Adriano Tavares de Freitas

Árvore de Subgradiente com Pré-Fase VNS-Lagrangeana para o Problema da Árvore Geradora Mínima com Restrição de Grau Máximo nos Vértices

Fortaleza – CE

Agosto/2011

Adriano Tavares de Freitas

Árvore de Subgradiente com Pré-Fase VNS-Lagrangeana para o Problema da Árvore Geradora Mínima com Restrição de Grau Máximo nos Vértices

Dissertação de mestrado apresentada como requisito para obtenção do título de mestre em Ciências da Computação pela Universidade Federal do Ceará.

Orientador:
Prof. Dr. Rafael Castro de Andrade

UNIVERSIDADE FEDERAL DO CEARÁ
DEPARTAMENTO DE COMPUTAÇÃO
MESTRADO E DOUTORADO EM CIÊNCIA DA COMPUTAÇÃO
PARGO - PARALELISMO, GRAFOS E OTIMIZAÇÃO COMBINATÓRIA

Fortaleza – CE

Agosto/2011

ÁRVORE DE SUBGRADIENTE COM PRÉ-FASE VNS-LAGRANGEANA PARA A ÁRVORE GERADORA COM RESTRIÇÃO DE GRAU MÁXIMO NOS VÉRTICES

Dissertação apresentada ao Curso de Mestrado em Ciência da Computação da Universidade Federal do Ceará, como parte dos Requisitos para a obtenção do Grau de Mestre em Ciência da Computação do aluno Adriano Tavares de Freitas.

Composição da Banca Examinadora:

Prof. Dr. Rafael Castro de Andrade (Presidente) (DEMA/UFC)

Prof. Dr. Celso da Cruz Carneiro Ribeiro (UFF- Rio)

Prof. Dr. Ricardo Cordeiro Corrêa (DC/UFC)

Prof. Dr. Manoel Bezerra Campelo Neto (DEMA/UFC

Aprovada em 19 de agosto de 2011

Agradecimentos

Primeiramente, gostaria de agradecer a Deus por tantas bençãos derramadas na minha vida e por ter colocado tantas pessoas maravilhosas no meu caminho. Verdadeiros anjos que sempre trouxeram conselhos, mensagens de ensino e momentos de alegria. Pessoas que foram de suma importância na minha vida pessoal e acadêmica: familiares, amigos e professores.

Em seguida, agradeço à minha família. Sou muito grato aos meus queridos e amados pais Pedro e Teresinha que sempre se esforçaram para me dar uma educação de qualidade, que sempre estiveram presentes me transmitindo seus valores e moldando meu caráter. Sou muito feliz por ter também meu amado irmão Anderson em minha caminhada. Com ele, aprendi características como companheirismo e amizade. Ele sempre me acompanhou e com ele compartilhei meus medos e desejos, meus sonhos e frustrações.

Expresso minha total gratidão ao professor e orientador Rafael Castro de Andrade que assumiu papel importantíssimo na minha formação acadêmica. Sempre acompanhando meus passos desde minha graduação. Formando-me o intelecto, despertando-me a curiosidade. Agradeço pela sua disposição e prontidão para me ouvir, retirar minhas dúvidas e me auxiliar nas diversas etapas que passei e ainda passo na minha trajetória nesta universidade.

Deixo meus sinceros agradecimentos aos demais participantes do Grupo de Pesquisa ParGO. O meu obrigado a todos os professores, alunos de iniciação científica, de mestrado, de doutorado e também a todos os colaboradores do grupo. Pessoas com as quais compartilhei muitas horas de laboratório e diversos seminários. Em especial, gostaria de agradecer aos professores Ricardo Corrêa e Manoel Campêlo pela participação nas bancas das etapas anteriores e pelos bons conselhos e sugestões.

Agradeço também aos amigos e companheiros de curso. Muitas das dúvidas foram sanadas graças a eles. Com eles, compartilhei muitos momentos de tensão, felicidade e decepção. Cito alguns poucos com a permissão da minha memória: Álinson, Rafael, Luis Henrique, Renan, Robson, Atílio, Jonathan, Marcos, Suelen, Karol, Arthur, Vinícius, Márcio, Phablo, Pablo, Max, Thiago, Emanuel, Nelson e Mayara.

Não poderia esquecer os meus amigos do grupo Nova Semente que muitas vezes me deram novo ânimo para continuar e prosseguir com minhas atividades. Eles são minha segunda família

e são deles os ouvidos que pacientemente me escutaram quando eu não encontrava ninguém para conversar. Eis seus nomes: Edilene, Eduardo, Eliandro e Vanessa, Mariana, Alisson, Iury, Natália, Deyse, Elias, Elane e Flaviane.

Meus agradecimentos aos membros da banca, os professores Celso Ribeiro, Manoel Campêlo, Rafael Castro e Ricardo Corrêa. Obrigado pela sua disposição e dedicação do seu tempo na leitura e avaliação deste trabalho. Obrigado também por seu papel desempenhado na sociedade, na pesquisa e na formação de pessoas.

Por último, mas não menos importante, gostaria de agradecer ao CNPq por apoiar este projeto de pesquisa através do financiamento da bolsa de estudos. Aproveito também para a agradecer a todo o povo brasileiro que indiretamente ajudaram na minha formação acadêmica.

Resumo

O problema das árvores geradoras mínimas com restrição de grau (*AGMRG*) consiste em encontrar uma árvore geradora mínima num grafo que respeite um limite superior do número de arestas incidentes em cada vértice da árvore. O problema é NP-Difícil e tem aplicações práticas no projeto de redes (de computadores ou de telecomunicações) com nós capacitados.

A heurística Lagrangeana (Andrade et al. 2006) utiliza a informação dual obtida em cada iteração do método do subgradiente para obter soluções viáveis (limites superiores) para o problema *AGMRG*. Incorporamos um mecanismo de fuga de ótimos locais nessa heurística: o método de busca em vizinhança variável (*VNS* do inglês Variable Neighborhood Search). Utilizamos essa nova heurística como pré-fase de um método de resolução exato que chamamos de árvore de subgradiente (*ASG*). É um método de divisão e conquista a exemplo de um algoritmo *Branch and Bound* (*B&B*). Para cada nó da *ASG*, utilizamos os multiplicadores de Lagrange e a fixação das variáveis do processo de ramificação para obter uma solução Lagrangeana para o subproblema do nó em questão.

No presente trabalho, apresentamos uma adaptação do método de descida em vizinhança variável (*VND* do inglês Variable Neighborhood Descent) aplicado ao problema *AGMRG* (Ribeiro e Souza 2002). O objetivo é permitir que essa técnica de busca local possa explorar elementos nas vizinhanças de algumas instâncias Hamiltonianas. Mostramos também um novo esquema de ramificação para o método exato que permite excluir a solução obtida no nó pai, diferentemente do que pode acontecer num esquema clássico de ramificação por dicotomia (aresta fixada em 0 ou 1) de soluções inteiras 0-1 na relaxação do nó.

Alguns grupos de instâncias são utilizados nos experimentos computacionais. No primeiro deles (instâncias Euclidianas), conseguimos melhorar os resultados e fechamos os intervalos de erro na maioria das instâncias. Quanto ao segundo grupo (instâncias Hamiltonianas), nosso algoritmo é bastante competitivo com os melhores resultados presentes na literatura. Inclusive fechando os intervalos de erro para algumas instâncias. Dois outros grupos também são utilizados nos experimentos: as instâncias DE e DR.

PALAVRAS-CHAVE: Árvore Geradora, VNS, VND, Árvore de Subgradiente, Heurística VNS-Lagrangeana.

Abstract

The degree constrained minimum spanning tree (*DCMST*) problem consists of finding a minimum spanning tree T in a graph. The tree T must satisfy, for each vertex v, an upper limit on the number of edges incident to v. The problem is NP-Hard and it has practical applications in (computer or telecommunications) network design.

The Lagrangian heuristic (Andrade et al. 2006) uses the dual information obtained in each iteration of the subgradient method to obtain feasible solutions (upper bounds) to the *DCMST* problem. We introduce a mechanism to overcome local optima into this heuristic: the variable neighborhood search (*VNS*) method. We use this new heuristic as a pre-phase for an exact resolution method called subgradient tree (*SGT*). It is a divide-and-conquer method such as a Branch and Bound (*B&B*) algorithm. For each node p of the subgradient tree, we use Lagrange multipliers and the variable fixing of the branching process to obtain a Lagrangian solution for the subproblem of the node p.

In this work, we present an adaptation of the variable neighborhood descent (*VND*) method applied to the *DCMST* problem (Ribeiro and Souza 2002). The goal is to allow the local search technique to explore elements in the neighborhoods of some Hamiltonian instances. We also show a new branching scheme for the exact method that excludes the solution obtained in the father node. This may not happen in a classic scheme of branching by dichotomy (where the edge is fixed in 0 or 1) when relaxed node solutions are integer 0-1.

Some groups of benchmark instances were used in the computational experiments. In the first one (Euclidean instances), we improve the upper bounds and close the gap for most instances. For the second group (Hamiltonian instances), our algorithm is very competitive with the best results in the literature. We also close some gaps for these instances. Two others groups of instances are also used in the experiments: the DE instances and the DR instances.

KEYWORDS: Spanning Tree, VNS, VND, Subgradient Tree, VNS-Lagrangian Heuristic.

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Sumário

1 Introdução

O problema da árvore geradora mínima com restrição de grau (máximo) nos vértices, *AGMRG*, possui como objetivo encontrar uma árvore geradora de menor custo em um dado grafo em que a cada vértice associamos um limite superior para o número máximo de arestas que podem incidir nesse vértice. O problema *AGMRG* é NP-Difícil [10] e tem aplicações práticas no projeto de redes de computadores, de telecomunicações e de transporte.

Neste trabalho, desenvolvemos algoritmos heurísticos e exatos para o problema *AGMRG*. Mais precisamente, incorporamos técnicas de busca em vizinhança variável (*VNS* do inglês *Variable Neighborhood Search*) na heurística Lagrangeana de [4], já que a mesma não dispõe de mecanismos de fuga de ótimos locais. Essa nova heurística, que pode ser classificada como sendo do tipo *VNS-Lagrangeana*, é usada como pré-fase em um método exato, denominado de *Árvore de Subgradiente* (*ASG*), conceito introduzido em [2] e cuja eficiência é comprovada em [3] para o problema *AGMRG*. Trata-se de um método de divisão e conquista, inspirado no método *Branch and Bound* (*B&B*) tradicional, no qual determina-se uma sequência de multiplicadores de Lagrange e uma solução Lagrangeana para cada nó da árvore de busca. Os multiplicadores de um nó são atualizados utilizando os multiplicadores de seu nó pai, se houver, e sua solução Lagrangeana, além da fixação de variáveis no processo de ramificação do método. Tais multiplicadores são empregados para perturbar os custos originais das arestas. O custo da árvore geradora mínima da solução Lagrangeana dos nós pode ser usado para obter um limite inferior na solução do problema original. O objetivo da *ASG* é melhorar as cotas inferiores da solução do problema enquanto determina soluções viáveis de boa qualidade para o mesmo. Propomos, para o método da *ASG*, um novo esquema de ramificação de nós, extensível a métodos exatos (tipo B&B) em que a relaxação do problema nos nós é inteira. Esse esquema permite encontrar soluções viáveis para o problema, gerando um número menor de nós na árvore de busca. A inovação dessa técnica de ramificação reside no fato de a partição do espaço de soluções viáveis excluir a solução ótima relaxada do nó ramificado, o que pode não acontecer num esquema clássico de ramificação por dicotomia (fixar uma aresta em 0 ou em 1), já que as soluções relaxadas dos nós devem ser árvores (solução binária).

Quanto à pré-fase heurística, para contornar a dificuldade de diversificação apresentada pela heurística Lagrangeana de Andrade et al. [4], usamos a técnica *VNS* (utilizada com êxito em Souza e Martins [29]), que explora o espaço de soluções de um determinado problema verificando as regiões de vizinhança de uma certa solução, procurando melhorá-la. Caso não encontre melhora, progressivamente explora espaços de vizinhança maiores. Nosso algoritmo procurará iterativamente, a partir de uma certa solução, remover uma quantidade k de arestas (com k variando entre 1 e um dado valor máximo), inserir outras k arestas para recompor a árvore e, em seguida, aplicar uma busca local a partir dessa nova árvore. O objetivo é diversificar a busca, fugindo de mínimos locais. Para tanto, sugerimos uma adaptação da estrutura de vizinhança encontrada em Ribeiro e Souza [27] para o problema *AGMRG*.

Eventualmente, o método aqui proposto pode ser acrescido de estratégias que possam melhorar a qualidade dos limites inferiores e superiores na solução do problema. Um exemplo é o fortalecimento da relaxação Lagrangeana do problema *AGMRG* com as desigualdades de Blossom como é mostrado em Cunha e Lucena [8].

O problema da *AGMRG* já possui mais de três décadas de estudo na literatura. Em 1979, Garey e Johnson [10] mostram que o problema é NP-Difícil. Basta considerar um caso particular do problema da *AGMRG*: a árvore geradora mínima com restrição de grau k nos vértices (*AGMRG-k*). No problema da *AGMRG-k*, a restrição de grau k é imposta para todos os vértices do grafo. Logo, se $k = 2$, o problema se reduz a encontrar um caminho Hamiltoniano de custo mínimo no grafo.

As primeiras estratégias para solucionar o problema da *AGMRG* são propostas por Narula e Ho [24] em 1980. Elas consistem em um algoritmo B&B (baseado no método de Held e Karp [15, 16] para o problema do caixeiro viajante) e algoritmos gulosos primais e duais. Entre tais algoritmos, uma heurística construtiva para obter soluções viáveis é sugerida. Trata-se de uma adaptação do algoritmo de Prim [25] permitindo lidar com as restrições de grau. Em 1982, Gavish [11] sugere a utilização da relaxação Lagrangeana e, por meio de otimização por subgradiente, encontra limites inferiores para o problema. No ano de 1985, Savelsbergh e Volgenant [28] propõem um algoritmo B&B com novas regras de ramificação e um esquema de troca de arestas. Em 1989, Volgenant [30] integra as estratégias de troca de arestas de [28] a um esquema de relaxação Lagrangeana. Além disso, ele melhora o B&B de [28] com um método dual ascendente para obter limites inferiores Lagrangeanos para o problema. Em 1995, Boldon et al. [5] introduzem um algoritmo dual simplex baseado no algoritmo de Prim e também apresentam um método heurístico baseado na técnica de arrefecimento simulado. No ano seguinte, Craig et al. [7] propõem um algoritmo B&B e diversas heurísticas baseadas em redes neurais, arrefecimento simulado, algoritmos genéticos e mecanismos de busca no espaço de

soluções do problema. Em 1997, Zhou e Gen [31] sugerem um algoritmo genético que utiliza o número de Prüfer para codificar uma árvore. Em 2000, Knowles e Corne [19] apresentam uma estratégia aleatória primal para o problema e a integram às técnicas de arrefecimento simulado, *multi-start hill-climbing* e algoritmos genéticos. Krishnamoorthy et al. [20] comparam computacionalmente as estratégias sugeridas em [7] em 2001. Um gerador de instâncias é proposto e algumas das instâncias utilizadas, conhecidas como grupo shrd com até 30 vértices, tornam-se padrão para testes de referência e desempenho. Algoritmos genéticos também são apresentados por Raidl e Julstron [26] e Li [22].

Caccetta e Hill [6] sugerem um algoritmo *branch-and-cut* onde as restrições de eliminação de sub-rotas são geradas iterativamente, na medida que elas se tornam violadas na relaxação linear do problema. O algoritmo também envolve uma eficiente fase de pré-processamento baseada em relaxação Lagrangeana e uma heurística *multi-start* Lagrangeana. Ribeiro e Souza [27], em 2002, apresentam um procedimento bastante eficaz baseado na metaheurística de busca em vizinhança variável (*VNS*). Eles definem também 3 estruturas de vizinhança que são utilizadas na descida em vizinhança variável (*VND* do inglês Variable Neighborhood Descent), método utilizado na fase de busca local do *VNS*. Em 2006, Andrade et al. [4] utilizam a informação dual Lagrangeana para perturbar os custos das arestas do grafo original e, a partir daí, utilizando uma heurística gulosa baseada no algoritmo de Kruskal, encontrar soluções viáveis para o problema. Um gerador de instâncias também é proposto e novas instâncias são apresentadas. O primeiro grupo delas são chamadas de instâncias Euclidianas e consistem de grafos com até 2000 vértices com restrições de grau entre 1 e 4. As instâncias Hamiltonianas constituem o segundo grupo. Tratam-se de instâncias com até 500 vértices com restrições de grau entre 1 e 2 (no máximo dois vértices apresentam restrição de grau igual a 1). Cunha e Lucena [8] propõem, em 2007, um esquema onde os limites inferiores são obtidos a partir de um algoritmo *non-delayed relax and cut* Lagrangeano que utiliza as desigualdades de Blossom, enquanto os limites superiores são obtidos por meio de uma heurística Lagrangeana multi-start. No mesmo ano, Souza e Martins [29] apresentam variações do método VNS proposto em [27]. Estratégias baseadas no algoritmo de Segunda Ordem [18] são apresentadas e utilizadas na fase de agitação do *VNS*, e funções que expressam diferenças estruturais das árvores são consideradas no processo de atualização da solução candidata. Tais procedimentos foram bastante eficientes na melhoria dos limites primais para as instâncias Euclidianas e Hamiltonianas. Devido a esses bons resultados, adaptamos e incorporamos neste trabalho algumas das estratégias apresentadas em [27, 29], permitindo assim, obter melhores limites superiores na pré-fase da *ASG*, método que também é melhorado com um novo esquema de ramificação dos nós em sua árvore de busca.

