UNIVERSIDADE FEDERAL FLUMINENSE

ROGÉRIO MELO NEPOMUCENO

ITERATED LOCAL SEARCH PARA O PROBLEMA DA ÁRVORE GERADORA EUCLIDIANA MÍNIMA COM DIÂMETRO LIMITADO

Niterói

2018

ROGÉRIO MELO NEPOMUCENO

Iterated Local Search para o Problema da Árvore Geradora Euclidiana Mínima com Diâmetro Limitado

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Doutor. Área de Concentração: Algoritmos e Otimização.

Orientadores:

Prof. Dr. Fábio Protti

Profª Drª Isabel Cristina Mello Rosseti

Niterói

2018

Ficha catalográfica automática - SDC/BEE

N441i Nepomuceno, Rogério Melo
Iterated Local Search para o Problema da Árvore Geradora Euclidiana Mínima com Diâmetro Limitado / Rogério Melo Nepomuceno ; Fábio Protti, orientador ; Isabel Cristina Mello Rosseti, coorientadora. Niterói, 2018.
102 f.

Tese (doutorado)-Universidade Federal Fluminense, Niterói, 2018.

DOI: http://dx.doi.org/10.22409/PGC.2018.d.51747278653

1. Heurísitca. 2. Grafo. 3. Otimização (Computação). 4. Produção intelectual. I. Título II. Protti,Fábio, orientador. III. Rosseti, Isabel Cristina Mello, coorientadora. IV. Universidade Federal Fluminense. Escola de Engenharia.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

ROGÉRIO MELO NEPOMUCENO

Iterated Local Search para o Problema da Árvore Geradora Euclidiana Mínima com Diâmetro Limitado

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Doutor. Área de Concentração: Algoritmos e Otimização.

Aprovada em novembro de 2018.

BANCA EXAMINADORA

Prof. Dr. Fábio Protti (Orientador) – Universidade Federal Fluminense

Prof. Dra. Isabel Cristina Mello Rosseti (Coorientadora) – Universidade Federal Fluminense

Prof. Dr. Luís Felipe Ignácio Cunha – Universidade Federal do Rio de Janeiro

Prof. Dr. Mitre Costa Dourado – Universidade Federal do Rio de Janeiro

Prof. Dra. Sulamita Klein – COPPE/Universidade Federal do Rio de Janeiro

Prof. Dr. Carlos Alberto de Jesus Martinhon – Universidade Federal Fluminense

Profa. Dra. Simone de Lima Martins – Universidade Federal Fluminense

Niterói

2018

Dedico este trabalho à minha família.

AGRADECIMENTOS

Agradeço, em primeiro lugar, a Deus, que me iluminou nesta caminhada. Agradeço a minha família, que sempre esteve ao meu lado nos momentos mais difíceis. Agradeço também aos meus orientadores Fábio Protti e Isabel Rosseti, que me guiaram nesta jornada.

"*A tarefa não é tanto ver aquilo que ninguém viu, mas pensar o que ninguém ainda pensou sobre aquilo que todo mundo vê.*" (Arthur Schopenhauer)

RESUMO

A árvore geradora de custo mínimo de um grafo G não direcionado, conexo e com custos nas arestas é uma árvore geradora onde o somatório dos custos das arestas é mínimo. O diâmetro dessa árvore é definido como o número máximo de arestas dentre todos os caminhos da árvore. O problema da árvore geradora mínima com diâmetro limitado (*Bounded-Diameter Minimum Spanning Tree Problem* – BDMSTP) consiste em encontrar uma árvore geradora para G cujo diâmetro não exceda a um limite D, $2 \leq D \leq n - 1$, onde n é o número de vértices de G, e que tenha custo mínimo. Esse problema é NP-difícil para $4 \leq D < n - 1$. Aplicações da árvore geradora de custo mínimo com diâmetro limitado surgem em problemas de engenharia, como projetos de redes de telecomunicações e de fibra ótica, problemas de compressão de dados e projeto de sistemas distribuídos, quando considerada a exclusão múltipla. Métodos exatos para resolver o BDMSTP só conseguem resolver pequenas instâncias do problema, com no máximo 161 vértices e diâmetro menor ou igual a oito. Desse modo, heurísticas construtivas e heurísticas baseadas em metaheurísticas tentam encontrar uma solução próxima do ótimo para o problema em um tempo aceitável. Se os vértices do grafo de entrada estão associados a pontos no plano Cartesiano, temos o *Euclidean Bounded-Diameter Minimum Spanning Tree Problem* – EBDMSTP. Esta tese apresenta e analisa as heurísticas mais conhecidas na literatura para o EBDMSTP, propõe uma nova heurística construtiva para o problema, e ainda outra heurística baseada no método *Iterated Local Search* (ILS), analisando os seus resultados com os existentes na literatura.

Palavras-chave: Árvore geradora, Árvore geradora Euclidiana de custo mínimo com diâmetro limitado, Iterated Local Search, ILS, Heurísticas.

ABSTRACT

The minimum spanning tree of an undirected graph G is a spanning tree where the sum of the edge costs is minimum. The diameter of this tree is set as the maximum number of edges in a path of the tree. The *Bounded-Diameter Minimum Spanning Tree Problem* (BDMSTP) consists of finding a spanning tree for G whose diameter does not exceed a limit D, $2 \leq D \leq n$ - 1, where n is the number of vertices of G, and has minimum cost. This problem is NP-hard for $4 \leq D <$ n – 1. Applications of the bounded-diameter minimum spanning tree arise in engineering problems, such as projects of telecommunication networks and optical fibers, data compression problems, and design of distributed systems, when considering multiple exclusion. Exact methods to solve the BDMST can only deal with small instances, up to 161 vertices and diameter less or equal to 8. Thus, constructive heuristics and heuristic-based metaheuristics try to find solutions close to the optimal in an acceptable time. If the vertices of the input graph are associated with points in the Cartesian plane, we have the *Euclidean Bounded-Diameter Minimum Spanning Tree Problem* (EBDMSTP). This thesis presents and analyzes the best known heuristics in the literature for the EBDMSTP, proposes a new constructive heuristic for the problem, and a new heuristic based on the Iterated Local Search (ILS) method, comparing the results with those existing in the literature.

Keywords: Spanning trees, Euclidean bounded-diameter minimum spanning tree, Iterated Local Search, ILS, Heuristics.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

SUMÁRIO

CAPÍTULO 1 – INTRODUÇÃO

Seja $G = (V, E)$ um grafo, com $n = |V|$ vértices, $m = |E|$ arestas e um valor real positivo w_{uv} associado a cada aresta $(u, v) \in E$ que indica o custo de ir de um vértice u a v.

A árvore geradora de custo mínimo de um grafo não direcionado conexo com custos nas arestas (*minimum spanning tree* – MST) é uma árvore geradora[1] onde o somatório dos custos das arestas da árvore é mínimo. Destacam-se dois algoritmos para determinar a MST em tempo polinomial: o algoritmo de Prim (PRIM, 1957)[2] e o algoritmo de Kruskal (KRUSKAL, 1956)[3]. Outros exemplos de algoritmos de tempo polinomial para o cálculo da MST são o de Borůvka (NEÄ; MILKOVÃ; NEÄ, 2001) e o de Karger (KARGER; KLEIN; TARJAN, 1995).

Em uma árvore a excentricidade de um vértice v é o número máximo de arestas em um caminho a partir de v para outro vértice na árvore. A

Figura 1 ilustra a excentricidade de cada vértice de uma árvore. Dado um grafo G não direcionado, conexo e com custos nas arestas, o diâmetro da sua árvore geradora é definido como o número máximo de arestas dentre todos os caminhos da árvore (isto é, a excentricidade máxima). O problema da árvore geradora mínima com diâmetro limitado (*Bounded-Diameter Minimum Spanning Tree Problem* – BDMSTP) consiste em encontrar uma árvore geradora para G cujo diâmetro não exceda a um limite D, $2 \leq D \leq n - 1$, e que tenha custo mínimo. Esse problema é NP-difícil para $4 \leq D < n - 1$ (GAREY; JOHNSON, 1979).

---

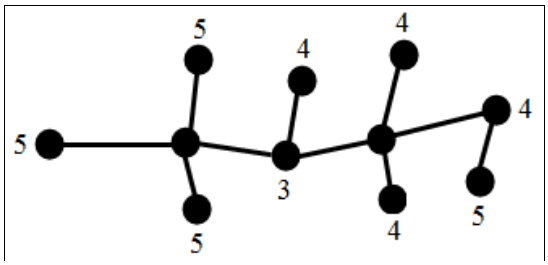
[1] Uma *árvore geradora* de um grafo G, não direcionado e conexo, é qualquer subgrafo de G que seja uma árvore e que contenha todos os vértices de G.

[2] O algoritmo de Prim inicia a construção da árvore geradora mínima a partir de um vértice inicial arbitrário e repetitivamente adiciona novos vértices através de arestas de menor custo que conectam um vértice que não está na árvore a outro que está.

[3] Para a construção da árvore geradora mínima pelo algoritmo de Kruskal, primeiro as arestas do grafo são incluídas em uma lista ordenada pelos custos. A seguir, percorre-se ordenadamente a lista e adiciona-se cada aresta à árvore, desde que isto não produza um ciclo.

Aplicações do BDMSTP surgem em problemas de engenharia, como projetos de redes de telecomunicações e de fibra ótica (BALA; PETROPOULOS; STERN, 1993), problemas de compressão de dados (BOOKSTEIN; KLEIN, 1991) e projeto de sistemas distribuídos, quando considerada a exclusão múltipla (RAYMOND, 1989).

Figura 1: Exemplo de Excentricidade.



Métodos exatos para resolver o BDMSTP, os quais só podem ser aplicados a instâncias relativamente pequenas, com no máximo 161 vértices e diâmetro menor ou igual a oito, são encontrados em (SANTOS; LUCENA; RIBEIRO, 2004), (GRUBER; RAIDL, 2005a), (GRUBER; RAIDL, 2008) e (GOUVEIA; SIMONETTI; UCHOA, 2011). Desse modo, heurísticas construtivas e heurísticas baseadas em metaheurísticas tentam encontrar para o problema uma solução próxima do ótimo em um tempo aceitável, ou até mesmo encontrar uma solução ótima.

No *Problema da Árvore Geradora Euclidiana Mínima com Diâmetro Limitado* (*Euclidian Bounded-Diameter Minimum Spanning Tree Problem* – EBDMSTP[4]), os vértices do grafo são associados com pontos no plano cartesiano, e cada custo w_{uv} é precisamente a distância euclidiana entre os vértices u e v; neste caso assume-se que todas as possíveis distâncias são dadas, isto é, o grafo de entrada é completo, correspondendo a uma "instância Euclidiana".

[4] Neste caso a árvore gerada é denominada de EBDMST (*Euclidian Bounded-Diameter Minimum Spanning Tree*).

As heurísticas construtivas para o BDMSTP e o EBDMSTP são normalmente baseadas no algoritmo de Prim (PRIM, 1957), o qual determina a árvore geradora mínima. Elas iniciam a partir de um vértice, considerado como vértice central (se o diâmetro é par), ou a partir de uma aresta (se o diâmetro é ímpar), e conectam, iterativamente, os demais vértices com as arestas de menor custo, desde que a restrição de diâmetro não seja violada. Como exemplo, podem ser citados os seguintes algoritmos, todos considerados como algoritmos gulosos, que são bem conhecidos na literatura: OTTC – *One-Time Tree Construction* (ABDALLA; DEO; GUPTA, 2000), CBTC – *Center Base Tree Constrution* (JULSTROM, 2009), RGH – *Random Greedy Heuristic* (RAIDL; JULSTROM, 2003a), RTC – *Randomized Center-Based Tree Construction* (JULSTROM, 2009) e CBRC – *Center-Based Recursive Clustering* (BINH et al., 2009).

Várias heurísticas baseadas em metaheurísticas também foram propostas para resolver o EBDMSTP, visando encontrar soluções melhores que as dos algoritmos construtivos. Dentre elas, podem-se citar os algoritmos evolucionários (JULSTROM; RAIDL, 2003; RAIDL; JULSTROM, 2003a; SINGH; GUPTA, 2007), *Variable Neighborhood Search* (GRUBER; RAIDL, 2005b) e algoritmos híbridos, como o *Greedy Randomized Adaptive Search* – GRASP – combinado com o *Iterated Local Search* – ILS (LUCENA; RIBEIRO; SANTOS, 2010).

O ILS é uma metaheurística que tem sido utilizada com sucesso em problemas de otimização nas áreas acadêmica e industrial (LOURENÇO; MARTIN; STÜTZLE, 2003). Algumas áreas de atuação bem sucedidas do ILS são o roteamento heterogêneo de veículos (PENNA; SUBRAMANIAN; OCHI, 2013), roteamento de veículos com múltiplas viagens (DA SILVA; DE MENEZES FROTA; SUBRAMANIAN, 2012), formação de células de manufatura (MARTINS et al., 2015), coloração de grafos (CHIARANDINI; STÜTZLE, 2002), problemas de escalonamento (DONG; HUANG; CHEN, 2009), entre outros exemplos.

Neste trabalho é apresentada uma heurística híbrida para o EBDMSTP baseada no ILS (LOURENÇO; MARTIN; STÜTZLE, 2003) com uso do método *Random Variable Neighborhood Descent* (Random VND ou RVND) como busca local (MARTINS *et al*., 2015). Cinco novas vizinhanças são propostas para a busca local, as quais são uma variação do método VND (*Variable Neighborhood Descent*), juntamente com quatro perturbações. Resultados computacionais mostraram que o algoritmo proposto foi extremamente competitivo em relação aos demais algoritmos existentes na literatura, em termos de qualidade das soluções.

O restante do texto encontra-se assim estruturado. No Capítulo 2 é feita a caracterização do problema e uma revisão da literatura para o EBDMSTP e o BDMSTP. O Capítulo 3 apresenta uma proposta de um algoritmo construtivo para o EBDMSTP. O Capítulo 4 ilustra uma proposta de desenvolvimento de um ILS acoplado a RVND para o EBDMSTP. O Capítulo 5 apresenta os resultados computacionais, enquanto que no Capítulo 6 conclusões são realizadas. No apêndice, anexamos o artigo científico produzido a partir desta tese.

CAPÍTULO 2 – REVISÃO DA LITERATURA

Seja $G = (V, E)$ um grafo não direcionado, com $n = |V|$ vértices, $m = |E|$ arestas, e um valor real w_{uv} positivo associado a cada aresta $(u, v) \in E$ que indica o custo de alcançar o vértice v a partir de u, com u e $v \in V$. Dado um valor inteiro D, $2 \leq D \leq n - 1$, uma árvore geradora com diâmetro limitado é uma árvore geradora $T = (V, E_T)$ de G, $E_T \subseteq E$, cujo diâmetro não é maior que D.

O custo total $c(T)$ da árvore geradora T é o somatório dos custos de suas arestas:

$$c(T) = \sum_{(u,v) \in E_T} w_{uv}$$

O problema da árvore geradora de custo mínimo com diâmetro limitado (BDMSTP) consiste em encontrar uma árvore geradora de G que minimize $c(T)$, desde que o diâmetro não ultrapasse o valor D. Para os casos onde $D = 2$, 3 e $(n - 1)$, ou quando todas as arestas têm o mesmo custo, o problema pode ser resolvido em tempo polinomial (ABDALLA; DEO; GUPTA, 2000)[5]. Já para $4 \leq D < n - 1$ o problema é NP-Difícil (GAREY; JOHNSON, 1979). Na prática, somente pequenas instâncias do problema, com no máximo 161 vértices e diâmetro menor ou igual a oito, podem ser solucionadas por abordagens exatas em um tempo aceitável. Devido a esse fato, heurísticas construtivas e heurísticas baseadas em metaheurísticas são opções interessantes para prover boas soluções para o BDMSTP.

2.1 MÉTODOS EXATOS PARA A SOLUÇÃO DO BDMSTP

Várias formulações para o BDMSTP encontram-se descritas na literatura. Achuthan *et al*. (1994) apresentam uma formulação de programação linear inteira mista (*Mixed Integer*

[5] Para o caso onde $D = 2$, a árvore formada é uma estrela. Quando $D = 3$, a árvore resultante é composta por duas estrelas interligadas por uma aresta que une os centros das estrelas. Já para $D = n - 1$, tem-se uma árvore geradora de custo mínimo.

Linear Programming – MILP), bem como alguns procedimentos baseados no método *branch-and-bound*, e analisam a sua aplicação em instâncias de grafos completos com até 50 vértices e diâmetro igual a quatro. Gouveia e Magnati (2003) apresentam um modelo baseado em fluxo de redes (*network flow*) que resolve o problema para instâncias com 60 vértices e 600 arestas, com um diâmetro menor ou igual a oito. Santos, Lucena e Ribeiro (2004) propõem uma melhoria no método apresentado por (ACHUTHAN et al., 1994), conseguindo resolver instâncias de grafos completos com até 25 vértices e diâmetro menor ou igual a 10, e instâncias com 60 vértices, 150 arestas e diâmetro igual a cinco. Grüber e Raidl (2005a) apresentam um novo modelo de programação linear inteira 0-1 e conseguem resolver o problema para instâncias com 40 vértices, 100 arestas e diâmetro menor ou igual a nove com um tempo bem menor que o proposto por (SANTOS; LUCENA; RIBEIRO, 2004). Grüber e Raidl (2008) apresentam uma formulação de programação linear inteira baseada em um algoritmo *branch-and-cut*, o qual utiliza heurísticas construtivas, busca local e busca tabu para auxiliar na solução do BDMSTP, conseguindo resolver instâncias com 60 vértices e 600 arestas com diâmetro oito. Gouveia, Simonetti e Uchoa (GOUVEIA; SIMONETTI; UCHOA, 2011) apresentam um novo modelo de algoritmo *branch-and-cut* capaz de resolver instâncias de grafos completos com até 161 vértices e diâmetro menor ou igual a oito.

2.1.1 UMA FORMULAÇÃO PARA A SOLUÇÃO DO BDMST

A seguir é apresentada uma formulação de Santos, Lucena e Ribeiro (2004) para resolver o BDMSTP. Ela faz uso de um grafo direcionado $G' = (V, A)$ obtido do grafo não direcionado $G = (V, E)$ original, o qual é construído da seguinte forma: para cada aresta $(i, j) \in E$, com $i < j$, há duas arestas (i, j) e $(j, i) \in A$, com custo $c_{ij} = c_{ji}$ e $L = D / 2$, se D é par, e $L = (D - 1) / 2$, se D é ímpar. A seguir são apresentadas duas formulações, uma para quando D é par e outra para D ímpar.

Considere o primeiro caso, onde D é par. Introduza um vértice artificial r em G', obtendo o grafo $G'' = (V', A')$, com $V' = V \cup \{r\}$ e $A' = A \cup \{(r, 1), ..., (r, |V|)\}$. A seguir, associe uma variável binária x_{ij} a toda aresta $(i, j) \in A'$, que será utilizada para identificar uma árvore geradora, e uma variável não negativa u_i para todo vértice $i \in V'$, que será usada para indicar o número de arestas contidas no caminho de r até $i \in V$. A formulação para a BDMST é a seguinte:

$$\min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{1}$$

$$\sum_{j \in V} x_{rj} = 1 \tag{2}$$

$$\sum_{(i,j)\in A'} x_{ij} = 1 \,, \forall j \in V \tag{3}$$

$$u_i - u_j + (L+1)x_{ij} \leq L, \forall (i,j) \in A' \tag{4}$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A' \tag{5}$$

$$0 \leq u_i \leq L+1, \forall i \in V' \tag{6}$$

A equação (2) garante que o vértice artificial r esteja conectado a exatamente um vértice de V, isto é, o vértice central da BDMST. A restrição (3) determina que exatamente uma aresta seja incidente a cada vértice de V. As restrições (4) e (6) asseguram que o caminho de r até cada vértice $i \in V$ tem no máximo $L + 1$ arestas. Já a restrição (5) estabelece os requisitos de integralidade. As arestas $(i, j) \in E$, tal que $x_{ij} = 1$ ou $x_{ji} = 1$ em uma possível solução para (2) – (6) definem uma árvore geradora T de G com diâmetro menor ou igual a D.

Considere agora o caso onde D é ímpar. Neste caso, o vértice r deverá ser conectado a exatamente dois vértices de V, os quais são os vértices centrais da BDMST, interligados por uma aresta central. Esta situação é modelada pela seleção de uma aresta central $(p, q) \in E$ e forçando explicitamente que as arestas artificiais (p, r) e (q, r) apareçam na solução. Uma

possível árvore geradora T para G é obtida eliminando as duas arestas incidentes sobre r e conectando as suas extremidades por meio de uma aresta central (p, q). Para isso, considere uma variável binária z_{ij} associada a cada aresta $(i, j) \in E$, com $i < j$. Sempre que $z_{ij} = 1$, a aresta (i, j) é selecionada como a aresta central da árvore geradora. Caso contrário, $z_{ij} = 0$. Introduza também as variáveis x_{ri}, para todo $i \in V$, representando as arestas associadas com o vértice artificial r. Uma aresta $(i, j) \in E$ é selecionada como a aresta central da árvore se, e somente se, as arestas (r, i) e (r, j) são também selecionadas. A formulação para a BDMST, quando D é ímpar, é dada a seguir:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{(i,j)\in E} c_{ij}z_{ij} \tag{7}$$

$$\sum_{j \in V} x_{rj} = 2 \tag{8}$$

$$\sum_{(i,j)\in A'} x_{ij} = 1\,, \forall j \in V \tag{9}$$

$$\sum_{(i,j)\in E} z_{ij} = 1 \tag{10}$$

$$z_{ij} \geq x_{ri} + x_{rj} - 1, \forall (i,j) \in E \tag{11}$$

$$z_{ij} \leq x_{ri}, \forall (i,j) \in E \tag{12}$$

$$z_{ij} \leq x_{rj}, \forall (i,j) \in E \tag{13}$$

$$u_i - u_j + (L+1)x_{ij} \leq L, \forall (i,j) \in A' \tag{14}$$

$$0 \leq u_i \leq L + 1, \forall i \in V \tag{15}$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A' \tag{16}$$

$$z_{ij} \in \{0,1\}, \forall (i,j) \in E \tag{17}$$

A equação (8) estabelece que o vértice central artificial *r* está conectado a exatamente dois vértices de *V*. A restrição (9) especifica que há exatamente uma aresta incidente em cada vértice de *V*. As restrições de (10) a (13) fornecem a linearização de $z_{ij} = x_{ri} \cdot x_{rj}$ para cada aresta $(i, j) \in E$. Finalmente, as restrições de (14) a (15) garantem que o caminho a partir do vértice artificial *r* a cada vértice $i \in V$ tenha no máximo $L + 1$ arestas. As restrições (16) e (17) são requisitos de integralidade.

2.2 HEURÍSTICAS CONSTRUTIVAS PARA A SOLUÇÃO DO BDMSTP E DO EBDMSTP

Vários algoritmos que fazem uso de heurísticas construtivas encontram-se descritos na literatura. Esta seção relaciona aqueles mais conhecidos.

O primeiro deles é o *One-Time Tree Construction* – OTTC, (ABDALLA; DEO; GUPTA, 2000), o qual é baseado no algoritmo de Prim (PRIM, 1957). Basicamente, ele inicia a construção de uma solução para o BDMSTP a partir de um vértice qualquer de *G* e adiciona iterativamente novos vértices, desde que esses já não estejam na árvore. Além disso, as arestas inseridas não devem violar o diâmetro e serem as de menor custo. Para garantir que o diâmetro não seja violado, o algoritmo mantém os comprimentos dos caminhos entre os vértices e as suas excentricidades, sendo esses valores atualizados a cada nova inclusão de um vértice à árvore. Trata-se, portanto, de um algoritmo guloso, já que somente as arestas de menor custo são inseridas na árvore a cada iteração, sendo sua complexidade $O(n^3)$. Entretanto, a qualidade da árvore gerada é diretamente dependente de qual vértice é escolhido como sendo o inicial. Assim, os autores recomendam que o algoritmo seja executado para cada um dos vértices de *G*, devendo ser escolhida a árvore resultante de menor custo, o que eleva a complexidade do método para $O(n^4)$.

Segundo Raidl e Julstrom (2003a), quando o OTTC é aplicado a instâncias do problema cujos vértices são pontos em um espaço euclidiano[6] e cujos custos das arestas são dados pela distância euclidiana entre os pontos (EBDMSTP), a heurística em geral produz árvores geradoras cujos custos são muito maiores que o mínimo. Isso é particularmente verdade quando o diâmetro é pequeno se comparado a n. Diante desse fato, os autores propuseram uma nova heurística gulosa, a *random greedy heuristic* – RGH. Nessa, se o diâmetro for par, um vértice aleatório (v_0) é escolhido como o centro da árvore a ser construída. Caso contrário, dois vértices (v_0 e v_1) são escolhidos aleatoriamente e a aresta que os une se torna a primeira aresta da árvore. Para evitar que a restrição de diâmetro seja violada, a profundidade de cada vértice da árvore (o número de arestas do caminho do centro até ele) é mantida. Esse valor é determinado quando o vértice é inserido e não muda posteriormente. Um vértice somente poderá ser inserido na árvore se a profundidade dele até o centro não é maior que $\lfloor D/2 \rfloor$ (HANDLER, 1973). A heurística RGH é descrita pelo Algoritmo 1. Para facilitar a notação, consideramos a árvore geradora T sendo construída como um conjunto de arestas, enraizadas a partir do centro v_0, caso o diâmetro seja par, ou dos centros v_0 e v_1, caso o diâmetro seja ímpar. Para controlar a geração de T, U conterá os vértices que ainda não foram conectados a T, e C conterá os vértices já conectados e cuja profundidade (*depth*) não ultrapasse $\lfloor D/2 \rfloor$. A seguir é analisado o algoritmo.

Na linha 1, a árvore geradora T é inicializada, não contendo inicialmente nenhuma aresta. Na linha 2, um vértice aleatório v_0 de V é escolhido, sendo este considerado como o centro da árvore a ser construída. Na linha 3, U é inicializado com todos os vértices de V, exceto v_0, que já foi utilizado. Na linha 4, C é inicializado com o vértice v_0, o centro da árvore, o qual poderá ser utilizado para conectar outros vértices à árvore que será construída. Como v_0 foi escolhido como o centro, então sua profundidade (*depth*) é zero. A linha 6

[6] Um quadrado unitário onde a cada vértice é associado uma coordenada (x, y).

verifica se o diâmetro D da árvore sendo construída é ímpar. Nesse caso, a árvore terá dois centros, v_0 e v_1, sendo este último escolhido aleatoriamente de U (linha 7), e a árvore geradora T terá a sua primeira aresta (v_0, v_1) (linha 8). Como v_1 já foi escolhido, então ele é retirado de U (linha 9) e adicionado a C (linha 10). Sendo v_1 um centro, então sua profundidade deve ser zero (linha 11). A seguir, todos os vértices deverão ser retirados gradativamente de U e utilizados para compor arestas de T, o que é feito pelas instruções contidas no laço formado pelas linhas 13 a 22. Na linha 14, um vértice v é escolhido aleatoriamente de U e utilizado para compor uma aresta (u, v), com $u \in C$, desde que o custo $w(u, v)$ seja o menor possível (linha 15). Nesse caso, a aresta (u, v) é incluída em T (linha 16) e v é retirado de U (linha 17), sendo a sua profundidade inicializada com uma unidade a mais que a profundidade de u, pois u já estava na árvore (linha 18). Para garantir que o diâmetro não seja violado, se a profundidade de v for menor que $\lfloor D/2 \rfloor$ (linha 19), então v poderá ser utilizado para conectar vértices que serão escolhidos de U, sendo então incluído em C. O laço (linhas 13 a 22) se repete até que não reste mais nenhum vértice em U, culminando com o retorno da árvore geradora obtida T (linha 23), finalizando o algoritmo.

A complexidade de RGH é $O(n^2)$, onde n é o número de vértices do grafo. De acordo com Raidl e Julstrom (2003a), para se obter uma BDMST de menor custo o algoritmo deve ser executado n vezes, o que resulta em uma complexidade de $O(n^3)$, sendo essa menor que a do OTTC. O OTTC e o RGH foram comparados utilizando instâncias Euclidianas da OR-Library[7]. Cada instância é composta por 15 grafos completos. Foram utilizadas as cinco primeiras instâncias de cada, para n = 50, 100, 250, 500 e 1000 vértices, com diâmetros de 5, 10, 15, 20 e 25, respectivamente. Em todos os casos o RGH produziu árvores de menor custo que o OTTC.

