35$

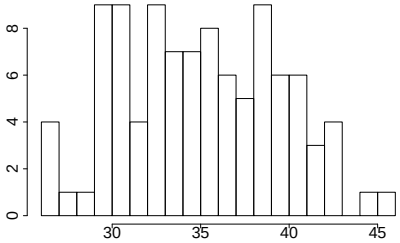

Figura 6.43. Histograma da instância `steind12` (Conjunto V): $IR_{min} = 26$, $IR_{\bar{x}} = 35.46$, $IR_{max} = 46$, $IR_{med} = 35$, $IR_{mod} = 30, 31, 33, 39$; $EWS_{val} = 40$, $NCH_{val} = 36$

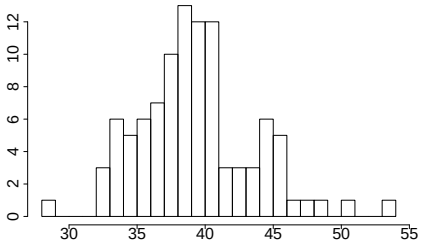

Figura 6.44. Histograma da instância `steind13`, (Conjunto V): $IR_{min} = 28$, $IR_{\bar{x}} = 39.76$, $IR_{max} = 54$, $IR_{med} = 39$, $IR_{mod} = 38$; $EWS_{val} = 40$, $NCH_{val} = 35$

6.4.6 Conjunto de Instâncias VI (Leighton (1979))

6.4.6.1 *Benchmark*

O Conjunto de Instâncias IV consiste em 12 grafos criados por Frank Leighton no artigo *A Graph Coloring Algorithm for Large Scheduling Problem* (Leighton [1979]). Consistem em arquivos que descrevem grafos no formato DIMACS, e que possuem as características dadas pela Tabela 6.13.

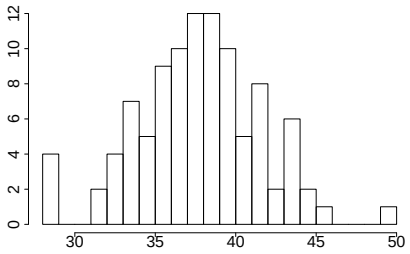

Figura 6.45. Histograma da instância `steind14` (Conjunto V): $IR_{min} = 28$, $IR_{\bar{x}} = 38.19$, $IR_{max} = 50$, $IR_{med} = 38$, $IR_{mod} = 37,38$; $EWS_{val} = 34$, $NCH_{val} = 33$

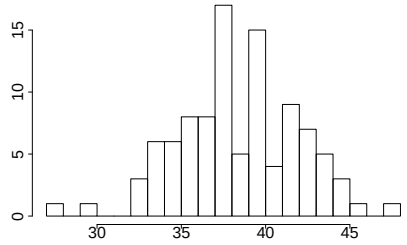

Figura 6.46. Histograma da instância `steind15` (Conjunto V): $IR_{min} = 27$, $IR_{\bar{x}} = 38.87$, $IR_{max} = 48$, $IR_{med} = 38.5$, $IR_{mod} = 37$; $EWS_{val} = 45$, $NCH_{val} = 40$

Tabela 6.13. Detalhes sobre as instâncias do Conjunto VI

Instância	n	m	$d(\%)$	$\lvert K_n \rvert$
`le450_5a.col`	450	5714	6	101025
`le450_5b.col`	450	5734	6	101025
`le450_5c.col`	450	9803	10	101025
`le450_5d.col`	450	9757	10	101025
`le450_15a.col`	450	8186	8	101025
`le450_15b.col`	450	8169	8	101025
`le450_15c.col`	450	16680	17	101025
`le450_15d.col`	450	16750	17	101025
`le450_25a.col`	450	8260	8	101025
`le450_25b.col`	450	8263	8	101025
`le450_25c.col`	450	17343	17	101025
`le450_25d.col`	450	17425	17	101025

6.4.6.2 Análise dos Resultados

A Tabela 6.14 apresenta os resultados da execução dos algoritmos EWS, NCH e IR para o Conjunto VI. Para esse conjunto de instâncias, o algoritmo IR conseguiu resolver 50% dos casos com número médio de vértices *branch* inferior aos resultados computados pelas heurísticas EWS e NCH.