O restante deste manuscrito é organizado da seguinte maneira. No Capítulo 2, tratamos da

formulação do problema. No Capítulo 3, descrevemos os métodos de resolução estudados e propostos: os procedimentos de busca local para melhorar as soluções viáveis do problema, os mecanismos que permitem a fuga de ótimos locais, o método do subgradiente, a heurística *VNS-Lagrangeana*, a árvore de subgradiente (*ASG*) e o novo esquema de ramificação. No Capítulo 4, mostramos nossos resultados e comparamos com outros resultados da literatura. Por fim, concluímos com um breve resumo dos trabalhos que já realizamos e o que ainda pode ser feito para melhorar o método da *ASG* e aplicação em outros problemas.

2 *Formulação do Problema AGMRG*

2.1 Árvore Geradora Mínima

Sejam $G = (V, E)$ um grafo e $c : E \to \mathbb{R}_+$ uma função que atribui custos às arestas de G.

Uma ***Árvore Geradora Mínima*** é um subgrafo gerador do grafo G que é uma árvore e cujo custo é igual ou menor ao custo de qualquer outro subgrafo gerador de G.

O problema de encontrar uma árvore geradora mínima em um grafo admite algoritmos de tempo polinomial, tais como os que são encontrados em Prim [25] e Kruskal [21]. Ambos utilizam estratégias gulosas.

Na Figura 2.1, os valores próximos às arestas representam seus custos. Temos um grafo G à esquerda e sua árvore geradora mínima T à direita, composta pelas três arestas de menor custo.

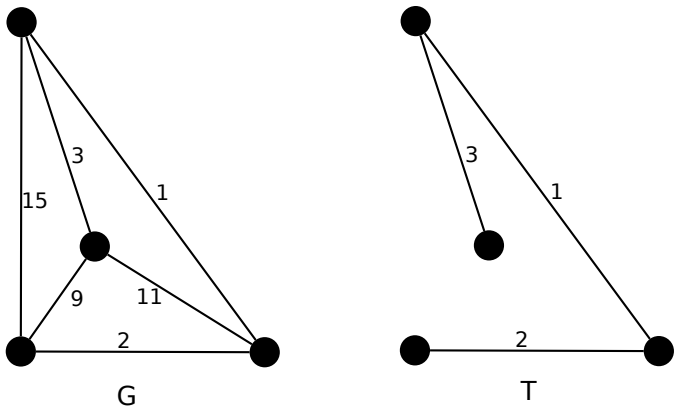

Figura 2.1: Árvore Geradora Mínima T do grafo G.

Uma formulação matemática para esse problema é dada por

$$
\begin{aligned}
(P_0) \quad \min \quad & \sum_{e \in E} c_e x_e && (2.1)\\
s.a. \quad & \sum_{e \in E} x_e = \mid V \mid -1, && (2.2)\\
& \sum_{e \in E(S)} x_e \leq \mid S \mid -1, \;\; S \subset V, && (2.3)\\
& x_e \in \{0,1\}, \quad e \in E. && (2.4)
\end{aligned}
$$

Para cada aresta $e \in E$, c_e representa o valor da função de custo aplicada à aresta e. A uma aresta e, associamos uma variável binária x_e. O valor de x_e é 1 se a aresta estiver presente na árvore geradora de G, caso contrário, é 0. Na formulação acima, $E(S) \subseteq E$, quando $S \subseteq V$, denota o conjunto de arestas de G com ambas extremidades em S.

A restrição (2.2) garante o número de arestas para formar a árvore, enquanto as restrições (2.3) garantem a inexistência de ciclos.

2.2 Árvore Geradora Mínima com Restrição de Grau Máximo

Considere dados, além do grafo G e da função de custo c, um valor $d_v \in \mathbb{Z}_+^*$ para cada $v \in V$, indicando o máximo grau permitido a v em uma árvore geradora de G.

O problema da ***Árvore Geradora Mínima com Restrição de Grau nos Vértices*** (*AGMRG*) consiste em encontrar, se houver, uma árvore geradora T com custo mínimo (soma dos custos de suas arestas) que satisfaça as restrições de grau, ou seja, com cada um de seus vértices v apresentando grau, em T, inferior ou igual a d_v.

Como já foi dito no capítulo anterior, o problema é NP-Difícil. A redução é feita a partir do problema do caminho Hamiltoniano [10].

A Figura 2.2 mostra um grafo G (à esquerda) e sua árvore geradora mínima T com restrição de grau nos vértices (à direita). Os valores próximos aos vértices representam suas restrições de grau máximo e os valores próximos às arestas, seus custos.

A seguir, mostramos uma formulação matemática para o problema:

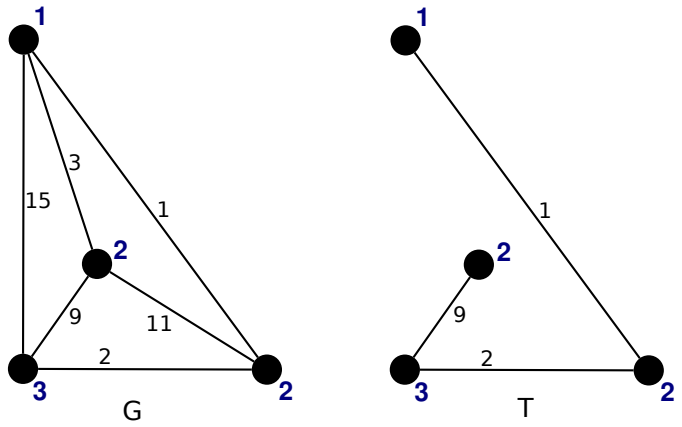

Figura 2.2: AGMRG T do grafo G

$$
\begin{aligned}
(P) \quad \min \quad & \sum_{e \in E} c_e x_e \\
s.a. \quad & (2.2), (2.3) \; e \; (2.4), \\
& \sum_{e \in \delta(v)} x_e \leq d_v, v \in V.
\end{aligned} \tag{2.5}
$$

em que $\delta(v) \subseteq E$, para $v \in V$, representa o conjunto de arestas que possuem v como uma de suas extremidades. As inequações (2.5) forçam que as restrições de grau não sejam violadas.

Para obter cotas inferiores para a solução do problema, associamos multiplicadores de Lagrange $\lambda \in \mathbb{R}_+^{|V|}$ às restrições (2.5) e as levamos à função objetivo de (P). Temos assim a relaxação Lagrangeana do problema *AGMRG*:

$$
\begin{aligned}
(P_\lambda) \quad \min \quad & \sum_{e=(i,j) \in E} (c_e + \lambda_i + \lambda_j) x_e - \sum_{i \in V} \lambda_i d_i \\
s.a. \quad & (2.2), (2.3) \; e \; (2.4).
\end{aligned} \tag{2.6}
$$

Vale ressaltar que o problema acima é de árvore geradora mínima, mencionado na Seção 2.1, com uma nova função objetivo. Seja z_λ o valor da solução ótima em (P_λ), para um dado $\lambda \geq 0$. Note que z_λ fornece um limite inferior para o problema (P) [4] . Logo, o melhor limite inferior pode ser dado pela resolução do dual Lagrangeano:

$$(D) \quad \max \quad z_\lambda \qquad (2.7)$$
$$s.a. \quad \lambda \geq 0.$$

3 *Métodos de Resolução*

Neste capítulo, apresentamos os métodos de resolução estudados. Inicialmente, mostramos o algoritmo utilizado para gerar soluções viáveis para o problema *AGMRG*. Em seguida, abordamos os métodos de busca local que podem ser aplicados juntamente com os mecanismos de fuga de ótimos locais (abordados posteriormente) e na heurística VNS-Lagrangeana, durante as iterações do método do subgradiente. Em seguida, mostramos como integramos o método *Skewed VNS* [29] à heurística Lagrangeana de Andrade et al. [4]. A árvore de subgradiente (*ASG*) [2, 3] e seu novo esquema de ramificação também são abordados nesse capítulo.

3.1 Construindo Árvores Geradoras com Restrição de Grau

Andrade et al. [4] apresentam um algoritmo para gerar árvores geradoras que respeitam as restrições de grau. A exemplo do algoritmo de Kruskal, a operação básica é a utilização da aresta de menor custo para conectar duas componentes de uma floresta. No entanto, algumas estratégias complementares são incorporadas para reduzir as chances de gerar uma árvore inviável ao final do algoritmo.

Considere um grafo $G = (V,E)$. Sejam $T_1 = (V_1,E_1)$, com $V_1 \subseteq V$ e $E_1 \subseteq E$, uma árvore contida em G e $x \in \mathbb{B}^{|E|}$ o vetor de incidência correspondente, tal que $x_e = 1$ se e somente se $x_e \in E_1$. Assim, o grau de T_1 é definido como

$$\sigma(T_1) = \sum_{i \in V_1} \sum_{e \in \delta(i)} x_e \tag{3.1}$$

enquanto a capacidade máxima de grau de T_1 é definida como:

$$d(T_1) = \sum_{i \in V_1} d_i. \tag{3.2}$$

Se $\sigma(T_1) = d(T_1)$, dizemos que a árvore T_1 é *saturada*. Veja que se T_1 é saturada, então os vértices em V_1 devem ser saturados (apresentam grau em T_1 igual ao valor de sua restrição de

grau) e não é possível, a partir de T_1, encontrar uma árvore geradora que respeita as restrições de grau. Se T_1 é uma árvore geradora de G que respeite as restrições de grau e $T_2 = (V_2, E_2)$ é uma sub-árvore própria de T_1, então T_2 não pode ser saturada e $\sigma(T_2) < d(T_2)$.

Sejam $T_1 = (V_1, E_1)$ e $T_2 = (V_2, E_2)$ duas árvores disjuntas em vértices contidas em G. Suponha que T_1 e T_2 não sejam saturadas e que exista uma aresta $e \in E$ de maneira tal que $T_3 = (V_1 \cup V_2, E_1 \cup E_2 \cup e)$ seja uma árvore não saturada de G. Temos então que

$$\sigma(T_3) = \sigma(T_1) + \sigma(T_2) + 2 < d(T_3). \tag{3.3}$$

A Desigualdade (3.3) é válida desde que T_3 é não saturada e pelo fato de que ao se conectar T_1 a T_2 por meio da aresta $e = (i, j) \in E$ aumentamos em uma unidade o grau de i e o grau de j.

O Algoritmo *Kruskal Modificado* (Algoritmo 1) toma por base esse fato mencionado anteriormente para determinar uma floresta que satisfaz as restrições de grau máximo. Ele é iniciado com uma floresta formada por $|V|$ componentes disjuntas, cada uma delas contendo um dos vértices de V. Em seguida, a cada iteração, executamos operações para unir componentes. A desigualdade (3.3) deve ser satisfeita para cada operação de união executada, com exceção do caso da inclusão na $(n-1)$-ésima aresta, caso ocorra. Uma árvore viável é retornada se o grafo G é completo. Se o subgrafo retornado for conexo, então ele é uma solução viável do problema *AGMRG* (o que é tanto mais difícil de ocorrer quanto o grafo for esparso). Uma alternativa para garantir que uma solução viável seja encontrada é trabalhar com arestas artificiais com custo alto. Geraríamos uma *solução Kruskal incompleta* e em seguida aplicaríamos o procedimento de melhora mostrado na Seção 3.2.3.

O parâmetro k^* mostrado no Algoritmo 1 é utilizado para definir o problema *AGMRG* reduzido. Trata-se de um subconjunto $E' \subseteq E$. Considere a ordem das arestas $\{e_1, e_2, ..., e_m\}$ utilizada no algoritmo *Kruskal Modificado* e seja k^* o índice da última aresta inserida na solução inicial gerada. O conjunto de arestas $E' = \{e_1, e_2, ..., e_{m^*}\}$ é então definido, onde $m^* = \text{Min}\{k^* + \lfloor perc \cdot k^* \rfloor, m\}$, para um dado percentual $perc$ das arestas utilizadas para gerar a solução inicial. Depois de alguns experimentos, utilizamos $perc = 10\%$. Veja que todas as arestas na lista de arestas original do grafo G até o índice m^* são utilizadas para definir o problema reduzido, tanto as que foram inseridas na solução inicial como as que não foram.

3.2 Métodos de Busca Local

Seja x uma solução viável para um certo problema e seja $\bar{c}$ o custo dessa solução. Técnicas de busca local consistem na verificação de uma vizinhança de x, analisando os custos de

Algoritmo 1: Pseudo-Código: Kruskal_Modificado (Andrade et al. [4])

Entrada: Grafo G, vetor c com os custos das arestas e vetor d com as restrições de grau
Resultado: *AGMRG* $T^1 = (V, E^1)$ e k^*

```
1  Ordene as arestas {e_1, e_2, ..., e_m} em ordem não decrescente dos seus custos.;
2  d̄_i ← 0, ∀i ∈ V;
3  T^1 ← (V, E^1), onde E^1 = ∅;
4  k ← 1;
5  enquanto (| E^1 | < | V | −1) faça
6      Seja e_k = (i, j);
7      se (d̄_i < d_i) e (d̄_j < d_j) então
8          T^2 ← (V, E^1 ∪ {e_k});
9          se (T^2 é acíclico) então
10             se (| E^1 ∪ {e_k} | = | V | −1) então
11                 E^1 ← E^1 ∪ {e_k};
12             senão
13                 se (as componentes em E^1 ∪ {e_k} são não saturadas) então
14                     E^1 ← E^1 ∪ {e_k};
15                     d̄_i ← d̄_i + 1;
16                     d̄_j ← d̄_j + 1;
17     k ← k + 1;
18 k* ← k − 1;
```

suas soluções vizinhas. Na esperança de eventualmente melhorar o valor da função objetivo (minimização ou maximização) ao encontrar uma solução x' com custo $\bar{c}'$ melhor (inferior ou superior) ao custo $\bar{c}$ de x.

As técnicas de busca local exigem a definição de um parâmetro máximo k_{max} de estruturas de vizinhança de uma dada solução x. Denotamos por $N_k(x), k = 1, ..., k_{max}$, o conjunto de soluções viáveis vizinhas a x que possuem $m - k$ das arestas presentes também em x (onde m é o número de arestas da solução x). Algumas heurísticas de busca local utilizam apenas $N_1(x)$, ou seja, definem apenas uma estrutura de vizinhança. É o que acontece em heurísticas como *1-Opt* (utilizada para melhorar as cotas superiores em [4]) e *2-Opt* [6], ambas casos especiais do método *k-Opt*. Em outras situações, são definidas mais de uma estrutura de vizinhança, como ocorre no *VND_DCMST* em [27].

Utilizamos algumas dessas técnicas de busca local nos mecanismos de fuga de ótimos locais mostrados na Seção 3.3. A seguir, abordamos as estratégias de busca local empregadas neste trabalho.

3.2.1 1-Opt

O *1-Opt* é um caso especial do *K-Opt*. Apenas a vizinhança $N_1(T)$ é definida da seguinte forma: dada uma solução viável T, $N_1(T)$ é composto das soluções viáveis T' que possuem $m-1$ arestas em comum com T.

A Figura 3.1 exemplifica elementos da vizinhança $N_1(x)$. A árvore T^2 é uma árvore viável que possui 2 arestas em comum com T^1, logo $T^2 \in N_1(T^1)$. Também temos que $T^3 \in N_1(T^2)$.

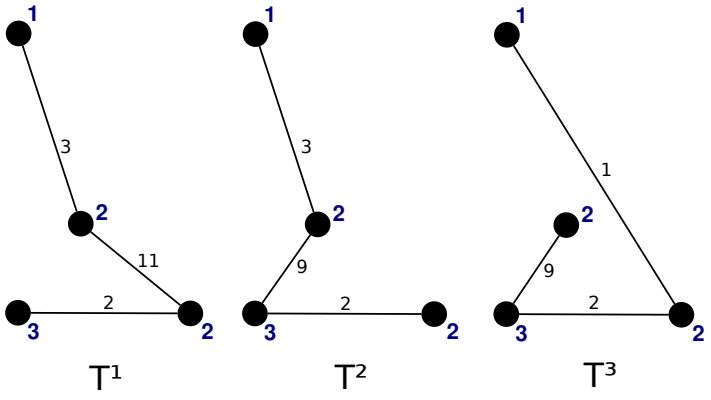

Figura 3.1: Elementos da vizinhança N_1 de T^1 e de T^2

Como o número de árvores viáveis em $N_1(T)$ é da ordem de $O(n \cdot m)$, onde n é o número de vértices e m o de arestas do grafo, não é um processo oneroso analisar essa vizinhança em busca de uma árvore com custo inferior ao custo de T. Tal processo é descrito no Algoritmo 2.