[7] Beasley's OR-Library. Disponível em http://people.brunel.ac.uk/~mastjjb/jeb/orlib/esteininfo.html

Algoritmo 1: RGH

Entrada: um grafo completo e conexo $G = (V, E)$ e o diâmetro D
Saída: uma BDMST T

```
1.  T ← ∅
2.  escolha um vértice aleatório v0 de V
3.  U ← V − {v0}
4.  C ← {v0}
5.  depth[v0] ← 0
6.  se D é ímpar então
7.      escolha um vértice aleatório v1 de U
8.      T ← {(v0, v1)}
9.      U ← U − {v1}
10.     C ← C ∪ {v1}
11.     depth[v1] ← 0
12. fim-se
13. enquanto U ≠ ∅ faça
14.     escolha um vértice aleatório v de U
15.     escolha um vértice u de C cuja aresta (u, v) tenha o menor custo w((u, v))
16.     T ← T ∪ {(u, v)}
17.     U ← U − {v}
18.     depth[v] ← depth[u] + 1
19.     se depth[v] < ⌊D/2⌋ então
20.         C ← C ∪ {v}
21.     fim-se
22. fim-enquanto
23. retorne T
```

Uma outra heurística para construir uma BDMST é a *Center-Based Tree Constrution* – CBTC (JULSTROM, 2009), descrita pelo Algoritmo 2. Nesse algoritmo, uma árvore T para o BDMSTP é construída a partir de um vértice central (v_0), se o diâmetro é par (linhas de 1 a 4), ou a partir de dois vértices centrais (v_0 e v_1), se o diâmetro é ímpar (linhas 6 e 10), sendo v_1 um vértice de V o mais próximo de v_0 (linha 6). Nesse último caso, assim como no RGH, a aresta (v_0, v_1) se torna a primeira aresta de T (linha 9). Para controlar a profundidade de cada vértice v da árvore, um vetor *depth* é mantido. No caso do vértice central (ou vértices centrais, se o diâmetro é ímpar) a profundidade é zero (linha 2, se o diâmetro é par, ou linhas 7 e 8, se o diâmetro for ímpar). Um vértice v da árvore T é considerado elegível se $depth[v] < \lfloor D/2 \rfloor$. Assim, novos vértices somente poderão ser inseridos na árvore por meio de uma aresta com um vértice elegível. Para controlar quais vértices são elegíveis, um vetor *near* é mantido.

Dessa forma, *near*[*u*] identifica o vértice elegível mais próximo de *u*, onde *u* é um vértice que ainda não está na árvore. O custo da aresta (*u*, *near*[*u*]) é mantido no vetor *wnear*, na posição *wnear*[*u*]. Tais vetores são inicializados inicialmente na linha 3 (se o diâmetro é par), onde apenas v_0 é elegível, ou na linha 10 (se o diâmetro é ímpar), sendo que nesse caso os vértices v_0 e v_1 são os únicos vértices elegíveis. As linhas de 12 a 24 estabelecem um laço de repetição, cujas instruções nele contidas irão introduzir novos vértices à *T*. Esse laço é executado enquanto o número de vértices de *T* for menor que $n - 1$, já que *T* pode ter no máximo $n - 1$ arestas. Na linha 13 é selecionado um vértice *v* que não esteja em *T* e que tenha o menor *wnear*[*v*], ou seja, aquele que tenha o menor custo para o seu vértice elegível *near*[*v*]. A seguir, a aresta (*v*, *near*[*v*]) é incluída em *T* (linha 14), sendo a profundidade de *v* (*depth*[*v*]) inicializada como sendo uma unidade a mais que o seu vértice elegível *near*[*v*] (linha 15). A seguir, na linha 16, é verificado se *v* é um vértice elegível, ou seja, se $depth[v] < \lfloor D/2 \rfloor$. Se esse for o caso, então todo vértice *u* que não esteja em *T* (linha 17) terá como vértice elegível o vértice *v*, desde que o custo da aresta ligando *u* a *v* ($w(u, v)$) seja menor que o custo da aresta ligando *u* a seu vértice elegível *near*[*u*], isto é, desde que $w(u, v) < wnear[u]$ (linha 18), resultando na atualização de *near*[*u*] com *v* (linha 19) e *wnear*[*u*] com o custo $w(u, v)$ (linha 20). Após o término do laço, a árvore resultante *T* é retornada (linha 25), finalizando o algoritmo.

Analisando o algoritmo, observa-se que o custo do laço contido na linha 12 é $O(n)$. Além disso, a cada inserção de uma aresta na árvore, os dois vetores *near* e *wnear* devem ser atualizados, sendo o custo dessa atualização $O(n)$, resultando em um custo $O(n^2)$ para o algoritmo. Segundo Julstrom (2009), a qualidade da árvore que o algoritmo identifica depende do vértice escolhido como centro (ou vértices, caso o diâmetro seja ímpar). Dessa forma, para identificar a árvore de menor custo o algoritmo deve ser executado *n* vezes, uma para cada vértice do grafo, o que resulta em um tempo de $O(n^3)$.

Algoritmo 2: CBTC/RTC

Entrada: um grafo completo e conexo $G = (V, E)$, um vértice v_0 de V e o diâmetro D
Saída: uma BDMST T

```
1.  se D é par então
2.      depth[v0] ← 0
3.      inicialize os vetores near[.] e wnear[.]
4.      T ← ∅
5.  senão
6.      escolha um vértice v1 o mais próximo de v0 (*)
7.      depth[v0] ← 0
8.      depth[v1] ← 0
9.      T ← {(v0, v1)}
10.     inicialize os vetores near[.] e wnear[.]
11. fim-se
12. enquanto o número de vertices de T for menor que n – 1 faça
13.     escolha um vértice v que não esteja em T e tenha o menor wnear[v] (*)
14.     T ← T ∪ {(v, near[v])}
15.     depth[v] ← 1 + depth[near[v]]
16.     se depth[v] < ⌊D/2⌋ então
17.         para cada vértice u que não esteja em T faça
18.             se w(u, v) < wnear[u] então
19.                 near[u] ← v
20.                 wnear[u] ← w(u, v)
21.             fim-se
22.         fim-para
23.     fim-se
24. fim-enquanto
25. retorne T;
```

De acordo com Julstrom (2009), para se obter a BDMST, as heurísticas OTTC e CBTC são "extremamente gulosas" quando aplicadas a instâncias euclidianas com diâmetros pequenos. Isso porque, em ambos os casos, cada novo vértice é sempre aquele mais próximo a um vértice elegível da árvore, o que resulta em uma espinha dorsal[8] mais curta e em arestas longas conectando as folhas da árvore. Assim, é proposta uma nova heurística, a *Randomized Center-Based Tree Construction* – RTC, onde os vértices a serem inseridos na árvore são escolhidos aleatoriamente, assim como na heurística RGH (RAIDL; JULSTROM, 2003a). Apesar disso, o método ainda se mantém guloso, visto que um novo vértice somente é inserido na árvore por meio de um vértice elegível. O algoritmo que implementa essa

[8] Árvore geradora resultante da retirada de suas folhas e arestas que as conectam.

heurística é o mesmo apresentado no Algoritmo 2, trocando-se os pontos marcados por (*) por uma escolha aleatória dos vértices, o que resulta, também, em uma complexidade $O(n^2)$. Como o algoritmo deve ser executado n vezes (uma vez para cada vértice) e escolhe-se a BDMST de menor custo, a complexidade resultante é também $O(n^3)$.

As heurísticas OTTC, CBTC e RTC foram comparadas utilizando instâncias onde metade são euclidianas e metade tiveram custos das arestas escolhidos aleatoriamente, com uma variedade de diâmetros (JULSTROM, 2009). Em instâncias euclidianas com diâmetros pequenos a RTC identificou árvores mais curtas e com menores custos que a CBTC, enquanto que essa última superou a OTTC. Com diâmetros maiores, mais próximos do diâmetro da árvore geradora de custo mínimo e sem restrições, a CBTC identificou árvores mais curtas e com custos menores que a OTTC e a RTC. Já para instâncias onde os custos das arestas foram gerados aleatoriamente, a CBTC encontrou, em geral, árvores de menores custos que a OTTC, sendo que ambas retornaram árvores com custos bem menores que a RTC.

A heurística *Center-Based Recursive Clustering* – CBRC (BINH et al., 2009), foi baseada em RGH/CBTC e encontra-se descrita pelo Algoritmo 3. Nesse algoritmo, as linhas de 2 a 13 são equivalentes às do RGH no tocante a escolha do(s) vértice(s) central(is), inicialização da árvore T, dos conjuntos U e C e do vetor *depth*. A seguir, o algoritmo interliga todos os vértices u que estão em U, e portanto não estão na árvore T, ao(s) vértice(s) central(is) de T, formando um agrupamento com uma estrutura do tipo estrela[9], todos tendo profundidade (*depth*) igual a um (linhas de 15 a 31). No caso do diâmetro ser par (linha 16), o vértice u é interligado ao vértice v_0 de T, tornado-se seu filho (linha 17); caso contrário, se a distância[10] de u a v_0 for menor que a distância de u a v_1 (linha 21), então o vértice u é interligado ao vértice v_0, tornando-se filho de v_0 (linha 22); senão, ele é interligado a v_1 (linha

[9] Segundo os autores, boas soluções para o problema da BDMST tem normalmente a estrutura de uma estrela.

[10] De acordo com os autores, dois métodos para o cálculo de *distância*(u, v) podem ser adotados: (1) considerando o custo $w(u, v)$ ou (2) considerando o custo do caminho mais curto contendo as k arestas entre u e v, sendo este último o método mais indicado para instâncias não-euclidianas.

26), tornando-se filho de v_1 (linha 26). Em ambos os casos, a aresta obtida, (u, v_0) ou (u, v_1), é incluída em T (linhas 24 ou 28). Posteriormente, e de forma iterativa (linhas de 32 a 46), a função *EscolhaCentro*()[11] escolhe um vértice v de U, que seja uma folha de T e tenha *depth*[v] $< \lfloor D/2 \rfloor$, para ser o centro do novo agrupamento a ser gerado (linhas 33 e 34), sendo v retirado de U (linha 38). Caso não seja possível obter esse vértice v, o laço é encerrado (linhas 35 a 37) e a árvore T gerada é retornada, encerrando o algoritmo (linha 47). Caso contrário, para cada vértice u em U que seja folha de T (linhas de 39 a 45), é verificado se a distância de u a v é menor que a distância de u ao seu pai na árvore (linha 40). Se esse for o caso, então u é desconectado de seu pai em T e conectado a v (linha 41), sendo a profundidade de u corrigida para ser uma unidade a mais que a profundidade de v (linha 42), culminando com a posterior retirada da aresta (u, *pai*(u)) de T e a inclusão da aresta (u, v) a T (linha 43). Analisando o algoritmo, verifica-se que a sua complexidade é de no máximo $O(n^3)$, dependendo de como se implementa as funções *distância*() e *EscolhaCentro*(), sendo essa complexidade idêntica a do RGH e do CBTC.

O algoritmo CBRC foi comparado com OTTC, RGH e CBTC, todos utilizando instâncias euclidianas da OR-Library e instâncias não-euclidianas, ambas representando grafos completos. No caso das instâncias euclidianas, foram utilizadas as cinco primeiras de cada problema com n = 50, 100, 250, 500 e 1000 vértices, com diâmetros de 5, 10, 15, 20 e 25, respectivamente, totalizando 25 instâncias. Já para as não-euclidianas foram utilizadas cinco instâncias de cada problema com n = 100, 250, 500 e 1000 vértices, com diâmetros de 10, 15, 20 e 25, totalizando 20 instâncias. Nesse caso, os custos das arestas foram gerados aleatoriamente no intervalo de 0,01 a 0,99. O método para escolha dos centros foi classificar

[11] De acordo com os autores, são propostas quatro possíveis implementações para a função *EscolhaCentro*(): (1) v é um centro de U se $\forall u \in U$ o somatório de *distância*(v, u) é mínima; (2) ranqueie todos os vértices $u \in U$, a partir do somatório da *distância*(u, v) e escolha v aleatoriamente a partir do primeiro h% dos vértices; (3) faça a seleção de v a partir de h-torneios, onde a aptidão de cada vértice v é dada pelo somatório de *distância*(v, u), com $u \in U$; (4) escolha v aleatoriamente.

cada nó elegível por meio do custo de uma árvore estrela enraizada naquele nó e então selecionar aleatoriamente a partir dos 20% com menores custos. Em instâncias euclidianas, o peso da aresta (u, v) é a distância Euclidiana. Já para instâncias não-euclidianas a distância é o custo das arestas no caminho de u a v. Segundo os autores, em instâncias euclidianas o CBRC obteve soluções para o BDMST com menores custos quando comparado com os demais – cerca de 10% melhor que o RGH, que obteve os segundos melhores resultados. Já para instâncias não-euclidianas, o CBRC obteve praticamente os mesmos resultados que OTTC e CBTC, enquanto o RGH apresentou o pior resultado – cerca de 41% pior que o CBRC.

2.3 HEURÍSTICAS BASEADAS EM METAHEURÍSTICAS PARA A SOLUÇÃO DO BDMST

Várias heurísticas baseadas em metaheurísticas para resolver o problema da BDMST encontram-se descritas na literatura. Essa seção relaciona as mais conhecidas. Em todos os casos citados, D é o diâmetro da BDMST a ser construída.

Raidl e Julstrom (2003a) propuseram um algoritmo evolucionário onde cada cromossomo é representado por meio de uma lista codificada de arestas (*Edge-Set-Code Evolutionary Algorithm* – ESCEA). Além disso, o cromossomo contém também o vértice central (ou os vértices centrais, quando o diâmetro é ímpar) da árvore geradora que ele representa e a sua aptidão (*fitness*), sendo essa dada pelo custo total de suas arestas. O ESCEA utiliza uma operação de recombinação e quatro operadores de mutação: *Edge-Delete*, *Center-Move*, *Greedy-Edge-Replace* e *Subtree-Optimize*.

Algoritmo 3: CBRC

Entrada: um grafo completo e conexo $G = (V, E)$
Saída: uma BDMST T

```
1.  // Escolha do(s) vértice(s) central(is)
2.  T ← ∅
3.  Escolha aleatoriamente um vértice v0 em V
4.  U ← V – {v0}
5.  C ← {v0}
6.  depth[v0] ← 0
7.  se D é ímpar então
8.     escolha aleatoriamente um vértice v1 em U
9.     T ← {(v0, v1)}
10.    U ← U – {v1}
11.    C ← C ∪ {v1}
12.    depth[v1] ← 0
13. fim-se
14. // Agrupe os vértices em U em cluster(s) com os centros em v0 ou v1
15. para cada vértice u em U faça
16.    se D é par então
17.       torne u um filho de v0
18.       depth[u] = 1
19.       T ← T ∪ {(u, v0)}
20.    senão // D é ímpar
21.       se distância(u, v0) ≤ distância(u, v1) então
22.          torne u um filho de v0
23.          depth[u] ← 1
24.          T ← T ∪ {(u, v0)}
25.       senão
26.          torne u um filho de v1
27.          depth[u] ← 1
28.          T ← T ∪ {(u, v1)}
29.       fim-se
30.    fim-se
31. fim-para
32. enquanto verdade = verdade faça
33.    V* ← todos os vertices em U que são folhas de T e tenham depth < ⌊D/2⌋
34.    v ← EscolhaCentro(V*)
35.    se v é vazio então
36.       saia do laço
37.    fim-se
38.    U ← U - {v}
39.    para cada vértice folha u em U faça
40.       se distancia(u,v) ≤ distancia(u, pai(u)) então
41.          torne u um filho de v
42.          depth[u] ← depth[v] + 1
43.          T ← T - {(u, pai(u))} ∪ {(u, v)}
44.       fim-se
45.    fim-para
46. fim-enquanto
47. retorne T
```

A operação de recombinação seleciona aleatoriamente um vértice central (se o diâmetro é par) ou dois vértices centrais (se o diâmetro é ímpar) dos centros dos pais. Assim como no algoritmo RGH, ela mantém um conjunto U de vértices não conectados à BDMST e um conjunto C de vértices já conectados e que possuam profundidade menor que $\lfloor D/2 \rfloor$. Uma aresta somente poderá ser conectada a um vértice de C se a profundidade resultante não for maior que $\lfloor D/2 \rfloor$. Dois novos conjuntos, *A1* e *A2*, mantêm as arestas incidentes a vértices em C que podem ser utilizados para estender a árvore. Enquanto *A1* contém as arestas encontradas em ambos os pais, *A2* contém as arestas contidas em apenas um dos dois. Para estender a árvore é escolhida uma aresta aleatoriamente de *A1* ou *A2*, caso *A1* esteja vazio. Se *A2* também estiver vazio, é criada uma aresta a partir de dois vértices aleatórios de C e U. A complexidade da operação de recombinação é $O(n)$.

Os resultados do ESCEA foram comparados com o OTTC e o RGH, utilizando instâncias euclidianas da OR-Library com n = 50, 100, 250, 500 e 1000, todas representando grafos completos, com diâmetros 5, 10, 15, 20 e 25, respectivamente. Em todos os casos o ESCEA superou os resultados do OTTC e do RGH.

Julstrom e Raidl (JULSTROM; RAIDL, 2003) propuseram outro algoritmo evolutivo que codifica árvores geradoras como permutação de seus vértices (*Permutation-Coded EA* – PCEA). Nessa codificação o primeiro vértice listado (se o diâmetro é par) ou os dois primeiros vértices (se o diâmetro é ímpar) formam o centro da árvore geradora que a permutação representa. Para gerar os demais vértices da permutação é utilizado o RGH. Toda permutação representa uma árvore válida. Os cromossomos da população inicial são permutações aleatórias. Cada posição de um vértice no cromossomo determina quando o RGH o incluirá na árvore geradora. Assim, o cruzamento é parcialmente mapeado (GOLDBERG; LINGLE, 1985), o qual tende a preservar as posições dos símbolos dos pais

no filho. A mutação troca os vértices em duas posições aleatórias no cromossomo do pai, mudando a forma em que os vértices são incluídos na árvore geradora.

A estrutura do PCEA é a mesma do ESCEA. Nela os cromossomos são selecionados dos pais por meio de torneios com trocas[12]; a recombinação é aplicada algumas vezes e a mutação é aplicada sempre. O cromossomo resultante substitui o pior da população, desde que isso não gere duplicidade. O PCEA e o ESCEA foram comparados utilizando 25 instâncias euclidianas da OR-Library. Foram utilizados para os testes apenas os cinco primeiros grafos de cada instância, todos representando grafos completos. As instâncias usadas tinham n = 50, 70, 100, 250 e 500 vértices. Para cada uma delas foram adotados os diâmetros 5, 7, 10, 15 e 20, respectivamente. O algoritmo foi executado 50 vezes para cada instância. Excetuando-se o caso de n = 50, onde o resultado foi ambíguo, o PCEA obteve melhores resultados que o ESCEA, identificando a BDMST mais curta. Entretanto, como a heurística para decodificar a permutação requer tempo O(n^2), o PCEA é mais lento, e essa desvantagem de tempo cresce com o tamanho da instância do problema.

Utilizando o conceito de chaves aleatórias proposto por Bean (1994), onde todo o passo de codificação e decodificação de um cromossomo utiliza chaves aleatórias, Julstrom (2004) apresentou um algoritmo evolucionário para resolver o BDMSTP, *Random Key Evolutionary Algorithm* – RKEA, onde um decodificador baseado no algoritmo de Prim (1957) é utilizado para construir a árvore representada pelos cromossomos. Nesse algoritmo, é utilizado um cruzamento de dois pontos e uma mutação ponto a ponto, sendo a escolha dos pais feita por torneios com trocas. O RKEA foi comparado com o PCEA utilizando 25 instâncias para grafos completos euclidianos da OR-Library, com n = 50, 70, 100, 250 e 500, e diâmetros 5, 7, 10, 15 e 20, respectivamente. Em ambos os casos, o desempenho dos dois algoritmos e as soluções geradas para o BDMST foram semelhantes.

[12] No torneio com trocas k indivíduos são escolhidos aleatoriamente da população. Aquele que apresentar o melhor *fitness* é escolhido para participar da operação de recombinação.

Gruber e Raidl (2005b) propuseram uma heurística baseada em *Variable Neighborhood Search* (VNS), utilizando *Variable Neighborhood Descent* (VND) como busca local (GLOVER; KOCHENBERGER, 2003; HANSEN; MLADENOVIĆ, 2001, 2005; MLADENOVIĆ; HANSEN, 1997) para resolver o BDMSTP, a qual utiliza quatro estruturas de vizinhança: *Edge Exchange Neighborhood*, *Node Swap Neighborhood*, *Level Change Neighborhood* e *Center Exchange Level Neighborhood* (GRUBER; RAIDL, 2005b).

O algoritmo proposto foi comparado com ESCEA, PCEA e RKEA, utilizando instâncias de grafos completos da OR-Library, com n = 100, 250, 500 e 1000, e os diâmetros 10, 15, 20 e 25. Em todos os casos, o algoritmo superou os evolucionários obtendo as melhores soluções e as melhores médias.

Em Gruber, Hermert e Raidl (2006) o VNS proposto anteriormente é comparado com um novo algoritmo evolucionário, denominado de *Level Encoded Evolutionary Algorithn* (LEEA) e uma otimização de colônia de formigas (*Ant Colony* – ACO). No LEEA, a solução gerada para o BDMSTP é representada por meio de uma cadeia de níveis, onde cada gene representa o nível de um vértice correspondente na árvore. O cruzamento é aplicado uniformemente, sendo que cada gene dos pais possui a mesma probabilidade de seleção, enquanto que a mutação gera um novo nível para cada gene com uma probabilidade de *1/n*, excetuando-se os centros, os quais são trocados com outros vértices selecionados aleatoriamente. O LEEA utiliza torneios com trocas, eliminando os piores indivíduos, sendo o cruzamento e a mutação sempre aplicados. O ACO utiliza uma matriz de feromônios onde cada posição representa um valor correspondente a um vértice em um nível i, $0 \leq i \leq \lfloor D/2 \rfloor$.

Nos testes realizados foram utilizadas instâncias de grafos completos da OR-Library com n = 100, 250, 500 e 1000, e os diâmetros 10, 15, 20 e 25, respectivamente. Quando o tempo não era prefixado, o ACO apresentou os melhores resultados quando comparados com o VNS e o LEEA, exceto para n = 100, onde os resultados foram semelhantes. Já para os

testes com o tempo prefixado, o LEEA apresentou os melhores resultados, superando o ACO e o VNS.

Em Singh e Gupta (2007) foi proposta uma melhoria do PCEA, denominada de PEA-I, onde uma permutação linear dos vértices do grafo é utilizada para representar um cromossomo. A aptidão de um cromossomo é dada pelo somatório dos custos das arestas da BDMST que ele representa. O cruzamento é realizado por meio de uma ordenação uniforme (DAVIS, 1991) e a mutação é feita por meio da troca dos símbolos localizados em dois pontos do cromossomo escolhidos aleatoriamente.

O PEA-I foi executado e os resultados foram comparados com o ESCEA e o PCEA, utilizando instâncias de grafos completos da OR-Library, com n = 50, 100, 250, 500 e 1000, e os diâmetros 5, 10, 15, 20 e 25. Em todos os casos a heurística proposta superou os dois algoritmos, obtendo os melhores resultados para as instâncias com 100, 250, 500 e 1000.

Um algoritmo híbrido, combinado com os princípios do *Greedy Randomized Adaptive Search* – GRASP (LOURENÇO; MARTIN; STÜTZLE, 2003) e do *Iterated Local Search* – ILS (LOURENÇO; MARTIN; STÜTZLE, 2003), denominado por GILS, foi proposto em (LUCENA; RIBEIRO; SANTOS, 2010). Os componentes principais do algoritmo proposto pelos autores são um método heurístico construtivo para gerar as soluções factíveis, denominado de OTT-M, uma busca local composta por quatro estruturas de vizinhança (*Restricted 1-opt Neighborhood*, *Restricted 2-opt Neighborhood*, *Subtree Adoption Neighborhood* e *Path Exchange Neighborhood*) em esquema similar ao VND (GLOVER; KOCHENBERGER, 2003; HANSEN; MLADENOVIĆ, 2001, 2005; MLADENOVIĆ; HANSEN, 1997), e dois tipos de perturbações as quais são aplicadas às soluções obtidas. As perturbações propostas pelos autores envolvem a troca de um vértice central, quando o diâmetro é par, ou uma extremidade de uma aresta, quando o diâmetro é ímpar, sendo ambas aplicadas alternadamente em toda iteração.

Em instâncias de grafos completos euclidianos da OR-Library, com n = 50, 70, 100, 250, 500, 1000 e diâmetros iguais a 5, 7, 10, 15, 20, 25, respectivamente, o algoritmo híbrido conseguiu obter melhores resultados em 50% dos casos quando comparados com os melhores resultados apresentados pelo VNS (GRUBER; RAIDL, 2005b), ESCEA (RAIDL; JULSTROM, 2003a) e PCEA (RAIDL; JULSTROM, 2003b).

CAPÍTULO 3 – UMA NOVA HEURÍSTICA CONSTRUTIVA PARA O EBDMSTP

Neste capítulo, a nova heurística construtiva proposta é baseada na ideia de que todo corpo no espaço possui um centro de massa, ou centroide. Essa ideia é estendida para um conjunto de pontos em um espaço cartesiano.

Seja um conjunto de pontos P_1, P_2, ..., P_n em um plano cartesiano ortogonal cujas massas são m_1, m_2, ..., m_n, respectivamente. Sabendo-se que as coordenadas dos pontos P_1, P_2, ..., P_n no plano são (x_1, y_1), (x_2, y_2), ..., (x_n, y_n), então a coordenada (x_{cm}, y_{cm}) é definida como o *centróide* do sistema de pontos, sendo calculada por:

$$x_{cm} = \frac{m_1 x_1 + m_2 x_2 + ... + m_n x_n}{m_1 + m_2 + ... + m_n} \tag{18}$$

$$y_{cm} = \frac{m_1 y_1 + m_2 y_2 + ... + m_n y_n}{m_1 + m_2 + ... + m_n} \tag{19}$$

Considerando que a massa de cada ponto é um, o centroide (x_{cm}, y_{cm}) pode ser calculado por:

$$x_{cm} = \frac{x_1 + x_2 + ... + x_n}{n} \tag{20}$$

$$y_{cm} = \frac{y_1 + y_2 + ... + y_n}{n} \tag{21}$$

Seja $G = (V, E)$ um grafo não direcionado, conexo e completo, onde cada vértice é representado por um ponto em um espaço euclidiano, cuja massa é um, e o custo de cada aresta $(u, v) \in E$ é dado pela distância euclidiana entre os pontos u e v, com u e $v \in V$. Logo, a heurística proposta, chamada de CM, que solucione o EBDMSTP, pode ser implementada

substituindo-se a escolha aleatória de um vértice $v \in U$, no Algoritmo 1 (RGH), por uma que localiza v próximo ao *centroide* (linha 2, 7 e 14), conforme ilustrado pelo Algoritmo 4.

Algoritmo 4: CM

Entrada: um grafo complete e conexo $G = (V, E)$
Saida: uma BDMST T

1. $T \leftarrow \emptyset$
2. localize um vértice v_0 em V o mais próximo ao centróide
3. $U \leftarrow V - \{v_0\}$
4. $C \leftarrow \{v_0\}$
5. $depth[v_0] \leftarrow 0$
6. se D é ímpar então
7. localize um vértice v_1 em U o mais próximo ao centróide
8. $T \leftarrow \{(v_0, v_1)\}$
9. $U \leftarrow U - \{v_1\}$
10. $C \leftarrow C \cup \{v1\}$
11. $depth[v_1] \leftarrow 0$
12. fim-se
13. enquanto $U \neq \emptyset$ faça
14. localize um vértice v em U o mais próximo ao centróide
15. localize um vértice u em C com o menor custo $w((u, v))$
16. $T \leftarrow T \cup \{(u, v)\}$
17. $U \leftarrow U - \{v\}$
18. $depth[v] \leftarrow depth[u] + 1$
19. se $depth[v] < \lfloor D/2 \rfloor$ então
20. $C \leftarrow C \cup \{v\}$
21. fim-se
22. fim-enquanto
23. retorne T

O custo para se processar todos os vértices em U é O(n). O custo para se localizar um vértice v próximo ao centroide é O(n). Da mesma forma, o custo para se localizar o vértice u em C cuja aresta (v, u) tenha o menor peso, é O(n). Logo, o custo total do algoritmo proposto é O(n^2).

Os resultados computacionais comparando esta heurística com OTTC, CBTC, RTC e RGH encontram-se descritos no 0.

CAPÍTULO 4 – ITERATED LOCAL SEARCH

O ILS é uma metaheurística para resolver, de maneira aproximada, problemas de otimização, a qual se baseia na ideia que um ótimo local obtido por um procedimento de busca local pode ser sempre melhorado por meio da aplicação de um procedimento de perturbação, visando a exploração do espaço de soluções em busca de um ótimo global. Trata-se, portanto, de um método de busca local que procura diversificar a busca em um pequeno subconjunto do espaço de soluções que são ótimos locais (LOURENÇO; MARTIN; STÜTZLE, 2003).

O ILS é composto por uma função para a construção de uma solução inicial (*GeraSoluçãoInicial*), uma função de busca local (*BuscaLocal*), uma função de perturbação (*Perturbação*) e uma função para o critério de aceitação (*CriterioAceitação*), conforme ilustrado no Algoritmo 5. Os procedimentos de *Perturbação*(), *BuscaLocal*() *e CritérioAceitação*() são executados iterativamente até que uma condição de término seja atingida (linha 12), ocasionando, neste caso, o fim do procedimento ILS com o retorno da solução *s** obtida (linha 13).