Importante ressaltar que em muitos casos encontrou-se um resultado mínimo computado pelo algoritmo IR cuja solução corresponde à um caminho hamiltoniano, isto é, uma árvore geradora sem nenhum vértice *branch*. (ver coluna 'Min' para as instâncias

le450_5c, le450_5d, le450_15c, le450_15d, le450_25c e le450_25d). Independente disso, o algoritmo IR conseguiu encontrar ao menos uma solução com qualidade melhor do que a melhor solução computada pelas heurísticas de Cerulli et al. [2009] em cada instância do conjunto.

Mesmo para os casos onde o método IR não se sobressaiu sobre ambas heurísticas considerando seu resultado médio, ainda assim existe uma boa probabilidade de conseguir um resultado igual ou superior em qualidade do que as heurísticas EWS e NCH a partir de uma única execução deste algoritmo. De acordo com os histogramas apresentados nas Figuras 6.47 - 6.58, temos as seguintes probabilidade aproximadas de sacar um resultado no intervalo $IR_{min} \leq IR_{val} \leq argmin\{EWS_{val}, NCH_{val}\}$ em uma única execução do algoritmo IR: 55% para a instância le450_5a, 55% para a instância le450_5b, 90% para a instância le450_5c, 87% para a instância le450_5d, 70% para a instância le450_15a, 95% para a instância le450_15b, 80% para a instância le450_15c, 90% para a instância le450_15d, 26% para a instância le450_25a, 42% para a instância le450_25b, 87% para a instância le450_25c e 85% para a instância le450_25d.

Considerando tais probabilidades, nota-se que existem casos onde o critério adotado de comparar o resultado médio do algoritmo IR com os resultados determinísticos das heurísticas EWS e NCH pode levar à uma interpretação injusta da Tabela 6.14. Instâncias que foram 'prejudicadas' por essa análise consistem nas instâncias le450_15a e le450_25d. Dessa forma, se tomarmos o critério de que considera-se um sucesso os casos onde uma execução do algoritmo IR retorna um resultado menor ou igual ao menor dentre os resultados das heurísticas EWS e NCH com probabilidade de pelo menos 60%, temos que para esse conjunto o algoritmo IR teria resolvido com sucesso oito das doze instâncias (66% dos casos).

Tabela 6.14. Comparação dos 'tempos de execução' e 'função objetivo' dos métodos EWS, NCH e IR para o Conjunto VI

Benchmark	Heurísticas Cerulli et al				Algoritmo Refinamento Iterativo											
	EWS		NCH		FUNÇÃO OBJETIVO									RUNTIME		
Instâncias	*Val*	*Tempo(s)*	*Val*	*Tempo(s)*	*Min*	*1Q*	*Média*	*Mediana*	*3Q*	*Max*	*Melhor*	*Var*	*Dev*	*Min*	*Média*	*Max*
le450_5a	3	13,581	3	11,501	1	3	4,23	4	5	8		1,51	2,28	2	3,05	4,82
le450_5b	4	14,069	5	11,713	1	3	4,15	4	5	7		1,31	1,7	1,82	2,98	4,47
le450_5c	**3**	**28,878**	**3**	**20,465**	**0**	**1**	**1,91**	**2**	**3**	**4**	**X**	**1,08**	**1,17**	**3,02**	**3,79**	**5,13**
le450_5d	**2**	**28,946**	**3**	**20,233**	**0**	**1**	**1,86**	**2**	**2**	**5**	**X**	**1,02**	**1,03**	**2,91**	**3,82**	**5,87**
le450_15a	7	22,133	6	16,421	4	5	6,63	6	8	10		1,5	2,26	2,58	4,71	8,34
le450_15b	**10**	**22,145**	**9**	**16,357**	**3**	**6**	**7,62**	**8**	**9**	**12**	**X**	**1,75**	**3,07**	**2,92**	**5,46**	**8,38**
le450_15c	**3**	**64,696**	**3**	**34,990**	**0**	**0**	**1,12**	**1**	**2**	**3**	**X**	**0,9**	**0,81**	**3,35**	**5,49**	**7,79**
le450_15d	**3**	**65,608**	**3**	**35,126**	**0**	**1**	**1,06**	**1**	**1,25**	**3**	**X**	**0,78**	**0,6**	**3,6**	**5,43**	**7,29**
le450_25a	12	22,993	11	17,049	8	12	13,66	13,5	15	19		2,44	5,96	3,75	8	15,08
le450_25b	10	22,997	7	17,229	4	8	8,9	9	10	13		2,06	4,25	3,64	6,14	10,14
le450_25c	**4**	**69,820**	**3**	**36,414**	**0**	**1**	**2,02**	**2**	**3**	**5**	**X**	**1,16**	**1,35**	**5,42**	**7,15**	**10,39**
le450_25d	1	69,620	1	36,434	0	1	1,49	1	2	4		0,97	0,94	5,26	6,77	9,14