Algoritmo 2: Pseudo-Código: 1-Opt

Entrada: *AGMRG T*
Resultado: melhor *AGMRG* T^* na vizinhaça $N_1(T)$

```
1 c(T*) ← ∞; // custo de T*
2 para cada aresta e ∈ E(T) faça
3     T' ← T \ {e};
4     para cada aresta e' ∉ E(T) faça
5         T'' ← T' ∪ {e'};
6         se T'' é viável então
7             se c(T'') ≤ c(T*) então
8                 T* ← T'';
9 retorna T*;
```

Inspirado na temática do *1-Opt*, Andrade et al. [4] propuseram um *procedimento de melhora* (descrito posteriormente) que age em conjunto com a heurística Lagrangeana e é utilizado para auxiliar na busca de soluções viáveis, melhorando assim as cotas superiores do problema.

3.2.2 K-Opt

O *K-Opt* generaliza o procedimento descrito na seção anterior. O método, a priori, foi desenvolvido para o problema do caixeiro viajante. Nesse contexto, k arestas são retiradas do percurso. Em seguida, tentamos reconectar as componentes conexas geradas de maneira a diminuir o custo total do percurso. No problema do caixeiro viajante, podemos enxergar uma solução viável como um caminho, que se torna desconexo se dele são retiradas k arestas. Veja que cada componente conexa obtida também é um caminho e como queremos reconstruir uma solução viável, trabalhamos apenas com as extremidades de cada componente durante o processo de reconstrução. Quando mudamos de perspectiva e consideramos árvores, e não apenas caminhos, temos um número maior de possibilidades na reconexão das componentes, pois podemos trabalhar com as arestas incidentes em quaisquer vértices, satisfazendo às restrições de grau.

Se T é uma solução viável, $N_k(T)$ representa o conjunto de todas as árvores viáveis que possuem $m-k$ arestas em comum com a árvore T. O método consiste em explorar a vizinhança N_k, procurando encontrar uma árvore T' com custo $c(T')$ inferior ao de $c(T)$.

3.2.3 Procedimento de Melhora

Em Andrade et al. [4], um procedimento heurístico de melhora baseado nas estruturas de vizinhança do *1-Opt* (mostrado na Seção 3.2.1) é apresentado. Seja uma árvore viável T, passada como entrada do procedimento. Exploramos estruturas de vizinhança $N_1^e(T)$ que consistem de todas as árvores viáveis que possuem $m-1$ arestas em comum com T. A aresta pré-selecionada e (que pertence ao conjunto de arestas $\hat{E}$ da árvore original T) não está presente em nenhuma árvore viável do conjunto $N_1^e(T)$.

O número de regiões de vizinhança exploradas é, portanto, $n-1$. Basicamente, para cada aresta $e \in \hat{E}$, retiramos e da árvore T e procuramos reconectar as componentes T_1 e T_2 geradas de maneira que as restrições de grau não sejam violadas. Então, selecionamos a aresta e' que reconecta T_1 e T_2 com menor custo $c_{e'}$, gerando uma árvore T^*. Se o custo de T^* for menor que o custo de T, atualizamos $T \leftarrow T^*$. Veja, porém, que a lista $\hat{E}$ continua inalterada, garantindo assim que o procedimento termine em $n-1$ iterações.

Esse procedimento de melhora pode ser aplicado na heurística Lagrangeana quando encontramos uma árvore viável durante a busca por soluções viáveis para o problema *AGMRG* reduzido.

O Algoritmo 3 mostra como realizar tal busca nas vizinhanças N_1^e de uma *AGMRG* T.

Algoritmo 3: Pseudo-Código: Procedimento de Melhora

Entrada: *AGMRG T*
Resultado: *AGMRG T*

```
1  Ê ← E(T);
2  para cada aresta e ∈ Ê faça
3      {T1, T2} ← T \ {e}; // obtemos duas sub-árvores T1 e T2
4      MENORCUSTO ← ∞;
5      para cada aresta e' ∉ E(T1) ∪ E(T2) que conecta T1 e T2 sem violar as restrições de grau faça
6          se c_e' < MENORCUSTO então
7              T* ← T1 ∪ T2 ∪ {e'};
8              MENORCUSTO ← c_e';
9      se c(T*) ≤ c(T) então
10         T ← T*;
11 retorna T;
```

3.2.4 VND

O método de *Descida em Vizinhança Variável (VND)* é uma estratégia de busca local que é caracterizada por procurar efetuar melhoras sucessivas a partir de uma solução viável inicial. Caso nenhuma melhora seja encontrada em uma certa estrutura de vizinhança, variamos a vizinhança corrente de acordo com uma ordem predeterminada. Essa técnica é usada na etapa de busca local do *Skewed VNS* de Souza e Martins [29] como podemos ver na Seção 3.3.2.

Em [27], Ribeiro e Souza definem estruturas de vizinhança e mostram o *VND* aplicado ao problema da *AGMRG*, o *VND_DCMST*. Dada uma árvore viável T de um grafo $G = (V, E)$, um vértice v está *saturado* quando o número de arestas incidentes a v em T é igual ao valor de sua restrição de grau d_v. Quando o grau de v em T for superior à sua restrição de grau, dizemos que v está violado. Dessa maneira, classificamos as arestas que não estão em T em três grupos:

- E_0: arestas com ambas as extremidades não saturadas.
- E_1: arestas com exatamente uma extremidade saturada.
- E_2: arestas com ambas as extremidades saturadas.

De maneira que $E(G) = E(T) \cup E_0 \cup E_1 \cup E_2$. Uma operação de troca de arestas é definida por um par $(e, \bar{e})$ de arestas, onde $e \in E(T)$ e $\bar{e} \in E \setminus E(T)$, tal que o grafo $T' = (V, (E(T) \cup \{\bar{e}\}) \setminus \{e\})$ ainda é uma árvore geradora de G. Definimos as estruturas de vizinhança:

N_1^{vnd} : definida para todas as operações de troca $(e, \bar{e})$ que preservam a viabilidade da nova

árvore. Se $\bar{e} \in E_0$, então a aresta e a ser removida é aquela de maior custo no ciclo gerado pela inserção de $\bar{e}$. Se $\bar{e} \in E_1$, e é a aresta do ciclo adjacente ao vértice violado.

N_2^{vnd} : definida por sequências de duas operações de troca: $(e_1, \bar{e}_1)$ e $(e_2, \bar{e}_2)$. A primeira troca, geralmente, torna a árvore inviável, enquanto a segunda restaura a viabilidade. Se $\bar{e}_1 \in E_1$, então a aresta e_1 a ser removida é aquela de maior custo no ciclo gerado. Se $\bar{e}_1 \in E_2$, então e_1 é a aresta de maior custo no ciclo gerado entre aquelas que incidem nas extremidades de $\bar{e}_1$. Ao final da primeira troca, um vértice p pode ainda estar violado. Portanto, uma aresta e_2 incidente em p será removida e uma aresta $\bar{e}_2 \in E_0 \cup E_1$ é inserida (obedecendo às restrições de grau) para reconectar as duas componentes geradas pela remoção de e_2.

N_3^{vnd} : definida por sequências de três operações de troca: $(e_1, \bar{e}_1)$,$(e_2, \bar{e}_2)$ e $(e_3, \bar{e}_3)$. A primeira troca leva a uma solução com um ou dois vértices violados, o segundo movimento restaura a viabilidade de um deles e com o terceiro restabelecemos a viabilidade total da árvore. Nesse caso, a primeira aresta inserida $\bar{e}_1$ pertence a E_2, e_1 é a aresta de maior custo no ciclo gerado pela inserção de $\bar{e}_1$. Após a troca, temos, no mínimo, um vértice p violado (a inserção de $\bar{e}_1$ viola dois vértices e a retirada de e_1 pode restaurar um deles). A segunda troca remove uma aresta e_2 incidente em p e uma aresta $\bar{e}_2 \in E_0 \cup E_1$ é inserida para reconectar as duas componentes, respeitando as restrições de grau. Todos os pares $(e_2, \bar{e}_2)$ possíveis são considerados nessa segunda troca. Se ainda existe algum vértice q violado, realizamos o terceiro movimento, que na árvore corrente, remove uma aresta e_3 incidente a q e adiciona $\bar{e}_3 \in E_0 \cup E_1$ para reconectar as componentes que foram geradas. Também verificamos todos os pares $(e_3, \bar{e}_3)$ possíveis nessa terceira troca, sempre respeitando as restrições de grau.

Algoritmo 4: Pseudo-Código: VND_DCMST

Entrada: *AGMRG T*
Resultado: *AGMRG T' com custo menor ou igual ao de T*

```
1  T' ← T;
2  l ← 1;
3  enquanto l ≤ 3 faça
4      T̄ ← Melhor solução em N_l^vnd(T');
5      se c(T̄) < c(T') então
6          T' ← T̄;
7          l ← 1;
8      senão
9          l ← l + 1;
10 retorna T';
```

O Algoritmo 4 recebe como entrada uma árvore T candidata a solução. Essa árvore é armazenada em T'.

Vasculhamos inicialmente a primeira vizinhança de T' e só examinamos as vizinhanças superiores se não encontrarmos nenhuma melhora na vizinhança atual. Se alguma melhora é encontrada, atualizamos a árvore candidata para a solução $\bar{T}$ de custo inferior a T' e reiniciamos o processo, examinando novamente a primeira estrutura de vizinhança da nova solução. O algoritmo termina ao realizar uma passada pelas três estruturas de vizinhança e não encontrar melhora na solução corrente.

A Figura 3.2 mostra a operação de troca na vizinhança N_1^{vnd}. Na primeira linha, temos uma árvore T e a classificação das arestas que não estão em T à direita. As bolinhas preenchidas representam vértices saturados e as bolinhas em branco vértices não saturados. A segunda linha nos mostra a troca de arestas. Adicionamos a única aresta $e \in E_1$, provocando a violação de um vértice (representado pela bolinha circulada). No ciclo gerado, retiramos a aresta adjacente a esse vértice violado e obtemos a árvore T_1.

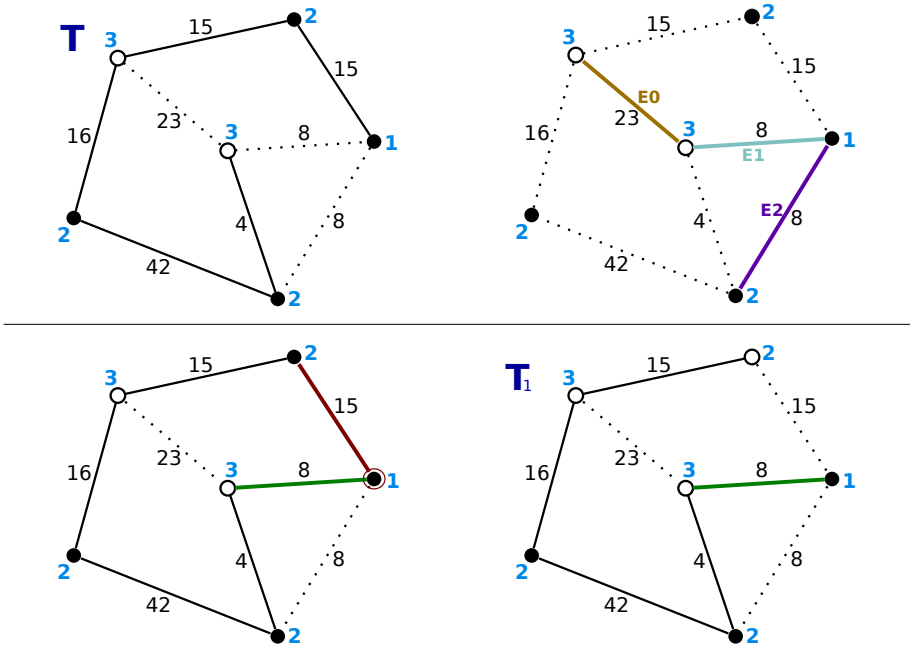

Figura 3.2: VND - Troca na vizinhança N_1^{vnd}

A Figura 3.3 mostra as operações de troca na vizinhança N_2^{vnd}. Na primeira linha, temos uma árvore T e, à direita, o grafo gerado após a adição de uma aresta $e \in E_2$. A inserção da aresta acarreta a violação de dois vértices. Na segunda linha, temos a árvore T_1, gerada após a primeira operação de troca (remoção da aresta de maior custo incidente aos vértices violados no ciclo). Em seguida, temos a árvore T_2, resultante da segunda operação de troca que visa corrigir a violação do vértice violado restante.

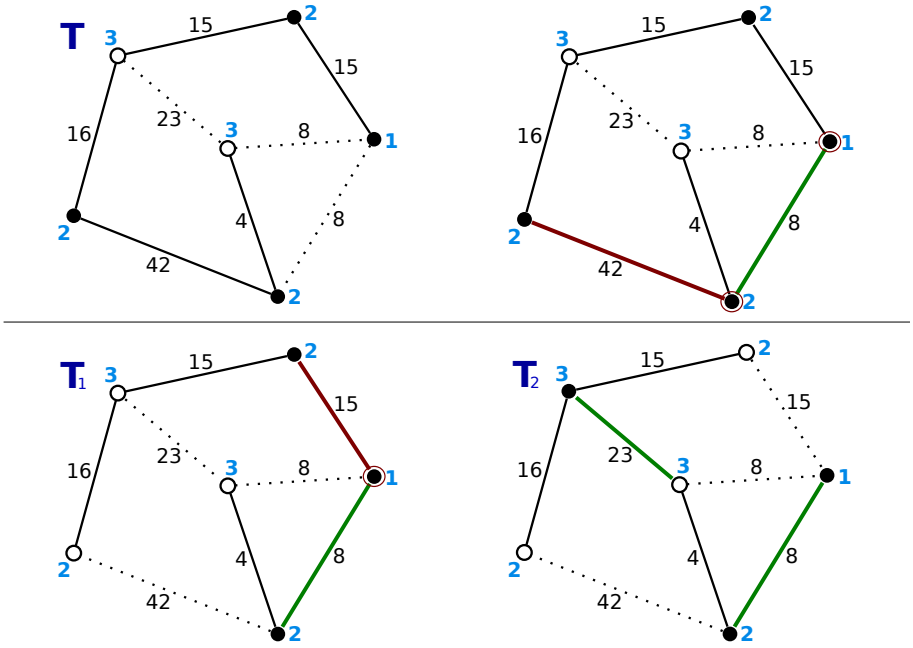

Figura 3.3: VND - Trocas na vizinhança N_2^{vnd}

3.2.5 VND Dinâmico

O que chamamos de *VND Dinâmico* é uma adaptação do *VND* mostrado na seção anterior. O *VND Dinâmico* é utilizado na etapa de busca local da nossa variação do *Skewed VNS* e este, por sua vez, incorporado à *Heurística Lagrangeana* como pode ser verificado na Seção 3.5.

Encontramos alguns obstáculos ao tentar reproduzir sem sucesso o *VND* relatado em [27]. Ele é utilizado como busca local no mecanismo de fuga de ótimos locais de [29], o *Skewed VNS*.

Inicialmente, constatamos que o *VND*, tal qual estava implementado, não contemplava movimentos de troca clássicos do *TSP*, como o 2-Opt (exemplificado na Figura 3.4, onde as arestas pontilhadas são trocadas pelas tracejadas), para as instâncias Hamiltonianas com restrições de duas folhas (tratam-se de instâncias com restrição de grau de valor 1 para exatamente dois vértices e de valor 2 para os demais vértices do grafo).

Percebemos que o problema estava na classificação das arestas que não estão na árvore corrente. Para uma dada árvore T, classificamos as arestas que não estão em T em três grupos: E_0, E_1 e E_2. Note que, para tais instâncias Hamiltonianas, T é um caminho e todos os seus vértices estão saturados. Logo, temos:

$$E_0 = \emptyset \qquad E_1 = \emptyset \qquad E_2 = E - E(T)$$

Isso compromete as operações de troca de arestas do *VND* proposto em [27] para esse tipo

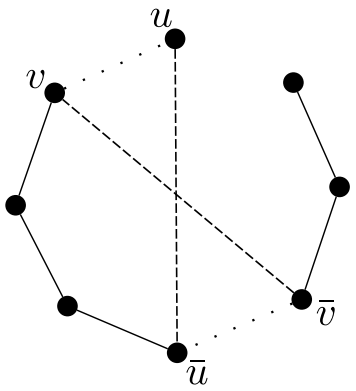

Figura 3.4: Exemplo de troca de arestas do 2-Opt

de instância. Não temos arestas para procurar na primeira vizinhança N_1^{vnd}. As vizinhanças N_2^{vnd} e N_3^{vnd} também não retornam nenhuma outra solução pois, em sua definição, utilizam os conjuntos E_0 e E_1.