Algoritmo 5: Estrutura do ILS

```
1.  procedimento ILS
2.
3.  entrada: um grafo completo e conexo G = (V, E)
4.  saída: uma BDMST s*
5.
6.      s0 ← GeraSoluçãoInicial()
7.      s* ← BuscaLocal(s0)
8.      repita
9.          s' ← Perturbação(s*, histórico)
10.         s*' ← BuscaLocal(s')
11.         s* ← CritérioAceitação(s*, s*', histórico)
12.     até que uma condição de término seja atingida
13.     retorne s*
14.
15. fim-procedimento
```

A função *GeraSoluçãoInicial*(), contida na linha 6, permite determinar uma solução inicial s_0, no espaço de soluções, que será utilizada como ponto de partida para a busca local. Tal função pode ser construída, por exemplo, utilizando-se qualquer método aleatório ou qualquer heurística gulosa. Entretanto, quanto melhor for a solução inicial, mais rápido será a convergência para um ótimo local.

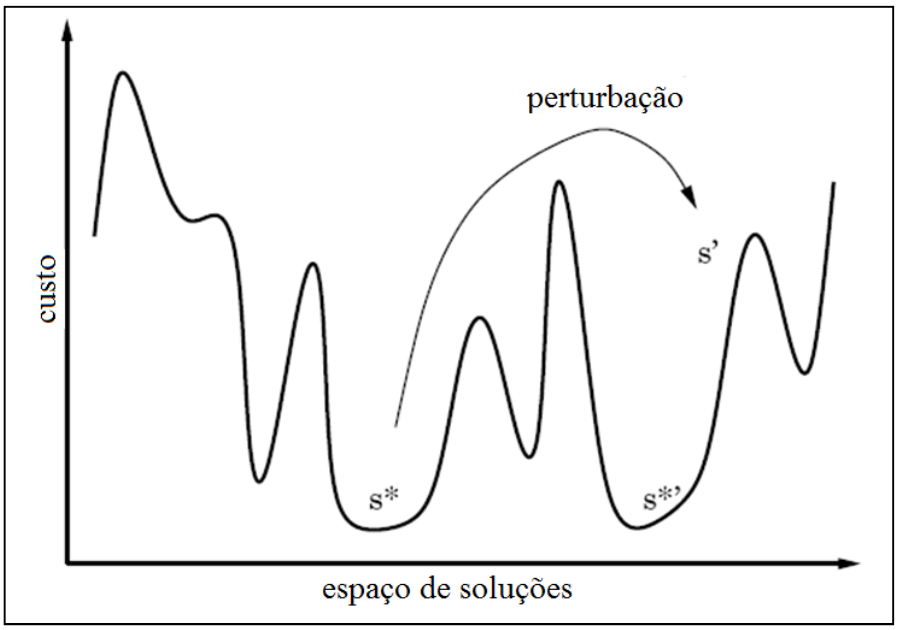
A busca local (*BuscaLocal*()) permite explorar o espaço de soluções em busca de um ótimo local. Basicamente, ela consiste em melhorar uma solução inicial *s* por meio da exploração das soluções vizinhas a *s*. Dessa forma, se uma solução vizinha *s*' é melhor que *s*, então *s* é substituída por *s*' (*s* = *s*') e a busca é reiniciada a partir de *s*. Esta etapa se repete até que nenhuma melhora seja possível, isto é, quando *s* é um ótimo local. Tal fato pode ser observado na linha 7, onde a função *BuscaLocal*(s_0) é aplicada a solução inicial s_0, gerando uma nova solução s*, e na linha 10, onde ela é aplicada a solução *s*' (*BuscaLocal*(*s*')), obtida pela função *Perturbação*(), gerando a solução *s**'.

O problema com a busca local é que ela pode ficar estagnada em um ótimo local. A perturbação (*Perturbação*()) permite que o ILS escape desse ótimo local, possibilitando que diferentes regiões do espaço de soluções sejam exploradas. Basicamente, o que a perturbação faz é modificar a solução corrente s*, gerando uma solução intermediária s', cujo objetivo é sair do ótimo local, conforme indicado pela Figura 2. Entretanto, a perturbação deve ser suficientemente forte para permitir que a *busca local* explore diferentes soluções e fraca o suficiente para evitar um reinício aleatório. Além disso, para que o espaço de soluções seja melhor explorado, um histórico das soluções visitadas anteriormente pode ser mantido, conforme indicado no Algoritmo 5 pela aplicação da função *Perturbação*(*s**, *histórico*) na linha 9.

A função *CritérioAceitação*(*s**, *s*'*, *histórico*), vista na linha 11, verifica se a solução corrente *s** será ou não substituída por uma nova solução *s*'*, correspondendo a um novo

ótimo local gerado pela *BuscaLocal*(), baseada ou não em um *histórico* das computações. Isto é, a nova solução *s**' se torna a solução atual *s**, se ela melhorar a solução anterior *s**.

Figura 2: Exemplo de aplicação da perturbação.



4.1 UM ILS PARA O EBDMSTP

Um ILS para resolver o EBDMSTP foi implementado utilizando uma variante do VND (HANSEN; MLADENOVIĆ, 2005) como busca local, denominado de *Random Variable Neighborhood Descent* – Random VND ou RVND (MARTINS et al., 2015), que explora aleatoriamente a estrutura de vizinhança da solução atual, aceitando somente soluções que a melhorem e retornando o melhor vizinho que corresponde à melhor solução encontrada.

O Algoritmo 6 apresenta a estrutura do procedimento *BuscaLocal*. Inicialmente, *V* é inicializado com a coleção das *r* vizinhanças, $\{V^1, ..., V^r\}$, as quais são ordenadas aleatoriamente (linha 4). Basicamente, o algoritmo percorre, de forma ordenada, a estrutura de vizinhança V^k da solução corrente *A**, em busca de uma melhor solução *A*' para o EBDMSTP (linha 7). Neste caso, ou seja, se o custo de *A*' é menor que o de *A** (linha 8), *A*' passa a ser a

solução corrente A^* (linha 9) e o processo se reinicia ($k \leftarrow 1$). Caso contrário, uma nova vizinhança é visitada ($k \leftarrow k + 1$) – linha 12. O processo se repete até que nenhuma melhoria seja mais possível, isto é, quando todas as vizinhanças tenham sido visitadas ($k > r$). O algoritmo é finalizado com o retorno da melhor solução obtida (linha 15).

Algoritmo 6: Estrutura da Busca Local

```
1.  procedimento BuscaLocal

2.  Entrada: uma EBDMST A*
3.  Saída: uma EBDMST A*

4.  inicialize V com as vizinhanças {V^1, ..., V^r}, ordenadas aleatoriamente
5.  k ← 1
6.  repita
7.      A' ← V^k(A*)
8.      se custo(A') < custo(A*) então
9.          A* ← A'
10.         k ← 1
11.     senão
12.         k ← k + 1
13.     fim-se
14. até k > r
15. return A*

16. fim-procedimento
```

4.2 VIZINHANÇAS

As estruturas de vizinhanças utilizadas na implementação da *BuscaLocal* são a *Edge Exchange* e a *Node Swap* (GRUBER; RAIDL, 2005b), *Subtree Optimize* (RAIDL; JULSTROM, 2003a), *Hierarchy Exchange*, *Hierarchy Rotation*, *Leaf Swap*, *Father Swap* e *Level Change*. Estas últimas cinco são contribuições desta tese.

Em todos os casos, uma árvore com n nós, enraizada em um centro, se o diâmetro é par, ou em dois centros, se o diâmetro é ímpar, representa uma solução possível para o EBDMSTP. Os demais nós não poderão estar a uma profundidade maior que $\lfloor D/2 \rfloor$ (HANDLER, 1973). Além disso, se o diâmetro é ímpar, uma aresta que interliga os nós

centrais existe. Em todos os casos, a árvore que representa uma solução possível para o EBDMSTP é orientada a partir do centro (ou centros) em direção as folhas, formando uma relação de predecessores e sucessores. O centro (ou os centros) possui profundidade zero, enquanto que as folhas têm profundidade de no máximo $\lfloor D/2 \rfloor$.

4.2.1 EDGE EXCHANGE

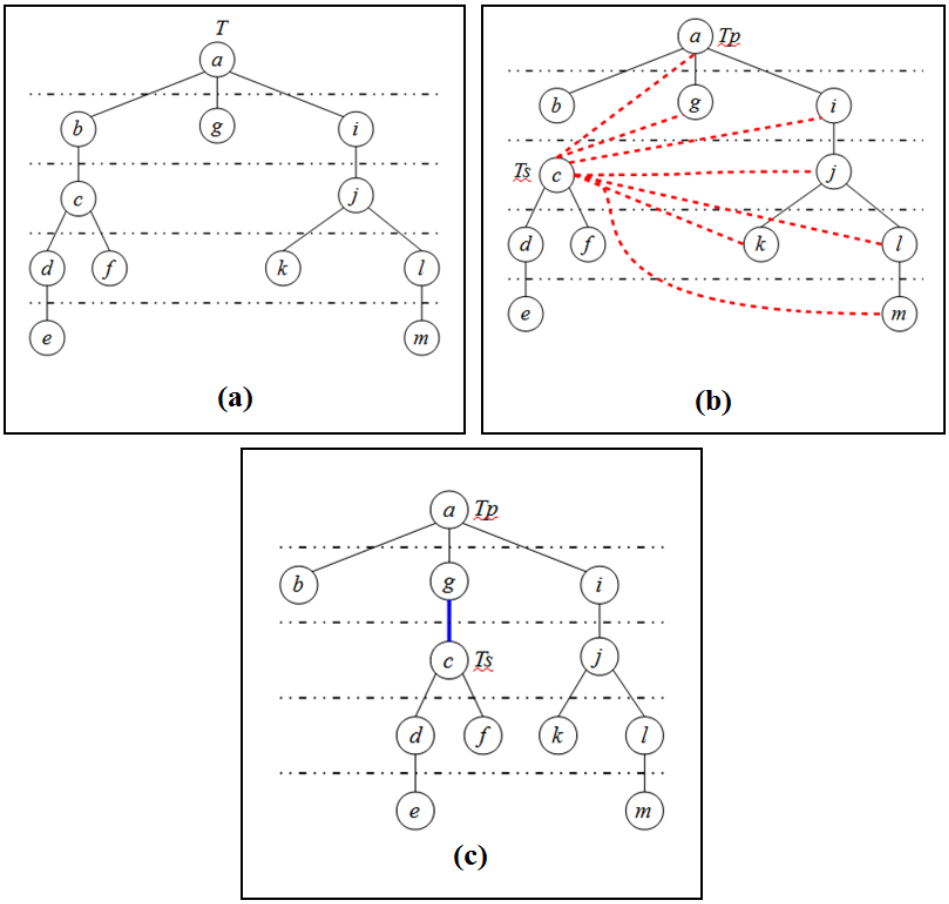
Esta vizinhança é uma variação da apresentada em (GRUBER; RAIDL, 2005b). Seja T uma possível solução para o EBDMSTP. A vizinhança *Edge Exchange* consiste em explorar todas as possíveis árvores obtidas a partir da desconexão de uma subárvore de T (por meio da remoção de uma aresta) e a sua conexão em outra posição da árvore, desde que o custo seja minimizado e o diâmetro não seja violado. Todas as possíveis subárvores são examinadas, num esquema de *best improvement* (GLOVER; KOCHENBERGER, 2003), visando determinar a conexão de menor custo[13].

Considere a árvore da Figura 3 (a). A desconexão da aresta (b, c) de T produz duas subárvores, T_p (arvore primária) e T_s (árvore secundária), conforme ilustrado pela Figura 3 (b). Posteriormente, o algoritmo verifica se a criação de uma aresta da raiz c de T_s com um dos nós de T_p minimiza o custo de T e não viola o diâmetro, Figura 3 (b). A aresta que produz o menor custo dentre todas as possibilidades é escolhida, nesse caso a aresta (g, c), formando a nova árvore apresentada pela Figura 3 (c).

O tempo para se percorrer todos os nós de T é O(n). O tempo para se examinar todas as possíveis trocas, visando identificar a de menor custo, é O(n), o que resulta em um tempo total de O(n^2) para esta vizinhança.

[13] Na proposta apresentada em (GRUBER; RAIDL, 2005b) a primeira árvore resultante da operação que tenha o custo minimizado é selecionada, num esquema de *first improvement*.

Figura 3: Exemplo de Edge Exchange.



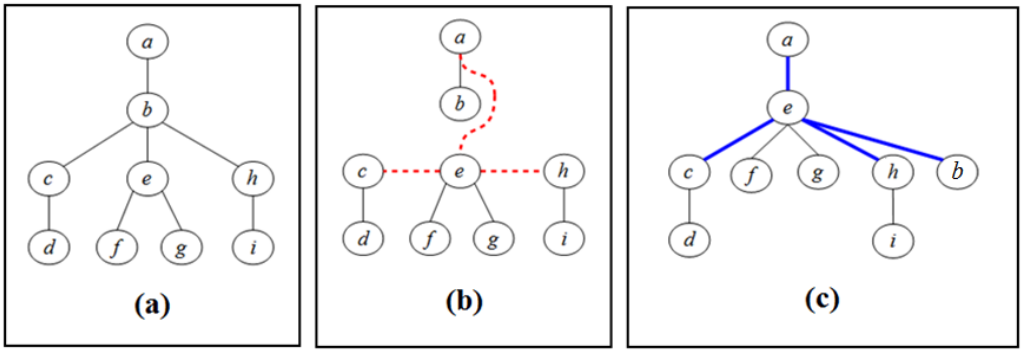
4.2.2 NODE SWAP

Esta vizinhança é uma variação da apresentada em (GRUBER; RAIDL, 2005b). Esta vizinhança explora a relação entre um nó de uma BDMST, incluindo o seu centro (ou um dos centros, se o diâmetro é ímpar)[14] e seus sucessores diretos. Uma solução para essa vizinhança é uma árvore resultante da substituição de um nó *v* por um de seus sucessores, por exemplo, um nó *u*. Neste caso, a árvore resultante dessa operação é obtida tornando *v*, e seus sucessores, sucessores de *u* e fazendo o predecessor de *v* ser o predecessor de *u*. Caso *v* seja um centro, então *u* não terá um predecessor. Note que a operação realizada por esta vizinhança não viola o diâmetro.

[14] No modelo de vizinhança proposto em (GRUBER; RAIDL, 2005b), a operação não afeta o centro (ou um dos centros, se o diâmetro é ímpar).

Seja T a árvore ilustrada pela Figura 4 (a). Assuma que $v = b$ e $u = e$. De acordo com a Figura 4 (b), a operação faz com o nó a passe a ser o predecessor direto de e, enquanto b e todos os seus sucessores passam a ser sucessores de e, conforme pode ser verificado pela Figura 4 (c).

Figura 4: Exemplo de Node Swap.



O tempo para se verificar todos os pares (v, u) de T é de O(n). O tempo para se verificar todos os sucessores de um nó v é $d(v)$, onde $d(v)$ é o grau de v. Logo, o tempo total gasto por essa vizinhança é de O($n\Delta$), onde Δ é o grau máximo de um nó.

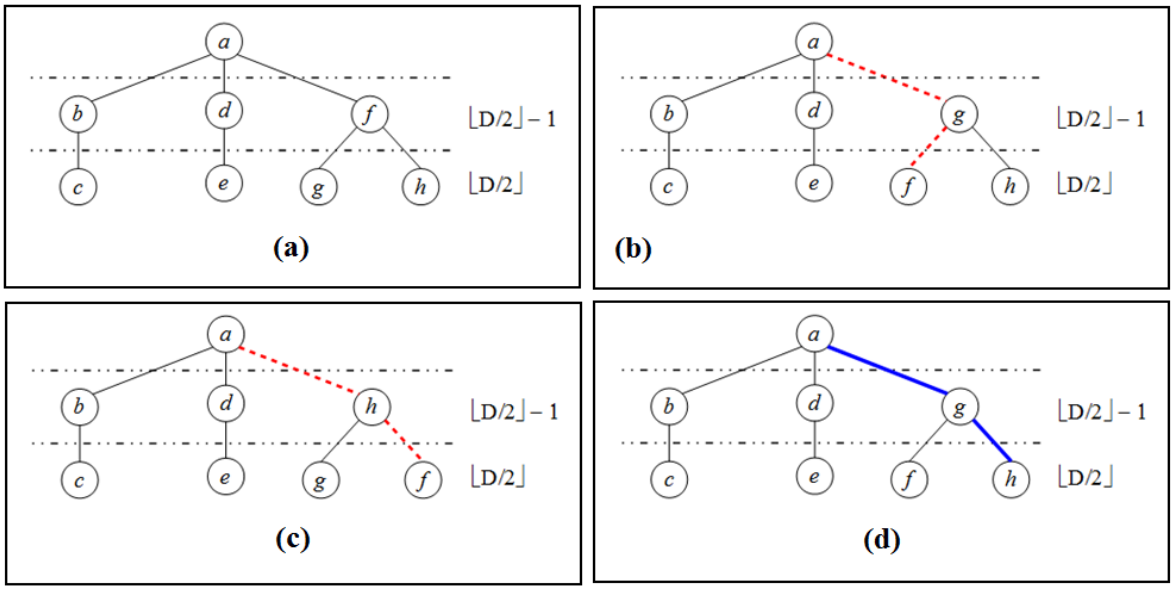
4.2.3 SUBTREE OPTIMIZE

A *Subtree Optimize* (RAIDL; JULSTROM, 2003a) explora a relação entre os nós com profundidade $\lfloor D/2 \rfloor$ e seu predecessor direto v, cuja profundidade é $\lfloor D/2 \rfloor - 1$. Basicamente, esta vizinhança rearranja a subárvore enraizada em v de forma ótima. Seja S o conjunto de vértices nesta subárvore e p o predecessor direto de v. A operação tenta, para cada vértice u em $S - \{v\}$, tornar u a raiz de uma subárvore, conectando-o a p. Neste caso, v toma o lugar ocupado por u antes da mudança. O custo da subárvore com raiz em u é dado pela expressão:

$$w_s(u) = w_{pu} + \sum_{x \in S - \{u\}} w_{ux}$$

A subárvore que minimiza este custo é escolhida. Se nenhuma melhora no custo for obtida, nenhum rearranjo local é feito. Note que essa operação não viola o diâmetro.

Figura 5: Exemplo de Subtree Optimize.



Seja a árvore T indicada pela Figura 5 (a). A operação seleciona o nó f, cuja profundidade é $\lfloor D/2 \rfloor - 1$. A seguir, cada um dos sucessores de f (neste caso g e h), são colocados como a raiz de uma nova subárvore e o custo é verificado, conforme ilustrado pelas Figura 5 (b) e (c). A Figura 5 (d) ilustra a árvore resultante, onde a comutação do nó g com f produz uma árvore resultante de menor custo.

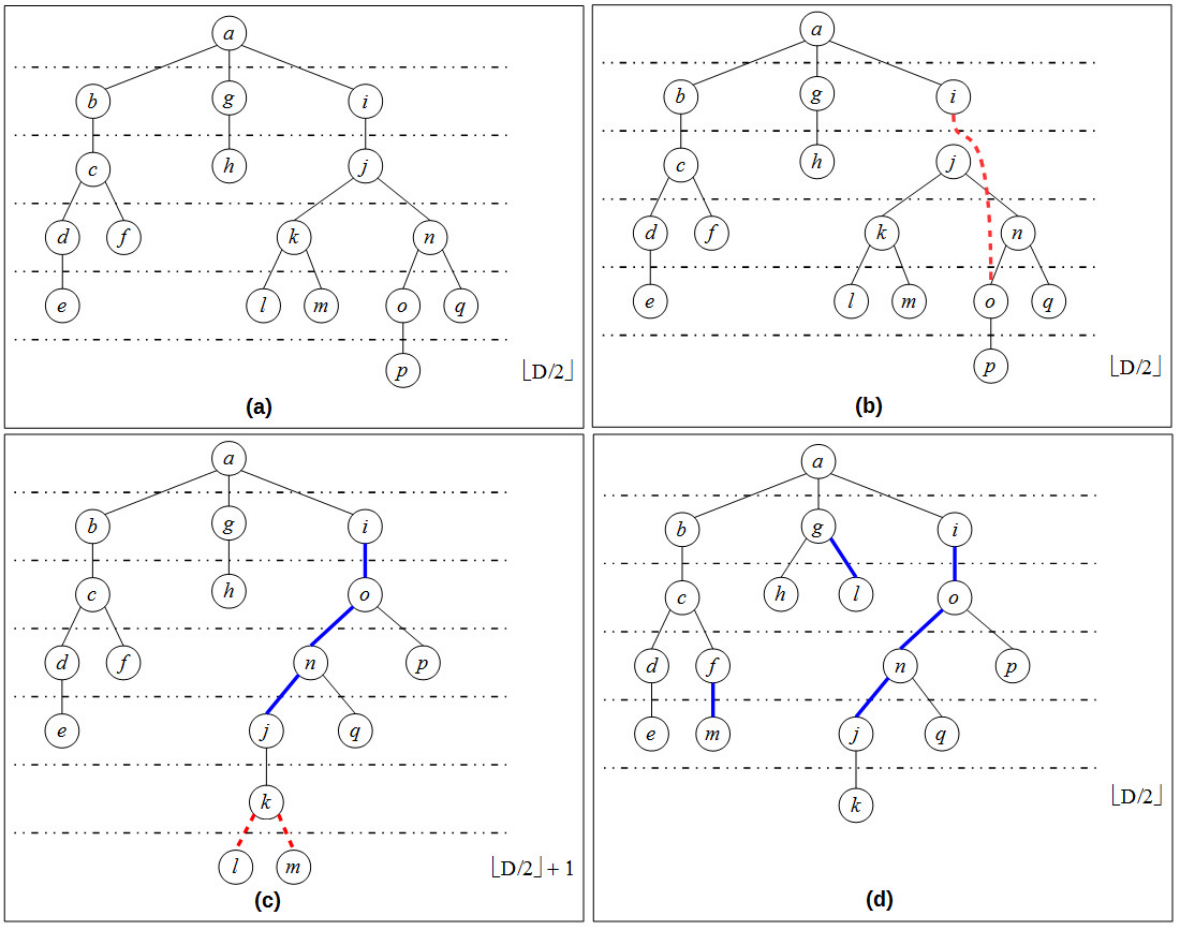
Para um nó fixo v, determinar o custo de uma configuração de subárvore requer um tempo de $O(|S|)$, onde $|S| = d(v) + 1$. Visto que existem $O(d(v))$ possíveis configurações a serem verificadas, então a complexidade desta operação é $O(d(v)^2)$.

4.2.4 HIERARCHY EXCHANGE

Esta vizinhança explora a inversão gradual na ordem hierárquica de um nó v e seus sucessores. A operação é aplicada a todos os nós da BDMST até encontrar uma árvore resultante que minimize o custo e não viole o diâmetro. A operação é descrita a seguir.

Seja T uma BDMST, e p, v, u_1, u_2, …, u_{k-1}, u_k, nós de T, onde v é a raiz de uma subárvore, p é o predecessor direto de v, u_1 é o sucessor direto de v, u_2 é o sucessor direto de u_1, e assim sucessivamente. Suponha que o custo da aresta (p, u_k) seja menor que o custo da aresta (p, v). Então, a operação torna u_k sucessor direto de p, u_{k-1} sucessor direto de u_k, u_{k-2} sucessor direto de u_{k-1}, e assim sucessivamente até v, o qual se torna o sucessor direto de u_1. Se após essa operação algum nó violar o diâmetro, então ele será desconectado da árvore e reconectado em outro local utilizando a aresta de menor custo possível.

Figura 6: Exemplo de Hierarchy Exchange.



A Figura 6 (a) ilustra a BDMST T inicial. Para iniciar a operação, o nó j é escolhido como a raiz de uma subárvore, sendo o nó i o seu predecessor direto. A seguir verifica-se que a aresta (i, o) tem um custo menor que a (i, j), sendo o um descendente de j, conforme

ilustrado pela Figura 6 (b); então é aplicada uma inversão na ordem hierárquica entre o nó *o* e os seus antecessores até o nó *j*. Neste caso, *n* passa a ser o sucessor direto de *o*, já que *n* é o antecessor direto de *o*; *j* torna-se o sucessor direto de *n*, já que *j* é antecessor direto *de n*. O resultado desta operação pode ser vista pela Figura 6 (c). Neste ponto, verifica-se que os vértices *l* e *m* violam o diâmetro, devendo serem desconectados da árvore e reconectados em outras posições usando arestas de menor custo possível, que são (*g*, *l*) e (*f*, *m*), respectivamente, conforme ilustrado pela Figura 6 (d).

Cada operação de inversão de hierarquia entre predecessores e sucessores gasta O(n). Ao final, se o diâmetro de um nó é violado, o tempo para reconectá-lo em outro local da árvore é de O(n). No pior caso, O(n) nós necessitam serem reconectados, produzindo uma complexidade de tempo O(n^2) para a inversão e redistribuição dos nós. Visto que há O(n) nós que podem ser o ponto de partida da operação de inversão da hierarquia, então a complexidade geral para esta vizinhança é O(n^3).

4.2.5 HIERARCHY ROTATION

Esta vizinhança é similar à anterior, mas explora a inversão gradual na ordem hierárquica de um nó *v* e seus predecessores, e a distribuição da subárvore resultante. Novamente, a operação é aplicada a todos os nós da BDMST até encontrar uma árvore resultante que minimize o custo e não viole o diâmetro. A operação é descrita a seguir.

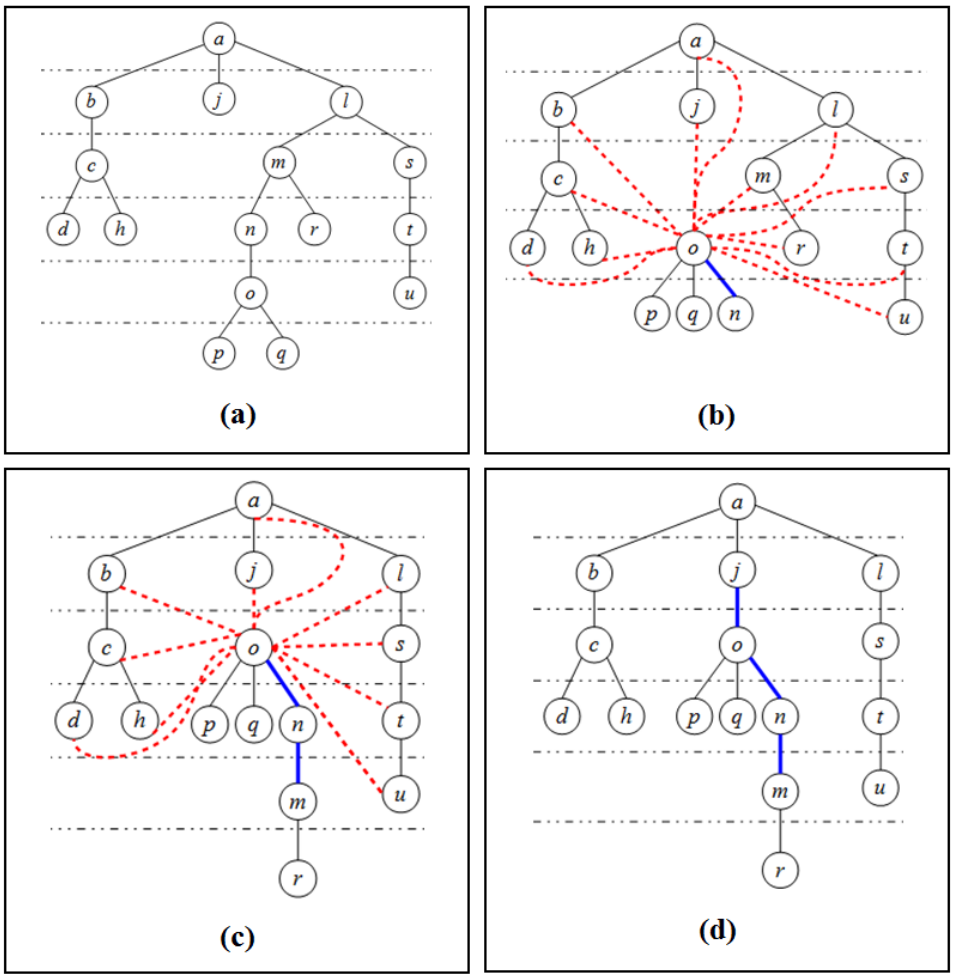
Seja *T* uma BDMST, e *c*, *v*, u_1, u_2, u_3, …, u_{k-1}, u_k nós de *T*, onde *v* é a raiz de uma subárvore, u_1 é o predecessor direto de *v*, u_2 é o predecessor direto de u_1, e assim sucessivamente até u_k, o qual é o predecessor direto de u_{k-1} e sucessor direto do centro *c* (ou um dos centros, se o diâmetro é ímpar). A operação inicia-se tornando u_1 sucessor direto de *v* e removendo a aresta (u_1, u_2). A subárvore obtida, cuja raiz é *v*, é reconectada em outra posição da árvore, desde que o custo diminua e o diâmetro não seja violado. Caso o custo da BDMST resultante não diminua ou o diâmetro seja violado, então a aresta (u_3, u_2) é removida

e u_2 se torna o sucessor direto de u_1, formando a aresta (u_1, u_2). Novamente, a subárvore resultante, cuja raiz é *v,* é reconectada em outra posição da árvore, desde que o custo diminua e o diâmetro não seja violado. Se o custo da árvore não diminuir, o processo é repetido até que o nó u_k seja analisado; nesse caso, a aresta (c, u_k) é removida e u_k é feito o sucessor direto de u_{k-1}, formando a aresta (u_{k-1}, u_k), e subárvore resultante é conectada em outra posição, desde que o custo diminua e o diâmetro não seja violado.