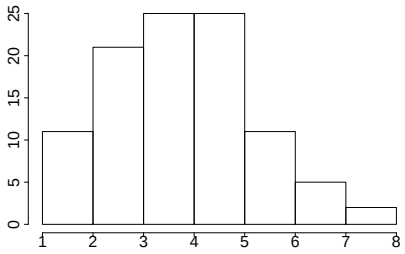

Figura 6.47. Histograma da instância `le450_5a` (Conjunto VI): $IR_{min} = 1$, $IR_{\bar{x}} = 4.23$, $IR_{max} = 8$, $IR_{med} = 4$, $IR_{mod} = 3,4$; $EWS_{val} = 3$, $NCH_{val} = 3$

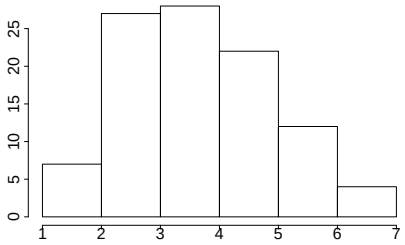

Figura 6.48. Histograma da instância `le450_5b` (Conjunto VI): $IR_{min} = 1$, $IR_{\bar{x}} = 4.15$, $IR_{max} = 7$, $IR_{med} = 4$, $IR_{mod} = 3$; $EWS_{val} = 4$, $NCH_{val} = 5$

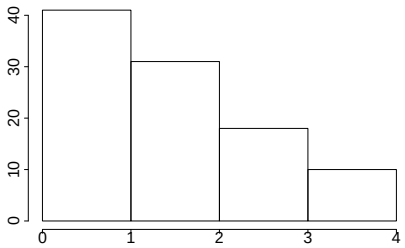

Figura 6.49. Histograma da instância `le450_5c` (Conjunto VI): $IR_{min} = 0$, $IR_{\bar{x}} = 1.91$, $IR_{max} = 4$, $IR_{med} = 2$, $IR_{mod} = 0$; $EWS_{val} = 3$, $NCH_{val} = 3$

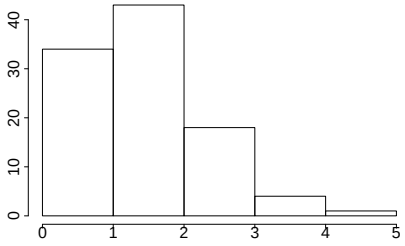

Figura 6.50. Histograma da instância `le450_5d` (Conjunto VI): $IR_{min} = 0$, $IR_{\bar{x}} = 1.86$, $IR_{max} = 5$, $IR_{med} = 2$, $IR_{mod} = 1$; $EWS_{val} = 2$, $NCH_{val} = 3$