A Figura 3.5 nos permite exemplificar o comportamento do *VND* para os três tipos de instâncias Hamiltonianas. Na primeira situação ($d_v = 2\ \forall v$) o conjunto E_1 não é vazio e trocas de arestas podem ser executadas. Por exemplo, considerando as operações de troca de N_2^{vnd}, no primeiro movimento de troca, adicionamos a aresta $\bar{u}v$ e eliminamos a aresta $v\bar{v}$ no ciclo gerado. Na segunda troca, retiramos a aresta $u\bar{u}$ para restaurar a viabilidade do vértice $\bar{u}$ e adicionamos a aresta $u\bar{v}$ para reconectar a árvore. Temos, ao final, outra árvore viável para o problema.

A segunda situação apresenta $d_v = 1$ para exatamente um vértice e $d_v = 2$ para os demais. O conjunto E_1 também não é vazio nesse caso. Basta executarmos as mesmas trocas citadas na primeira situação para obter uma árvore viável.

Na terceira situação, temos exatamente dois vértices $v \in V$ com $d_v = 1$, digamos v_1 e v_2, enquanto os demais vértices $v \neq v_1, v_2$ têm $d_v = 2$. Nesse caso, como todos os vértices estão saturados, $E_1 = \emptyset$ e todas as arestas estão no conjunto E_2. Isso compromete a segunda operação de troca de N_2^{vnd}. Uma solução para esse problema seria considerar também o conjunto E_2 na segunda operação de troca. Assim, podemos aplicar o mesmo exemplo citado nas situações anteriores.

No *VND Dinâmico* que propomos para contornar esse problema, resolvemos mudar um pouco a definição dos conjuntos E_0, E_1 e E_2. Tais conjuntos agora são definidos de forma dinâmica. Consideramos a árvore corrente para sua redefinição, em vez de defini-los com base apenas na árvore inicial T.

Durante o processo do *VND*, após a primeira troca (u_1, v_1), temos uma árvore T^1. É ela que

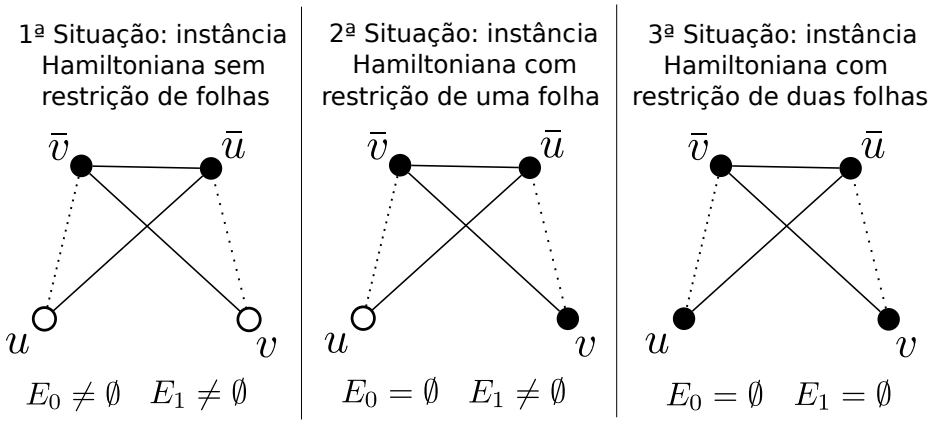

Figura 3.5: Troca de arestas nas instâncias Hamiltonias

consideramos para definição de E_0, E_1 e E_2 da próxima troca (u_2, v_2), gerando uma árvore T^2. Se estamos em N_3^{vnd}, utilizamos T^2 para redefinir os conjuntos para a terceira troca, resultando na última árvore T^3. Assim, passamos a explorar algumas árvores nas vizinhanças N_2^{vnd} e N_3^{vnd}. A vizinhança N_1^{vnd}, contudo, possui cardinalidade zero, já que utilizamos a definição de E_0, E_1 e E_2 dada pela árvore inicial T,

A Figura 3.6 mostra um exemplo de uma instância Hamiltoniana com restrição de duas folhas. Temos uma árvore viável T à esquerda e a classificação das arestas que não estão em T à direita. Todos os vértices estão saturados em T, portanto os conjuntos E_0 e E_1 são vazios. Por esta razão, não podemos realizar a operação de troca da vizinhança N_1^{vnd}.

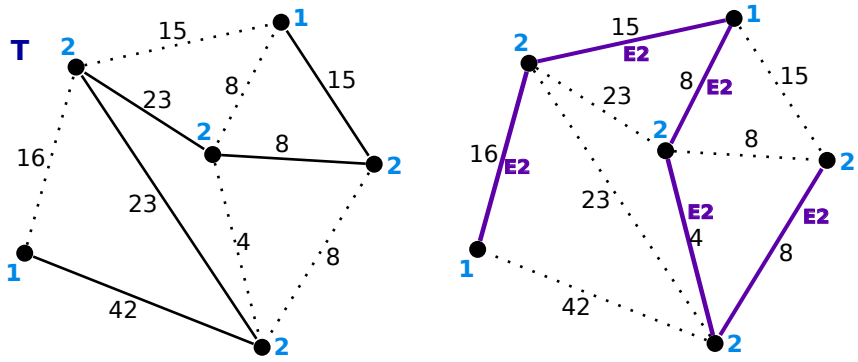

Figura 3.6: Instância Hamiltoniana com restrição de 2 folhas

A Figura 3.7 mostra a primeira operação de troca para a vizinhança N_3^{vnd}. O grafo considerado é o mesmo da Figura 3.6. Inserimos a aresta $e \in E_2$ de custo 8, violando dois vértices e gerando um ciclo. Em seguida, selecionamos a aresta (distinta de e) de maior custo nesse ciclo para ser retirada (uma das arestas de custo 23). Nesse momento, temos uma árvore intermediária T_1. Redefinimos os conjuntos E_0, E_1 e E_2 utilizando a informação de saturação dos vértices da árvore T_1.

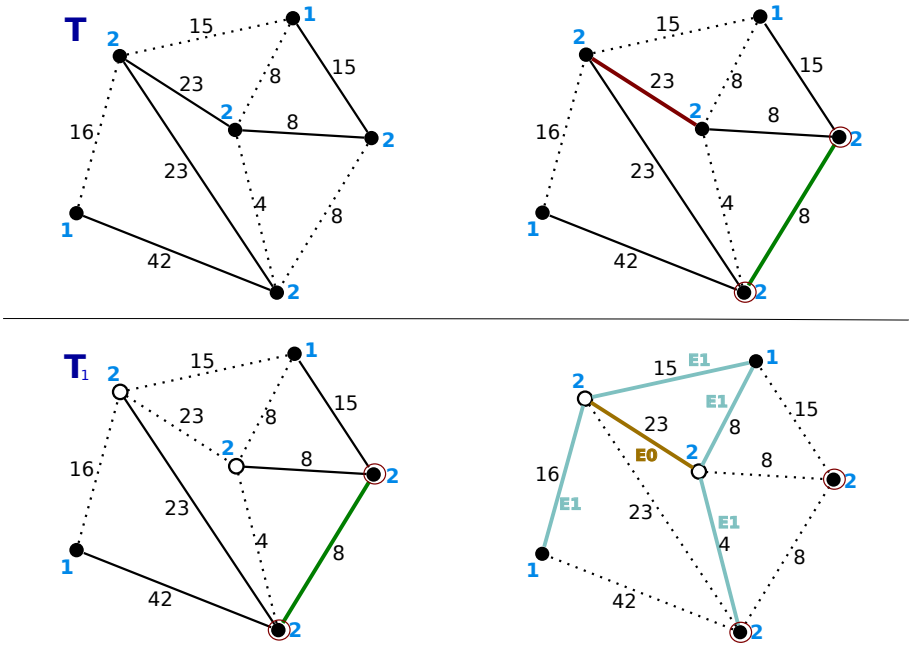

Figura 3.7: VND Dinâmico - 1ª troca na vizinhança N_3^{vnd}

A Figura 3.8 mostra a segunda e terceira operações de troca para a vizinhança N_3^{vnd}. Partimos da árvore T_1 da Figura 3.7 e dos conjuntos E_0, E_1 e E_2 redefinidos a partir de T_1. Nosso objetivo é corrigir as violações dos dois vértices violados indicados nessa figura. A segunda operação de troca corrige um deles, retirando a aresta de custo 42 incidente ao vértice violado mais abaixo e adicionando outra de custo 16 que reconecta a árvore (essa inserção deve respeitar as restrições de grau, por isso trabalhamos com as arestas dos conjuntos E_0 e E_1). Nesse momento, temos uma árvore intermediária T_2. Redefinimos os conjuntos E_0, E_1 e E_2 e em seguida, realizamos a terceira operação de troca para corrigir a violação do vértice violado restante. Ao final, temos a árvore viável T_3.

3.3 Mecanismos de Fuga de Ótimos Locais

Com o objetivo de melhorar as cotas superiores do nosso problema, estudamos e implementamos alguns mecanismos de fuga de ótimos locais. Tais estratégias possuem uma fase de agitação acompanhada de alguma busca local. Em seguida, a solução candidata é analisada e atualizada. Incorporamos tais mecanismos à heurística Lagrangeana de Andrade et al. (2006) [4].

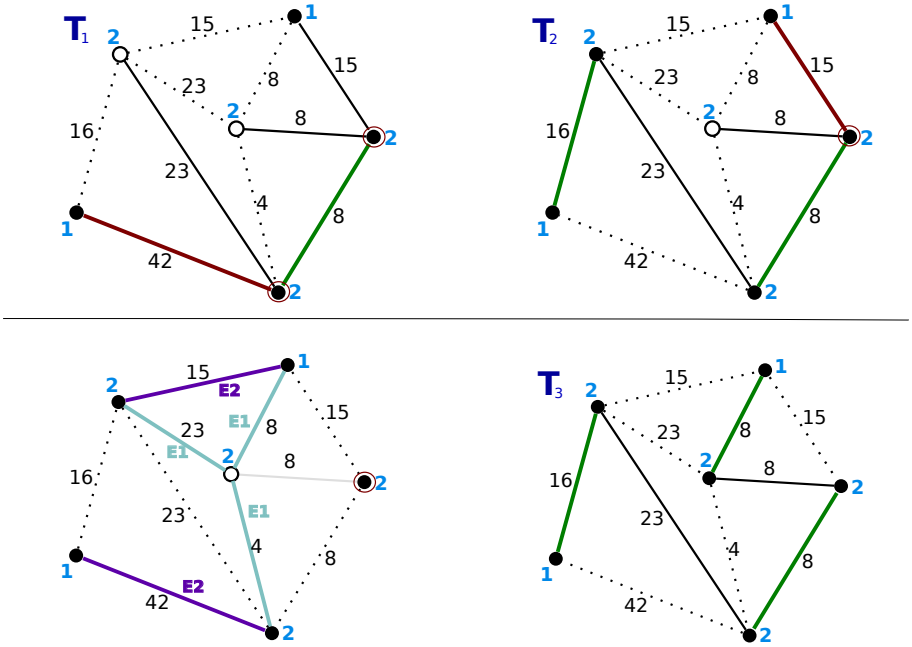

Figura 3.8: VND Dinâmico - 2ª e 3ª trocas na vizinhança N_3^{vnd}

3.3.1 VNS

Abordamos brevemente a técnica *VNS*. Apresentamos o método *VNS* tradicional para facilitar o entendimento das variações que veremos em seguida.

O método de *Busca em Vizinhança Variável (VNS)* foi proposto inicialmente por Hansen e Mladenović (1997) [23]. Trata-se de uma metaheurística cuja ideia básica é a troca sistemática de vizinhanças com aplicação de uma busca local dentro de cada vizinhança. Tais estruturas são pré-selecionadas e percorremos o espaço de soluções definido por elas.

Uma iteração do *VNS* pode ser dividida em três partes:

- Agitação
- Busca local
- Atualização da solução candidata

Na fase de agitação, uma solução é obtida a partir da solução candidata e conforme as estruturas de vizinhança definidas. Na fase de busca local, as vizinhanças são utilizadas para melhorar, de maneira determinística, a solução encontrada na fase anterior. Na última fase, comparamos a solução retornada pela busca local com a solução candidata e a atualizamos se possível.

É preciso definir, portanto, dois grupos de estruturas de vizinhança. O primeiro grupo é

utilizado na primeira fase do método, enquanto o segundo grupo é utilizado nos mecanismos de busca local.

O método VNS para o problema *AGMRG* pode ser visto no Algoritmo 5, conforme mostrado em [27].

Algoritmo 5: Pseudo-Código: VNS_DCMST (Ribeiro e Souza [27])

Entrada: *uma solução inicial* T_0
Resultado: *AGMRG T'*

```
1  T ← T_0;
2  Defina k_max para algum valor em {2,...,| V |};
3  para i = 1,...,seqs faça
4      k ← 1;
5      enquanto k ≤ k_max faça
6          Gere arbitrariamente T' ∈ N_k^vns(T);
7          Obtenha T̄ aplicando um procedimento de busca local em T';
8          se c(T̄) < c(T) então
9              T ← T̄;
10             k ← 1;
11         senão
12             k ← k + 1;
```

$N_1^{VNS},...,N_{k_{max}}^{VNS}$ é a sequência de estruturas de vizinhança definidas para a fase de agitação, onde $N_k^{VNS}(T)$ é o conjunto de soluções vizinhas a T que diferem de T exatamente em k arestas. O Algoritmo 5 termina depois de realizar *seqs* vezes a exploração dessa sequência sem encontrar nenhuma melhora. O parâmetro *seqs* é obtido de forma empírica e definido como 1 [27].

Na fase de busca local, podemos utilizar qualquer procedimento determinístico que melhore a solução encontrada na fase de agitação. Em [27], os autores usam o *VND* [27] e a atualização da árvore candidata só ocorre se a solução retornada pela busca local possuir um custo inferior ao dela.

3.3.2 Skewed-VNS

Em [29], algumas ideias foram desenvolvidas para melhorar a performance da *busca em vizinhança variável* mostrado na seção anterior. As estratégias elaboradas atuam justamente nas diferentes fases do *VNS*.

Na fase de agitação, a busca na vizinhança é guiada com o auxílio de *Algoritmos de Segunda Ordem (SO)*, classe de algoritmos introduzida em [18].

Na fase de busca local, o algoritmo utilizado é o *VND* de Ribeiro e Souza [27] e em seguida, na atualização da árvore candidata, funções de absorção são utilizadas para comparar as

diferenças estruturais entre a árvore em questão e a árvore encontrada na *descida em vizinhança variável* (VND).

Procurando reproduzir os resultados encontrados na literatura, implementamos as ideias de [27] e [29]: o método de descida variável (*VND*), as variações dos algoritmos de segunda ordem, e as funções de absorção na fase da atualização da solução candidata.

Agitação

Os algoritmos de segunda ordem (*SO*) geram uma sequência de problemas restritos. A cada iteração, iniciamos com um problema restrito P e terminamos com um novo problema restrito P'. Durante cada iteração, uma restrição é adicionada a P, de maneira que o conjunto viável de P' esteja contido no conjunto viável de P.

Na abordagem de [29] para o problema *AGMRG*, adicionar uma restrição ao problema P significa fixar uma aresta na solução do problema P'. Na fase de agitação, k arestas são retiradas aleatoriamente da árvore viável corrente T gerando uma floresta F. O *SO* é utilizado na reconexão de F. A cada iteração, uma aresta é selecionada entre um conjunto de arestas candidatas para ser adicionada a F. Ao final das k iterações, temos uma nova árvore viável T'. As arestas candidatas devem satisfazer a algumas condições: suas extremidades devem estar em componentes diferentes; sua inserção não viola as restrições de grau; e se $\mid E \mid \leq \mid V \mid - 3$, então pelo menos um vértice da componente resultante não se torna saturado.

Em [29], duas variações de como reconectar a floresta utilizando o *SO* são apresentadas:

SO_rd: O primeiro passo dessa variação é, a cada iteração, selecionar q arestas candidatas. O segundo passo é a construção de q árvores viáveis. Inicialmente, adicionamos uma das q arestas candidatas. Em seguida, chamamos o *Kruskal Modificado* para terminar de construir cada uma das árvores. Seja $\bar{q}$ a aresta candidata associada à árvore de menor custo. Adicionamos $\bar{q}$ ao problema restrito P, do início da nossa iteração, gerando um grafo acíclico com uma componente a menos que as do grafo anterior.

SO_gr: Nessa abordagem, um conjunto $R = \{e_1, e_2, ..., e_q\}$ de q arestas é selecionado de maneira que $F \cup R$ é uma árvore viável. Definimos também os subconjuntos $R_u \subset R$, $u = 1, ..., q$, onde R_u é obtido por meio da eliminação da aresta e_u de R. A cada iteração, inserimos temporariamente as arestas de R_u, e com o *Kruskal Modificado*, encontramos apenas uma aresta que completa a árvore. Portanto, temos q arestas encontradas pelo *Kruskal Modificado*. Aquela aresta com o menor custo é escolhida para ser adicionada permanentemente em F na iteração corrente.