A Figura 7 (a) ilustra a BDMST *T*. Primeiramente, o nó *o* é escolhido para iniciar a operação, o nó *n* (predecessor de *o*) é feito sucessor direto de *o* e a aresta (*m*, *n*) é removida. A seguir a operação verifica se a conexão da subárvore enraizada em *o* em outro local da BDMST diminui o custo, conforme ilustrado pela Figura 7 (b). Se o custo não diminuir, então a operação é continuada e o nó *m*, antigo predecessor de *n*, passa a ser o sucessor direto de *n*, o que pode ser visto pela Figura 7 (c). Novamente, a subárvore resultante, com raiz *o*, é testada em todas as possíveis posições (Figura 7 (c)), visando diminuir o custo e não violar o diâmetro. De acordo com a Figura 7 (d), tornando *j* o predecessor direto de *o*, o custo da BDMST diminui e o diâmetro não é violado. Então *o* passa a ser o sucessor direto de *j*, formando a aresta (*j*, *o*), e o processo é finalizado.

Há O(n) possíveis nós para iniciar a operação de rotação da hierarquia. Cada operação envolve O(n) inversões entre pares de predecessores e sucessores. Cada operação de inversão necessita de O(n) testes de reconexões de subárvores para verificar se o custo da árvore diminuiu. Assim, o custo total dessa operação é O(n^3).

Figura 7: Exemplo de Hierarchy Rotation.

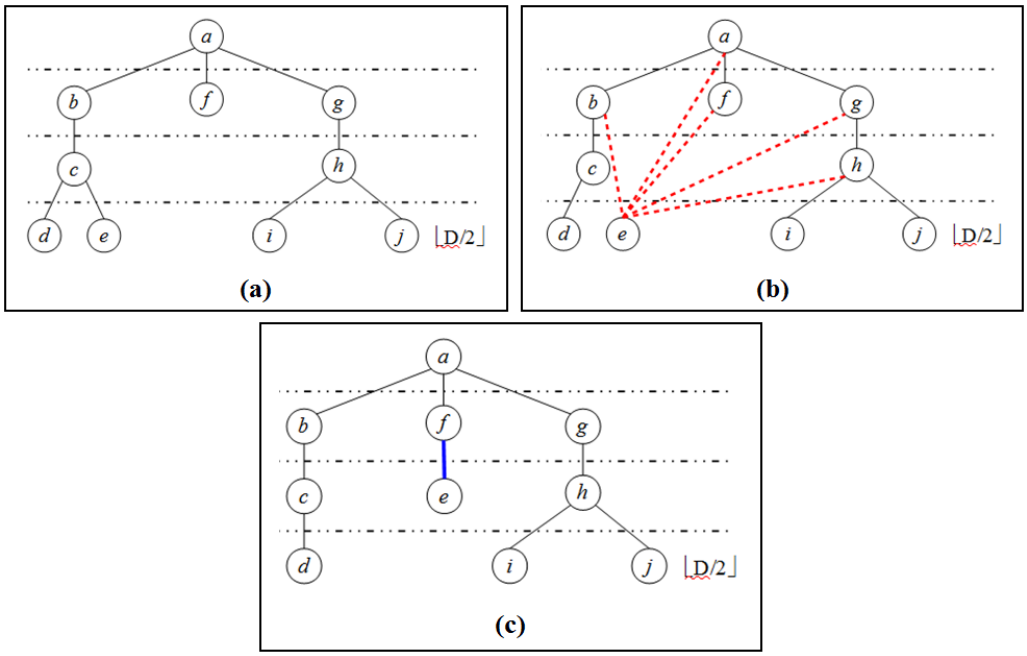


4.2.6 LEAF SWAP

Esta vizinhança explora a distribuição de uma folha ao longo de toda a BDMST, visando diminuir o custo, desde que o diâmetro não seja violado.

Seja *T* a BDMST da Figura 8 (a). A operação seleciona uma folha de *T*, nesse caso o nó *e*. Posteriormente, a operação verifica se colocar o nó *e* como um sucessor direto de outro nó não folha de *T* diminui o seu custo e não viola o diâmetro, conforme ilustrado pela Figura 8 (b). Nesse caso, a escolha de *e* como sucessor direto de *f* diminui o custo da árvore, sendo essa a BDMST resultante, conforme ilustrado pela Figura 8 (c).

Figura 8: Exemplo de Leaf Swap.



Explorar todas as folhas da BDMST leva um tempo O(n), enquanto que testar uma folha em todas as posições da árvore leva tempo O(n), o que resulta em um tempo total de O(n^2) para essa vizinhança.

4.2.7 FATHER SWAP

Esta vizinhança explora se a conexão de uma subárvore com raiz v, contendo uma única folha u como sua sucessora direta, em outra posição da árvore reduz o seu custo. Caso o custo da árvore resultante não seja reduzido, a operação inverte a relação entre os nós v e u (isto é, u passa a ser a nova raiz da subárvore e v o seu sucessor direto) e refaz o mesmo teste, tentando mover a nova subárvore para outro local da árvore, visando reduzir o seu custo. Em cada operação o diâmetro não deve ser violado.

Seja a BDMST T da Figura 9 (a). Primeiro, a operação seleciona o nó d, visto que ele possui apenas um nó folha como sucessor direto. A seguir, a operação verifica se mover a subárvore com raiz em d para outra posição da árvore reduz o seu custo, conforme ilustrado pela Figura 9 (b). Caso isso não ocorra, o nó e passa a ser o predecessor direto de d, e d passa

a ser o sucessor direto de *e*, Figura 9 (c). A seguir, o mesmo teste ilustrado pela Figura 9 (b) é realizado, mas agora com a subárvore tendo *e* como sua raiz – vide a Figura 9 (c). De acordo com a Figura 9 (d), tornar o nó *e* um sucessor direto de *f* reduz o custo da árvore e, portanto, a operação é finalizada.

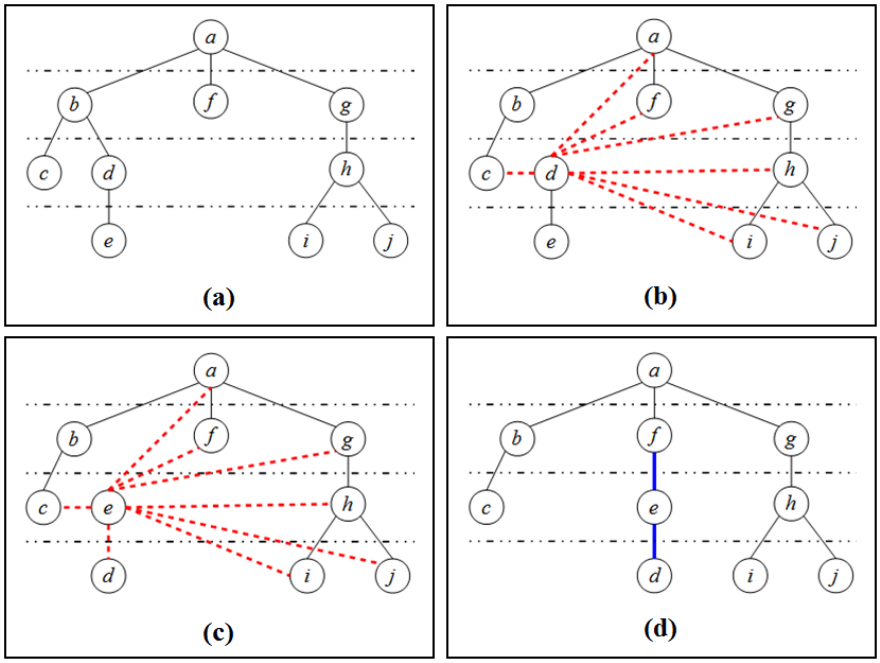
Explorar todos os nós de *T* para selecionar aquele que tem somente um nó folha como sucessor direto leva um tempo O(n). Verificar se a conexão do nó selecionado em outro local da árvore reduz o seu custo gasta um tempo O(n). A operação de inversão entre predecessor e sucessor consome um tempo constante, enquanto que verificar se mover a subárvore resultante dessa inversão para outro local da árvore reduz o custo, toma um tempo O(n). Logo, o custo total dessa vizinhança é O(n^2).

4.2.8 LEVEL CHANGE

Esta vizinhança verifica se mover uma subárvore, cuja raiz é *v* e que está no nível *k*, para outra posição na árvore, onde *v* se torna um sucessor direto de um nó *u* que está no nível *l*, $l \geq k$, reduz o custo da árvore e não viola o diâmetro.

Seja *T* a BDMST da Figura 10 (a). A operação seleciona o nó *i* (no nível três) como raiz da subárvore. A seguir verifica-se se conectar *i* como sucessor direto de um nó no nível três (*d*, *e*, *m* ou *o*) ou de um nó no nível quatro (*p*) reduz o custo da árvore – veja a Figura 10 (b). No exemplo, conectar a subárvore ao nó *e* reduz o custo da árvore e não viola o diâmetro; logo esse nó é selecionado para ser o predecessor direto de *i*, sendo a árvore resultante a indicada pela Figura 10 (c).

Figura 9: Exemplo de Father Swap.

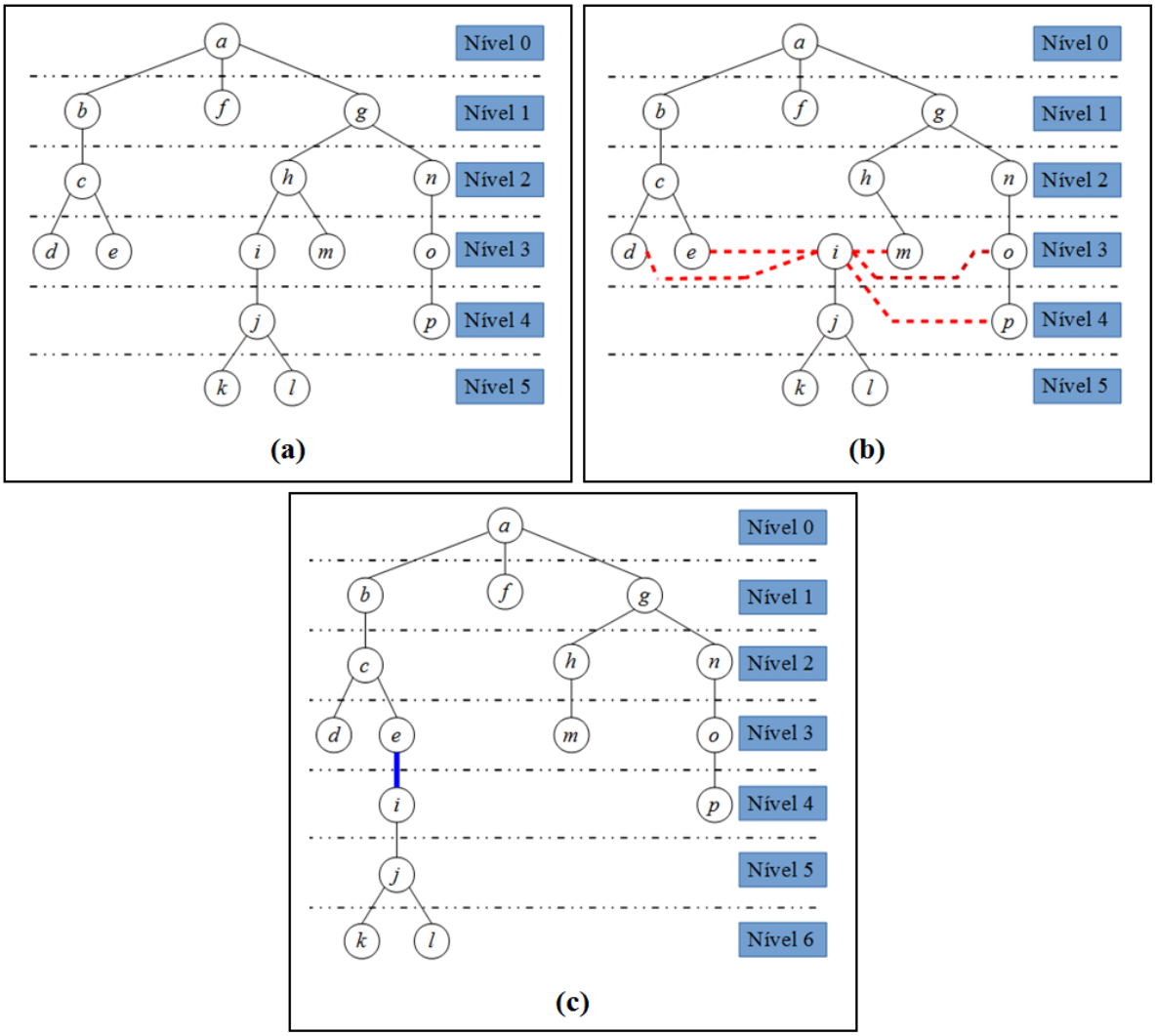


Há O(n) possíveis subárvores a serem selecionadas para a operação. Para um subárvore selecionada, o tempo para verificar se conectá-la aos nós no mesmo nível ou nos níveis inferiores reduz o custo da árvore é O(n), enquanto que a conexão da subárvore toma um tempo constante. Portanto, o tempo gasto para esta vizinhança é O(n^2).

4.3 PERTURBAÇÃO

Como rotinas de perturbação, foram empregadas variações das vizinhanças *Edge Exchange* e *Node Swap* (GRUBER; RAIDL, 2005b), onde a escolha da raiz da subárvore a ser utilizada na operação é aleatória. Também foram utilizadas variações da *Edge Delete* (RAIDL; JULSTROM, 2003a), onde a reconexão de um vértice desconectado da árvore é feita através da aresta de menor custo, e *Center Exchange* (GRUBER; RAIDL, 2005b), onde o antigo centro passa a ser sucessor de um nó escolhido aleatoriamente, sendo esta vizinhança denominada de *Center Change*. A escolha de qual vizinhança será aplicada em cada chamada da perturbação é feita aleatoriamente. O histórico das soluções exploradas anteriormente não foi considerado, conforme discutido na seção 5.3.

Figura 10: Exemplo de Level Change.

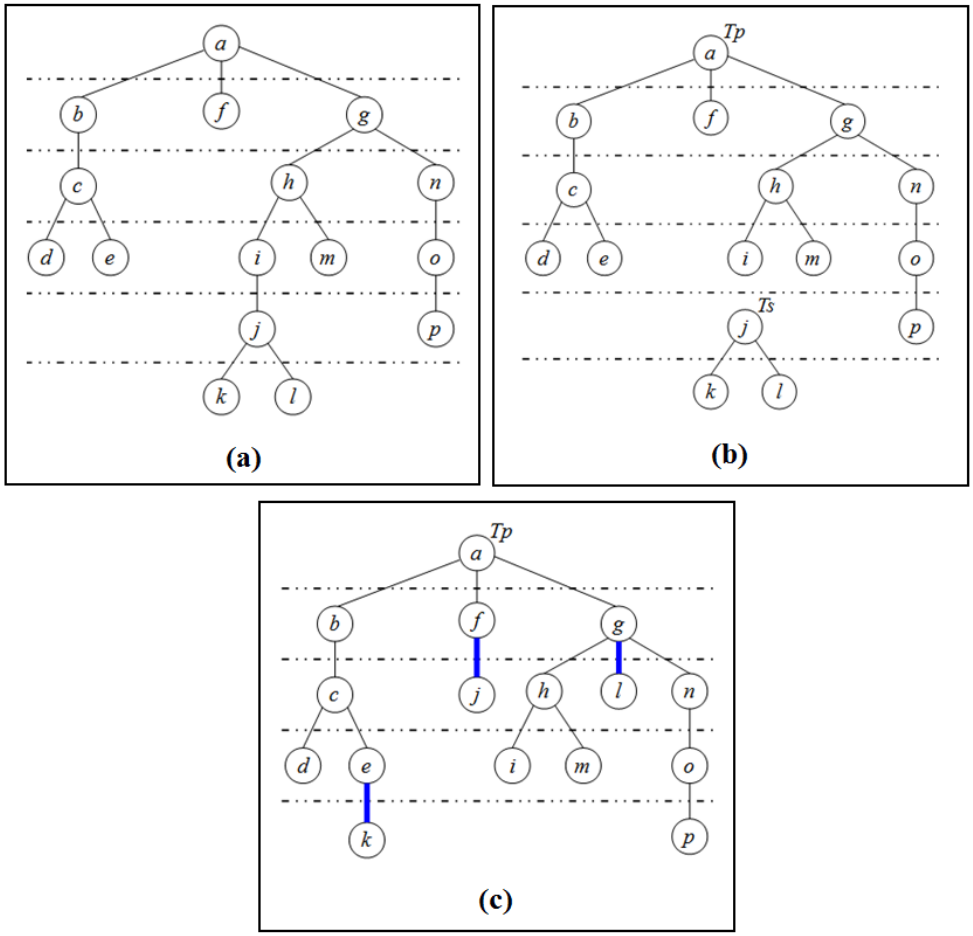


4.3.1 EDGE DELETE

Seja T a BDMST da Figura 11 (a). Esta vizinhança remove aleatoriamente uma aresta de T, formando duas subárvores T_p e T_s, o que pode ser visto pela Figura 11 (b), onde a aresta (i, j) foi removida. A seguir todos os nós de T_s são desconectados e individualmente reconectados em outros locais de T_p usando arestas de menor custo possível o que é ilustrado pela Figura 11 (c). Como sempre, o diâmetro da árvore resultante não deve ser violado.

O tempo para se selecionar aleatoriamente um dos nós de T é constante. A subárvore T_s pode conter O(n) nós, cada um necessitando de um tempo O(n) para ser conectado em alguma posição de T_p. Isto resulta em um tempo total de O(n^2) para esta vizinhança.

Figura 11: Exemplo de Edge Delete.

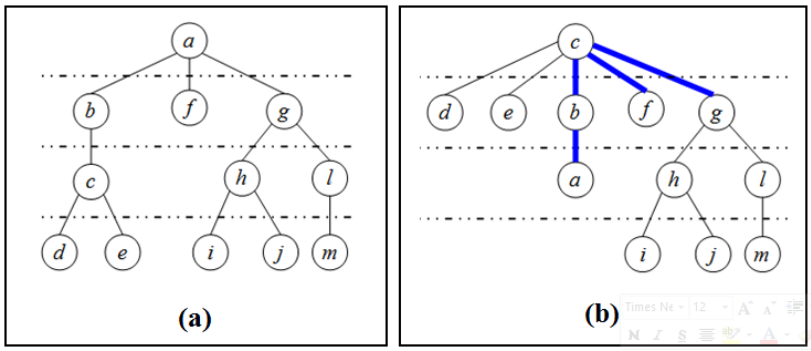


4.3.2 CENTER CHANGE

Seja T uma BDMST. Esta vizinhança seleciona aleatoriamente um nó de T (suponha que c seja o nó selecionado, conforme ilustrado pela Figura 12 (a)) e o torna o novo centro, se o diâmetro é par, ou um dos centros, se o diâmetro é ímpar. Os sucessores do antigo centro (o nó a, no exemplo) passam a serem sucessores diretos de c, enquanto que a passa a ser sucessor direto de um nó escolhido aleatoriamente (no exemplo, de acordo com a Figura 12 (b), a se torna o sucessor direto do nó b), desde que o diâmetro não seja violado.

As Figura 12 (a) e 11 (b) ilustram a operação dessa vizinhança.

Figura 12: Exemplo de Center Change.



O tempo para se selecionar aleatoriamente um nó x de T é constante. O tempo para tornar x o novo centro da árvore é constante, enquanto que tornar os sucessores do antigo centro y sucessores de x é $O(\Delta)$, onde Δ é o número de sucessores diretos de y. O tempo para se conectar o antigo centro y como um sucessor direto de um nó da árvore escolhido aleatoriamente é constante. Logo, o tempo total desta vizinhança é $O(\Delta)$, onde Δ é o grau máximo de um nó na árvore.

4.4 IMPLEMENTAÇÃO DO ILS

A implementação do ILS proposto é apresentada no Algoritmo 7. A heurística construtiva RGH (RAIDL; JULSTROM, 2003a) foi utilizada para a geração da solução inicial (linha 5), já que para diâmetros pequenos ela produz melhores resultados que a CM . O critério de aceitação adotado foi simplesmente a comparação dos custos da solução corrente s^* com a nova solução $s^{*'}$, gerada pela busca local, sendo a solução de menor custo tornada a solução corrente s^* (linha 10). O histórico das soluções exploradas anteriormente não foi considerado. A variável *it* é utilizada para contar o número de vezes que a busca local não foi capaz de melhorar a solução perturbada s' (linhas 4, 12 e 14), a qual poderá ou não ser usada como uma das condições de término do laço de repetição (linha 16), conforme discutido no 0.

Algoritmo 7: Implementação do ILS

```
1.  procedimento ILS

2.  Entrada: um grafo completo e conexo G = (V, E)
3.  Saída: uma BDMST

4.      it ← 1
5.      s0 ← RGH()
6.      s* ← BuscaLocal(s0)
7.      repita
8.          s' ← Perturbação(s*)
9.          s*' ← BuscaLocal(s')
10.         se custo(s*') < custo(s*) então
11.             s* ← s*'
12.             it ← 1
13.         senão
14.             it ← it + 1
15.         fim-se
16.     até a condição de término ser atingida
17.     retorne s*
18. fim-procedimento
```

O Algoritmo 9 apresenta a implementação da *busca local*. Nele, *table* é uma matriz bidimensional a qual contém todas as 8! (40.320)[15] permutações de vizinhanças possíveis de serem aplicadas (linha 9). Esta matriz é inicializada antes da chamada do procedimento ILS, sendo o tempo destinado a esta inicialização desconsiderado no cálculo do tempo de processamento do ILS. A variável *V* corresponde a um índice, escolhido aleatoriamente, para acesso a uma das permutações de vizinhanças contidas em *table* (linha 4). Da mesma forma que o Algoritmo 6, *k* (linha 5) é utilizado para indicar qual vizinhança será aplicada a solução corrente, enquanto *r* indica o número total de vizinhanças a serem utilizadas pela busca local, que no caso são 8 (linha 6). A vizinhança a ser aplicada é obtida acessando-se *table*[*V*][*k*] (linha 9) e avaliando-se a estrutura caso (linha 10). O restante do algoritmo é idêntico ao Algoritmo 6, o qual já foi discutido.

[15] Como há oito vizinhanças a serem utilizadas pela busca local, então tem-se 8! permutações possíveis.

Algoritmo 9: Implementação da Busca Local

```
1.  prodedimento BuscaLocal

2.  entrada: uma BDMST A
3.  saída: uma BDMST A*

4.      inicialize aleatoriamente V com um valor entre 1 e 8! = 40320
5.      k ← 1
6.      r ← 8
7.      A* ← A
8.      repita
9.          index ← table[V][k]
10.         caso index for
11.             1: A' ← EdgeExchange(A*)
12.             2: A' ← NodeSwap(A*)
13.             3: A' ← SubtreeOptmize(A*)
14.             4: A' ← LevelChange(A*)
15.             5: A' ← LeafSwap(A*)
16.             6: A' ← FatherSwap(A*)
17.             7: A' ← HierarchyExchange(A*)
18.             8: A' ← HierarchyRotation(A*)
19.         fim-caso
20.         se custo(A') < custo(A*) então
21.             A* ← A'
22.             k ← 1
23.         senão
24.             k ← k + 1
25.         fim-se
26.     até k > r
27.     retorne A*

28. fim-procedimento
```

Algoritmo 8: Implementação de Perturbação

```
1.  procedimento Perturbação

2.  entrada: uma BDMST A
3.  saida: uma BDMST A*

4.      inicialize aleatoriamente k com um dos valores: 1, 2, 3, 4
5.      caso k for
6.          1: A* ← NodeSwap Aleatório(A)
7.          2: A* ← EdgeExchange Aleatório(A)
8.          3: A* ← EdgeDelete(A)
9.          4: A* ← CenterChange(A)
10.     fim-caso
11.     return A*

12. fim-procedimento
```

A implementação da *perturbação* é apresentada pelo Algoritmo 8. Nela, *k* é a vizinhança a ser aplicada, a qual é escolhida aleatoriamente (linha 4), e aplicada pela avaliação da estrutura caso (linha 5). O histórico das soluções exploradas anteriormente não é considerado.

CAPÍTULO 5 – RESULTADOS COMPUTACIONAIS

5.1 RESULTADOS DA HEURÍSTICA CONSTRUTIVA PROPOSTA

A heurística construtiva proposta no Capítulo 3, denominada de CM, foi implementada em Java e executada em um computador com processador Intel I7, 2.7 GHz, 4 GB RAM, rodando o sistema operacional Windows, e os resultados obtidos foram comparados com os resultados apresentados em (JULSTROM, 2009) para OTTC, CBTC e RTC. O algoritmo RGH foi reimplementado neste trabalho, já que ele gera boas soluções para o EBDMSTP. Para obtenção dos resultados foram utilizadas as 15 instâncias de grafos completos oriundos da OR-Library, onde o custo das arestas é a distância euclidiana entre dois pontos, e 15 instâncias cujos custos das arestas foram gerados de maneira aleatória, utilizando o intervalo de 0,01 a 0,99. A Tabela 1 apresenta os resultados desta comparação. A coluna *Instâncias* descreve duas informações: o número de vértices do grafo a ser considerado para cada instância, que no caso são n = 100, 250, 500 e 1000 vértices; e *min d*, que identifica o diâmetro mínimo das árvores geradoras de custo mínimo e sem restrições para o valor de n considerado. A coluna D indica os diâmetros a serem utilizados, os quais devem ser inferiores a *min d*. As colunas OTTC, CBTC, RTC, RGH e CM descrevem a média dos melhores resultados obtidos para cada uma das 30 instâncias testadas e diâmetros especificados.

Analisando a Tabela 1 verifica-se que, para diâmetros pequenos, OTCC, CBTC e CM apresentam os piores resultados, enquanto RTC e RGH apresentam os melhores resultados. Já quando os diâmetros tendem a se aproximarem do diâmetro mínimo de uma árvore geradora de custo mínimo e sem restrições, OTTC, CBTC e CM passam a ter um melhor desempenho, sendo os resultados melhores que RTC e RGH, sendo que, nesses casos, os resultados do CM são melhores que os demais.

Tabela 1: comparação entre as heurísticas OTTC, RGH, CBTC, RTC e CM.

Instâncias	*D*	OTTC	RGH	CBTC	RTC	CM
$n = 100$	5	29,38	15,35	26,48	15,3	28,12
$min\ d = 29$	10	18,43	9,79	15,59	9,38	16,14
	15	12,84	9,20	10,95	9,25	10,61
	25	8,06	9,15	7,69	9,19	7,46
$n = 250$	10	58,2	16,86	49,07	16,89	52,51
$min\ d = 48$	15	41,59	15,24	36,44	15,3	37,57
	20	32,55	14,97	26,17	15,05	25,03
	40	14,43	14,96	12,51	14,98	11,61
$n = 500$	15	106,87	22,11	94,38	22,26	98,48
$min\ d = 91$	30	58,82	21,36	46,51	21,54	40,53
	45	32,23	21,36	25,63	21,43	18,87
	60	20,33	21,36	17,72	21,45	16,12
$n = 1000$	20	217,71	31,23	195,96	31,15	215,11
$min\ d = 152$	40	124,21	30,82	99,57	30,85	86,65
	60	60,83	30,82	50,97	30,84	32,05
	100	28,95	30,82	23,41	30,84	22,59

Para exemplificar melhor os resultados, a Figura 14 mostra a BDMST de menor custo gerada para a segunda instância de $n = 250$ com $D = 15$: (a) ilustra o resultado de OTCC, com um custo de 52,38, (b) mostra a solução obtida por CBTC, com custo 32,44, (c) apresenta RTC, com custo 15,08, (d) mostra RGH, com custo 15,23, e (e) exibe CM, com custo 38,30. Pela análise da figura pode-se verificar que OTTC, CBTC e CM têm uma espinha dorsal mais curta, o que faz com que as folhas se liguem à árvore por meio de arestas mais longas, aumentando o custo final. Este fato não ocorre com RTC e RGH, os quais apresentam uma espinha dorsal mais longa, o que proporciona um custo menor para a árvore gerada.

Figura 14: BDMST para OTTC, CBTC, RTC, RGH e CM – n = 250 e D = 15.