Os tempos de execução dos algoritmos EWS, NCH e IR para as instâncias do Conjunto VI são dados na Tabela 6.14. O algoritmo IR foi o método mais rápido dentre os avaliados considerando-se a comparação dos tempos consumidos pelas heurísticas EWS e NCH com o tempo gasto pelo algoritmo IR aplicado na repetição que mais demorou

para retornar um resultado em cada instância do *benchmark* (coluna 'Max' do bloco `RUNTIME`). Se considerarmos os resultados de tempo apenas para as heurísticas EWS e NCH, verifica-se pela Tabela 6.14 que a diferença entre a ordem de grandeza desses resultados fica mais evidente nos grafos `le450_5c.col`, `le450_5d.col`, `le450_15c.col`, `le450_15d.col`, `le450_25c.col` e `le450_35d.col`. Esses grafos são os que possuem mais arcos dentre os contidos no Conjunto de Instâncias VI; de fato, possuem cerca de 9, 16 e 17 mil arcos, respectivamente.

6.5 Considerações Finais

Os experimentos descritos nessa seção apresentam resultados sobre seis diferentes conjuntos de instâncias, de onde foi mostrado que o algoritmo IR foi capaz de obter um resultado médio considerando a quantidade de vértices *branch* das suas soluções resultantes em 100 execuções independentes de cada instância para a maioria das instâncias presentes em quatro desses seis conjuntos testados se comparados com os resultados obtidos pelas heurísticas EWS e NCH.

Utilizar um resultado médio como critério de comparação pode desvalorizar os resultados obtidos pelo algoritmo IR pois essa medida de tendência central é afetada pelos valores extremos da observação (no caso, das 100 observações obtidas nas 100 execuções independentes do algoritmo IR), 'prejudicando' a análise de resultados de alguns desses conjuntos de instâncias.

Entretanto, existe outra maneira de avaliar esses resultados, que é considerar cada execução independente do algoritmo IR como sendo uma iteração de um método multi-partida. De acordo com Martí [2003], métodos multi-partida são métodos constituídos por duas fases: a primeira na qual uma solução é gerada, e a segunda onde a solução é tipicamente melhorada. Dessa forma, cada iteração produz uma solução e a melhor dentre todas as soluções geradas é retornada como saída do algoritmo.

Analisando os resultados obtidos nessa seção sobre esse novo ponto de vista, e generalizando que cada execução independente consiste em uma iteração de um método multi-partida, teríamos que o resultado IR_{min} consiste na solução obtida para cada instância dos conjuntos testados. Se as Tabelas 6.3, 6.5, 6.6, 6.8, 6.10, 6.12 e 6.14 forem revistas sob a luz desse novo fato então temos que o algoritmo IR conseguiu obter melhores resultados que as heurísticas EWS e NCH em 100% dos casos testados. Embora essa abordagem não tenha sido utilizada nesse trabalho, ela é sugerida como trabalhos futuros no Capítulo 7.

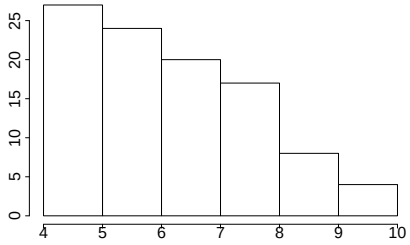

Figura 6.51. Histograma da instância `le450_15a` (Conjunto VI): $IR_{min} = 4$, $IR_{\bar{x}} = 6.63$, $IR_{max} = 10$, $IR_{med} = 6$, $IR_{mod} = 4$; $EWS_{val} = 7$, $NCH_{val} = 6$

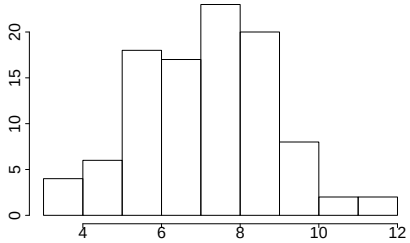

Figura 6.52. Histograma da instância `le450_15b` (Conjunto VI): $IR_{min} = 3$, $IR_{\bar{x}} = 7.62$, $IR_{max} = 12$, $IR_{med} = 8$, $IR_{mod} = 7$; $EWS_{val} = 10$, $NCH_{val} = 9$