Algoritmo 6: Pseudo-Código: SO_rd (Souza e Martins [29])

Entrada: *número k de arestas removidas aleatoriamente e a floresta* $F = (V, \bar{E})$ *gerada por tais remoções*
Resultado: *AGMRG T construída a partir de F*

```
1  i ← | V | − k;
2  enquanto | Ē | < | V | −1 faça
3      q ← | V | − i;
4      Selecione aleatoriamente q arestas candidatas, e_1, e_2, ..., e_q, de E − Ē;
5      para u = 1, ..., q faça
6          F' ← F ∪ {e_u};
7          z_u ← KruskalModificado(F');
8      q̄ ← argmin{z_u : u = 1, ...., q};
9      Ē ← Ē ∪ {e_q̄};
10     i ← i + 1;
```

Algoritmo 7: Pseudo-Código: SO_gr (Souza e Martins [29])

Entrada: *número k de arestas removidas aleatoriamente e a floresta* $F = (V, \bar{E})$ *gerada por tais remoções*
Resultado: *AGMRG T construída a partir de F*

```
1  i ← | V | − k;
2  enquanto | Ē | < | V | −2 faça
3      q ← | V | − i;
4      Selecione aleatoriamente conjunto R de q arestas, e_1, e_2, ..., e_q, de E − Ē tal que F' = (V, Ē ∪ R) é uma
       árvore viável;
5      c̄ ← ∞;
6      para u = 1, ..., q faça
7          F' ← F ∪ R − {e_u};
8          Seja e a aresta que completa uma árvore viável encontrada pelo KruskalModificado;
9          se c_e < c̄ então
10             c̄ ← c_e;
11             ē ← e;
12     Ē ← Ē ∪ {ē};
13     i ← i + 1;
14 Selecione aleatoriamente aresta ē de E − Ē tal que F' = (V, Ē ∪ {ē}) é uma árvore viável;
15 Ē ← Ē ∪ {ē};
```

Os Algoritmos 6 e 7 mostram as ideias apresentadas anteriormente. Procuramos reproduzir tais ideias e, dentre as dificuldades encontradas, a que se destaca é justamente a seleção das q arestas em cada iteração dos algoritmos de segunda ordem. Esse procedimento aumenta o custo dessa abordagem em relação a uma simples seleção arbitrária na vizinhança corrente.

Na variação *SO_gr*, implementamos um mecanismo de retorno para facilitar a seleção do conjunto R. Em certas ocasiões, algumas combinações de arestas não permitem encontrar uma solução viável facilmente, ou seja, o processo para completar as q arestas de R é muito difícil e demorado. Portanto, ao atingir um número máximo de tentativas, ignoramos as arestas encontradas até então, nosso conjunto R é zerado e iniciamos o processo de seleção novamente.

Em algumas situações, verificamos que o *SO_rd* não consegue encontrar facilmente as q arestas candidatas. Nesses casos, ao atingir um número máximo de tentativas, simplesmente fornecemos uma árvore viável T' encontrada pela remoção aleatória de k arestas da árvore candidata e pela reconexão da floresta gerada utilizando o algoritmo *Kruskal Modificado*.

Busca Local

Para o processo de busca local, foi utilizado o método de descida em vizinhança variável (VND) de Ribeiro e Souza [27]. Maiores detalhes sobre o mesmo podem ser encontrados nessa referência.

Aceitação da Árvore

Essa última etapa consiste na definição dos critérios de aceitação da solução viável retornada na fase de busca local. Em outras palavras, determinamos se a solução candidata será atualizada.

Como pode ser visto no Algoritmo 8, o *VNS* tradicional [27] simplesmente verifica se o custo da solução retornada pela busca local é melhor que o custo da solução candidata, atualizando em caso positivo ou incrementando a vizinhança em caso negativo.

Algoritmo 8: Pseudo-Código: Atualização da árvore candidata no VNS tradicional [27]

```
se c(T̄) < c(T) então
    T ← T̄;
    k ← 1;
senão
    k ← k + 1;
```

O *Skewed VNS* foi proposto originalmente por Hansen et al (2000) [12]. A ideia principal é

tornar-se mais tolerante na aceitação de uma solução retornada pela busca local, não necessariamente melhorando a solução candidata. Uma árvore com custo um pouco pior pode ser aceita, porém tais limites de aceitação precisam ser definidos. Devido a isso, precisamos guardar a melhor solução encontrada até então.

No Algoritmo 9, podemos ver o esquema de atualização da solução candidata do *Skewed VNS* mostrado por Souza e Martins (2007) [29].

Algoritmo 9: Pseudo-Código: Atualização da árvore candidata no Skewed VNS [29]

```
se c(T̄) < c(T*) então
    T* ← T̄;
se c(T̄) − c(T) < β · ρ(T, T̄) então
    T ← T̄;
    k ← 1;
senão
    k ← k + 1;
```

Esse algoritmo utiliza uma função ρ para medir a distância entre T e $\bar{T}$ e um parâmetro positivo β, ambos definidos mais adiante. Note que a melhor solução é armazenada em T^*.

Em Souza e Martins [29], os autores propõem uma definição para a função ρ e para o parâmetro β. A função ρ é definida como a variação dos graus dos vértices total.

Sejam σ_i e $\bar{\sigma}_i$ os graus do vértice i em T e $\bar{T}$ respectivamente. Definimos ρ como

$$\rho(T,\bar{T}) = \sum_{i \in V} \mid \sigma_i - \bar{\sigma}_i \mid$$

A ideia da função ρ é permitir que árvores com estruturas diferentes possam ser aceitas mais facilmente. Se duas árvores T e $\bar{T}$ apresentam estrutura semelhante, os respectivos graus σ_i e $\bar{\sigma}_i$ de um dado vértice i serão muito próximos. Logo, a tendência é que a função $\rho(T,\bar{T})$ se aproxime de zero. Assim, o fator de erro β não terá muita força e a atualização da árvore candidata recaí simplesmente na verificação dos custos das árvores em questão.

O parâmetro β é definido como uma função quadrática do quartil de k_{max}.

Seja $q_k \in \{1,2,3,4\}$ o quartil de k_{max} que contém o valor de k corrente da fase de agitação. Então define-se

$$\beta(q_k) = \bar{\beta} q_k^2$$

onde $\bar{\beta}$ representa o percentual de erro que estamos dispostos a aceitar.

Critérios de Parada

Sabemos que as estruturas de vizinhança $N_1^{VNS}, ..., N_{k_{max}}^{VNS}$ são definidas para a fase de agitação do *VNS* (*Skewed VNS*). Quando o *VNS* (*Skewed VNS*) faz uma passada completa por todas as estruturas, encerramos sua execução.

O valor de k_{max} é definido como uma porcentagem do número de arestas na árvore, portanto, dependendo do tamanho da instância. O *VNS* (*Skewed VNS*) pode demorar a ser finalizado e áreas de vizinhança com índice maior podem não ser facilmente alcançadas. Diante dessa problemática, Hansen e Mladenovic ([13], [14]) sugerem um novo parâmetro chamado k_{step}. Ao invés de incrementarmos o índice da vizinhança em uma unidade, a cada iteração do *VNS*, incrementamos em k_{step} unidades.

Além disso, como podemos atualizar a solução candidata sem melhorar seu custo, adicionamos também um outro critério de parada. Trata-se de um número máximo de iterações que o algoritmo executa sem encontrar nenhuma melhora. Dessa maneira, evitamos que o algoritmo fique continuamente aceitando soluções que não melhoram nossos limites.

3.4 Método do Subgradiente

No Capítulo 2, definimos o problema *AGMRG*, sua relaxação Lagrangeana e o problema dual Lagrangeano, cuja resolução corresponde a encontrar um conjunto ótimo de multiplicadores λ^* que forneça o melhor limite inferior para o problema.

$$
\begin{aligned}
(P) \quad \min \quad & \sum_{e \in E} c_e x_e && (2.1)\\
s.a. \quad & (2.2), (2.3) \; e \; (2.4), \\
& \sum_{e \in \delta(v)} x_e \leq d_v, \forall v \in V.
\end{aligned}
$$

$$
\begin{aligned}
(P_\lambda) \quad \min \quad & \sum_{e=(i,j) \in E} (c_e + \lambda_i + \lambda_j) x_e - \sum_{i \in V} \lambda_i d_i && (2.6)\\
s.a. \quad & (2.2), (2.3) \; e \; (2.4).
\end{aligned}
$$

$$(D) \quad \max \quad z_\lambda \qquad\qquad (2.7)$$
$$s.a. \quad \lambda \geq 0.$$

Uma forma de se obter limites inferiores para o problema é construindo uma sequência de multiplicadores (λ^t) que se espera convergir para λ^*.

Tal sequência (λ^t) pode ser determinada utilizando o **Método do Subgradiente** (*MSG*) [17].

Seja z_{ub} o valor de uma solução viável para o problema (P). Denote por $\lambda^p \in \mathbb{R}_+^{|V|}$ o conjunto de multiplicadores de Lagrange gerado na iteração p do *MSG*. Seja x^p uma solução ótima (de valor $z(\lambda^p)$) para a função objetivo da relaxação Lagrangeana (2.6) associada a λ^p. O subgradiente s^p de (2.6) em x^p é dado por $s_i^p = \sum_{e \in \delta(i)} x_e^p - d_i$, para todo $i \in V$.

O *MSG* consiste em determinar um novo conjunto de multiplicadores de Lagrange de forma iterativa [4]:

$$\lambda_i^{p+1} = \max\{0, \lambda_i^p + t_p s_i^p\}, \; \forall \, i \in V \; e \; p \geq 0, \qquad (3.4)$$

onde $t_p = \alpha_p \frac{z_{ub} - z(\lambda^p)}{||s^p||^2}$, com $\alpha_p = 2$ para $p = 0$.

O valor de α_p é dividido por um fator de redução $f \in (0,1)$ sempre que um número ρ de iterações sem melhora é atingido.

A ideia é determinar um conjunto λ^p de multiplicadores de Lagrange e resolver o problema (P_λ). Determinando uma árvore geradora com custos modificados por λ^p, obtemos um limite inferior de valor $z(\lambda^p)$ para o problema original. Se $z(\lambda^p)$ não for ótimo, sua árvore x^p associada é usada para determinar um novo conjunto de multiplicadores λ^{p+1} e assim sucessivamente até uma condição de parada ser atingida [4].

3.5 Heurística VNS-Lagrangeana

A *Heurística Lagrangeana* de Andrade et al. [4], além de fornecer boas cotas duais para o problema, tem como objetivo construir soluções primais a partir da informação dual obtida a cada iteração do método do subgradiente. Para essas soluções primais viáveis, aplica-se algum mecanismo de busca local. Em particular, é utilizado o *procedimento de melhora* (mostrado na Seção 3.2.3) baseado nas vizinhanças do *1-Opt*. A *Heurística Lagrangeana* [4], porém, não

apresenta mecanismos de fuga de ótimos locais. Devido aos bons resultados obtidos por Ribeiro e Souza [27] e Souza e Martins [29], incorporamos algumas de suas estratégias à *Heurística Lagrangeana*. A esse método resultante, damos o nome de Heurística VNS-Lagrangeana.

Podemos dividir a Heurística VNS-Lagrangeana nas seguintes partes:

1. Determinar uma solução viável para o problema e definir um problema de *AGMRG* reduzido com as arestas usadas para obter essa solução. (Algoritmo 1 - *Kruskal Modificado*)
2. Busca por soluções viáveis para o problema *AGMRG* reduzido. (Método do Subgradiente com busca de soluções viáveis [4] e Método *Skewed VNS*)
3. Obter limites duais e primais para o problema *AGMRG* original. (Método do Subgradiente [4])

3.5.1 Solução viável inicial

Esta fase consiste em determinar um limite superior para a solução do problema. Encontramos uma solução viável para o problema *AGMRG* usando o algoritmo *Kruskal Modificado* [4] (Algoritmo 1) e a partir das arestas utilizadas para construir essa solução definimos o conjunto de arestas do problema reduzido que será utilizado na segunda fase da Heurística, como é explicado na Seção 3.1.

3.5.2 Busca por soluções viáveis para um problema AGMRG reduzido

Durante esta etapa, trabalhamos apenas com um subconjunto de arestas do problema original para determinar soluções viáveis para o problema reduzido. Isso possibilita reduzir o tamanho do problema e aplicar mecanismos heurísticos de geração de soluções primais viáveis de forma eficiente.

Com o auxílio de multiplicadores de Lagrange, a cada iteração i do *método do subgradiente*, construímos soluções Lagrangeanas, isto é, a solução da relaxação Lagrangeana para os multiplicadores da iteração i. Trata-se de uma árvore geradora mínima encontrada no grafo com os custos das arestas modificados. Esse processo nos permite obter cotas inferiores do problema reduzido. Além disso, em cada iteração, se a solução Lagrangeana T gerada for primal viável, podemos aplicar um mecanismo de busca local para melhorar a cota superior encontrada. Caso T não seja primal viável, usamos o algoritmo *Kruskal Modificado* para construir uma nova solução primal viável T' (para o grafo com custos modificados utilizando os multiplicadores) e em seguida podemos tentar melhorá-la com uma busca local.

Ao final do *método do subgradiente*, submetemos a uma variação do método *Skewed VNS* a árvore $\hat{T}$ de menor custo encontrada durante o processo. No método *Skewed VNS* em questão, geramos um vizinho arbitrário na vizinhança N_k^{vns} da árvore candidata durante a fase de agitação, utilizamos o *VND Dinâmico* semelhante ao *VND* de Ribeiro e Souza [27] na fase de busca e utilizamos as funções de absorção propostas por Souza e Martins [29].

3.5.3 Obter limites primais e duais para o problema AGMRG original

O processo é semelhante ao da etapa anterior. A diferença é que o *método do subgradiente* é aplicado utilizando todo o conjunto de arestas do grafo e que não procuramos construir, como o fazemos para o problema reduzido, soluções viáveis para o grafo de custos modificados.

O Algoritmo 10 mostra um resumo dos procedimentos em cada etapa da *heurística VNS-Lagrangena*. Detalhes computacionais, como a utilização dos parâmetros do método do Subgradiente, podem ser vistos em Andrade et al. [4], onde a *heurística Lagrangeana* é descrita em detalhes.

Na primeira parte da *heurística VNS-Lagrangena*, temos a obtenção de uma árvore viável e a definição do problema *AGMRG* reduzido. Em seguida, trabalhamos nesse subconjunto de arestas aplicando o *método do Subgradiente*, obtendo soluções viáveis e aplicando o método *Skewed VNS* na melhor árvore viável encontrada. Ao final, trabalhamos com o conjunto original de arestas, encontrando cotas primais e duais para o problema através do *método do subgradiente*.

Trabalhamos com duas variações da Heurística VNS-Lagrangeana:

***Heurística VNS-Lagrangeana v1*:** Nesta variação, não utilizamos os algoritmos de segunda ordem na fase de agitação do *Skewed VNS*. Simplesmente, geramos um vizinho arbitrário na vizinhança N_k^{vns}, removendo aleatoriamente k arestas e utilizando o *Kruskal Modificado* para reconectar a floresta gerada. Tal procedimento demanda menor custo computacional.

Ainda no *Skewed VNS*, na fase de busca local, utilizamos o *VND dinâmico* proposto anteriormente (Seção 3.2.5). Na atualização da árvore candidata, utilizamos as funções ρ e β definidas por Souza e Martins [29] (Seção 3.3.2). Atribuímos 30% ao parâmetro $\bar{\beta}$ utilizado no cálculo de β.

Os valores 40% e 8% do número de arestas da árvore são atribuídos aos parâmetros k_{max} e k_{step} respectivamente e um dos critérios de parada (além da passada completa pelas vizinhanças) é passar 5 iterações do algoritmo *Skewed VNS* sem obter nenhuma melhora.

Algoritmo 10: Pseudo-Código: heurística VNS-Lagrangeana

```
// Solução viável inicial
Encontramos T viável inicial com Kruskal Modificado;
Definimos o conjunto de arestas do problema reduzido;
// Busca por soluções viáveis para um problema AGMRG reduzido
para cada iteração i do Método do Subgradiente faça
    Seja T_i a solução Lagrangeana da iteração corrente;
    se T_i é viável então
        Aplicar procedimento de busca local em T_i;
    senão
        Encontramos T̄_i viável com Kruskal Modificado no grafo reduzido pertubardo por λ^i;
        se custo original de T̄_i melhorar limite superior então
            Aplicar procedimento de busca local em T̄_i;
Aplicamos método Skewed VNS em T̂; // T̂ é a melhor árvore viável encontrada.
// Limites primais e duais para o problema AGMRG original
Aplicamos o Método do Subgradiente no problema original com os multiplicadores de Lagrange obtidos
ao final das iterações do problema reduzido;
```

Esses valores são definidos de forma empírica baseado em vários experimentos computacionais.

***Heurística VNS-Lagrangeana v2*:** A única diferença desta variação em relação a anterior é a aplicação de uma busca local durante a heurística VNS-Lagrangeana quando encontramos soluções viáveis para o problema *AGMRG* reduzido. Tais soluções podem ser encontradas quando a solução ótima da relaxação Lagrangeana, em uma dada iteração do método do Subgradiente, é também primal viável ou quando construímos uma solução viável no grafo reduzido perturbado pelos multiplicadores de Lagrange da iteração. Aplicamos o procedimento de melhora do tipo 1-Opt mostrado na Seção 3.2.3. Na variação *Heurística VNS-Lagrangeana v1*, nenhuma busca local é realizada quando uma solução viável é encontrada no método do subgradiente.