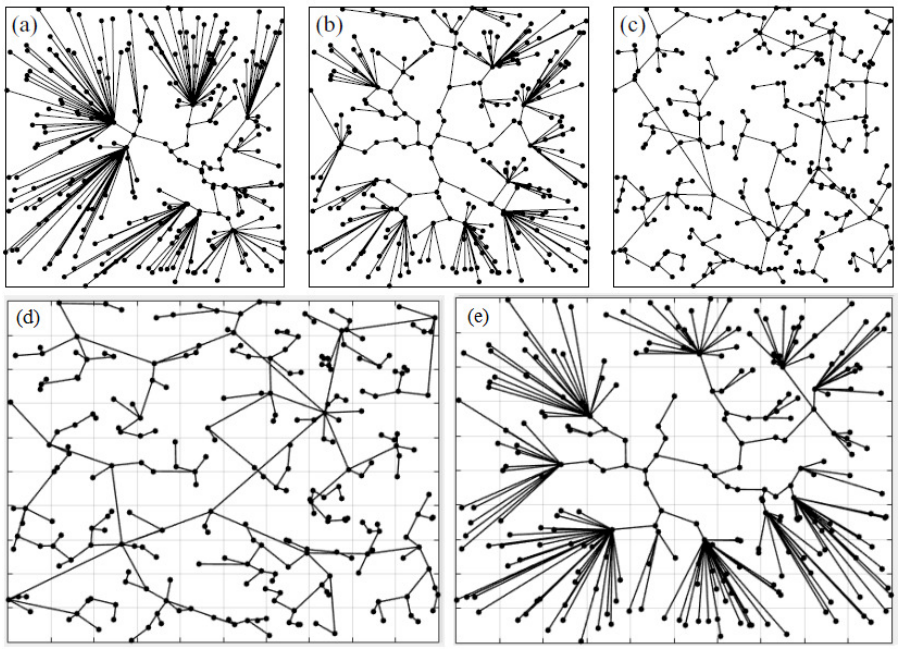
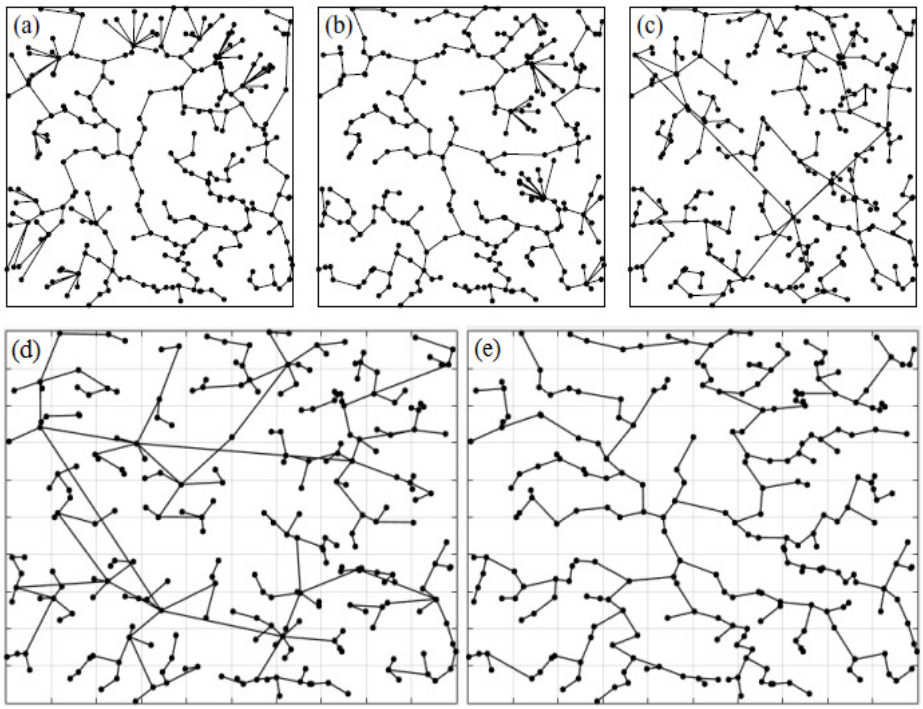


Figura 13: BDMST para OTTC, CBTC, RTC, RGH e CM – n = 250 e D = 40.



A Figura 13 ilustra a BDMST de menor custo gerada para a segunda instância de n = 250 com D = 40: (a) ilustra o resultado de OTCC, com um custo de 14,43, (b) mostra a

solução obtida por CBTC, com custo 12,67, (c) apresenta RTC, com custo 14,84, (d) mostra RGH, com custo 14,85, e (e) exibe CM, com custo 11,28. Pela observação da figura, verifica-se que as espinhas dorsais de OTTC, CBTC e CM tornam-se mais longas, o que proporciona um melhor resultado, já que as folhas terão que se conectar à árvore por arestas mais curtas. Além disso, verifica-se que CM, com as configurações especificadas, apresenta um resultado melhor que todas as outras heurísticas neste caso.

5.2 ANÁLISE DAS VIZINHANÇAS PROPOSTAS

Para verificar o quanto uma vizinhança melhora o custo da EBDMST, o ILS proposto pelo Algoritmo 7 foi executado com as vizinhanças *Hiearchy Exchange*, *Hierarchy Rotation*, *Leaf Swap*, *Father Swap* e *Level Change* combinadas individualmente com as modificações das vizinhanças *Edge Exchange*, *Node Swap* e *Subtree Optmize*, presentes na literatura. Os resultados encontram-se descritos na Tabela 2. Para os testes foram utilizadas as cinco primeiras instâncias de grafos completos e conexos de 250 vértices, presentes na OR-Library, utilizando o diâmetro 15. Como critério de aceitação para o ILS foi adotado o tempo de um segundo. Na Tabela 2, a coluna "*Id*" indica qual instância de 250 vértices está sendo considerada e a coluna (1) corresponde à execução do RGH. As demais colunas apresentam os custos relativos à execução do ILS utilizando as vizinhanças *Edge Exchange*, *Node Swap* e *Subtree Optmize* em conjunto com as vizinhanças: *Hierarchy Exchange* (2), *Hierarchy Rotation* (3), *Leaf Swap* (4), *Father Swap* (5) e *Level Change* (6).

Pelos resultados presentes na tabela pode-se verificar que todas as combinações de vizinhanças melhoram os custos das EBDMST obtidas pelo RGH como soluções iniciais ao ILS. Além disso, também pode ser observado que a combinação de vizinhanças que mais melhora os custos da árvore é aquela que utiliza o *Hiearchy Rotation*, seguida pela *Hierarchy Exchange*, conforme indicado pelas colunas (3) e (2), respectivamente.

Tabela 2: análise das vizinhanças.

Id	(1)	(2)	(3)	(4)	(5)	(6)
1	15,32	13,00	12,89	13,46	13,39	13,44
2	15,23	12,87	12,65	13,30	13,22	13,37
3	15,01	12,81	12,67	13,07	13,04	13,01
4	15,48	13,51	13,23	13,60	13,53	13,73
5	15,36	13,05	12,94	13,47	13,29	13,44

5.3 PARÂMETROS PARA A EXECUÇÃO DO ILS

Para a execução do ILS proposto no Capítulo 4, foram utilizadas três abordagens para determinar a condição de parada (*condição de término*, no Algoritmo 7) para obtenção dos três resultados apresentados neste trabalho. No primeiro caso, foram utilizados como condição de parada os tempos de 1000, 2000, 3000, 4000 e 5000 segundos, para instâncias de 50, 100, 250, 500 e 1000 vértices, respectivamente, e 1000 iterações do ILS sem melhoria no custo da árvore, condições essas similares àquelas adotadas pelo VNS (GRUBER; RAIDL, 2005b), LEEA e ACO (GRUBER; VAN HEMERT; RAIDL, 2006) e GILS (LUCENA; RIBEIRO; SANTOS, 2010). No segundo caso foram utilizados apenas os tempos de 1, 10, 160, 1200 e 2000 segundos, para instâncias de 50, 100, 250, 500 e 1000 vértices, respectivamente, como condição de parada, tempos estes determinados empiricamente através de experimentos prévios. Já para o terceiro caso, foram, utilizados como condição de parada, apenas os tempos constantes da literatura, ou seja, 1000, 2000, 3000, 4000 e 5000, para instâncias de 50, 100, 250, 500 e 1000 vértices, respectivamente.

Conforme citado na seção 4.3, na implementação da perturbação não foi levado em consideração o histórico das computações passadas. Tal abordagem é semelhante à adotada por (LUCENA; RIBEIRO; SANTOS, 2010), (DA SILVA; DE MENEZES FROTA; SUBRAMANIAN, 2012), (PENNA; SUBRAMANIAN; OCHI, 2013) e (MARTINS et al., 2015), que obtiveram bons resultados para o ILS sem considerar o uso de histórico na

perturbação. Por esse motivo, foi adotado o mesmo procedimento no algoritmo proposto, além de que, nos experimentos realizados, os resultados obtidos pelo ILS já se apresentavam melhores que os da literatura.

Outro ponto a considerar é a não utilização do histórico de computações no critério de aceitação, sendo aceitas somente soluções que melhorem o custo da BDMST, conforme descrito na seção 4.4. Isto se deve ao fato de outros autores também não o terem utilizado e terem conseguido bons resultados, considerando apenas a comparação dos custos e escolhendo sempre o melhor, conforme pode ser verificado em (LUCENA; RIBEIRO; SANTOS, 2010), (PENNA; SUBRAMANIAN; OCHI, 2013), (DONG; HUANG; CHEN, 2009) e (MARTINS et al., 2015). Além disso, os experimentos realizados com o ILS sem o uso de um histórico no critério de aceitação já apresentaram resultados melhores que os da literatura, reforçando a não necessidade do seu uso.

5.4 RESULTADOS DO ILS PROPOSTO

O ILS proposto no Capítulo 4 foi aplicado a instâncias de grafos conexos, completos e euclidianos da OR-Library, sendo os resultados comparados com os melhores resultados encontrados na literatura: VNS (GRUBER; RAIDL, 2005b), LEEA e ACO (GRUBER; VAN HEMERT; RAIDL, 2006), PEA-I (SINGH; GUPTA, 2007) e GILS (LUCENA; RIBEIRO; SANTOS, 2010). Para a realização dos testes, foram utilizadas as cinco primeiras instâncias de cada conjunto de grafos com 50, 100, 250, 500 e 1000 vértices, com os diâmetros 5, 10, 15, 20 e 25, respectivamente. Para cada uma das cinco instâncias de cada conjunto de grafos foram feitas 50 execuções do ILS, similar ao esquema adotado pela literatura. Os algoritmos foram implementados em Java 7, executando no sistema operacional Ubuntu em um computador I7 de 3.4 GHz, com 16 GB de RAM. Os resultados podem ser visualizados na Tabela 4, Tabela 6 e Tabela 7. A coluna "*n*" indica o número de vértices do grafo de entrada;

"*D*" o diâmetro considerado; "*Id*" o número da instância; "*Ref.*" o método que obteve o melhor resultado encontrado na literatura; "*Melhor*" o valor da melhor solução obtida pelo método indicado pela coluna anterior; "*Média*" o valor da solução média obtida após várias execuções; "*Desvio Padrão*" o desvio padrão obtido após as execuções; "*Tempo Médio (s)*" a média de tempo gasto para cada execução da instância considerada (em segundos). As colunas *"Melhor*"*, "*Média*"* e *"Desvio Padrão*"* possuem o mesmo significado que as anteriores, mas aplicadas ao ILS proposto. A coluna "*Tempo Médio 1**" apresenta ou a média de tempo de processamento gasto para as 50 execuções (Tabela 4), ou o tempo para obter a melhor solução considerando o limite de tempo especificado para cada execução (Tabela 6 e Tabela 7), sendo todos eles resultantes da execução do ILS no computador especificado. Já "*Tempo Médio 2**" apresenta o tempo especificado em "*Tempo Médio 1**" normalizado, o qual foi obtido considerando-se processadores mais próximos àqueles utilizados na literatura, disponíveis no site "cpubenchmark.net/compare", em relação ao computador utilizado para executar os experimentos, sendo relacionados na Tabela 3[16].

Tabela 3: Relação de processadores utilizados para normalização dos tempos.

Método	*Processador*	*Processador Usado para Comparação*	*Fator de Normalização*
GILS	Intel Pentium IV 2.0 GHz	Intel Pentium D1508 2.2 GHz	1,61
PEA-I	Intel Pentium IV 2.4 GHz	Intel Pentium D1508 2.2 GHz	1,61
VNS	Intel Pentium IV 2.8 GHz	Intel Pentium G4560T 2.9 GHz	1,44
LEEA	Intel Pentium IV 2.8 GHz	Intel Pentium G4560T 2.9 GHz	1,44
ACO	Intel Pentium IV 2.8 GHz	Intel Pentium G4560T 2.9 GHz	1,44
ILS Proposto	Intel I7 3.4 GHz	Intel Core I7 7900X 3.3 GHz	*

[16] Para obtenção do tempo normalizado o tempo obtido pela execução do ILS é multiplicado por um fator obtido pela razão da velocidade da máquina sobre a menor velocidade encontrada entre os computadores comparados.

A Tabela 4 apresenta os resultados da execução do ILS, considerando a execução do Algoritmo 7, utilizando como condição de parada (*condição de término*) 1000 iterações do laço principal do ILS sem melhoria no custo da árvore, o que é feito pela análise do conteúdo da variável *it*, e os tempos de 1000, 2000, 3000, 4000 e 5000 segundos para instâncias de 50, 100, 250, 500 e 1000 vértices, respectivamente. Conforme pode ser observado, o ILS proposto foi capaz de melhorar todos os resultados encontrados na literatura (compare as colunas *"Melhor"* e "*Melhor*"*), exceto para a primeira instância de 1000 vértices, demonstrando a competitividade do método. Em relação ao tempo computacional, excetuando-se as instâncias de 1000 vértices, o maior tempo necessário para que o ILS supere o resultado da literatura para uma instância foi de 736,13 segundos, considerando-se a coluna *"Tempo Médio 1**", o que é razoável. Isto demonstra a competividade do método proposto. Além disso, deve ser notado que em cada linha da tabela o tempo médio normalizado gasto (coluna "*Tempo Médio 2**") pelo ILS proposto para encontrar a melhor solução (coluna "*Melhor*"*) é, em 68% dos casos (17 de 25), menor que o tempo especificado pela literatura (coluna "*Tempo Médio"*)[17]. Em todos os casos (exceto para a primeira instância de 1000 vértices), o ILS encontrou melhores soluções com um tempo menor que o tempo especificado como condição de parada. Novamente, para a primeira instância de 1000 vértices, o ILS não conseguiu superar o resultado da literatura. Em relação à coluna "*Média**", o ILS proposto foi capaz de superar em 40% dos casos os resultados da literatura (10 de 25). Os valores para o desvio padrão estão dentro de um intervalo de 0 a 0,082, conforme pode ser verificado pelas colunas "*Desvio Padrão*" e "*Desvio Padrão**", o que é razoável.

[17] Como os tempos para o PEA-I não foram especificados, eles foram excluídos da contagem.

Tabela 4: Resultados computacionais da execução do ILS com 1000 iterações sem melhoria do custo e os tempos de 1000, 2000, 3000, 4000 e 5000 segundos para as instâncias de grafos com 50, 100, 250, 500 e 1000 vértices.

Instância			Melhores Resultados da Literatura					Resultados Obtidos pelo ILS				
N	*D*	*Id*	*Ref*	*Melhor*	*Média*	*Desvio Padrão*	*Tempo Médio (s)*	*Melhor**	*Média**	*Desvio Padrão**	*Tempo Médio 1 (s)**	*Tempo Médio 2 (s)**
50	5	1	GILS	**7,60**	**7,60**	*não há*	10,00	**7,60**	7,61	0,024	**0,40**	**0,64**
50	5	2	GILS	**7,61**	**7,62**	*não há*	10,00	**7,61**	**7,62**	0,014	**0,46**	**0,74**
50	5	3	GILS	**7,24**	**7,24**	*não há*	10,00	**7,24**	7,26	0,033	**0,45**	**0,72**
50	5	4	GILS	**6,59**	**6,59**	*não há*	10,00	**6,59**	**6,59**	0,000	**0,36**	**0,58**
50	5	5	GILS	**7,25**	**7,25**	*não há*	10,00	**7,25**	**7,25**	0,000	**0,42**	**0,68**
100	10	1	ACO	**7,76**	**7,77**	0,020	27,78	**7,76**	7,83	0,032	**4,10**	**5,90**
100	10	2	LEEA	**7,85**	**7,86**	0,020	734,65	**7,85**	7,89	0,031	**3,23**	**4,65**
100	10	3	VNS	**7,90**	**7,96**	0,040	38,66	**7,90**	**7,96**	0,042	**4,83**	**6,96**
100	10	4	LEEA	**7,98**	8,09	0,030	732,83	**7,98**	**8,02**	0,039	**3,61**	**5,20**
100	10	5	ACO	**8,16**	**8,17**	0,000	25,45	**8,16**	8,22	0,052	**3,70**	**5,33**
250	15	1	ACO	12,23	**12,28**	0,020	174,17	**12,18**	12,29	0,054	**81,04**	**116,70**
250	15	2	ACO	**12,02**	**12,04**	0,010	156,71	**12,02**	12,15	0,064	**63,61**	**91,60**
250	15	3	ACO	12,00	**12,02**	0,010	145,29	**11,98**	12,04	0,045	**56,84**	**81,85**
250	15	4	PEA-I	**12,42**	12,57	0,050	*não há*	**12,42**	**12,52**	0,056	72,28	116,37
250	15	5	ACO	12,23	**12,29**	0,040	211,11	**12,21**	12,31	0,059	**70,35**	**101,30**
500	20	1	ACO	16,78	**16,85**	0,030	906,17	**16,77**	16,90	0,082	**735,87**	1059,65
500	20	2	ACO	**16,63**	**16,70**	0,030	1012,91	**16,63**	16,77	0,065	**618,15**	**890,14**
500	20	3	ACO	16,79	**16,84**	0,030	1069,84	**16,78**	16,88	0,058	**731,69**	**1053,63**
500	20	4	ACO	16,80	**16,92**	0,040	1010,91	**16,78**	16,93	0,065	**628,13**	**904,51**
500	20	5	ACO	16,42	**16,46**	0,020	947,26	**16,41**	16,53	0,069	**736,13**	1060,03
1000	25	1	LEEA	**23,43**	**23,57**	0,080	**565,38**	23,63	23,73	0,063	4821,77	6943,35
1000	25	2	LEEA	23,46	23,67	0,080	**561,49**	**23,36**	**23,48**	0,079	4953,84	7133,53
1000	25	3	PEA-I	23,61	23,76	0,080	*não há*	**23,15**	**23,28**	0,069	4777,56	7691,87
1000	25	4	LEEA	23,79	23,96	0,090	**602,30**	**23,56**	**23,69**	0,070	4847,89	6980,96
1000	25	5	PEA-I	23,75	23,90	0,070	*não há*	**23,25**	**23,40**	0,060	4861,67	7827,29

Para comparar os resultados do ILS contidos na Tabela 4, o método exato descrito na Seção 2.1.1 foi implementado em CPLEX e executado para as cinco primeiras instâncias de grafos completos euclidianos da OR-Library com 50 vértices e diâmetro cinco. Para cada instância, o tempo de 24 horas foi utilizado como condição de parada. Os resultados obtidos encontram-se descritos na

Tabela 5. A melhor solução viável encontrada é indicada pela coluna *"Limite Superior*". Em todos os casos, não foi possível provar a otimalidade de nenhuma instância. Além disso, comparando os resultados obtidos com os da Tabela 4, o ILS proposto apresentou melhores soluções para as instâncias de 50 vértices com um tempo computacional bem menor, apesar dos limites superiores marcados com (*) terem sido os mesmos obtidos pela coluna "*Melhor**".

Tabela 5: resultados da execução do CPLEX.

Instância	*Limite Inferior*	*Limite Superior*
1	6,16	7,73
2	6,06	7,62
3	5,91	7,55
4	5,82	6,59 (*)
5	6,05	7,25 (*)

A Tabela 6 apresenta os resultados da execução do ILS utilizando como condição de parada (*condição de término*, no Algoritmo 7) somente os tempos de 1, 10, 160, 1200 e 2000 segundos para cada uma das 50 execuções das instâncias de 50, 100, 250, 500 e 1000 vértices, respectivamente. Conforme pode ser observado, o ILS proposto foi capaz de melhorar todos os resultados encontrados na literatura (compare as colunas "*Melhor"* e "*Melhor*"*), exceto para a primeira instância de 1000 vértices, demonstrando a competividade do método proposto. Em relação ao tempo computacional, excetuando-se as instâncias de 1000 vértices, o maior tempo necessário para que o ILS supere o resultado da literatura para uma instância foi de 1076,61 segundos, considerando-se a coluna *"Tempo Médio 1**", o que é razoável. Além disso, deve ser notado que, em cada linha da tabela, o tempo médio normalizado gasto (coluna "*Tempo Médio 2**") pelo ILS proposto para encontrar a melhor solução (coluna "*Melhor**") é, em 52% dos casos (13 de 25), menor que o tempo especificado pela literatura (coluna "*Tempo Médio"*). Em todos os casos (exceto para a primeira instância de 1000 vértices), o ILS encontrou melhores soluções com um tempo médio (coluna "*Tempo Médio*

*1**") menor que a condição de parada. Novamente, para a primeira instância de 1000 vértices, o ILS não conseguiu superar o resultado da literatura. Em relação à coluna "*Média**", o ILS proposto foi capaz de superar em 59% dos casos os resultados da literatura (14 de 25). Os valores para o desvio padrão estão dentro de um 0 a 0,090, conforme pode ser verificado pelas colunas "*Desvio Padrão*" e "*Desvio Padrão**", o que é razoável.

Tabela 6: Resultados computacionais da execução do ILS com os tempos de 1, 10, 160, 1200 e 2000 segundos para as instâncias de grafos com 50, 100, 250, 500 e 1000 vértices.

Instância			Melhores Resultados da Literatura					Resultados Obtidos pelo ILS				
N	*D*	*Id*	*Ref*	*Melhor*	*Média*	*Desvio Padrão*	*Tempo Médio (s)*	*Melhor**	*Média**	*Desvio Padrão**	*Tempo Médio 1 (s)**	*Tempo Médio 2 (s)**
50	5	1	GILS	**7,60**	**7,60**	*não há*	10,00	**7,60**	7,61	0,024	**0,14**	**0,23**
50	5	2	GILS	**7,61**	7,62	*não há*	10,00	**7,61**	**7,61**	0,007	**0,34**	**0,55**
50	5	3	GILS	**7,24**	**7,24**	*não há*	10,00	**7,24**	7,26	0,033	**0,45**	**0,72**
50	5	4	GILS	**6,59**	**6,59**	*não há*	10,00	**6,59**	**6,59**	0,000	**0,05**	**0,08**
50	5	5	GILS	**7,25**	**7,25**	*não há*	10,00	**7,25**	**7,25**	0,000	**0,27**	**0,43**
100	10	1	ACO	**7,76**	**7,77**	0,020	27,78	**7,76**	7,82	0,030	**1,39**	**2,00**
100	10	2	LEEA	**7,85**	**7,86**	0,020	734,65	**7,85**	7,88	0,024	**3,48**	**5,01**
100	10	3	VNS	**7,90**	7,96	0,040	38,66	**7,90**	**7,94**	0,037	**2,43**	**3,50**
100	10	4	LEEA	**7,98**	8,09	0,030	732,83	**7,98**	**8,00**	0,028	**0,66**	**0,95**
100	10	5	ACO	**8,16**	**8,17**	0,000	25,45	**8,16**	8,20	0,047	**3,08**	**4,44**
250	15	1	ACO	12,23	12,28	0,020	174,17	**12,18**	**12,26**	0,047	**88,44**	**127,35**
250	15	2	ACO	12,02	**12,04**	0,010	156,71	**12,01**	12,12	0,060	**148,69**	214,11
250	15	3	ACO	12,00	**12,02**	0,010	145,29	**11,97**	**12,02**	0,034	**137,27**	197,67
250	15	4	PEA-I	**12,42**	12,57	0,050	*não há*	**12,42**	**12,49**	0,045	159,68	257,08
250	15	5	ACO	12,23	**12,29**	0,040	211,11	**12,19**	**12,29**	0,053	**117,79**	**169,62**
500	20	1	ACO	16,78	**16,85**	0,030	**906,17**	**16,74**	16,88	0,077	1073,28	1545,52
500	20	2	ACO	16,63	**16,70**	0,030	**1012,91**	**16,62**	16,74	0,058	1163,48	1675,41
500	20	3	ACO	16,79	**16,84**	0,030	1069,84	**16,78**	16,86	0,046	**544,31**	**783,81**
500	20	4	ACO	16,80	16,92	0,040	1010,91	**16,75**	**16,90**	0,062	**914,04**	1316,22
500	20	5	ACO	16,42	**16,46**	0,020	**947,26**	**16,40**	16,51	0,066	1076,61	1550,32
1000	25	1	LEEA	**23,43**	**23,57**	0,080	**565,38**	23,72	23,83	0,069	1801,18	2593,70
1000	25	2	LEEA	23,46	23,67	0,080	**561,49**	**23,45**	**23,60**	0,090	1874,62	2699,45
1000	25	3	PEA-I	23,61	23,76	0,080	*não há*	**23,22**	**23,38**	0,077	1899,84	3058,74
1000	25	4	LEEA	23,79	23,96	0,090	**602,30**	**23,66**	**23,81**	0,071	1930,30	2779,63
1000	25	5	PEA-I	23,75	23,90	0,070	*não há*	**23,35**	**23,52**	0,073	1788,12	2878,87

Visando verificar se o aumento no tempo da condição de parada melhorava os resultados, foram testados os tempos de 1000, 2000, 3000, 4000 e 5000 segundos como condição de parada para as instâncias de 50, 100, 250, 500 e 1000 vértices, respectivamente, seguindo parâmetros de tempo similares àqueles utilizados pelo VNS (GRUBER; RAIDL, 2005b) e GILS (LUCENA; RIBEIRO; SANTOS, 2010). Analisando a Tabela 7, nota-se que os melhores resultados (coluna "*Melhor**") superaram os da literatura, exceto para a primeira instância de 1000 vértices. Nota-se também que as médias (coluna "*Média**") apresentaram resultados melhores que os da literatura, exceto em cinco instâncias das 25, correspondendo a 80% dos casos. Os valores de desvio padrão ficaram em um intervalo razoável, entre 0 e 0,076.

Tabela 7: Resultados computacionais da execução do ILS com os tempos de 1000, 2000, 3000, 4000 e 5000 segundos, para as instâncias de grafos com 50, 100, 250, 500 e 1000 vértices.

Instância			Melhores Resultados da Literatura					Resultados Obtidos pelo ILS				
N	*D*	*Id*	*Ref*	*Melhor*	*Média*	*Desvio Padrão*	*Tempo Médio (s)*	*Melhor**	*Média**	*Desvio Padrão**	*Tempo Médio 1 (s)**	*Tempo Médio 2 (s)**
50	5	1	GILS	**7,60**	**7,60**	*não há*	10,00	**7,60**	7,61	0,022	**0,14**	**0,23**
50	5	2	GILS	**7,61**	7,62	*não há*	10,00	**7,61**	**7,61**	0,000	**0,34**	**0,55**
50	5	3	GILS	**7,24**	**7,24**	*não há*	10,00	**7,24**	**7,24**	0,000	**0,45**	**0,72**
50	5	4	GILS	**6,59**	**6,59**	*não há*	10,00	**6,59**	**6,59**	0,000	**0,05**	**0,08**
50	5	5	GILS	**7,25**	**7,25**	*não há*	10,00	**7,25**	**7,25**	0,000	**0,27**	**0,43**
100	10	1	ACO	**7,76**	**7,77**	0,020	27,78	**7,76**	7,79	0,033	**1,39**	**2,00**
100	10	2	LEEA	**7,85**	7,86	0,020	734,65	**7,85**	**7,85**	0,016	**3,48**	**5,01**
100	10	3	VNS	**7,90**	7,96	0,040	38,66	**7,90**	**7,92**	0,028	**2,43**	**3,50**
100	10	4	LEEA	**7,98**	8,09	0,030	732,83	**7,98**	**7,98**	0,016	**0,66**	**0,95**
100	10	5	ACO	**8,16**	**8,17**	0,000	25,45	**8,16**	**8,17**	0,027	**3,08**	**4,44**
250	15	1	ACO	12,23	12,28	0,020	**174,17**	**12,16**	**12,21**	0,031	1547,08	2227,80
250	15	2	ACO	12,02	**12,04**	0,010	**156,71**	**11,99**	12,06	0,048	1140,80	1642,75
250	15	3	ACO	12,00	12,02	0,010	**145,29**	**11,95**	**11,99**	0,027	2520,55	3629,59
250	15	4	PEA-I	12,42	12,57	0,050	*não há*	**12,37**	**12,44**	0,038	1848,96	2976,83
250	15	5	ACO	12,23	12,29	0,040	**211,11**	**12,18**	**12,26**	0,054	2426,69	3494,43
500	20	1	ACO	16,78	**16,85**	0,030	**906,17**	**16,73**	**16,85**	0,072	3248,05	4677,19
500	20	2	ACO	16,63	**16,70**	0,030	**1012,91**	**16,58**	**16,70**	0,054	3860,47	5559,08
500	20	3	ACO	16,79	16,84	0,030	**1069,84**	**16,73**	**16,82**	0,046	3100,59	4464,85
500	20	4	ACO	16,80	16,92	0,040	**1010,91**	**16,75**	**16,86**	0,059	911,70	1312,85
500	20	5	ACO	16,42	**16,46**	0,020	**947,26**	**16,39**	16,48	0,061	3993,31	5750,37

1000	25	1	LEEA	**23,43**	**23,57**	0,080	**565,38**	23,63	23,73	0,062	4943,70	7118,93
1000	25	2	LEEA	23,46	23,67	0,080	**561,49**	**23,36**	**23,48**	0,076	4658,96	6708,90
1000	25	3	PEA-I	23,61	23,76	0,080	*não há*	**23,15**	**23,27**	0,066	4486,63	7223,47
1000	25	4	LEEA	23,79	23,96	0,090	**602,30**	**23,56**	**23,69**	0,068	4538,38	6535,27
1000	25	5	PEA-I	23,75	23,90	0,070	*não há*	**23,25**	**23,39**	0,055	3527,48	5679,24

A Tabela 8 apresenta os resultados da aplicação do teste de Wilcoxon, utilizando a plataforma R, para a análise da significância estatística dos grupos de instâncias com 50, 100, 250, 500 e 1000 vértices. Em todos os casos, o teste de normalidade falhou. As colunas "*#V*", "*#D*" e "*#E*" indicam, respectivamente, o número de vezes que os resultados do ILS superaram os da literatura, o número de vezes que os resultados do ILS foram piores que os da literatura e o número de vezes que os resultados do ILS foram iguais ao da literatura. Os resultados marcados com "–" indicam que o teste não pôde ser aplicado para o grupo de instâncias, enquanto "N" indica a ausência de significância estatística e "S" a presença, sendo que neste caso o valor de *p-value* se encontra entre parênteses. Em todos os casos, o grau de confiança é de 95%.