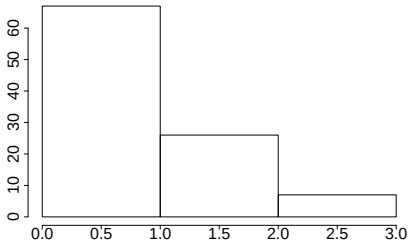

Figura 6.53. Histograma da instância `le450_15c` (Conjunto VI): $IR_{min} = 0$, $IR_{\bar{x}} = 1.12$, $IR_{max} = 3$, $IR_{med} = 1$, $IR_{mod} = 0$; $EWS_{val} = 3$, $NCH_{val} = 3$

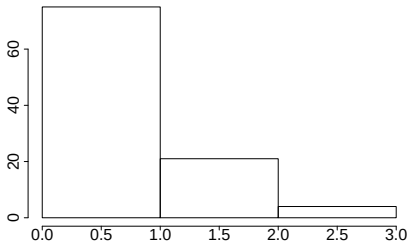

Figura 6.54. Histograma da instância `le450_15d` (Conjunto VI): $IR_{min} = 0$, $IR_{\bar{x}} = 1.06$, $IR_{max} = 3$, $IR_{med} = 1$, $IR_{mod} = 0$; $EWS_{val} = 3$, $NCH_{val} = 3$

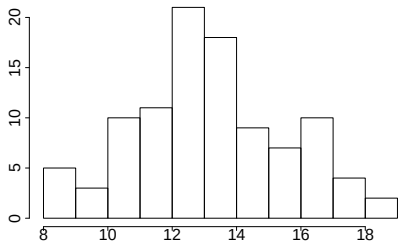

Figura 6.55. Histograma da instância `le450_25a` (Conjunto VI): $IR_{min} = 8$, $IR_{\bar{x}} = 13.66$, $IR_{max} = 19$, $IR_{med} = 13.5$, $IR_{mod} = 13$; $EWS_{val} = 12$, $NCH_{val} = 11$

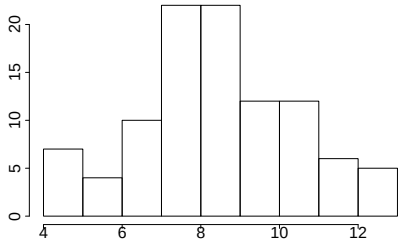

Figura 6.56. Histograma da instância `le450_25b` (Conjunto VI): $IR_{min} = 4$, $IR_{\bar{x}} = 8.9$, $IR_{max} = 13$, $IR_{med} = 9$, $IR_{mod} = 7, 8$; $EWS_{val} = 10$, $NCH_{val} = 7$

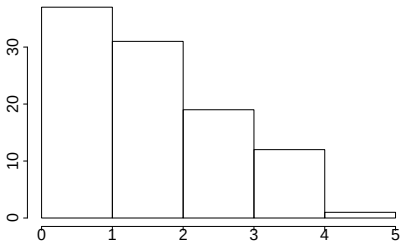

Figura 6.57. Histograma da instância `le450_25c` (Conjunto VI): $IR_{min} = 0$, $IR_{\bar{x}} = 2.02$, $IR_{max} = 5$, $IR_{med} = 2$, $IR_{mod} = 0$; $EWS_{val} = 4$, $NCH_{val} = 3$

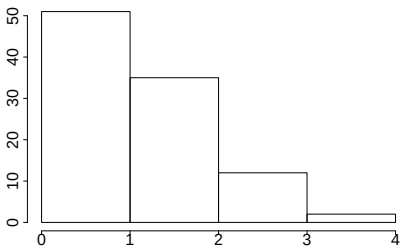

Figura 6.58. Histograma da instância `le450_25d` (Conjunto VI): $IR_{min} = 0$, $IR_{\bar{x}} = 1.49$, $IR_{max} = 4$, $IR_{med} = 1$, $IR_{mod} = 0$; $EWS_{val} = 1$, $NCH_{val} = 1$

Capítulo 7

Conclusões

7.1 Conclusões

O presente trabalho aborda o problema $\mathcal{NP}$-Completo conhecido como problema da árvore geradora com número mínimo de vértices *branch* (MBV) por meio de uma proposta de algoritmo de refinamento iterativo. Tal algoritmo computa árvores geradoras 'restritas' para o problema MBV usando refinamentos sucessivos como artifício para transformar uma árvore geradora inicialmente irrestrita em uma árvore geradora final cujo objetivo é possuir a menor quantidade possível de vértices $v \in V$ tais que $\delta(v) \geq 3$, denominados vértices *branch*.