3.6 Árvore de Subgradiente com pré-fase VNS-Lagrangeana

Após o término da heurística VNS-Lagrangeana (pré-fase), o procedimento da *Árvore de Subgradiente (ASG)* [2, 3] é aplicado. Ele utiliza a ramificação de variáveis e avaliação de nós tal qual ocorre em uma árvore de *Branch and Bound (B&B)* tradicional.

A diferença é que, associadas a cada nó da *Árvore de Subgradiente*, temos uma subsequên-

cia de multiplicadores de Lagrange e uma solução Lagrangeana. Tais multiplicadores são atualizados de um nó pai para um nó filho na *ASG*, conforme a atualização tradicional do subgradiente, utilizando os multiplicadores do nó pai e a solução Lagrangeana do nó filho após a fixação de variáveis obtida no processo de ramificação.

Por meio da solução Lagrangeana de um nó pai, geramos uma perturbação nos custos das arestas (do grafo original) por intermédio dos multiplicadores de Lagrange e obtemos uma solução (uma árvore geradora T) para cada nó filho (obedecendo a fixação das variáveis). Cada aresta $e = (u, v)$ de T, em um dado nó na ASG, terá um novo custo $\bar{c_e}$ que será a soma do seu custo original c_e com os valores dos multiplicadores associados aos vértices u e v desse nó. A árvore T fornece uma direção do subgradiente que será utilizada para obter os multiplicadores iniciais desse nó na ASG.

Os multiplicadores de Lagrange de um dado nó são atualizados k vezes internamente nesse nó da *ASG*, ou seja, realizamos o método do Subgradiente dentro de cada nó limitado a k iterações. Isso nos permite encontrar melhores cotas inferiores para a solução de cada nó, enquanto subproblema de *AGMRG* com arestas fixas e, consequentemente, melhores cotas inferiores para o problema original.

Nessa técnica, introduzimos o conceito de penalizadores primais que são utilizados quando a fixação de uma aresta em 1 (obrigando a presença da aresta na árvore) é responsável pela saturação de um vértice. Quando um vértice já está saturado, não há necessidade de mais nenhuma aresta incidir naquele vértice. Logo, seria conveniente que o restante das arestas nele incidentes fossem automaticamente fixadas em 0 (obrigando a sua ausência na árvore). Tal procedimento permite reduzir várias ramificações do *B&B* e economizar na alocação de memória por não aumentar o tamanho das listas de arestas fixas em 0.

Quando um vértice torna-se saturado, o penalizador primal daquele vértice recebe o valor $+\infty$. Dessa maneira, quando inserimos arestas na árvore solução de cada nó, se um vértice v já está saturado, todas as demais arestas no grafo original que incidem em v e não fixadas em 1, nunca farão parte da árvore apesar de não serem fixadas em 0. Dessa maneira, uma economia de memória resulta do processo, já que não precisamos armazenar, para um dado nó, a lista de variáveis fixadas em 0. Quando um vértice v torna-se saturado, o seu respectivo multiplicador de Lagrange recebe o valor 0. Isso ocorre porque não é preciso adicionar mais nenhum custo às arestas incidentes ao vértice v, pois todas as arestas já estão fixadas, ou em 1, que são as arestas responsáveis pela saturação do vértice, ou em 0, que são as demais arestas que não podem mais ser inseridas. Podemos ver isso como a remoção da restrição de grau desse vértice da formulação do problema após a fixação de variáveis.

No Algoritmo 11 podemos verificar a descrição geral do procedimento.

Algoritmo 11: Pseudo-Código: Árvore de Subgradiente

```
// Pré-fase
Heurística VNS-Lagrangeana
// Árvore de Subgradiente
Utilizamos os multiplicadores (λ^subg) do fim do método do subgradiente como multiplicadores (λ^0) iniciais da ASG;
Geramos o nó raiz (solução da relaxação de grau) da árvore de busca e colocamos na lista 𝒮 de nós a serem avaliados;
enquanto existir nó p ∈ 𝒮 faça
    Ramificamos o nó p em outros nós {f_1, f_2, ...} e atualizamos seus multiplicadores primais {β^f1, β^f2, ...} e Lagrangeanos {λ^f1, λ^f2, ...};
    para cada nó filho f_i faça
        Resolvemos a relaxação de grau de f_i e obtemos o valor v_1;
        Resolvemos a relaxação Lagrangeana de f_i e obtemos o valor v_2;
        se solução x^fi da relaxação de grau é viável para o problema original então
            Se possível, atualizamos os limites e realizamos os procedimentos de poda;
        senão
            se as arestas fixadas em 1 não tornam a solução x^fi inviável para o problema original então
                Aplicamos MSG limitado a k iterações e atualizamos o valor v_2;
                se max{v_1,v_2} < UB então
                    Adicionamos f_i a lista 𝒮 de nós a serem avaliados;
```

3.7 Novo esquema de ramificação do método ASG

Propomos neste trabalho um novo esquema de ramificação do nó que é avaliado no método da *Árvore de Subgradiente*. Esse esquema de particionamento permite eliminar a solução que é encontrada no nó que é ramificado.

Ao analisarmos um nó i da *ASG*, temos uma árvore T_i que é a solução relaxada (Lagrangeana) do subproblema levando em consideração a fixação das arestas na *ASG*. Verificamos em T_i a existência de pelo menos um vértice violado, ou seja, um vértice cujo grau em T_i seja maior que a sua restrição de grau. Quando mais de um vértice violado é encontrado, consideramos para a ramificação o vértice v que apresenta menor violação, isto é, cujo o grau em T_i mais se aproxima da sua restrição de grau. Procuramos, com essa escolha, eliminar a violação de grau do vértice v mais rapidamente.

Sejam v o vértice escolhido, $\delta'(v)$ o conjunto de arestas ainda não fixadas que incidem em

v e $|\delta'(v)| = p$. A ideia básica é reduzir a violação de v em cada ramificação em uma unidade. Fixamos, inicialmente, uma das arestas em $\delta'(v)$ em zero (fixação fora da árvore). Para cada aresta $e_j \in \delta'(v)$, geramos um nó filho n_j onde a aresta e_j é fixada em zero.

Na Figura 3.9, temos um vértice violado v (com restrição $d_v = 1$) que dá origem a três nós filhos. Os traços pontilhados representam as arestas fixadas em 0 (supomos aqui que nenhuma aresta incidente em v esteja fixada em 1 no nó pai).

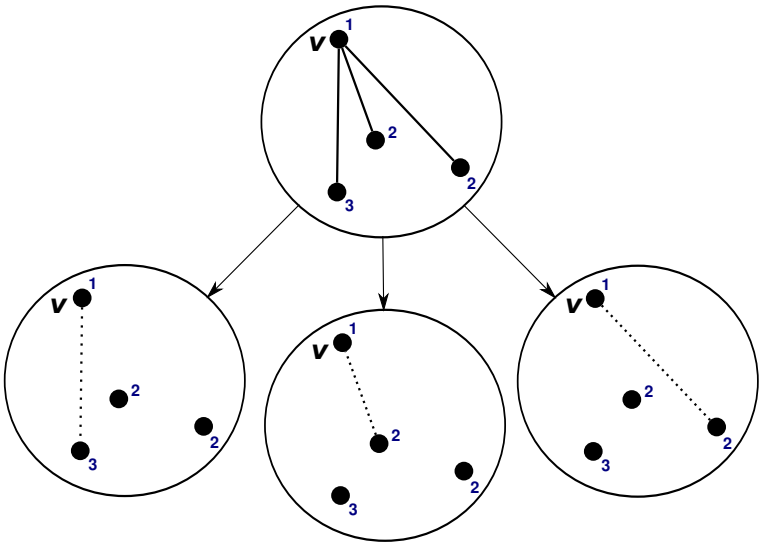

Figura 3.9: Ramificação utilizando fixações em 0

Percebemos que tal particionamento embora não contenha a solução do nó que está sendo ramificado, não gera partições disjuntas. Para resolver esse problema, utilizamos simultaneamente a fixação de arestas em 1 (fixação dentro da árvore).

Tomamos então as arestas de $\delta'(v)$: $L_v = \langle e_1, e_2, ..., e_p \rangle$ na ordem em que estão armazenadas na lista (estrutura de dados) de arestas da árvore. Para cada $j \in 1, ..., p$, geramos um nó filho no qual a aresta e_j é fixada em 0 e as arestas $e_1, ...e_{j-1}$ são fixadas em 1. Dessa maneira, as partições tornam-se disjuntas embora elas fiquem desbalanceadas. A fixação em zero permite excluir a solução do nó pai, o que não ocorreria caso fixássemos uma aresta de cada vez em 0 ou em 1 por dicotomia.

A Figura 3.10 exemplifica o processo. Os traços pontilhados representam as arestas fixadas em 0 e os tracejados as fixadas em 1. É possível ver também que o esquema pode gerar subproblemas inviáveis: é o caso do nó filho mais à direita.

Na Figura 3.11, apresentamos o esquema geral do novo procedimento de ramificação onde podemos verificar que p nós filhos são gerados. As bolas brancas representam tais nós. Veja que, para uma certa aresta e_j, o ramo da árvore da *ASG* enraizado em n_j apresenta soluções

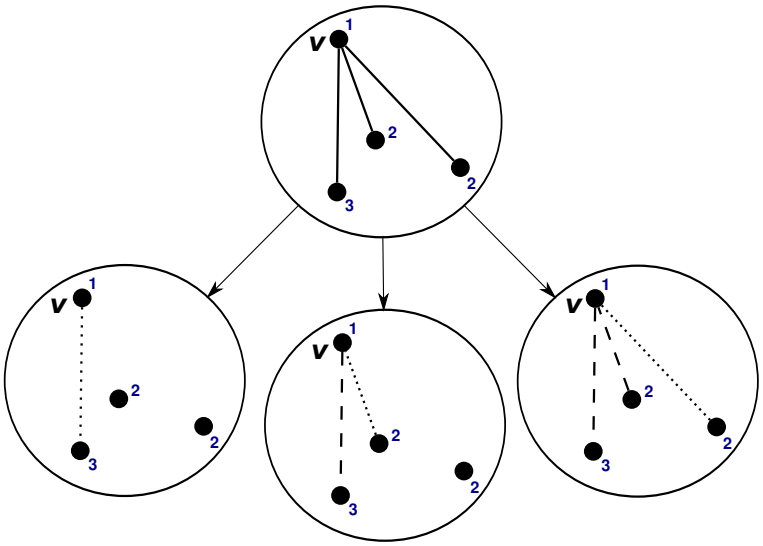

Figura 3.10: Ramificação utilizando fixações em 0 e em 1

onde a aresta e_j está fixada em 0. Os nós $n_{j+1}, n_{j+2}, ..., n_p$, por sua vez, enraízam os ramos onde a aresta e_j está fixada em 1. Chamamos de *in* a lista de arestas fixadas em 1 e chamamos de *out* a lista de arestas fixadas em 0. Note que a ideia é sempre deixar uma aresta fixada em 0, para que o vértice violado no nó pai não apresente o mesmo grau nos nós filhos.

Proposição 1. *As partições (subconjuntos) de soluções viáveis associadas aos nós filhos n_1,...,n_p da ASG na Figura 3.11 são disjuntas e não contêm a solução do nó pai que as gerou.*

Demonstração. As partições são claramente disjuntas. Basta verificar que para um dado nó n_i, a aresta e_i é fixada em 0, enquanto nos nós $n_{i+1}, ...n_p$, a aresta e_i é fixada em 1. A sub-árvore da Figura 3.11 só contém, na verdade, filhos nas ramificações do lado direito. O único filho (ramo) não presente nessa sub-árvore é o correspondente à fixação de e_p em 1 (o que levaria à solução do nó pai). Portanto, essa solução é separada do conjunto de soluções viáveis do problema. □

Note que o esquema de ramificação da Figura 3.11 é bem geral e pode ser aplicado a qualquer esquema de resolução do tipo B&B onde a relaxação do subproblema em um nó é um problema combinatório (solução inteira 0-1).

Perceba também que, no processo de ramificação de um nó, ao selecionarmos um vértice v de grau d_v, podemos parar o processo de criar novos nós filhos quando o número de arestas fixas em 1 saturar o vértice v (os demais nós gerados forneceriam soluções inviáveis).

Assim, efetivamente em nosso esquema de ramificação, no máximo d_v nós filhos são gerados por vez. Isso ocorre quando o vértice v, de grau d_v, não possui nenhuma aresta fixa em 1 incidente nele quando o mesmo é escolhido para ser ramificado.

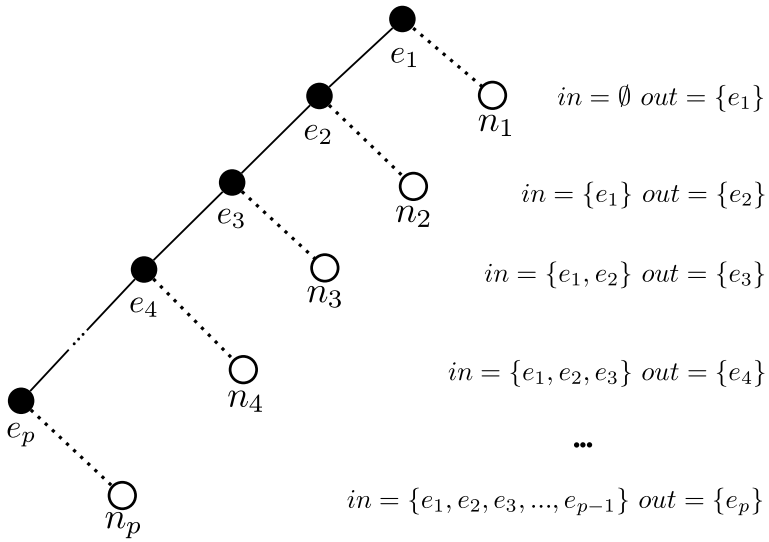

Figura 3.11: Esquema da ramificação

4 Resultados

Neste capítulo, mostramos os resultados dos algoritmos desenvolvidos. Apresentamos os resultados da aplicação da *ASG* para as instâncias Euclidianas tais quais em Andrade e Freitas [3]. Em seguida, utilizando as instâncias Euclidianas e Hamiltonianas [4], mostramos os resultados dos procedimentos desenvolvidos neste trabalho. Ao final, realizamos um comparativo com os resultados de Souza e Martins [29] e os nossos resultados. Pretendemos brevemente comparar nossos resultados com as instâncias usadas por Cunha e Lucena [8].

Os programas foram desenvolvidos em C++ em sistema operacional linux (Ubuntu) utilizando o compilador g++ de versão 4.4.3, e os experimentos computacionais foram realizados em computadores Core 2 Duo com 2.4 GHz e 3GB de RAM. Infelizmente, devido ao elevado tempo computacional para processar todos os conjuntos de instâncias do problema e à dificuldade de disponibilidade de máquinas, optamos por não parametrizar os tempos de *CPU* obtidos na resolução das mesmas utilizando as diferentes máquinas, o que permitiria uma comparação justa com os tempos de *CPU* relatados na literatura. Acreditamos que isso não torna os resultados aqui obtidos menos interessantes, principalmente tendo em vista a qualidade dos mesmos. Dessa forma, evitamos comparar tempos de processamento na obtenção das soluções das instâncias aqui consideradas com os da literatura.

4.1 Instâncias Euclidianas

Nesta seção, mostramos os resultados computacionais para um conjunto de instâncias Euclidianas do problema da *AGMRG*. Tais instâncias foram geradas aleatoriamente no plano euclidiano e introduzidas na literatura por Andrade et al. [4]. Utilizamos as instâncias de até 2.000 vértices. As instâncias possuem restrições de grau d_i nos vértices distintas, inclusive com alguns vértices sendo restritos a folhas $d_i = 1$.

Na Tabela 4.1, vemos que o procedimento da *ASG*, sem o novo esquema de ramificação, já se mostrava competitivo com o algoritmo *SkVNSSO_gr_kstep* de Souza e Martins [29] para o problema AGMRG. Enquanto aquele apresenta intervalo de erro médio de 0,028 para as

instâncias Euclidianas, este apresenta 0,069.