Tabela 8: Resultados de significância estatística.

	50			100			250			500			1000		
	#V	*#D*	*#E*	*#V*	*#D*	*#E*	*#V*	*#D*	*#E*	*#V*	*#D*	*#E*	*#V*	*#D*	*#E*
Média	1	1	3	3	1	1	4	1	0	2	1	2	4	1	0
Sign. Estat.?	–			N			N			N			N		
Melhor	0	0	5	0	0	5	5	0	0	5	0	0	4	1	0
Sign. Estat.?	–			–			S (0,02724)			S (0,02724)			N		

CAPÍTULO 6 – CONCLUSÃO

Uma heurística baseada no ILS foi proposta para resolver o EBDMSTP. Cinco novas vizinhanças foram propostas para a busca local, que é uma variação do método *Variable Neighborhood Descent* (VND), chamado de Random VND, e quatro vizinhanças para a perturbação. Os testes realizados demonstraram que estas vizinhanças permitiram que o ILS proposto obtivesse melhores soluções que aquelas apresentadas na literatura, utilizando 25 instâncias de grafos da OR-Libray, com 50, 100, 250, 500 e 1000 vértices, excetuando-se um único caso (uma instância de 1000 vértices).

Baseados nos resultados obtidos, pode-se indicar que as novas vizinhanças propostas (*Hierarchy Exchange*, *Hierarchy Rotation*, *Leaf Swap*, *Father Swap* e *Level Change*), em conjunto com as modificações das vizinhanças apresentadas na literatura (*Edge Exchange*, *Node Swap*, *Subtree Optimize*, *Edge Delete* e *Center Change*), foram cruciais para o desempenho do ILS, melhorando significativamente os resultados apresentados pela literatura, tornando-o competitivo em relação aos demais métodos apresentados.

Como sugestão de trabalhos futuros, novas vizinhanças poderiam ser propostas e implementadas, em conjunto com as apresentadas, visando reduzir o custo e o tempo necessário para a obtenção de uma EBDMST com o menor custo que as já obtidas.

REFERÊNCIAS

ABDALLA, A.; DEO, N.; GUPTA, P. Random-tree diameter and the diameter-constrained MST. **Congressus Numerantium**, 144, 161–182, 2000.

ACHUTHAN, N. R.; CACCETTA, L.; CACCETTA, P.; GEELEN, J. F. Computational methods for the diameter restricted minimum weight spanning tree problem. **Australasian Journal of Combinatorics**, 10:51–71, 1994.

AKBARI TORKESTANI, J. An adaptive heuristic to the bounded-diameter minimum spanning tree problem. **Soft Computing**, 16:1977–1988, 2012.

BALA, K.; PETROPOULOS, K.; STERN, T. E. Multicasting in a linear lightwave network. **IEEE INFOCOM '93 The Conference on Computer Communications, Proceedings**, 1350–1358, San Francisco, CA, USA, 1993.

BEAN, J. C. Genetic algorithms and random keys for sequencing and optimization. **ORSA Journal on Computing**, 6:154-160, 1994.

BINH, H. T. T.; HAOI, N. X.; MCKAY, R. I. New heuristic and hybrid genetic algorithm for solving the bounded diameter minimum spanning tree problem. **Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation - GECCO '09**, 373-380, Hong Kong, China, 2009.

BOOKSTEIN, A.; KLEIN, S. T. Compression of correlated bit-vectors. **Information Systems**, 16:387–400, 1991.

CHIARANDINI, M.; STÜTZLE, T. An application of iterated local search to graph coloring problem. **Proceedings of the Computational Symposium on Graph Coloring and its Generalizations**, 1–13, 2002.

DA SILVA, D. M.; DE MENEZES FROTA, Y. A.; SUBRAMANIAN, A. Uma heurística para o problema de roteamento de veículos com múltiplas viagens. **Congresso Lationo-Iberoamericana de Investigación Operativa**, 1880–1891, Rio de Janeiro, RJ, Brasil, 2012.

DAVIS, L. **Handbook of Genetic Algorithms**. Van Nostrand Reinhold, New York, 1991.

DONG, X.; HUANG, H.; CHEN, P. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. **Computers & Operations Research**, 36:1664–1669, 2009.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: a Guide to the Theory of NP-Completeness**. San Francisco: WH Freeman & Co., 1979.

GLOVER, F.; KOCHENBERGER, G. A. **Handbook of Metaheuristics**. New York: KLUWER ACADEMIC PUBLISHERS, 2003.

GOLDBERG, D. E.; LINGLE, J. R. Alleles, loci, and the traveling salesman problem. In J. J. Greffenstette, editor, **Proceedings of the First International Conference on Genetic Algorithms**, 154–159, Hillsdale, NJ, USA, 1985.

GOUVEIA, L.; MAGNANTI, T. L. Network flow models for designing diameter-constrained minimum-spanning and Steiner trees. **Networks**, 41:159–173, 2003.

GOUVEIA, L.; SIMONETTI, L.; UCHOA, E. Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered graphs. **Mathematical Programming**, 128:123–148, 2011.

GRUBER, M.; RAIDL, G. R. A new 0–1 ILP approach for the bounded diameter minimum spanning tree problem. **Proceedings of the 2nd International Network Optimization Conference**, 178–185, 2005a.

GRUBER, M.; RAIDL, G. R. Variable neighborhood search for the bounded diameter minimum spanning tree problem. **Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search**, 1–12, Seattle, Washington, USA, 2005b.

GRUBER, M.; RAIDL, G. R. (Meta-) Heuristic Separation of Jump Cuts in a Branch & Cut Approach for the Bounded Diameter Minimum Spanning Tree Problem. **Work**, v. 8, n. September, p. 209–229, 2008.

GRUBER, M.; VAN HEMERT, J.; RAIDL, G. R. Neighbourhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA, and ACO. **Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06**, Seattle, Washington, USA, 2006.

HANDLER, G. Y. Minimax location of a facility in an undirected tree graph. **Transportation Science**, 7:287–293, 1973.

HANSEN, P.; MLADENOVIĆ, N. Variable neighborhood search: Principles and applications. **European Journal of Operational Research**, 130:449–467, 2001.

HANSEN, P.; MLADENOVIĆ, N. Variable neighborhood search. In: BURKE, E. K.; KENDALL, G. (Eds.). **SEARCH METHODOLOGIES**. Boston, MA: Springer US, 2005. 211–238.

JULSTROM, B. A. Encoding bounded-diameter spanning trees with permutations and with random keys. **Genetic and Evolutionary Computation Conference's Workshops Proceedings, Workshop on Analysis and Design of Representations**, 1272–1281, Springer, Berlin, Heidelberg, 2004.

JULSTROM, B. A. Greedy heuristics for the bounded diameter minimum spanning tree problem. **Journal of Experimental Algorithmics**, 14:1, 2009.

JULSTROM, B. A.; RAIDL, G. R. A permutation-coded evolutionary algorithm for the bounded-diameter minimum spanning tree problem. **Genetic and Evolutionary Computation Conference's Workshops Proceedings, Workshop on Analysis and Design of Representations**, 2–7, 2003.

KARGER, D. R.; KLEIN, P. N.; TARJAN, R. E. A Randomized Linear-Time Algorithm to Find Minimum Spanning Trees. **Journal of the ACM (JACM),** v. 42, n. 2, p. 321–328, 1995.

KRUSKAL, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. **Proceedings of the American Mathematical Society**, 7:48-50, Rhode Island, USA, 1956.

LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated local search. **Handbook of Metaheuristics**. Kluwer Academic Publishers, 320–353, 2003.

LUCENA, A.; RIBEIRO, C. C.; SANTOS, A. C. A hybrid heuristic for the diameter constrained minimum spanning tree problem. **Journal of Global Optimization**, 46:363–

381, 2010.

MARTINS, I. C.; PINHEIRO, R. G. S.; PROTTI, F.; OCHI, L. S. A hybrid iterated local search and variable neighborhood descent heuristic applied to the cell formation problem. **Expert Systems With Applications**, 42: 8947-8955 (2015).

MLADENOVIĆ, N.; HANSEN, P. Variable neighborhood search. **Computers & Operations Research**, 24:1097–1100, 1997.

NEÄ, J.; MILKOVÃ, E.; NEÄ, H. Otakar Bor uvka on minimum spanning tree problem Translation of both the 1926 papers , comments , history. **Discrete Mathematics**, v. 233, p. 3–36, 2001.

PENNA, P. H. V.; SUBRAMANIAN, A.; OCHI, L. S. An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. **Journal of Heuristics**, 19:201–232, 2013.

PRIM, R. C. Shortest connection networks and some generalizations. **Bell System Technical Journal**, 36:1389-1401, 1957.

RAIDL, G. R.; JULSTROM, B. A. Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. **Proceedings of the 2003 ACM symposium on Applied computing - SAC '03**, 747-752, Melbourne, Florida, USA, 2003a.

RAIDL, G. R.; JULSTROM, B. A. Edge sets: An effective evolutionary coding of spanning trees. **IEEE Transactions on Evolutionary Computation**, 7:225–239, 2003b.

RAYMOND, K. A tree-based algorithm for distributed mutual exclusion. **ACM Transactions on Computer Systems**, 7:61–77, 1989.

SANTOS, A. C.; LUCENA, A.; RIBEIRO, C. C. Solving diameter constrained minimum spanning tree problems in dense graphs. **Proceedings of the International Workshop on Experimental Algorithms**, 3059:458–467, Springer, Berlin, Heidelberg, 2004.

SINGH, A.; GUPTA, A. K. Improved heuristics for the bounded-diameter minimum spanning tree problem. **Soft Computing**, 11:911–921, 2007.

APÊNDICE – ARTIGO SUBMETIDO

Iterated Local Search for the Euclidean Bounded-Diameter Minimum Spanning Tree Problem

Rogério M. Nepomuceno[18] Fábio Protti[19] Isabel C. M. Rosseti[20]

ABSTRACT

Abstract. This paper presents a proposal for an Iterated Local Search approach for the Euclidean Bounded-Diameter Minimum Spanning Tree Problem (EBDMSTP), which appears in industry applications. In the EBDMSTP the vertices of the spanning tree must be connected with a minimum cost but within a limit of connections in a path. Five new neighborhoods are proposed for the local search, which is a variation of the Variable Neighborhood Descent method, and four for the perturbation. The tests performed showed that these neighborhoods allowed the proposed Iterated Local Search approach to obtain better results than those presented in the literature, using OR-Library graph instances.

Keywords: Spanning Trees, Bounded-Diameter Minimum Spanning Tree, Iterated Local Search, ILS, Heuristics.

[18] R. M. Nepomuceno
Federal Institute of Education, Science and Technology of the Triângulo Mineiro, Av. Doutor Florestan Fernandes, 131, Univerdecidade, Uberaba, MG, 38064-190, Brazil
e-mail: rogerionepomuceno@iftm.edu.br

[19] F. Protti
Institute of Computing, Federal Fluminense University, Rua Passo da Pátria 156, Niterói, RJ, 24210-240, Brazil
e-mail: fabio@ic.uff.br

[20] I. C. M. Rosseti
Institute of Computing, Federal Fluminense University, Rua Passo da Pátria 156, Niterói, RJ, 24210-240, Brazil
e-mail: rosseti@ic.uff.br

1 INTRODUCTION

Let $G = (V, E)$ be an undirected graph, with $n = |V|$ vertices, $m = |E|$ edges, and a real positive value w_{uv} associated to each edge $(u, v) \in E$ that indicates the cost from vertex u to v. For an integer D, $2 \leq D \leq n - 1$, a bounded-diameter spanning tree is a spanning tree $T = (V, E_T)$ of G, $E_T \subseteq E$, whose diameter is not greater than D. The total cost $c(T)$ of the spanning tree T is the sum of the costs of its edges. The Bounded-Diameter Minimum Spanning Tree Problem (BDMSTP) consists of finding the spanning tree of G that minimizes $c(T)$, as long as the diameter is not greater than the value D. For the cases $D = 2$, 3, and $n - 1$, or when all edges have the same cost, the problem can be solved in polynomial time (ABDALLA; DEO; GUPTA, 2000). For $4 \leq D < n - 1$ the problem is NP-Hard (GAREY; JOHNSON, 1979). Applications of the BDMSTP emerge in engineering problems, such telecommunication network and fiber optic projects (BALA; PETROPOULOS; STERN, 1993), data compression problems (BOOKSTEIN; KLEIN, 1991), and in distributed systems design, when multiple exclusion is considered (RAYMOND, 1989). In practice, only small instances of the problem, with a maximum of 161 vertices and diameter less than or equal to 8, can be solved by exact approaches in an acceptable time. Because of this fact, constructive heuristics and metaheuristics are interesting options to provide good solutions for the BDMSTP.

In the Euclidean Bounded-Diameter Minimum Spanning Tree Problem (EBDMSTP), vertices of the input graph are associated with points in the Cartesian plane, and each cost w_{uv} is precisely the distance between vertices u and v; in this case, we assume that all the possible distances are given, i.e., the input graph is complete ("Euclidean instance"). This paper presents a proposal for an Iterated Local Search approach for the EBDMSTP. Five new neighborhoods are proposed for the local search, which is a variation of the Variable Neighborhood Descent method ("Random VND", or simply RVND), and four for the perturbation. The tests performed showed that these neighborhoods allowed the proposed Iterated Local Search approach to obtain better results than those presented in the literature, using OR-Library graph instances. To the best of the authors' knowledge, this is the first time an ILS+RVND approach is proposed for the solution of the EBDMSTP.

This paper is organized as follows: Sections 2, 3, and 4 present exact methods, constructive heuristics, and metaheuristics described in the literature for the BDMSTP and the EBDMSTP. Section 5 describes the Iterated Local Search metaheuristic (ILS), while Section 6 presents an ILS for the solution of the EBDMSTP, analyzing the neighborhoods to be used

for local search and perturbation. Section 7 presents its implementation. Section 8 presents the computational results, and Section 9 our conclusions.

2 EXACT APPROACHES FOR SOLVING THE BDMSTP

Several formulations for the BDMSTP are described in the literature. Achuthan *et al*. [6] present a Mixed Integer Linear Programming (MILP) formulation, as well as some procedures based on branch-and-bound, and analyze their application in instances of complete graphs with up to 50 vertices and diameter less than or equal to four. Gouveia and Magnati [7] present a model based on network flows, which solves instances with up to 60 vertices and 600 edges, and diameter less than or equal to 8. Santos, Lucena, and Ribeiro [8] propose an improvement in the method presented in [6], being able to solve instances of complete graphs with up to 25 vertices and diameter less than or equal to 10, and instances with 60 vertices, 150 edges and diameter equal to five. Grüber and Raidl [9] present a 0-1 Integer Linear Programming model and can solve the problem for instances with 40 vertices, 100 edges and diameter less than or equal to nine with a time much shorter than that proposed by [8]. Grüber and Raidl [10] present an entire linear programming formulation based on a branch-and-cut algorithm, which uses constructive heuristics, local search and Tabu search to aid in the BDMSTP solution, being able to solve instances with 60 vertices and 600 edges with diameter eight. Gouveia, Simonetti, and Uchoa [11] introduce a branch-and-cut algorithm capable of solving complete graph instances with up to 161 vertices and diameter less than or equal to 8.

3 CONSTRUCTIVE HEURISTICS FOR THE EBDMSTP

Many algorithms that use constructive heuristics are described in the literature for the EBDMSTP. The most renowned is the *One-Time Tree Construction* (OTTC) (ABDALLA; DEO; GUPTA, 2000), which is based on Prim's algorithm (PRIM, 1957). Raidl and Julstrom (2003a) present the *Random Greedy Heuristic* (RGH), which produces lower cost solutions for the EBDMSTP when compared with OTTC. Julstrom (2009) presents the *Center-Based Tree Construction* heuristic (CBTC), which builds solutions for the EBDMSTP at lower costs than OTTC, especially when large diameters are considered. Still according to Julstrom (2009), in order to obtain solutions for the EBDMSTP, OTTC and CBTC heuristics are "extremely greedy" when applied to Euclidean distances of small diameters. Thus, the author

presents a *Randomized Center-Based Tree Construction* heuristic (RTC), which yields lower cost trees, when considering small diameters, if compared with OTTC and CBTC. In the case of larger diameters, CBTC presents lower cost trees than OTTC and RTC. The *Center-Based Recursive Clustering* (CBRC), presented in (BINH et al., 2009), produces lower cost trees when compared with OTTC, RGH, and CBTC.

4 METAHEURISTICS FOR THE EBDMSTP

Metaheuristics to solve the EBDMSTP are also described in the literature. Raidl and Julstrom (2003a) propose an evolutionary algorithm called *Edge-Set-Code Evolutionary Algorithm* (ESCEA), which presents better results than those presented by OTTC and RGH. Julstrom and Raidl (JULSTROM; RAIDL, 2003) propose another evolutionary algorithm called *Permutation-Coded Evolutionary Algorithm* (PCEA), which obtained better results than the ESCEA. Julstrom (2004) presented the *Random Key Evolutionary Algorithm* (RKEA), which was compared with PCEA, presenting similar solutions. Gruber and Raidl (2005b) present a heuristic based on *Variable Neighborhood Search* (VNS), using a *Variable Neighborhood Descent* (VND) method as local search (GLOVER; KOCHENBERGER, 2003; HANSEN; MLADENOVIĆ, 2001, 2005; MLADENOVIĆ; HANSEN, 1997), and better results than those obtained by ESCEA, PCEA, and RKEA were obtained. Gruber, Hermert, and Raidl (2006) present the *Level Encoded Evolutionary Algorithm* (LEEA) and an Ant Colony Optimization (ACO) approach. In tests performed with non-prefixed time, ACO showed the best results, when compared with VNS and LEEA. On the other hand, in tests with prefixed time, LEEA showed the best results, overcoming ACO and VNS. Singh and Gupta (2007) presented a PCEA improvement, called PEA-I, that obtains better results than ESCEA and PCEA. A hybrid algorithm, combined with the principles of the *Greedy Randomized Adaptive Search* (GRASP) and ILS (LUCENA; RIBEIRO; SANTOS, 2010), here called GILS, is proposed in (LUCENA; RIBEIRO; SANTOS, 2010), obtaining the best results in 50% of the cases when compared with the best results presented by VNS (GRUBER; RAIDL, 2005b), ESCEA (RAIDL; JULSTROM, 2003a), and PCEA (RAIDL; JULSTROM, 2003b).

5 ITERATED LOCAL SEARCH

Iterated Local Search (ILS) is a metaheuristic widely employed to solve combinatorial optimization problems, and is based on the idea that a locally optimal solution obtained in the local search procedure can be improved through the application of a perturbation procedure, aiming at exploring the space of solutions and seeking a global optimum. Therefore, this is a method of local search that aims at refining the search in a small solution space subgroup that is locally optimal (LOURENÇO; MARTIN; STÜTZLE, 2003).

ILS has been successfully applied to optimization problems in the academic and industrial areas (LOURENÇO; MARTIN; STÜTZLE, 2003). Some examples are heterogeneous vehicle routing (PENNA; SUBRAMANIAN; OCHI, 2013), multiple-travel vehicle routing (DA SILVA; DE MENEZES FROTA; SUBRAMANIAN, 2012), manufacturing cell formation (MARTINS et al., 2015), graph coloring (CHIARANDINI; STÜTZLE, 2002), scheduling problems (DONG; HUANG; CHEN, 2009), to name just a few.

ILS is composed by the construction of an initial solution (*GenerateInitialSolution*), a local search procedure (*LocalSearch*), a perturbation procedure (*Perturbation*), and an acceptance criterion (*AcceptanceCriterion*), according to Algoritmo 5. *Perturbation*, *LocalSearch* and *AcceptanceCriterion* procedures are iteratively carried out until a terminal condition is obtained.

Algorithm 10: ILS Structure

```
procedure ILS
   s0 = GenerateInitialSolution()
   s* = LocalSearch(s0)
   repeat
      s' = Pertubation(s*, history)
      s*' = LocalSearch (s')
      s* = AcceptanceCriterion(s*, s*', history)
   until termination condition is met
end
```

Procedure *GenerateInitialSolution* enables the determination of an initial solution s_0 in the solution space, which will be used as a starting point for the local search. Such procedure can be built, for example, using any random method or any greedy heuristic. However, the better the initial solution, the faster will be the convergence to a good solution.

Procedure *LocalSearch* explores the space of solutions seeking a local optimum. Basically, it consists of improving an initial solution *s* through the exploration of neighboring solutions to *s*. Thus, if a neighboring solution *s*' is better than *s*, then *s* is replaced by *s*' and the search is restarted from *s*. This process is repeated until no improvement is possible, i.e., when *s* is a local optimum.

An important point regarding the local search is that it can be stagnant in a local optimum. Perturbation allows ILS to escape from a local optimum, enabling different regions of the solution space to be explored. Besides, the perturbation must be strong enough to move from a local optimum, but not too heavily, otherwise it will behave like a random restart. On the other hand, it cannot be very weak, otherwise the local search can undo it, taking to a local optimum that has already been visited (LOURENÇO; MARTIN; STÜTZLE, 2003). In order to better explore the solution space, a history of previously visited solutions can be maintained. In the procedure *Perturbation* presented in Algoritmo 5, *s** represents the current solution and *s*' is the new solution generated.

The *AcceptanceCriterion* procedure verifies whether the current solution *s** will be replaced or not by a new solution *s**', corresponding to a new optimal place generated by *LocalSearch*, based or not on a history of past computations. That is, the new solution becomes the current solution as long as it brings an improvement.

6 AN ILS FOR THE EBDMSTP

In order to solve the EBDMSTP, we propose an ILS implemented using a VND variant (HANSEN; MLADENOVIĆ, 2005) as local search, called *Random Variable Neighborhood Descent* – RVND (MARTINS et al., 2015), which randomly explores the neighborhood structure of the current solution, by only accepting solutions that improve such structure, and returns the best neighbor, the one that corresponds to the best solution found.

Algorithm 11: Local Search Structure

```
procedure LocalSearch(A*)
  random initialize V = {V^1, ..., V^r}
  k = 1
  repeat
    A' = V^k(A*)
    if value(A') < value(A*) then
      A* = A'
      k = 1
    else
      k = k + 1
    end if
  until k = k_max
  return A*
end
```

Algorithm 11 presents the *LocalSearch* procedure, where $A*$ is the current solution of the EBDMSTP, *value*(A) stands for the cost of solution A, and $V = \{V^1, ..., V^r\}$ corresponds to the collection of neighborhood structures, which are randomly ordered. Basically, the algorithm goes through the neighborhood structure V^k of the current solution $A*$, searching for a cheapest solution A' for the EBDMSTP. In this case, A' becomes the current tree and the process is restarted ($k = 1$). Otherwise a new neighborhood is visited ($k = k + 1$). The process repeats until no improvement is possible, that is, when $k = k_{max}$.

6.1 NEIGHBORHOODS

The neighboring structures used in the implementation of the local search were *Edge Exchange* [18], *Node Swap* (GRUBER; RAIDL, 2005b), *Subtree Optimize* (RAIDL; JULSTROM, 2003a), *Hierarchy Exchange*, *Hierarchy Rotation*, *Leaf Swap*, *Father Swap*, and *Level Change*.

In all cases, an n-node tree rooted at one center represents a feasible solution of the EBDMSTP, if the diameter is even, or at two centers, if the diameter is odd. The remaining nodes cannot be at a depth greater than $\lfloor D/2 \rfloor$ (HANDLER, 1973). Besides, if the diameter is odd, an edge that connects the two central nodes exists. In all cases, the tree that represents a feasible solution of the EBDMSTP is oriented from the center (or centers) towards the leaves, forming a relation of predecessors and successors. The center (or centers) have depth zero, while the leaves have depth at most $\lfloor D/2 \rfloor$. From this point on, we sometimes refer to a feasible solution as a BDMST.

6.1.1 EDGE EXCHANGE

This neighborhood is in fact a slight variant of that presented in [18]. Let T be a feasible solution of the EBDMSTP. The *Edge Exchange* neighborhood consists of exploring all possible trees obtained from disconnecting a subtree of T (by removing an edge) and connecting it into another position of the tree, provided that the cost is reduced and the diameter not violated. All possible subtrees are examined, aiming at determining the connection of least cost.

Consider the tree of Fig. 1(a). The disconnection of edge (b, c) produces two subtrees T_p

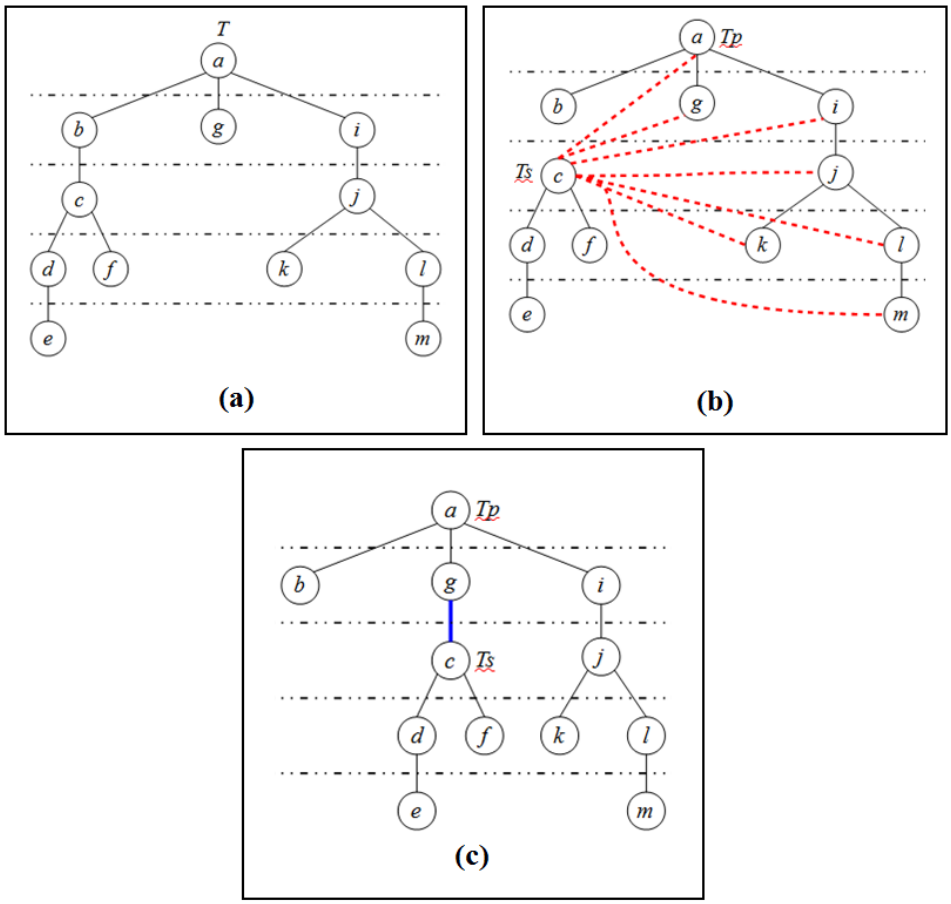
(primary tree) and T_s (secondary tree), as seen in Fig. 1(b). Subsequently, the algorithm verifies whether creating an edge linking the root node c of T_s with a node of T_p reduces the cost of T. The edge that produces the least cost among all the possibilities is chosen, in this case edge (g, c), forming the new tree presented in Fig. 1(c).