Destaca-se no trabalho a estratégia empregada pelo algoritmo para substituir arcos associados com as infrações por arcos 'vantajosos' de acordo com um critério que mede o grau de infração de um arco. Tal critério considera duas medidas de infração associadas com a incidência do arco em vértices *branch* e grau de seus vértices extremos.

Resultados experimentais envolvendo a comparação dos resultados médios obtidos sobre seis diferentes conjuntos de instâncias para o algoritmo de refinamento iterativo (IR) proposto com os resultados determinísticos computados pelas heurísticas de ponderação de arcos (EWS) e coloração de vértices (NCH) de Cerulli et al. [2009] sugerem que o algoritmo é mais bem sucedido nos casos em que os grafos de entrada são densos o suficiente para oferecer grande disponibilidade de arcos de substituição para a árvore, situação esta ocorrida em boa parte das instâncias presentes em quatro dos seis conjuntos testados. Nos dois conjuntos restantes o método não obteve sucesso na tentativa de sobrepor os resultados das heurísticas EWS e NCH em função das instâncias que compõem esses conjuntos possuírem baixa densidade de arcos (na ordem de 1%) ou possuírem topologia que facilite a obtenção de bons resultados pelas heurísticas propostas por Cerulli et al. [2009].

A análise dos tempos de execução apontou que o algoritmo IR conseguiu, para a

maioria dos conjuntos testados, resultados em um tempo de processamento menor do que as heurísticas EWS e NCH de Cerulli et al. [2009]. Essa situação ocorreu quando experimentando os conjuntos I, II, IV e VI. Além disso, os resultados experimentais mostraram que existe uma diferença no tempo de execução entre as heurísticas EWS e NCH, embora Cerulli et al. [2009] tenham apresentado uma análise de complexidade da ordem de $O(nm)$ para ambos métodos. Tais resultados sugerem que existe um fator de diferença entre esses métodos que cresce em função do número de arcos m dos grafos; tornando-se mais evidente em grafos com $m \geq 5000$.

Uma possível explicação para esse fato, não comprovada, consiste no fato de que esses autores podem ter considerado uma simplificação durante a análise de complexidade do método EWS. É possível que, por simplificação, tenham considerado que a construção da lista L (apresentada no Algoritmo 1 do Capítulo 3) consome um número constante de operações, isto é, possui ordem $O(1)$, enquanto que na prática ela necessita visitar todos os arcos do grafo G' e possivelmente ordená-los em ordem de seu peso a fim de incluir todos os arcos de G' que possuem peso mínimo e que não formem ciclos se inseridos na árvore em construção. Se a construção da lista L é feita dessa forma, então terá um custo variável em função de m, consumindo cerca de $O(mlogm)$ operações para ser realizada. Mesmo que essa hipótese não tenha sido comprovada no presente trabalho, os resultados experimentais obtidos mostraram que na prática essa diferença existe e fica mais evidente conforme o valor de m cresce assintoticamente.

Tais resultados, sejam na qualidade da solução, sejam no tempo de execução, destacam o método de refinamento iterativo apresentado como uma opção interessante para resolver o problema MBV diretamente ou como um sub-problema de problemas maiores. Cabe ressaltar que boa parte das instâncias testadas nos seis conjuntos analisados são artificiais, de forma que em futuros trabalhos o método deverá ser experimentado sobre instâncias reais, cujas redes possuam densidades de arcos e topologia diferentes das analisadas, assim como confrontar seus resultados com soluções ótimas resolvidas por métodos exatos aplicados sobre a formulação proposta por Cerulli et al. [2009].