		Heurística SkVNSSO_gr_kstep			ASG			
		UB	seg	erro(%)	LB	UB	seg	erro(%)
Instâncias Euclidianas	100_1	3790	8,66	0,00	3790	3790	0	0,00
	100_2	3829	9,34	0,00	3829	3829	0	0,00
	100_3	3916	6,03	0,00	3916	3916	0	0,00
	200_1	5316	65,41	0,00	5316	5316	0	0,00
	200_2	5651	33,23	0,07	5647	5647	1	0,00
	200_3	5703	25,95	0,09	5698	5698	1	0,00
	300_1	6477	198,78	0,00	6477	6477	2329	0,00
	300_2	6809	91,77	0,03	6807	6807	1034	0,00
	300_3	6435	119,77	0,08	6430	6430	0	0,00
	400_1	7414	469,10	0,00	7414	7414	1	0,00
	400_2	7793	40,32	0,19	7776	7782	*	0,05
	400_3	7614	160,25	0,16	7602	7605	*	0,04
	500_1	8272	802,01	0,00	8272	8272	1	0,00
	500_2	8407	335,18	0,30	8392	8392	8	0,00
	500_3	8506	332,85	0,05	8502	8502	7	0,00
	600_1	9037	619,33	0,00	9037	9037	2	0,00
	700_1	9786	997,99	0,00	9786	9786	159	0,00
	800_1	10333	1761,27	0,00	10333	10333	906	0,00
	900_1	10918	2160,96	0,00	10918	10918	12	0,00
	1000_1	11408	2165,89	0,01	11407	11409	*	0,02
	2000_1	15670	10850,35	0,00	15670	15670	59	0,00
	2000_2	16255	12486,61	0,10	16239	16270	*	0,19
	2000_3	16687	13526,58	0,11	16668	16696	*	0,17
	2000_4	16445	2076,86	0,47	16368	16390	*	0,13
	2000_5	16532	11607,89	0,07	16520	16537	*	0,10
Erro Médio				0,069				0,028

*Execução interrompida ao atingir o limite máximo de tempo (13000 segundos).

Tabela 4.1: Instâncias Euclidianas com a ASG sem o novo esquema de ramificação.

Os resultados da variação *Heurística VNS-Lagrangeana v1* para as instâncias Euclidianas podem ser verificados na Tabela 4.2. Vale ressaltar que os resultados são referentes apenas a uma das variações usadas na pré-fase da *Árvore de Subgradiente* com o novo esquema de ramificação (denotada por *ASGn*), ou seja, as instâncias cujos intervalos de erro não foram fechados ainda são submetidas à *ASGn*. A coluna (LBMetSubg Or) refere-se aos limites inferiores encontrados com a aplicação do método do subgradiente ao problema *AGMRG* original (com todas as arestas do grafo). A coluna (UBMetSubg Re) possui os limites superiores obtidos com o método do subgradiente aplicado ao problema *AGMRG* reduzido, enquanto a coluna (UBSkVNS) contém os limites superiores obtidos pelo procedimento de fuga de ótimos locais.

Comparando, na Tabela 4.2, com os melhores limites superiores de Souza e Martins para essas instâncias na coluna (UB [29]), temos que, dentre as 25 instâncias Euclidianas, em apenas uma (300_2) nosso resultado na pré-fase Lagrangeana (coluna UBSkVNS) foi inferior. Em 7 delas, nosso algoritmo apresentou melhores limites superiores e nas demais os resultados foram

equivalentes. Nossos resultados apresentam intervalo de erro médio de 0,0122 enquanto os resultados apresentados por Souza e Martins [29] (considerando os melhores resultados entre as duas estratégias apresentadas: *VNS* e *SkVNSSO_gr_kstep*) apresentam intervalo de erro médio de 0,0443 (utilizamos os novos limites inferiores obtidos na heurística VNS-Lagrangeana para calcular os intervalos de erro).

Testamos também a variação *Heurística VNS-Lagrangeana v2* com esse grupo de instâncias. Os resultados obtidos foram muito próximos aos da primeira variação e semelhantes em qualidade aos mostrados na Tabela 4.2, por essa razão, optamos por omiti-los. O intervalo de erro médio obtido por essa variação foi 0,0134.

		LBMetSubg Or	UBMetSubg Re	UB [29]	min:sec [29]	UBSkVNS	min:sec	erro(%)
Instâncias Euclidianas	100_1	3790	3790	3790	0:00	3790	0:00	0,000
	100_2	3829	3829	3829	0:00	3829	0:00	0,000
	100_3	3916	3916	3916	0:00	3916	0:01	0,000
	200_1	5316	5316	5316	0:06	5316	0:00	0,000
	200_2	5647	5647	5651	0:08	5647	0:01	0,000
	200_3	5698	5698	5699	0:06	5698	0:01	0,000
	300_1	6475	6477	6477	0:50	6477	0:11	0,030
	300_2	6803	6813	6807	0:55	6809	0:48	0,088
	300_3	6430	6430	6430	0:40	6430	0:01	0,000
	400_1	7414	7414	7414	3:01	7414	0:01	0,000
	400_2	7776	7783	7783	2:33	7783	1:43	0,090
	400_3	7602	7607	7604	3:19	7604	1:24	0,026
	500_1	8272	8272	8272	9:12	8272	0:03	0,000
	500_2	8392	8393	8396	10:56	8392	2:01	0,000
	500_3	8502	8505	8502	8:24	8502	2:07	0,000
	600_1	9037	9037	9037	10:19	9037	0:01	0,000
	700_1	9786	9786	9786	16:37	9786	0:02	0,000
	800_1	10333	10333	10333	29:21	10333	0:07	0,000
	900_1	10918	10918	10918	36:00	10918	0:07	0,000
	1000_1	11407	11413	11408	36:05	11408	2:58	0,008
	2000_1	15670	15673	15670	180:50	15670	12:23	0,000
	2000_2	16239	16273	16255	208:06	16243	59:03	0,024
	2000_3	16668	16702	16687	225:29	16670	54:18	0,011
	2000_4	16368	16398	16445	34:36	16370	64:12	0,012
	2000_5	16520	16546	16532	193:27	16523	37:54	0,018
Erro Médio								0,0122

Tabela 4.2: Resultados - instâncias Euclidianas - Heurística VNS-Lagrangeana v1.

Como podemos verificar na Tabela 4.2, das 25 instâncias Euclidianas, 9 delas apresentam intervalo de erro diferente de 0. Dentre estas, temos instâncias de grande porte com número de vértices variando entre 1000 e 2000. Submetemos tais instâncias à *ASGn*. É importante ressaltar que devido ao grande número de arestas e limitações computacionais, resolvemos trabalhar com as arestas do problema reduzido para as instâncias acima de 1000 vértices. Portanto, na Tabela 4.3, os resultados das instâncias de 300 e 400 vértices são referentes ao problema *AGMRG*

original, enquanto os resultados das instâncias de 1000 e 2000 vértices tratam de limites para o problema *AGMRG* reduzido. Obviamente, os limites superiores para o problema reduzido também valem para o problema original.

	LB	UB	min:sec	erro(%)	Problema AGMRG
300_1	6477	6477	00:01	0,000	Original
300_2	6807	6807	20:01	0,000	Original
400_2	7782	7782	25:12	0,000	Original
400_3	7604	7604	00:09	0,000	Original
1000_1	11407	11408	*	0,008	Reduzido
2000_2	16240	16242	*	0,012	Reduzido
2000_3	16668	16670	*	0,011	Reduzido
2000_4	16369	16370	*	0,006	Reduzido
2000_5	16521	16521	182:18	0,000	Reduzido

*Execução interrompida ao atingir o limite máximo de tempo (1440 minutos).

Tabela 4.3: Resultados - instâncias Euclidianas - ASGn.

Para as instâncias de 300 e 400 vértices, a *ASGn* reduz o intervalo de erro a 0. Como estamos trabalhando com as arestas do problema *AGMRG* original, encontramos a melhor solução e provamos sua otimalidade. Dentre as demais, as instâncias 1000_1 e 2000_2 não apresentam melhores limites superiores em relação aos resultados obtidos na pré-fase. Nas outras, reduzimos o intervalo de erro, encontrando soluções viáveis com custo menor para as instâncias 2000_3, 2000_4 e 2000_5.

Na Tabela 4.4, podemos verificar um comparativo geral das cotas obtidas pela *ASGn* com pré-fase VNS-Lagrangeana e dos resultados mostrados por Cunha e Lucena [9]. Os tempos correspondem ao tempo de execução total utilizado (*ASGn*/pré-fase e *NDRC/BC4* [9]).

Conseguimos atingir os mesmos limites superiores. Das 7 instâncias para as quais não há prova de otimalidade em Cunha e Lucena [9], provamos otimalidade para 3 delas. Nas 4 restantes, atingimos a mesma cota inferior em duas delas e nas outras a diferença é de apenas uma unidade. Quanto ao tempo de execução, em geral, as estratégias de Cunha e Lucena [9] apresentam melhor resultado para as instâncias acima de 1000 vértices.

Em resumo, a *ASGn* com pré-fase VNS-Lagrangeana encontra a solução ótima e prova sua otimalidade em 20 das 25 instâncias Euclidianas. Nas demais, provamos a otimalidade no problema *AGMRG* reduzido para uma instância e o intervalo de erro das outras 4 é bem pequeno, apresentando intervalo de erro médio igual a 0,0022. Note o ganho de qualidade dos resultados quando comparamos aos da Tabela 4.1 para a *ASG* de [3] sem pré-fase VNS-Lagrangeana e sem o novo esquema de ramificação.

		LB	UB	min:sec	erro(%)	LB [9]	UB [9]	min:sec [9]	erro(%) [9]
Instâncias Euclidianas	100_1	3790	3790	0:00	0,000	3790	3790	0:00	0,000
	100_2	3829	3829	0:00	0,000	3829	3829	0:00	0,000
	100_3	3916	3916	0:01	0,000	3916	3916	0:00	0,000
	200_1	5316	5316	0:00	0,000	5316	5316	0:00	0,000
	200_2	5647	5647	0:01	0,000	5647	5647	0:02	0,000
	200_3	5698	5698	0:01	0,000	5698	5698	0:01	0,000
	300_1	6477	6477	0:12	0,000	6477	6477	0:02	0,000
	300_2	6807	6807	20:49	0,000	6805	6807	0:31	0,029
	300_3	6430	6430	0:01	0,000	6430	6430	0:04	0,000
	400_1	7414	7414	0:01	0,000	7414	7414	0:02	0,000
	400_2	7782	7782	26:55	0,000	7779	7782	0:32	0,038
	400_3	7604	7604	1:33	0,000	7603	7604	0:38	0,013
	500_1	8272	8272	0:03	0,000	8272	8272	0:06	0,000
	500_2	8392	8392	2:01	0,000	8392	8392	0:18	0,000
	500_3	8502	8502	2:07	0,000	8502	8502	0:26	0,000
	600_1	9037	9037	0:01	0,000	9037	9037	0:09	0,000
	700_1	9786	9786	0:02	0,000	9786	9786	0:13	0,000
	800_1	10333	10333	0:07	0,000	10333	10333	0:25	0,000
	900_1	10918	10918	0:07	0,000	10918	10918	0:29	0,000
	1000_1	11407	11408	*	0,008	11407	11408	2:48	0,008
	2000_1	15670	15670	12:23	0,000	15670	15670	4:32	0,000
	2000_2	16239	16242	*	0,018	16242	16242	45:12	0,000
	2000_3	16668	16670	*	0,011	16669	16670	68:56	0,005
	2000_4	16368	16370	*	0,012	16369	16370	70:32	0,006
	2000_5	16520	16521	220:12	0,006	16520	16521	47:57	0,006
Erro Médio					0,0022				0,0042

*Execução interrompida ao ultrapassar o limite máximo de tempo (1440 minutos).

Tabela 4.4: Resultados - Comparativo com Cunha e Lucena [9].

4.2 Instâncias Hamiltonianas

Mostramos, nesta seção, os resultados computacionais para as instâncias Hamiltonianas para o problema da *AGMRG*. Essas instâncias também foram geradas aleatoriamente no plano euclidiano e introduzidas por Andrade et al. [4]. As instâncias possuem restrições de grau d_i nos vértices definidas entre {1,2}, com a restrição de que não mais que 2 vértices possuem restrição de grau igual a 1. Trabalhamos, portanto, com 3 tipos de instâncias Hamiltonianas:

***Sem restrição de folhas*:** Todos os vértices apresentam restrição de grau igual a 2.

***Com restrição de uma folha*:** Exatamente um vértice apresenta restrição de grau igual a 1, todos os demais possuem restrição de grau igual a 2.

***Com restrição de duas folhas*:** Exatamente dois vértices apresentam restrição de grau igual a 1, todos os demais possuem restrição de grau igual a 2 (nesse tipo de instância, o *VND-Dinâmico* apresenta vantagem em relação ao *VND* tradicional, embora o aumento do

número dos elementos nas vizinhanças também acarrete melhora em outros tipos de instância).

Inicialmente, testamos apenas as estratégias utilizadas na pré-fase da *ASGn*: a *Heurística VNS-Lagrangeana v1* e *Heurística VNS-Lagrangeana v2*. Os resultados de cada variação são mostrados nas Tabelas 4.5 e 4.6, respectivamente.

Os resultados da primeira variação apresentam intervalo de erro médio de 1,909 contra 1,098 dos resultados obtidos pelas estratégias utilizadas por Souza e Martins [29], enquanto a segunda variação apresenta intervalo de erro médio de 1,644 contra 1,097 deles. Tais intervalos médios são calculados utilizando os limites inferiores obtidos nos nossos testes e que são mostrados em cada tabela. Para essas instâncias, a *Heurística VNS-Lagrangeana v2* apresenta, em geral, melhores cotas superiores que a versão *v*1.

		LB MetSubg Or	UB MetSubg Re	UB SkVNS	min:sec	erro(%)
Instâncias Hamiltonianas	100	4274	4449	4334	0:28	1,403
	100_1	3874	3884	3883	0:37	0,232
	100_2	4419	4743	4489	1:02	1,584
	200	5599	5819	5652	5:36	0,946
	200_1	5697	6086	5769	6:08	1,263
	200_2	5971	6169	6035	18:35	1,071
	300	6860	7382	7024	32:45	2,390
	300_1	6850	7370	7058	47:48	3,036
	300_2	7087	7821	7360	54:26	3,852
	400	7924	8452	8128	270:00	2,574
	400_1	8123	8647	8254	585:21	1,612
	400_2	7734	8271	7898	215:49	2,120
	500	8771	9360	8961	16:58	2,166
	500_1	8815	9313	9013	1303:30	2,246
	500_2	8733	9602	8921	2046:45	2,152
Erro Médio						1,909

Tabela 4.5: Resultados - instâncias Hamiltonianas - Heurística VNS-Lagrangeana v1.

A Tabela 4.7 mostra um comparativo entre os limites primais obtidos utilizando nossas variações (utilizadas na pré-fase da *ASGn*) e os melhores limites superiores encontrados na literatura para essas instâncias, obtidos utilizando um dos três métodos de Souza e Martins [29].

Os limites superiores se mostram competitivos em relação aos apresentados por Souza e Martins [29] ainda na pré-fase da *ASGn*. Nosso intervalo de erro médio é 1,608, considerando as melhoras cotas obtidas em cada variação, e obtemos inclusive melhores limites superiores para as instâncias h400_1 e h500.

Na Tabela 4.8, apresentamos os resultados da *ASGn* para essas instâncias.

		LB MetSubg Or	UB MetSubg Re	UB SkVNS	min:sec	erro(%)
Instâncias Hamiltonianas	100	4274	4464	4334	0:26	1,403
	100_1	3872	3909	3886	0:27	0,361
	100_2	4419	4743	4498	0:59	1,787
	200	5599	5847	5650	14:16	0,910
	200_1	5698	6072	5757	8:05	1,035
	200_2	5971	6169	6039	16:30	1,138
	300	6860	7306	7019	46:19	2,317
	300_1	6849	7325	6980	84:27	1,912
	300_2	7087	7821	7320	77:33	3,287
	400	7924	8404	8078	255:28	1,943
	400_1	8124	8460	8198	237:19	0,910
	400_2	7734	8271	7898	161:54	2,120
	500	8773	9338	8868	6:36	1,082
	500_1	8815	9332	9026	634:10	2,393
	500_2	8733	9602	8914	2306:49	2,072
Erro Médio						1,644

Tabela 4.6: Resultados - instâncias Hamiltonianas - Heurística VNS-Lagrangeana v2.

Comparando os resultados obtidos na *ASGn* da Tabela 4.8 para o grupo de instâncias de 100 vértices com os resultados obtidos na pré-fase da Tabela 4.7, podemos verificar melhora nos limites superiores encontrados. Contudo, tais valores já haviam sido encontrados por Souza e Martins [29]. Ao se tratar das cotas inferiores, conseguimos provar a otimalidade de todas as instâncias desse primeiro grupo.

Quanto ao grupo de instâncias de 200 vértices, a *ASGn* consegue melhorar as cotas superiores obtidas na pré-fase. Ao compararmos com os limites superiores de Souza e Martins, obtemos melhor resultado para a instância h200 (provando otimalidade) e um pouco pior para as outras duas instâncias, onde a contribuição foi na melhora das cotas inferiores.

Para as demais instâncias, a *ASGn* não melhora os limites primais encontrados na pré-fase; contudo, nos fornece melhores limites inferiores para todas as instâncias, permitindo reduzir o intervalo de erro.

Em suma, a *ASGn* com pré-fase VNS-Lagrangeana nos permite provar a otimalidade de 4 instâncias, 3 delas cujos limites superiores já eram conhecidos e em uma delas tal limite foi melhorado. Além disso, conseguimos obter melhores cotas inferiores para todas as outras instâncias, melhorando a cota superior de duas delas. Ao final, nosso intervalo de erro médio para as instâncias Hamiltonianas é 1,1003 contra 0,6387 de Souza e Martins [29] (utilizando nossas novas cotas inferiores para o problema).