The time to examine all the edges of T is O(n). The time to examine all possible replacements in order to identify the least cost is also O(n), which results in an O(n^2) total time for this neighborhood.

6.1.2 NODE SWAP

This neighborhood is also a slight variant of that presented in [18]. It explores the relation between a tree node, including its center (or one of the centers, if the diameter is odd),

Fig. 1: Edge Exchange example.



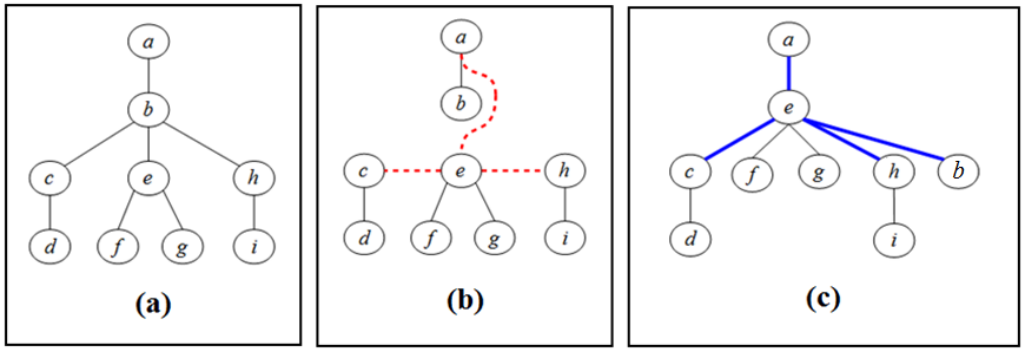
and its direct successors. A solution for this neighborhood is a tree resulting from replacing a node v by one of its successors, say, node u. In such case, the resulting tree of this operation is

obtained by making v, and its successors other than u, successors of u, and the predecessor of v is made the predecessor of u. If v is a center, then u will not have a predecessor.

Let be T the tree illustrated in Figura 4(a). Assume $v = b$ and $u = e$. According to Figura 4(b), the operation makes node a the direct predecessor of e, while b and all of its successors are made successors of e, as illustrated by Figura 4(c).

The time to verify all pairs (v, u) of the tree T is $O(n)$. The time to verify all successors of a node v is $d(v)$, where $d(v)$ is the degree of v. Thus, the total time spent by this neighborhood is $O(n\Delta)$, where Δ is the maximum degree of a node.

Fig. 2: Node Swap example.



6.1.3 SUBTREE OPTIMIZE

Subtree Optimize (RAIDL; JULSTROM, 2003a) explores the relation between sibling nodes with depth $\lfloor D/2 \rfloor$ and its direct predecessor v, whose depth is $\lfloor D/2 \rfloor - 1$. Basically, this neighborhood rearranges the subtree rooted at v in a suitable way. Let S be the set of vertices in this subtree, and p the direct predecessor of v. The operation tries, for each u in $S - \{v\}$, to set u as the subtree root, connecting it to p. Also, v takes the place occupied by u before the change. The cost of the new subtree with root u is easily given by the following expression:

$$w_S(u) = w_{pu} + \sum_{x \in S-\{u\}} w_{ux}$$

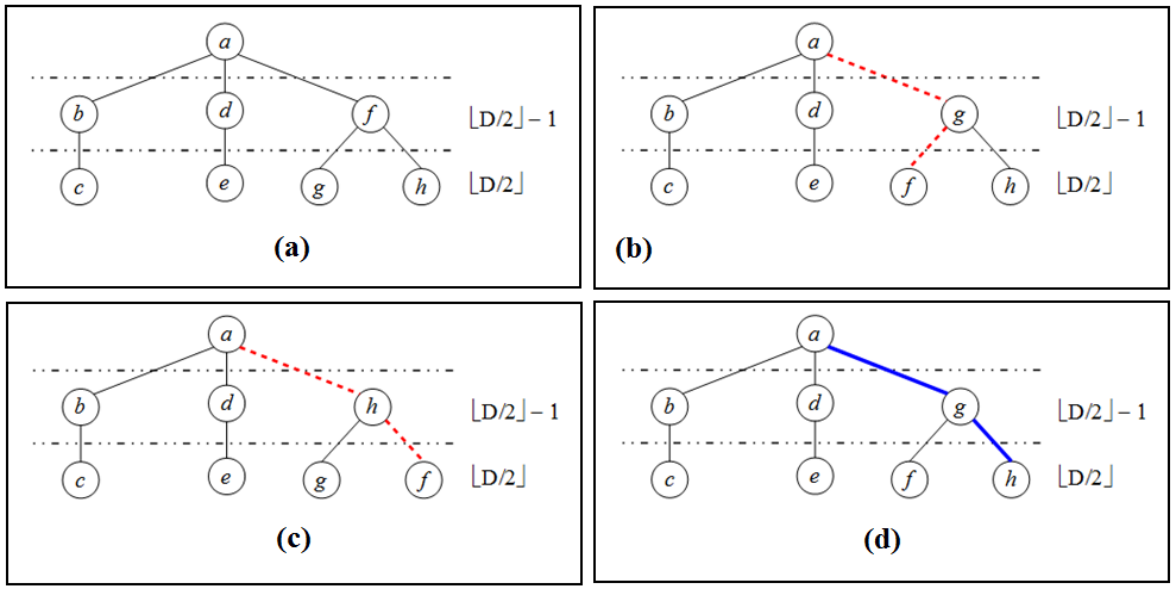
The subtree that minimizes this cost is chosen. If no cost improvement is achieved then no local rearrangement is done. Observe that this operation does not violate the diameter.

Let T be the tree indicated in Fig. 3(a). The operation selects node f, whose depth is

$\lfloor D/2 \rfloor - 1$. Next, each successor of f (in this case g or h) is placed at the root of the new subtree and the cost is verified, as shown in Figures 3(b) and 3(c). Fig. 4(d) shows the resulting tree, where the commutation of nodes g and f produces a resulting tree of lower cost.

For a fixed node v, determining the cost of a subtree configuration requires $O(|S|)$ time, where $|S|=d(v)+1$. In addition, there are $O(d(v))$ possible configurations to be examined. Thus, the complexity of this operation is $O(d(v)^2)$.

Fig. 3: Subtree Optimize example.



6.1.4 HIERARCHY EXCHANGE

This neighborhood explores the gradual inversion in the hierarchical order of a node v and its successors. The operation is applied to all nodes of the BDMST until finding a resulting tree that minimizes the cost of the whole operation. It is described as follows.

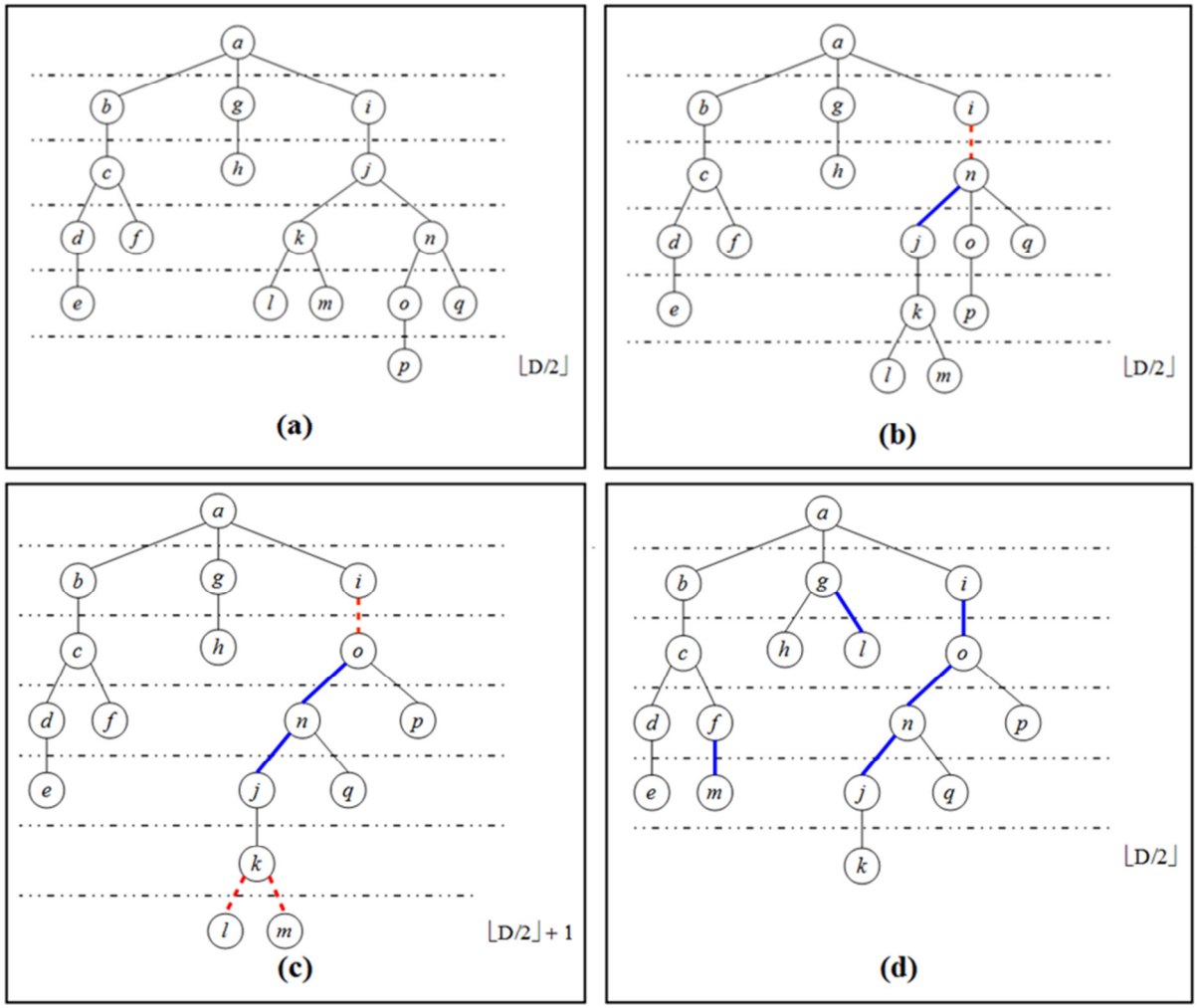
Let T be a BDMST, and let p, v, u_1, u_2, …, u_{k-1}, u_k be nodes of T such that v is the root of a subtree, p the direct predecessor of v, u_1 the direct successor of v, u_2 the direct successor of u_1, and so on. Assume that the cost of edge (p, u_k) is lower than the cost of edge (p, v). Thus, the operation makes u_k direct successor of p, u_{k-1} direct successor of u_k, u_{k-2} direct successor of u_{k-1}, and so on until v (which is made the direct successor of u_1). If after this process some node violates the diameter then it will be disconnected from the tree and reconnected to another position using an edge of lowest possible cost.

Figura 6(a) illustrates the BDMST T. Nodes j and n are chosen to start the operation.

Next, *j* becomes the direct successor of *n*, and *n* becomes the direct successor of *i*, as shown in Figura 6(b). Suppose that this operation does not reduce the cost of the tree; then node *o*, which is a direct successor of *n*, is chosen and made the direct successor of *i*, while *n* becomes the direct successor of *o*, as illustrated in Figura 6(c). At this point, suppose that the cost of the tree is lower; thus nodes that violate the diameter, in this case *l* and *m*, must be reconnected to other positions using edges of lowest possible cost, which are (*g*, *l*) and (*f*, *m*), respectively, as illustrated in Figura 6(d).

Each operation of hierarchy inversion between predecessors and successors clearly takes O(*n*) time. In the end, if the diameter of a node is violated, the time to reconnect it to another place of the tree is O(*n*); in the worst case, O(*n*) nodes need to be reconnected, yielding an O(n^2) time complexity for inversion plus redistribution of nodes. Since there are O(*n*) nodes that can be the starting point of an operation of hierarchy inversion, the overall complexity of this neighborhood is O(n^3).

Fig. 4: Hierarchy Exchange example.



6.1.5 HIERARCHY ROTATION

This neighborhood is similar to the previous one, but explores the gradual inversion in the hierarchy order of a node v node and its *predecessors*, and the distribution of the resulting subtree. Again, the operation is applied to all nodes of the BDMST until it finds a resulting tree that minimizes the cost of the operation. It is described below.

Let T be a BDMST, and let c, v, u_1, u_2, u_3, …, u_{k-1}, u_k be nodes of T, where v is a subtree root, u_1 is the direct predecessor of v, u_2 is the direct predecessor of u_1, and so on until u_k, which is the direct predecessor of u_{k-1} and direct successor of the center c (or one of the centers, if the diameter is odd). The operation begins by making u_1 a direct successor of v and removing edge (u_1, u_2). The obtained subtree, whose root is v, is reconnected to another position of the tree, provided that the cost is decreased and the diameter is not violated. If the cost of the resulting BDMST does not decrease, then edge (u_3, u_2) is removed and u_2 becomes the direct successor of u_1, using the edge (u_1, u_2). Again, the resulting subtree, whose root is v, is reconnected to another position of the tree, provided that the cost decreases and the diameter is not violated. This process is repeated repeat until node u_k is analyzed; then edge (c, u_k) is removed and u_k is made the direct successor of u_{k-1}, using the edge (u_{k-1}, u_k).

Fig. 5: Hierarchy Rotation example.

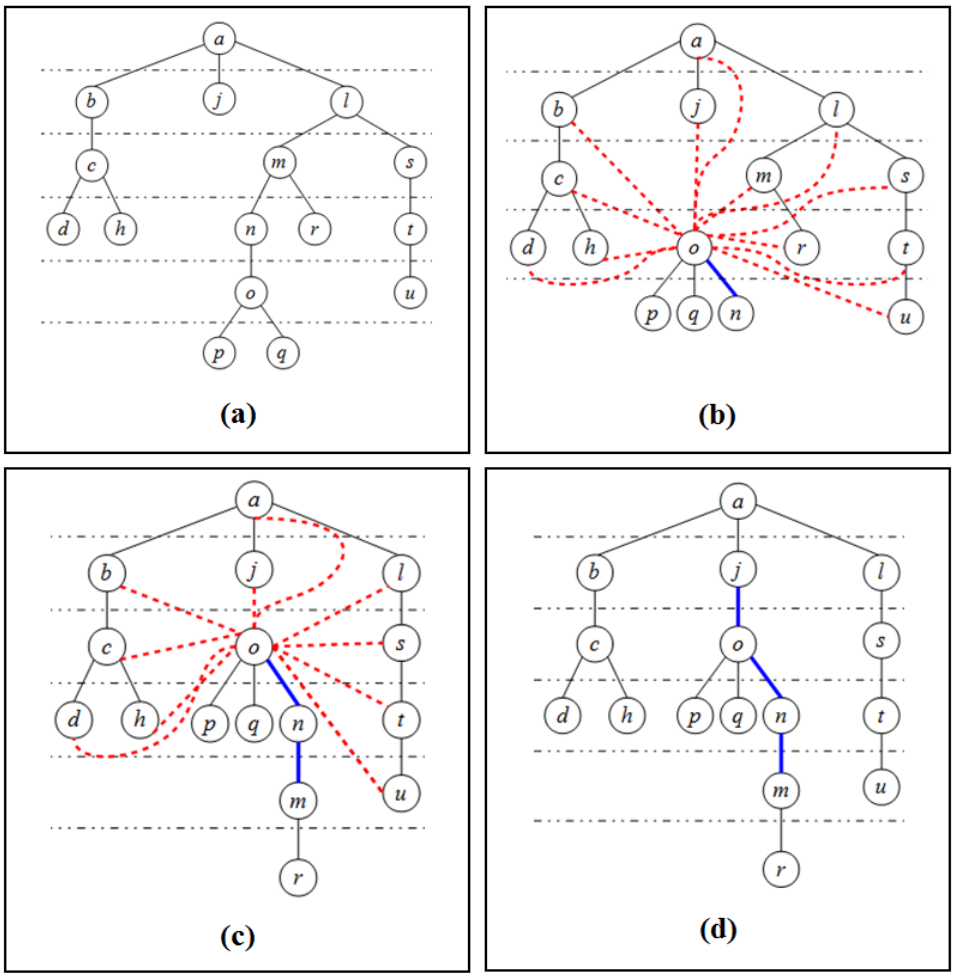


Figura 7(a) shows a BDMST *T*. First, node *o* is chosen to start the operation; node *n* becomes a direct successor of *o* and edge (*m*, *n*) is removed. Next, the operation checks whether connecting the subtree rooted at *o* to other place of the BDMST decreases the cost, as Figura 7(b) shows. If the cost of the BDMST does not decrease, then the operation is continued and node *m*, former predecessor of *n*, becomes the direct successor of *n*, as shown in Figura 7(c). Again, the resulting subtree with root *o* is tested in all possible positions (Figura 7(c)), aiming at decreasing the cost (provided that the diameter is not violated). According to Figura 7(d), by making *j* the direct predecessor of *o,* the cost of the BDMST is decreased and the diameter is not violated. Then *o* is made direct successor of *j*, using the edge (*j*, *o*), and the process finishes at this point.

There are O(*n*) possible nodes to start a single operation of hierarchy rotation. Each operation involves O(*n*) inversions of predecessor/successor pairs. In turn, each inversion needs O(*n*) subtree reconnection tests in order to decrease the cost. Thus, the total cost of this

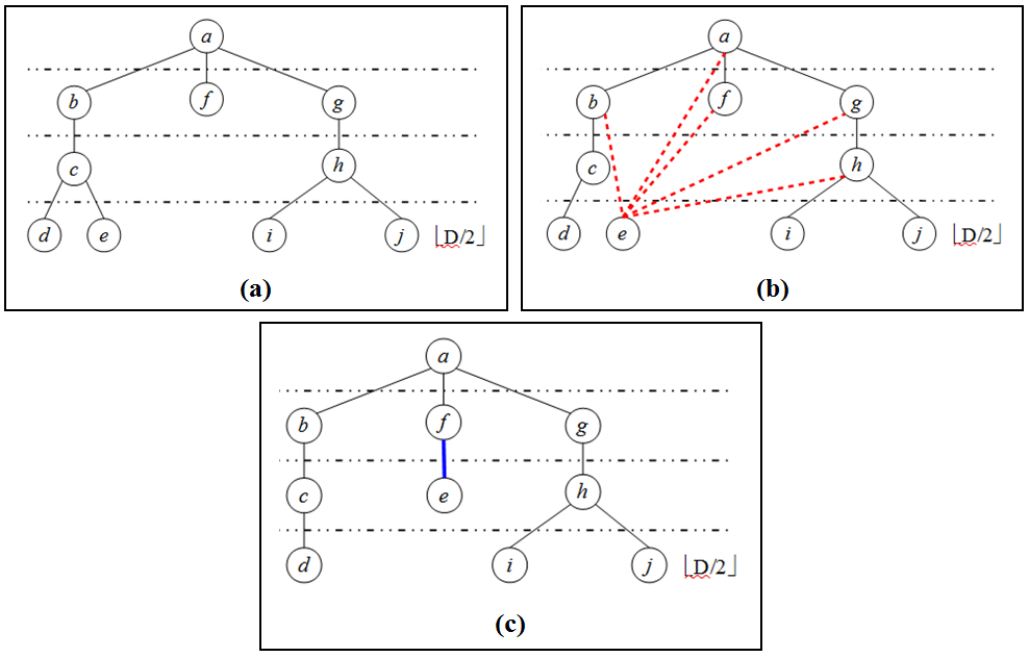
neighborhood is $O(n^3)$.

6.1.6 LEAF SWAP

This neighborhood explores the distribution of a leaf throughout the BDMST, aiming at decreasing the cost.

Let *T* be the BDMST of Figura 8(a). The operation selects a leaf of *T*, in this case, node *e*. Next, the operation verifies whether putting node *e* as a direct successor of other nodes in the tree will reduce its cost, as illustrated in Figura 8(b). In this case, making *e* a direct successor of node *f* reduces the cost of tree, and this is the resulting BDMST, as depicted in Figura 8(c).

Fig. 6: Leaf Swap example.



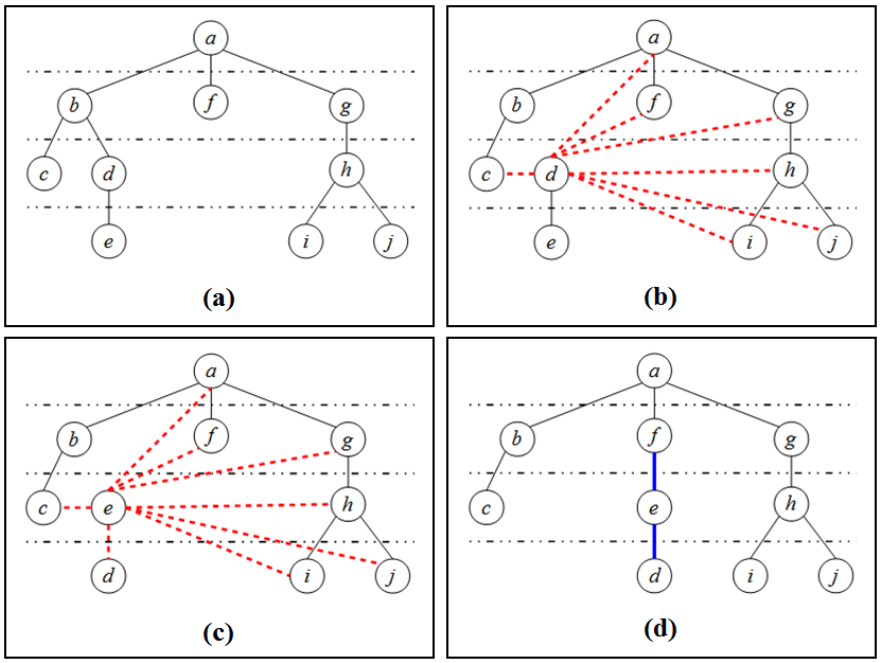
Exploring all the leaves takes $O(n)$ time, while testing a single leaf at other positions of the tree takes $O(n)$ time as well. This results in an $O(n^2)$ total time of for this neighborhood.

6.1.7 FATHER SWAP

This neighborhood tests whether moving a subtree with a root *v*, containing a single leaf *u* as its direct successor, to another place the tree reduces its cost. If the cost of the resulting tree does not decrease, the operation inverts the relation between nodes *v* and *u* (i.e., *u*

becomes the new root of the subtree and *v* its direct successor), and repeats the same test, trying to move the new subtree to another place of the tree in order to reduce its cost. In each operation the diameter should not be violated.

Fig. 7: Father Swap example.



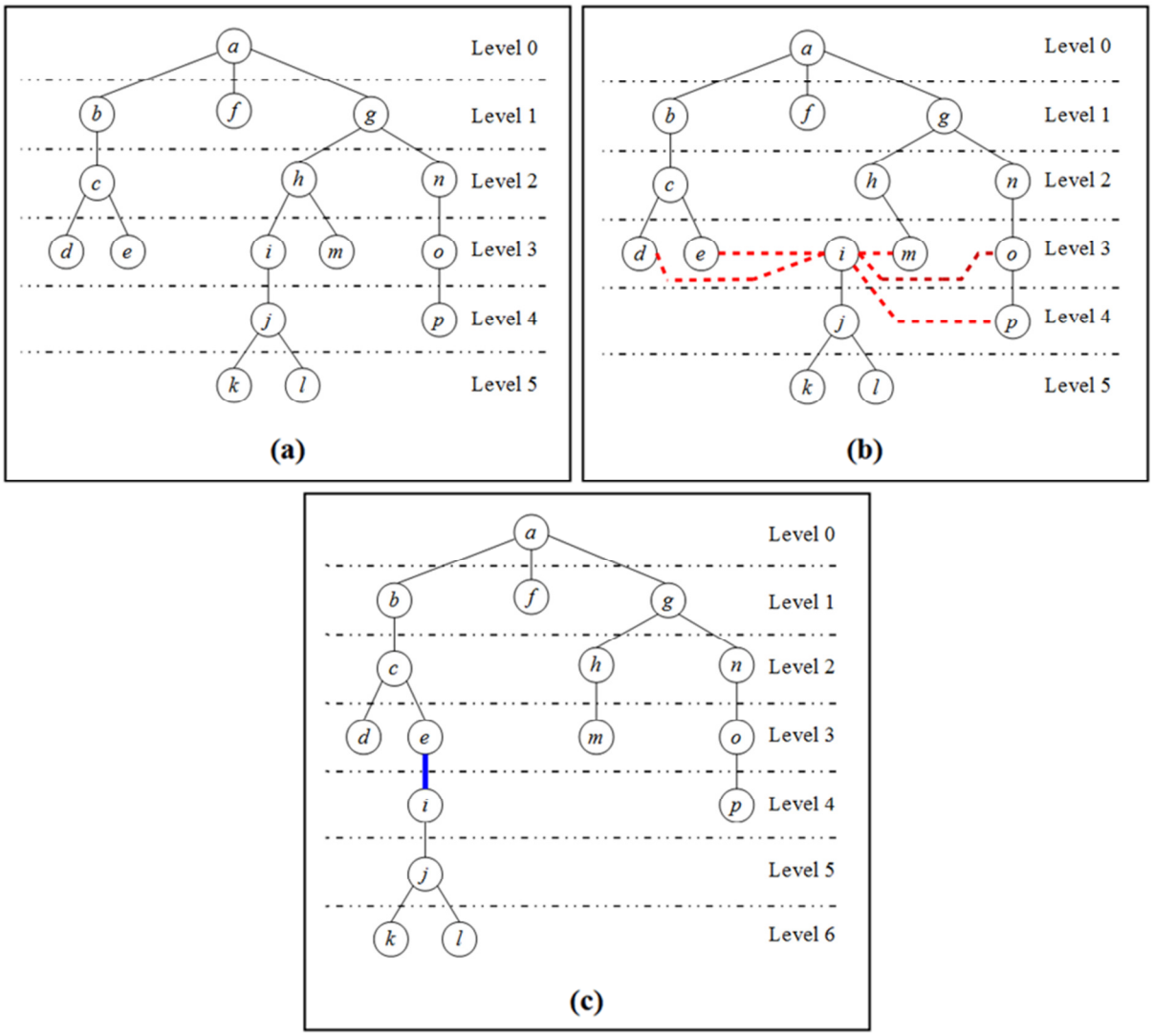
Le be *T* the BDMST of Figura 9(a). First, the operation selects node *d,* since it has a single leaf node as a direct successor. Next, the operation tests whether the subtree rooted at *d* can be moved to another position of the tree in order to reduce its cost, as shown in Figura 9(b). If this does not occur, node *e* becomes the direct predecessor of *d*, and *d* becomes the direct successor of *e* (see Fig. 7(c)). Then, the same test illustrated in Fig. 7(b) is performed, but now for the subtree having *e* as its root (see Fig. 7(c)). According to Figura 9(d), making node *e* a direct successor of *f* reduces the cost of the tree, and then the process finishes.

Exploring all the nodes of *T* to select those having only a leaf as a direct successor takes O(n) time. Verifying whether the connection of a single selected node to another position of the tree reduces its cost takes O(n) time. The inversion operation between predecessor and successor needs constant time, while verifying whether moving the resulting subtree from this inversion to another position of the tree in order to reduce the cost also takes O(n) time. Consequently, the total cost of this neighborhood is O(n^2).

6.1.8 LEVEL CHANGE

This neighborhood verifies whether moving a subtree rooted at a certain node v at a level k to another position in the tree such that v is made the direct successor of a node u at a level $l \geq k$ reduces the cost of the tree and does not violate the diameter.

Fig. 8: Level Change example.