7.2 Trabalhos Futuros

Essa seção apresenta propostas de trabalhos futuros que podem contribuir para a evolução e melhoria da abordagem de refinamento iterativo para o problema MBV. Estas incluem:

- Confrontar os resultados obtidos pelas heurísticas IR, EWS e NCH com o resultado exato obtido via resolução da formulação matemática de Cerulli et al. [2009] por um *solver*;

- Analisar outros métodos de seleção de arcos de corte e substituição para o algoritmo de refinamento;

- Propor novas medidas de infração dos arcos. Isso inclui modificar a medida σ_{ij} para expressar não a soma total do peso dos vértices em que o arco e_{ij} incide, mas sim o 'balanço' entre os graus dos vértices i e j de forma a priorizar a escolha de arcos de corte que sejam capazes de eliminar vértices *branch* mais rapidamente da árvore, reduzindo o esforço do algoritmo em número de iterações necessárias para melhorar a topologia da árvore;

- Experimentar partir o algoritmo de uma única solução do espaço de busca atribuindo pesos aos arcos e_{ij} do grafo G de maneira determinística, mas balizado pelo grau dos vértices i e j em que o arco incide;

- Implementar outros algoritmos para produzir soluções iniciais com menor quantidade de infrações para o algoritmo de refinamento iterativo. Uma das opções viáveis é implementar o algoritmo k-Prim de Narula e Ho [1980];

- Utilizar o algoritmo IR como parte de outras metaheurísticas pode resultar em formas poderosas de explorar o espaço de soluções, e portanto devem ser consideradas como melhorias no desenvolvimento do método e estudo do problema MBV. Uma possibilidade é utilizar esse algoritmo como *decoder* do algoritmo genético que emprega chaves aleatórias para problemas de otimização combinatória (Mendes et al. [2008]).

- Implementar o algoritmo IR como uma iteração de um método multi-partida (Martí [2003]) para o problema MBV, e obter a probabilidade desse algoritmo encontrar uma solução alvo em um dado intervalo de tempo através da análise de *time-to-target plots* (Aiex et al. [2006]).

Referências Bibliográficas

Abdalla, A.; Deo, N. e Gupta, P. (2000). Random-tree diameter and the diameter constrained MST. *Congressus Numerantium*, 144:161–182.

Ahuja, R. K.; Magnanti, T. L. e Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall.

Aiex, R. M.; Resende, M. G. C.; Celso e Ribeiro, C. (2006). Tttplots: A perl program to create time-to-target plots. *Optimization Letters*, 1:10–1007.

Bazaraa, M. S.; Sherali, H. D. e Jarvis, J. J. (1990). *Linear programming and network flows*. Wiley, New York, 2nd ed. edição.

Bazlamaçci, C. F. e Hindi, K. S. (2001). Minimum-weight spanning tree algorithms a survey and empirical study. *Comput. Oper. Res.*, 28:767–785.

Beasley, J. E. (1989). An SST-based algorithm for the Steiner problem in graphs. *Networks*, 19:1–16.

Bertsimas, D. e Tsitsiklis, J. (1997). *Introduction to Linear Optimization*. Athena Scientific, 1st edição.

Boldon, B.; Deo, N. e Kumar, N. (1995). Minimum-weight degree-constrained spanning tree problem: Heuristics and implementation on an SIMD parallel machine. Technical Report CS-TR-95-02, Department of Computer Science, University of Central Florida, Orlando, FL.

Bondy, J. A. e Murty, U. S. R. (1976). *Graph Theory with Applications*. Elsevier Science Ltd.

Cerulli, R.; Gentili, M. e Iossa, A. (2009). Bounded-degree spanning tree problems: Models and new algorithms. *Computational Optimization and Applications*, 42(3):353–370.

Cheriton, D. R. e Tarjan, R. E. (1976). Finding minimum spanning trees. *SIAM J. Comput.*, 5(4):724–742.

Cherkassky, B. V. e Goldberg, A. V. (1996). Negative-cycle detection algorithms. Technical Report 96-029, NEC Research Institute, Princeton, NJ.

Cooper, M. (2002). Advanced bash-scripting guide.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L. e Stein, C. (2001). *Introduction to Algorithms, second edition*. The MIT Press.