		UB H.VNSLagrV1	UB H.VNSLagrV2	UB [29]	min:seg [29]
Instâncias Hamiltonianas	100	4334	4334	4315	0:54
	100_1	3883	3886	3883	0:16
	100_2	4489	4498	4482	1:54
	200	5652	5650	5648	3:09
	200_1	5769	5757	5745	4:50
	200_2	6035	6039	6017	21:12
	300	7024	7019	6982	13:15
	300_1	7058	6980	6926	13:00
	300_2	7360	7320	7165	92:38
	400	8128	8078	8014	33:36
	400_1	8254	8198	8206	37:25
	400_2	7898	7898	7844	159:11
	500	8961	8868	8884	49:46
	500_1	9013	9026	8904	41:33
	500_2	8921	8914	8862	275:08

Tabela 4.7: Comparativo - Limites primais.

4.3 Instâncias DE e DR

As instâncias DE e DR são introduzidas por Cunha e Lucena [8]. As instâncias DE são geradas no plano Euclidiano como sugerido em Andrade et al. [4] com restrições de grau d_i variando no conjunto $\{1,2,3\}$. Já as DR são originadas a partir das DE ao ignorar os custos Euclidianos em favor de custos aleatórios obtidos a partir de uma distribuição uniforme no intervalo $[1,1000]$.

A Tabela 4.9 mostra um comparativo entre a *ASGn* usando a *heurística VNS-Lagrangeana v2* na pré-fase e o *NDRC/BC4* proposto por Cunha e Lucena [9] para as instâncias DE. Nossa contribuição, para esse grupo de instâncias, é a melhoria das cotas inferiores para a maioria das instâncias. A qualidade dos limites superiores é próxima mas ainda inferior à mostrada por Cunha e Lucena. Nosso erro médio é 0,3392 e o *NDRC/BC4* apresenta erro médio de 0,1091.

A Tabela 4.10 mostra os resultados para as instâncias DR. A *ASGn* encontra bons limites superiores e prova otimalidade para todas as instâncias do grupo. Constatamos assim que os limites superiores encontrados por Cunha e Lucena são ótimos, embora o erro médio apresentado seja 0,0619.

		LB	UB	min:sec	erro(%)	Problema AGMRG
Instâncias Hamiltonianas	100	4315	4315	168:01	0,000	Original
	100_1	3883	3883	01:18	0,000	Original
	100_2	4482	4482	1231:55	0,000	Original
	200	5643	5643	1492:36	0,000	Original
	200_1	5730	5755	*	0,436	Original
	200_2	6001	6034	*	0,549	Original
	300	6892	7019	*	1,842	Original
	300_1	6874	6980	*	1,542	Original
	300_2	7111	7320	*	2,939	Original
	400	7941	8078	*	1,725	Original
	400_1	8139	8198	*	0,724	Original
	400_2	7761	7898	*	1,765	Original
	500	8789	8868	*	0,898	Original
	500_1	8828	9013	*	2,095	Original
	500_2	8740	8914	*	1,990	Original
Erro Médio		1,1003				

*Execução interrompida ao atingir o limite máximo de tempo (2800 minutos).

Tabela 4.8: Resultados - instâncias Hamiltonianas - ASGn.

		ASGn com pré-fase VNS-LAGRv2				NDRC/BC4 [9]			
		LB	UB	min:sec	erro(%)	LB	UB	min:sec	erro(%)
Instâncias DE	100_1	8779	8779	00:50	0,000	8779	8779	00:05	0,000
	100_2	9629	9629	02:51	0,000	9629	9629	00:04	0,000
	100_3	10798	10798	23:33	0,000	10784	10798	00:18	0,129
	200_1	12031	12031	62:46	0,000	12029	12031	00:54	0,016
	200_2	12628	12665	*	0,292	12618	12645	03:13	0,213
	200_3	12229	12229	52:27	0,000	12211	12229	01:41	0,147
	300_1	14791	14796	*	0,033	14770	14792	04:20	0,148
	300_2	13931	13931	439:50	0,000	13921	13931	01:29	0,071
	300_3	14968	15009	*	0,273	14947	14997	05:10	0,334
	400_1	19239	19239	1731:44	0,000	19235	19239	16:29	0,020
	400_2	17404	17479	*	0,430	17387	17419	26:20	0,184
	400_3	16091	16091	5929:33	0,000	16072	16091	09:44	0,118
	500_1	19352	19445	*	0,480	19347	19357	13:14	0,051
	500_2	22397	22754	*	1,593	22395	22400	59:18	0,022
	500_3	19380	19493	*	0,583	19381	19411	130:11	0,154
	600_1	21911	22039	*	0,584	21908	21930	129:23	0,100
	600_2	21403	21663	*	1,214	21410	21449	199:26	0,182
	600_3	22229	22368	*	0,625	22226	22243	88:23	0,076
Erro Médio		0,3392							0,1091

*Execução interrompida ao atingir o limite máximo de tempo (6480 minutos).

Tabela 4.9: Resultados - instâncias DE - ASGn e NDRC/BC4.

		ASGn com pré-fase VNS-LAGRv2				NDRC/BC4 [9]			
		LB	UB	min:sec	erro(%)	LB	UB	min:sec	erro(%)
Instâncias DR	100_1	2186	2186	00:24	0,000	2178	2186	00:08	0,367
	100_2	2674	2674	00:02	0,000	2674	2674	00:06	0,000
	100_3	2689	2689	00:18	0,000	2689	2689	00:08	0,000
	200_1	2141	2141	00:45	0,000	2139	2141	01:01	0,093
	200_2	2471	2471	01:57	0,000	2470	2471	01:14	0,040
	200_3	2405	2405	01:41	0,000	2403	2405	01:02	0,083
	300_1	2462	2462	04:01	0,000	2460	2462	03:09	0,081
	300_2	2312	2312	03:32	0,000	2312	2312	01:47	0,000
	300_3	2585	2585	05:11	0,000	2584	2585	07:03	0,038
	400_1	2817	2817	70:59	0,000	2815	2817	23:15	0,071
	400_2	2443	2443	75:57	0,000	2441	2443	11:53	0,081
	400_3	2387	2387	06:21	0,000	2387	2387	02:41	0,000
	500_1	2597	2597	60:22	0,000	2596	2597	16:47	0,038
	500_2	2652	2652	58:01	0,000	2650	2652	87:13	0,075
	500_3	2593	2593	34:49	0,000	2592	2593	18:18	0,038
	600_1	2820	2820	82:13	0,000	2820	2820	36:00	0,000
	600_2	2662	2662	693:31	0,000	2660	2662	137:36	0,075
	600_3	2800	2800	216:42	0,000	2799	2800	54:56	0,035
Erro Médio					0,0000				0,0619

Tabela 4.10: Resultados - instâncias DR - ASGn e NDRC/BC4.

5 *Conclusão*

Neste trabalho, uma de nossas contribuições é a incorporação de mecanismos de fuga de ótimos locais à heurística Lagrangeana de Andrade et al. [4].

Com esse objetivo, elaboramos e implementamos diversas técnicas cujos resultados não apresentam qualidade suficiente para competir com as melhores abordagens da literatura. Por essa razão, tais estratégias foram omitidas neste trabalho. Para efeito de conhecimento, eis algumas: *K-Shake*, semelhante ao *VNS reduzido* por não aplicar busca local; *2-Opt Adaptado*, uma variação do *2-Opt*, onde inserimos fatores de absorção; *TSP Moves*, busca local que explorava algumas trocas de arestas próprias do problema do caixeiro viajante.

Os bons resultados obtidos por Souza e Martins nos apontaram uma direção. Procuramos reproduzir seus resultados, implementando o *Skewed VNS* [29] com os Algoritmos de Segunda Ordem, *VND* e funções de absorção. Porém, não obtivemos limites superiores com a mesma qualidade que os encontrados por eles. Com isso, começamos a realizar variações no esquema proposto. Ao final, chegamos a uma variação do *VNS* que escolhe simplesmente um vizinho arbitrário na fase de agitação (diminuindo assim o tempo computacional utilizado), utiliza o *VND-Dinâmico* na busca local (o que permite trabalhar com as instâncias Hamiltonianas com restrição de duas folhas) e faz uso das funções de absorção propostas por Souza e Martins (permitindo a aceitação de árvores baseada não somente na verificação dos seus custos mas também verificando suas estruturas pela leitura dos graus dos vértices). Incorporamos essa última variação à heurística de Andrade et al. [4] e passamos a chamá-la de *heurística VNS-Lagrangeana*.

A *heurística VNS-Lagrangeana* por si só mostra-se bastante competitiva. Para as instâncias Euclidianas, nossos resultados ou se igualam ou são melhores que os de Souza e Martins [29] (salvo uma instância dentre as 25). Para as instâncias Hamiltonianas, os resultados da *heurística VNS-Lagrangeana*, embora próximos, ainda não são melhores que os resultados de Souza e Martins [29] (salvo para duas instâncias das 15).

Contudo, utilizamos a *heurística VNS-Lagrangeana* como pré-fase de um método exato.

Nossa outra contribuição é a Árvore de Subgradiente (*ASG*). Aperfeiçoamos esse conceito introduzido por nós em 2008 [3] e que já mostrava bons resultados para as instâncias Euclidianas. Propomos um novo esquema de ramificação que elimina a solução do nó pai na árvore de busca, o que não era possível no antigo esquema de ramificação por dicotomia. Denotamos a *ASG* com a nova ramificação por *ASGn*. Esse esquema de particionamento pode ser utilizado em problemas com estruturas semelhantes aos de árvore, como por exemplo o problema de Árvore Geradora Mínima com Restrição Mínima de Grau, md-MST (do inglês Min-Degree Constrained Minimum Spanning Tree) [1] que pretendemos tratar brevemente. Uma melhora que pode ser realizada no método é trabalhar em paralelo com diferentes partições do problema.

A *ASGn* obteve bons resultados para as instâncias Euclidianas que continuaram em aberto após a pré-fase. Reduzimos o intervalo de erro médio das instâncias a praticamente zero, provando otimalidade para todas as instâncias com número de vértices inferior a 1000 e para uma de 2000 vértices. As demais apresentam intervalo de erro muito pequeno. Tais resultados são bastante competitivos e equivalentes aos mostrados por Cunha e Lucena [9]. Para as instâncias Hamiltonianas, conseguimos provar otimalidade para o grupo de 100 vértices e para uma de 200, a principal contribuição para as demais instâncias foi na melhora da cota inferior. Para estas, os melhores limites superiores continuam sendo encontrados pelas estratégias de Souza e Martins [29].

Aplicamos também a *ASGn* para outros 2 conjuntos de instâncias conhecidas da literatura [8]: as instâncias DE e DR. Nas instâncias DE, nossa contribuição é a melhora da qualidade dos limites inferiores. No segundo grupo, provamos otimalidade para todas as instâncias. Acreditamos assim ter desenvolvido um algoritmo muito eficiente para o problema da *AGMRG* que se mostra competitivo com os melhores algoritmos da literatura.

Referências Bibliográficas

[1] ALMEIDA, A. M., MARTINS, P., AND SOUZA, M. Min-degree constrained minimum spanning tree problem: Complexity, properties and formulations. Tech. Rep. 6/2006, CIO, 2006.

[2] ANDRADE, R. C., AND FREITAS, A. T. Subgradient tree optimization. In *Proc. 19th International Symposium on Mathematical Programming* (2006), pp. 76–77.

[3] ANDRADE, R. C., AND FREITAS, A. T. Otimização em árvore de subgradiente para a árvore geradora mínima com restrição de grau nos vértices. *XL SOBRAPO* (2008), 1751–1759.

[4] ANDRADE, R. C., LUCENA, A., AND MACULAN, N. Using lagrangian dual information to generate degree constrained spanning trees. *Discrete Appl. Math. 154*, 5 (2006), 703–717.

[5] BOLDON, B., DEO, N., AND KUMAR, N. Minimum-weight degree-constrained spanning tree problem: heuristics and implementation on an simd parallel machine. *Parallel Computing 22* (March 1996), 369–382.

[6] CACCETTA, L., AND HILL, S. A branch and cut method for the degree-constrained minimum spanning tree problem. *Networks 37*, 2 (2001), 74–83.

[7] CRAIG, G., KRISHNAMOORTHY, M., AND PALANISWAMI, M. Comparison of heuristics algorithms for the degree constrained minimum spanning tree. In *Metaheuristics: Theory and Applications - Osman, Ibrahim H.; Kelly, James P. (Eds.)* (Boston, MA, USA, March 1996), Kluwer Academic Publishers, pp. 83–96.

[8] CUNHA, A. S., AND LUCENA, A. Lower and upper bounds for the degree-constrained minimum spanning tree problem. *Networks 50* (August 2007), 55–66.

[9] CUNHA, A. S., AND LUCENA, A. A hybrid relax-and-cut/branch and cut algorithm for the degree-constrained minimum spanning tree problem. Tech. Rep. 6/2008, Departamento de Administração, Universidade Federal do Rio de Janeiro, 2008.

[10] GAREY, M. R., AND JOHNSON, D. S. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[11] GAVISH, B. Topological design of centralized computer networks - formulations and algorithms. *Networks 12*, 4 (1982), 355–377.

[12] HANSEN, P., JAUMARD, B., MLADENOVIĆ, N., AND PARREIRA, A. Variable neighborhood search for weight satisfiability problem. *Les Cahiers du GERARD - G-2000-62* (November 2000).

[13] HANSEN, P., AND MLADENOVIĆ, N. Developments of variable neighborhood search. In *Essay and Surveys in Metaheuristics - Ribeiro, C.C., Hansen, P. (Eds.)* (Norwell, MA, USA, 2002), Kluwer Academic Publishers, pp. 415–439.

[14] HANSEN, P., AND MLADENOVIĆ, N. Variable neighborhood search. In *Handbook of Metaheuristics - Glover, F., Kochenberger, G. (Eds)* (Norwell, MA, USA, January 2003), Kluwer Academic Publishers, pp. 145–184.

[15] HELD, M., AND KARP, R. M. The traveling salesman problem and minimum spanning trees. *Operations Research 18*, 6 (November-December 1970), 1138–1162.

[16] HELD, M., AND KARP, R. M. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming 1* (1971), 6–25. 10.1007/BF01584070.

[17] HELD, M., WOLFE, P., AND CROWDER, H. P. Validation of subgradient optimization. *Math. Programming 6* (1974), 62–88.

[18] KARNAUGH, M. A new class of algorithms for multipoint network optimization. *Communications, IEEE Transactions on 24*, 5 (May 1976), 500 – 505.

[19] KNOWLES, J., AND CORNE, D. A new evolutionary approach to the degree-constrained minimum spanning tree problem. *Evolutionary Computation, IEEE Transactions on 4*, 2 (July 2000), 125 –134.

[20] KRISHNAMOORTHY, M., ERNST, A. T., AND SHARAIHA, Y. M. Comparison of algorithms for the degree constrained minimum spanning tree. *Journal of Heuristics 7* (November 2001), 587–611.

[21] KRUSKAL, J. B. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society 7*, 1 (February 1956), 48–50.

[22] LI, Y. An effective implementation of a direct spanning tree representation in gas. In *Proceedings of the EvoWorkshops on Applications of Evolutionary Computing* (London, UK, UK, 2001), Springer-Verlag, pp. 11–19.

[23] MLADENOVIĆ, N., AND HANSEN, P. Variable neighborhood search. *Comput. Oper. Res. 24*, 11 (1997), 1097–1100.

[24] NARULA, S. C., AND HO, C. A. Degree-constrained minimum spanning tree. *Computers & Operations Research 7*, 4 (1980), 239 – 249.

[25] PRIM, R. C. Shortest connection networks and some generalizations. *Bell System Technology Journal 36* (November 1957), 1389–1401.

[26] RAIDL, G. R., AND JULSTROM, B. A. A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. In *Proceedings of the 2000 ACM symposium on Applied computing - Volume 1* (New York, NY, USA, 2000), SAC '00, ACM, pp. 440–445.

[27] RIBEIRO, C. C., AND SOUZA, M. C. Variable neighborhood search for the degree-constrained minimum spanning tree problem. *Discrete Appl. Math. 118*, 1-2 (2002), 43–54. Third ALIO-EURO Meeting on Applied Combinatorial Optimization (Erice, 1999).

[28] SAVELSBERGH, M., AND VOLGENANT, T. Edge exchanges in the degree-constrained minimum spanning tree problem. *Computers & Operations Research 12*, 4 (1985), 341 – 348.

[29] SOUZA, M. C., AND MARTINS, P. Skewed vns enclosing second order algorithm for the degree constrained minimum spanning tree problem. *European Journal of Operational Research 191*, 3 (2008), 677 – 690.

[30] VOLGENANT, A. A Lagrangean approach to the degree-constrained minimum spanning tree problem. *European J. Oper. Res. 39*, 3 (1989), 325–331.

[31] ZHOU, G., AND GEN, M. A note on genetic algorithms for degree-constrained spanning tree problems. *Networks 30*, 2 (1997), 91–95.