Deo, N. e Abdalla, A. (2000). Computing a diameter-constrained minimum spanning tree in parallel. In *Proceedings of the 4th Italian Conference on Algorithms and Complexity*, CIAC '00, pp. 17–31, London, UK. Springer-Verlag.

Deo, N. e Kumar, N. (1997). Computation of constrained spanning trees: A unified approach. *Network Optimization: Lecture Notes in Economics and Mathematical Systems*, 450:194–220.

Diestel, R. (2000). *Graph Theory (Graduate Texts in Mathematics)*. Springer.

Fredman, M. L. e Tarjan, R. E. (1987). Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM*, 34:596–615.

Garey, M. R. e Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York.

Gargano, L. e Hammar, M. (2003). There are spanning spiders in dense graphs (and we know how to find them). In *Proceedings of the 30th international conference on Automata, languages and programming*, ICALP'03, pp. 802–816, Berlin, Heidelberg. Springer-Verlag.

Gargano, L.; Hammar, M.; Hell, P.; Stacho, L. e Vaccaro, U. (2004). Spanning spiders and light-splitting switches. *Discrete Mathematics*, 285(1-3):83 – 95.

Gargano, L.; Hell, P.; Stacho, L. e Vaccaro, U. (2002). Spanning trees with bounded number of branch vertices. In *29th International Colloquium on Automata, Languages and Programming (ICALP)*, pp. 355–365.

Golumbic, M. C. e Hartman, I. B.-A. (2005). *Graph Theory, Combinatorics and Algorithms: Interdisciplinary Applications (Operations Research/Computer Science Interfaces Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Graham, R. L. e Hell, P. (1985). On the history of the minimum spanning tree problem. *IEEE Annals of the History of Computing*, 7(1):43–57.

Karger, D. R.; Klein, P. N. e Tarjan, R. E. (1995). A randomized linear-time algorithm to find minimum spanning trees. *J. ACM*, 42:321–328.

Klingman, D.; Napier, A. e Stutz, J. (1974). Netgen – A program for generating large scale (un)capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20:814–821.

Koutsofios, E.; North, S. C. e Gansner, E. R. (1991). Drawing graphs with dot - dot User's Manual. Technical Report 910904-59113-08TM, AT&T Bell Laboratories, Murray Hill, NJ.

Leighton, F. T. (1979). A graph colouring algorithm for large scheduling problems. *Journal of Research of the National Bureau of Standards*, 84(6):489–503.

Mao, L.-J.; Deo, N. e Lang, S.-D. (1997). A comparison of two parallel approximate algorithms for the degree-constrained minimum spanning tree problem. *Congressus Numerantium*, 123:15–32.

Martí, R. (2003). Multi-start methods. In Glover, F. e Kochenberger, G., editores, *Handbook of Metaheuristics*, pp. 354–368. Kluwer Academic Publishers.

Mendes, J. J. d. M.; Gonçalves, J. F. e Resende, M. G. C. (2008). A random key based genetic algorithm for the resource constrained project scheduling problem. *European Journal of Operational Research*, 189(3):1171–1190.

Narula, S. e Ho, C. (1980). Degree-constrained minimum spanning tree. In *Computers and Operations Research, vol 7*, pp. 239–249.

Noronha, T. F.; Santos, A. C. e Ribeiro, C. C. (2008). Constraint programming for the diameter constrained minimum spanning tree problem. *Electronic Notes in Discrete Mathematics*, 30:93–98.

R Development Core Team (2006). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.

Reinelt, G. (2008). TSPLIB 95. Disponível em http://www.iwr.uni-heidelberg.de/groups/comopt/software/tsplib95/.

Schildt, H. (1998). *C++: The Complete Reference*. Osborne/McGraw-Hill; 3rd edition.

Weidl, J. e Jazayeri, M. (1996). The standard template library tutorial 184.437 Wahlfachpraktikum (10.0).

Ziviani, N. (2004). *Projetos de Algoritmos com Implementações em PASCAL e C*. Editora Thomson.