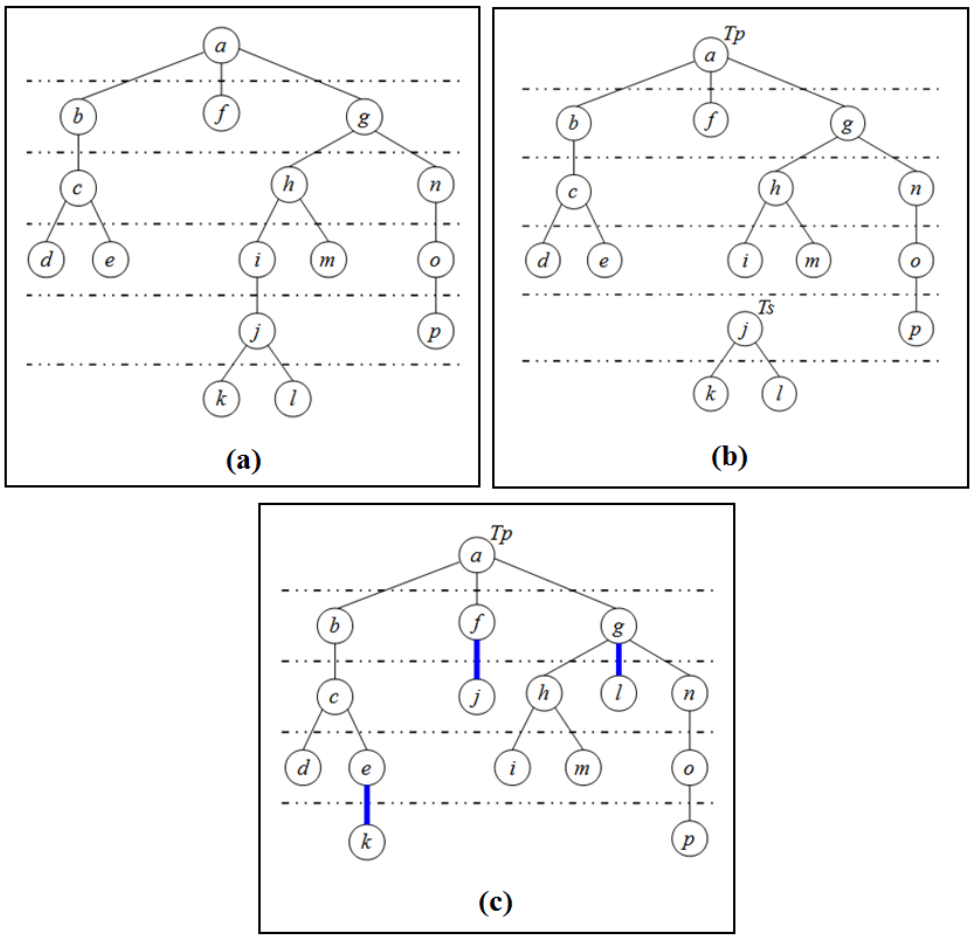
Let T be the BDMST of Figura 10(a). The operation selects node i (at level three) as the subtree root. Next, the operation verifies whether connecting i as a direct successor of a node at level three (d, e, m, or o) or a node at level four (node p) reduces the cost of the tree (see Figura 10(b)). In the example, connecting the subtree to node e reduces the cost of the tree, thus this node is selected to be the direct predecessor of i, and the resulting tree is the one indicated in Figura 10(c).

There are O(n) possible subtrees (roots) to be selected. For a single selected subtree, the time to verify whether connecting it to nodes at the same level or lower levels reduces the cost of the tree is O(n), while the connection of the subtree takes constant time. Thus, the total time for this neighborhood is O(n^2).

6.2 PERTURBATION

As the perturbation procedure, variants of *Edge Exchange* and *Node Swap* neighborhoods have been employed, where the choice of the subtree root to be used in the operation is random. Also, variants of *Edge Delete* [13] and *Center Exchange* [18] (here called *Center Change*) have been used. The choice of which neighborhood will be applied in each perturbation call is made randomly. The history of previous solutions is not considered.

Fig. 9: Edge Delete example.



6.2.1 EDGE DELETE

Let T be the BDMST of Figura 11(a). This neighborhood randomly removes an edge of

T, forming two subtrees T_p and T_s, as shown in Figura 11(b), where edge (i, j) is removed. Next, all nodes of T_s are disconnected and individually reconnected to other places of T_p using edges of lowest possible cost, which is illustrated by Figura 11(c).

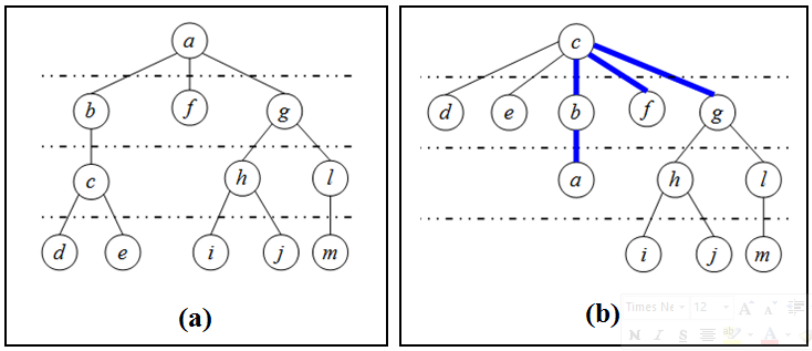
The time to randomly select one node of T is constant. The subtree T_s may contain $O(n)$ nodes, each requiring an $O(n)$ time to be reconnected to some position of T_p. This results in an $O(n^2)$ total time for this neighborhood.

6.2.2 CENTER CHANGE

Let T be a BDMST. This neighborhood randomly selects a node of T (in Fig. 10(a), suppose that c is the selected node) and makes it the new center, if the diameter is even, or one of the centers, if the diameter is odd. The successors of the old center (in the example, node a) become the direct successors of c, while a becomes the direct successor of a randomly chosen node (in the example, according to Fig. 10(b), a becomes the successor of node b).

Figura 12(a) and Figura 12(b) illustrate this neighborhood operation.

Fig. 10: Center Change example.



The time to randomly select a node x of T is constant. The time to make x the new center of the tree is constant, while making the successors of the old center y successors of x is $O(k)$, where k is the number of direct successors of y. The time to connect the old center y as a direct successor of a randomly chosen is constant. Therefore, this total time for this neighborhood is $O(\Delta)$, where Δ is the maximum degree of a node in the tree.

7 ILS IMPLEMENTATION

In the implementation of our ILS approach, the constructive heuristic RGH [13] was

used for the generation of the initial solution.

The acceptance criterion adopted was simply the comparison of the BDMST costs, represented by *s** and *s**' in Algoritmo 5, returning the one with the lowest cost.

Algorithm 12: Local Search

```
procedure LocalSearch(A)
  initialize table with all permutations from {1, 2, 3, 4, 5, 6, 7, 8}
  randomly initialize V with a value between 1 and 8!=40320
  k = 1
  k_max = 8
  A* = A
  repeat
    index = table[V][k]
    case index of
       1: A' = EdgeExchange(A*)
       2: A' = NodeSwap(A*)
       3: A' = SubtreeOptmize(A*)
       4: A' = LevelChange(A*)
       5: A' = LeafSwap(A*)
       6: A' = FatherSwap(A*)
       7: A' = HierarchyExchange(A*)
       8: A' = HierarchyRotation(A*)
    if value(A') < value(A*) then
      A* = A'
      k = 1
    else
      k = k + 1
  until k = k_max
  return A*
```

Algoritmo 9 presents the implementation of the local search. The bi-dimensional matrix *table* contains all possible 8! permutations of neighborhoods, while *V* is the index that corresponds to the permutation randomly chosen to access *table*. The function *value*(*A*) returns the cost of the BDMST *A*.

Algorithm 13: Perturbation

```
procedure Pertubation(A)
  randomly initialize k with a value 1, 2, 3, or 4
       case k:
          1: A* = nodeSwapAleatorio(A)
          2: A* = edgeExchangeAleatorio(A)
          3: A* = edgeDelete(A)
          4: A* = centerChange(A)

  return A*
```

Algorithm 13 presents the implementation of the perturbation. The neighborhood to be

applied is randomly chosen. The history of previously explored computations is not considered.

8 COMPUTATIONAL RESULTS

8.1 ILS PARAMETERS

The proposed ILS has been applied to Euclidean instances (complete graphs) of the OR-Library[21] and compared with the best results found in the literature. For the tests, the first five instances of each dataset with 50, 100, 250, 500, and 1000 vertices, with diameters 5, 10, 15, 20, and 25, respectively, were used. For each of the five instances of each set of graphs, 50 ILS executions were done, similar to the one adopted in the literature, and the tree with lower cost is select. For each test, we consider two cases. The first use as termination condition the times 1000, 2000, 3000, 4000 and 5000 seconds for instances of 50, 100, 250, 500 and 1000 vertices, respectively, and 1000 iterations of the ILS main loop without cost improvement, similar to the approach used by VNS (GRUBER; RAIDL, 2005b), LEEA e ACO (GRUBER; VAN HEMERT; RAIDL, 2006) and GILS (LUCENA; RIBEIRO; SANTOS, 2010) (Table 2). The second case use only the times 1, 10, 160, 1200 and 2000 seconds as termination condition for the instances of 50, 100, 250, 500 and 1000 vertices, respectively; these times were obtained through previous experiments (Table 4).

In the *perturbation* implementation we do not consider the computation historical. Such an approach is similar to that adopted by (LUCENA; RIBEIRO; SANTOS, 2010), (DA SILVA; DE MENEZES FROTA; SUBRAMANIAN, 2012), (PENNA; SUBRAMANIAN; OCHI, 2013) e (MARTINS et al., 2015), which obtained good results for ILS without considering the use of perturbation history. In addition, in the experiments the results obtained by the ILS were already better than those in the literature.

Another point to consider is the non-use of the computational history in the *acceptance criterion*, being accepted only solutions that improve the cost of the BDMST. This is due to the fact that other authors also did not use it and obtained good results, considering only the cost comparison and always choosing the best one, as can be seen in [25], [28], [32] and [30]. In addition, the experiments performed with the ILS without the use of a history in the criterion of acceptance already presented better results than those of the literature, reinforcing the no necessity of its use.

[21] Beasley's OR-Library. Available at http://people.brunel.ac.uk/~mastjjb/jeb/orlib/esteininfo.html

8.2 RESULTS OF THE ILS PROPOSED

The ILS proposed algorithms were implemented in Java 7, running on an Ubuntu system and an Intel I7 machine with 3.4 GHz and 16 GB RAM. The results of can be found in Table 2 and Table 4. Column *"N"* indicates the number of vertices of the input graph; *"D"* the diameter; *"Id"* the identification of the instance; *"Ref"* the method that achieved the best result found so far in the literature; *"Best"* the best solution value obtained by the method indicated in the previous column; "*Mean"* the average solution value obtained over several runs; "*StdDev"* the standard deviation obtained over the runs; "*Time"* (given in seconds) the average time spent in each run for the corresponding instance. Columns *Best**, *Mean** and *Std. Dev.** have the same meaning, but apply to our ILS approach. The *Time 1** column shows either the average processing time spent for the 50 runs (Table 2), or the time to obtain the best solution considering the time limit specified for each run (Table 4), all of which they result from running ILS on the specified computer. *Time 2** column presents the time specified in *Time 1** normalized, which was obtained considering processors closer to those used in the literature, available on the site "cpubenchmark.net/compare", as listed in Table 1.

Table 1: processors used for normalization of time.

Method	*Processor Used by the Method*	*Processor Used for Comparation*	*Normalization Factor*
GILS	Intel Pentium IV 2.0 GHz	Intel Pentium D1508 2.2 GHz	1.61
PEA-I	Intel Pentium IV 2.4 GHz	Intel Pentium D1508 2.2 GHz	1.61
VNS	Intel Pentium IV 2.8 GHz	Intel Pentium G4560T 2.9 GHz	1.44
LEEA	Intel Pentium IV 2.8 GHz	Intel Pentium G4560T 2.9 GHz	1.44
ACO	Intel Pentium IV 2.8 GHz	Intel Pentium G4560T 2.9 GHz	1.44
ILS	Intel I7 3.4 GHz	Intel Core I7 7900X 3.3 GHz	*

Table 2 presents the results of ILS execution using termination condition 1000 iterations without tree cost improvement and 1000, 2000, 3000, 4000 and 5000 second times for instances of 50, 100, 250, 500 and 1000 vertices, respectively. As can be observed, the proposed ILS was able to improve all the results found in the literature (compare the *Best* and *Best** columns), except for the first instance of 1000 vertices. Regarding the computational time, except the instances of 1000 vertices, the ILS was able to overcome the results of the literature in a reasonable time. This demonstrates the competitiveness of the proposed method. In addition, it should be noted that in each row of the table the average time spent

(*Time 2** column) by the ILS to find the best solution (*Best** column) is, in 68% of cases (17 of 25), less than the time specified by the literature (*Time* column). In all cases (except for the first instance of 1000 vertices), ILS has found better solutions with a time shorter than the time specified as a stop condition. Again, for the first instance of 1000 vertices the ILS failed to outperform the literature. In relation to the *Mean** column, the ILS was able to overcome in 40% of the cases the results of the literature (10 of 25). The values for the standard deviation are within a reasonable range, as can be verified by the columns "*Std. Dev.*" and "*Std. Dev**".

Table 2: computational results for ILS with 1000, 2000, 3000, 4000 and 5000 seconds, and 1000 iterations without improvement as termination condition. (*n.a.* = not available)

Instance			Bests Results in the Literature					ILS Results				
N	*D*	*Id*	*Ref*	*Best*	*Mean*	*Std. Dev.*	*Time* (*s*)	*Best**	*Mean**	*Std. Dev.**	*Time* 1 (*s*)*	*Time 2* (*s*)*
50	5	1	GILS	7.60	7.60	*n.a.*	10.00	**7.60**	7.61	0.024	0.40	0.64
50	5	2	GILS	7.61	7.62	*n.a.*	10.00	**7.61**	**7.62**	0.014	0.46	0.74
50	5	3	GILS	7.24	**7.24**	*n.a.*	10.00	**7.24**	7.26	0.033	0.45	0.72
50	5	4	GILS	6.59	6.59	*n.a.*	10.00	**7.59**	**6.59**	0.000	0.36	0.58
50	5	5	GILS	7.25	7.25	*n.a.*	10.00	**7.25**	**7.25**	0.000	0.42	0.68
100	10	1	ACO	7.76	**7.77**	0.020	27.78	**7.76**	7.83	0.032	4.10	5.90
100	10	2	LEEA	7.85	**7.86**	0.020	734.65	**7.85**	7.89	0.031	3.23	4.65
100	10	3	VNS	7.90	7.96	0.040	38.66	**7.90**	**7.96**	0.042	4.83	6.96
100	10	4	LEEA	7.98	8.09	0.030	732.83	**7.98**	**8.02**	0.039	3.61	5.20
100	10	5	ACO	8.16	**8.17**	0.000	25.45	**8.16**	8.22	0.052	3.70	5.33
250	15	1	ACO	12.23	**12.28**	0.020	174.17	**12.18**	12.29	0.054	81.04	116.70
250	15	2	ACO	12.02	**12.04**	0.010	156.71	**12.02**	12.15	0.064	63.61	91.60
250	15	3	ACO	12.00	**12.02**	0.010	145.29	**11.98**	12.04	0.045	56.84	81.85
250	15	4	PEA-I	12.42	12.57	0.050	*n.a.*	**12.42**	**12.52**	0.056	72.28	116.37
250	15	5	ACO	12.23	**12.29**	0.040	211.11	**12.21**	12.31	0.059	70.35	101.30
500	20	1	ACO	16.78	**16.85**	0.030	906.17	**16.77**	16.90	0.082	735.87	1059.65
500	20	2	ACO	16.63	**16.70**	0.030	1012.91	**16.63**	16.77	0.065	618.15	890.14
500	20	3	ACO	16.79	**16.84**	0.030	1069.84	**16.78**	16.88	0.058	731.69	1053.63
500	20	4	ACO	16.80	**16.92**	0.040	1010.91	**16.78**	16.93	0.065	628.13	904.51
500	20	5	ACO	16.42	**16.46**	0.020	947.26	**16.41**	16.53	0.069	736.13	1060.03
1000	25	1	LEEA	**23.43**	**23.57**	0.080	565.38	23.63	23.73	0.063	4821.77	6943.35
1000	25	2	LEEA	23.46	23.67	0.080	561.49	**23.36**	**23.48**	0.079	4953.84	7133.53
1000	25	3	PEA-I	23.61	23.76	0.080	*n.a.*	**23.15**	**23.28**	0.069	4777.56	7691.87
1000	25	4	LEEA	23.79	23.96	0.090	602.30	**23.56**	**23.69**	0.070	4847.89	6980.96
1000	25	5	PEA-I	23.75	23.90	0.070	*n.a.*	**23.25**	**23.40**	0.060	4861.67	7827.29

To compare the results of ILS, the exact method described in (2004) was implemented in CPLEX and executed for the first five instances of Euclidean complete graphs of the OR-

Library with 50 vertices and diameter five. For each instance, the time of 24 hours was used as the stop condition. The results obtained are shown in Table 3. The best viable solution found is indicated by the column "*Upper Bound*". In all cases, it was not possible to prove the optimality of any instance. In addition, comparing the results of Table 2 with those of Table 3, the ILS presented a better results for the instances of 50 vertices with a much shorter time, although the upper limits marked with (*) were the same as those obtained by the "*Best* *" column.

Table 3: results of CPLEX execution.

Instance	*Lower Bound*	*Upper Bound*
1	6,16	7,73
2	6,06	7,62
3	5,91	7,55
4	5,82	6,59 (*)
5	6,05	7,25 (*)

Table 4 presents the results of ILS execution using as the termination condition only the times of 1, 10, 160, 1200 and 2000 seconds for each of the 50 executions for the instances of 50, 100, 250 , 500 and 1000 vertices, respectively. Note that the proposed ILS was able to improve all the results found in the literature in a reasonable computational time, except for instances of 1000 vertices. In addition, it should be noted that in each row of the table the normalized time (*Time 2** column) spent by the ILS to find the best solution is, in 52% of cases (13 of 25), less than the time specified by the literature. In all cases (except for the first instance of 1000 vertices), the ILS found better solutions with a shorter time than the stop condition. Again, for the first instance of 1000 vertices the ILS failed to outperform the literature. In relation to the *Mean** column, the ILS proposed was able to overcome in 59% of the cases the results of the literature (14 of 25). The values for the standard deviation are also within a reasonable range.

Table 4: computational results for ILS with 1, 10, 160, 1200 and 2000 seconds as termination condition. (*n.a.* = not available)

Instance			Bests Results in the Literature					ILS Results				
N	*D*	*Id*	*Ref*	*Best*	*Mean*	*Std. Dev.*	*Time* (*s*)	*Best**	*Mean**	*Std. Dev.**	*Time* 1 (*s*)*	*Time 2* (*s*)*
50	5	1	GILS	7.60	7.60	*n.a.*	10.00	**7.60**	7.61	0.024	0.14	0.23
50	5	2	GILS	7.61	7.62	*n.a.*	10.00	**7.61**	**7.61**	0.007	0.34	0.55
50	5	3	GILS	7.24	7.24	*n.a.*	10.00	**7.24**	7.26	0.033	0.45	0.72

50	5	4	GILS	6.59	6.59	*n.a.*	10.00	**6.59**	**6.59**	0.000	0.05	0.08
50	5	5	GILS	7.25	7.25	*n.a.*	10.00	**7.25**	**7.25**	0.000	0.27	0.43
100	10	1	ACO	7.76	7.77	0.020	27.78	**7.76**	7.82	0.030	1.39	2.00
100	10	2	LEEA	7.85	7.86	0.020	734.65	**7.85**	7.88	0.024	3.48	5.01
100	10	3	VNS	7.90	7.96	0.040	38.66	**7.90**	**7.94**	0.037	2.43	3.50
100	10	4	LEEA	7.98	8.09	0.030	732.83	**7.98**	**8.00**	0.028	0.66	0.95
100	10	5	ACO	8.16	8.17	0.000	25.45	**8.16**	8.20	0.047	3.08	4.44
250	15	1	ACO	12.23	12.28	0.020	174.17	**12.18**	**12.26**	0.047	88.44	127.35
250	15	2	ACO	12.02	12.04	0.010	156.71	**12.01**	12.12	0.060	148.69	214.11
250	15	3	ACO	12.00	12.02	0.010	145.29	**11.97**	**12.02**	0.034	137.27	197.67
250	15	4	PEA-I	12.42	12.57	0.050	*n.a.*	**12.42**	**12.49**	0.045	159.68	257.08
250	15	5	ACO	12.23	12.29	0.040	211.11	**12.19**	**12.29**	0.053	117.79	169.62
500	20	1	ACO	16.78	16.85	0.030	906.17	**16.74**	16.88	0.077	1073.28	1545.52
500	20	2	ACO	16.63	16.70	0.030	1012.91	**16.62**	16.74	0.058	1163.48	1675.41
500	20	3	ACO	16.79	16.84	0.030	1069.84	**16.78**	16.86	0.046	544.31	783.81
500	20	4	ACO	16.80	16.92	0.040	1010.91	**16.75**	**16.90**	0.062	914.04	1316.22
500	20	5	ACO	16.42	16.46	0.020	947.26	**16.40**	16.51	0.066	1076.61	1550.32
1000	25	1	LEEA	23.43	23.57	0.080	565.38	23.72	23.83	0.069	1801,18	2593,70
1000	25	2	LEEA	23.46	23.67	0.080	561.49	**23.45**	**23.60**	0.090	1874.62	2699.45
1000	25	3	PEA-I	23.61	23.76	0.080	*n.a.*	**23.22**	**23.38**	0.077	1899.84	3058.74
1000	25	4	LEEA	23.79	23.96	0.090	602.30	**23.66**	**23.81**	0.071	1930.30	2779.63
1000	25	5	PEA-I	23.75	23.90	0.070	*n.a.*	**23.35**	**23.52**	0.073	1788.12	2878.87

9 CONCLUSIONS

This paper presents an ILS approach for the EBDMSTP. Five new neighborhoods are proposed for the local search, which is a variation of the Variable Neighborhood Descent method, called here RVND, and four for the perturbation. The tests performed showed that these neighborhoods allowed our ILS approach to obtain better solution values than those presented in the literature, using 25 OR-Library graph instances with 50, 100, 200, 500, and 1000 vertices, except in a single case (an instance with 1000 vertices). To the best of the authors' knowledge, this is the first time an ILS+RVND approach is proposed for the solution of the EBDMSTP.

Based on the results obtained, we can say that the new proposed neighborhoods (*Hierarchy Exchange*, *Hierarchy Rotation*, *Leaf Swap*, *Father Swap*, and *Level Change*) together with the variants of existing neighborhoods in the literature (*Edge Exchange*, *Node Swap*, *Subtree Optimize*, *Edge Delete*, and *Center Change*) were crucial to the ILS performance, making it extremely competitive in comparison with other existing methods.

10 REFERENCES

1. ABDALLA, A.; DEO, N.; GUPTA, P. Random-Tree Diameter and the Diameter-Constrained MST. **Congressus Numerantium**, n. 144, p. 161–182, 2000.
2. ACHUTHAN, N. R. et al. Computational methods for the diameter restricted minimum weight spanning tree problem. **Australasian Journal of Combinatorics**, v. 10, p. 51–71, 1994.
3. BALA, K.; PETROPOULOS, K.; STERN, T. E. Multicasting in a linear lightwave network. **IEEE INFOCOM '93 The Conference on Computer Communications, Proceedings**, p. 1350–1358, 1993.
4. BEAN, J. C. Genetic Algorithms and Random Keys for Sequencing and Optimization. **ORSA Journal on Computing**, v. 6, n. 2, p. 154–160, maio 1994.
5. BINH, T. T. H. et al. New heuristic and hybrid genetic algorithm for solving the bounded diameter minimum spanning tree problem. **Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09**, p. 373, 2009.
6. BOOKSTEIN, A.; KLEIN, S. T. Compression of Correlated Bit-Vectors. **Information systems**, v. 16, n. 4, p. 387–400, 1991.
7. CHIARANDINI, M.; STÜTZLE, T. An application of iterated local search to graph coloring problem. **Proceedings of the Computational Symposium on Graph Coloring and its Generalizations**, p. 1–13, 2002.
8. DA SILVA, D. M.; DE MENEZES FROTA, Y. A.; SUBRAMANIAN, A. Uma Heurística para o Problema de Roteamento de Veículos com Múltiplas Viagens. **Congresso Lationo-Iberoamericana de Investigación Operativa**, p. 1880–1891, 2012.
9. DAVIS, L. **Handbook of genetic algorithms**. Van Nostrand Reinhold, New York, 1991.
10. DONG, X.; HUANG, H.; CHEN, P. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. **Computers & Operations Research**, v. 36, n. 5, p. 1664–1669, 2009.
11. GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: a guide to the theory of NP-Completeness**. San Francisco: WH Freeman & Co., 1979.
12. GLOVER, F.; KOCHENBERGER, G. A. **Handbook of Metaheuristic**. Kluwer Academic Publishers, New York, 2003.
13. GOLDBERG, D. E.; LINGLE, J. R. **Alleles, loci, and the traveling salesman problem**. (J. J. Greffenstette, Ed.)Proceedings of the First International Conference on Genetic Algorithms. **Anais**...Lawrence Erlbaum, 1985
14. GOUVEIA, L.; MAGNANTI, T. L. Network flow models for designing diameter-constrained minimum-spanning and Steiner trees. **Networks**, v. 41, n. 3, p. 159–173, 2003.
15. GOUVEIA, L.; SIMONETTI, L.; UCHOA, E. Modeling hop-constrained and diameter-constrained minimum spanning tree problems as Steiner tree problems over layered

graphs. **Mathematical Programming**, v. 128, n. 1–2, p. 123–148, jun. 2011.

16. GRUBER, M.; RAIDL, G. R. A new 0–1 ILP approach for the bounded diameter minimum spanning tree problem. **Proceedings of the 2nd International Network Optimization Conference**, p. 178–185, 2005a.

17. GRUBER, M.; RAIDL, G. R. Variable Neighborhood Search for the Bounded Diameter Minimum Spanning Tree Problem. **Proceedings of the 18th Mini Euro Conference on Variable Neighborhood Search**, p. 1–12, 2005b.

18. GRUBER, M.; RAIDL, G. R. (Meta-) Heuristic Separation of Jump Cuts in a Branch & Cut Approach for the Bounded Diameter Minimum Spanning Tree Problem. **Work**, v. 8, n. September, p. 209–229, 2008.

19. GRUBER, M.; VAN HEMERT, J.; RAIDL, G. R. Neighbourhood searches for the bounded diameter minimum spanning tree problem embedded in a VNS, EA, and ACO. **Proceedings of the 8th annual conference on Genetic and evolutionary computation - GECCO '06**, p. 1187, 2006.

20. HANDLER, G. Y. Minimax location of a facility in an undirected tree graph. **Transportation Sci**, v. 7(3), p. 287–293, 1973.

21. HANSEN, P.; MLADENOVIĆ, N. Variable neighborhood search: Principles and applications. **European Journal of Operational Research**, v. 130, n. 3, p. 449–467, 2001.

22. HANSEN, P.; MLADENOVIĆ, N. Variable Neighborhood Search. In: BURKE, E. K.; KENDALL, G. (Eds.). . **SEARCH METHODOLOGIES**. Boston, MA: Springer US, 2005. p. 211–238.

23. JULSTROM, B. A. Encoding Bounded-Diameter Spanning Trees with Permutations and with Random Keys. **Genetic and Evolutionary Computation – GECCO 2004**, p. 1272–1281, 2004.

24. JULSTROM, B. A. Greedy heuristics for the bounded diameter minimum spanning tree problem. **Journal of Experimental Algorithmics**, v. 14, n. 1, p. 1.1, 2009.

25. JULSTROM, B. A.; RAIDL, G. R. A Permutation-Coded Evolutionary Algorithm for the Bounded-Diameter Minimum Spanning Tree Problem. **in 2003 Genetic and Evolutionary Computation Conference's Workshops Proceedings, Workshop on Analysis and Design of Representations**, p. 2–7, 2003.

26. KRUSKAL, J. B. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. **Proceedings of the American Mathematical Society**, v. 7, n. 1, p. 48, fev. 1956.

27. LOURENÇO, H. R.; MARTIN, O. C.; STÜTZLE, T. Iterated Local Search. In: **Handbook of Metaheuristics**. New York: KLUWER ACADEMIC PUBLISHERS, p. 320–353, 2003.

28. LUCENA, A.; RIBEIRO, C. C.; SANTOS, A. C. A hybrid heuristic for the diameter constrained minimum spanning tree problem. **Journal of Global Optimization**, v. 46, n. 3, p. 363–381, 2010.

29. MARTINS, I. C.; PINHEIRO, R. G. S.; PROTTI, F.; OCHI, L. S. A Hybrid Iterated Local Search and Variable Neighborhood Descent Heuristic Applied to the Cell Formation Problem. **Submited and Accepted to Elsevier Editorial System for Expert Systems With Applications**, 2015.

30. MLADENOVIĆ, N.; HANSEN, P. Variable Neighborhood Search. **Computers & Operations Research**, v. 24, n. 11, p. 1097–1100, 1997.

31. PENNA, P. H. V.; SUBRAMANIAN, A.; OCHI, L. S. An Iterated Local Search heuristic for the Heterogeneous Fleet Vehicle Routing Problem. **Journal of Heuristics**, v. 19, n. 2, p. 201–232, 2013.

32. PRIM, R. C. **Shortest Connection Networks And Some GeneralizationsBell System Technical Journal**, 1957. Disponível em: <http://dx.doi.org/10.1002/j.1538-7305.1957.tb01515.x>

33. RAIDL, G. R.; JULSTROM, B. A. Greedy heuristics and an evolutionary algorithm for the bounded-diameter minimum spanning tree problem. **Proceedings of the 2003 ACM symposium on Applied computing - SAC '03**, p. 747, 2003a.

34. RAIDL, G. R.; JULSTROM, B. A. Edge sets: An effective evolutionary coding of spanning trees. **IEEE Transactions on Evolutionary Computation**, v. 7, n. 3, p. 225–239, 2003b.

35. RAYMOND, K. A tree-based algorithm for distributed mutual exclusion. **ACM Transactions on Computer Systems**, v. 7, n. 1, p. 61–77, 1 jan. 1989.

36. SANTOS, A. C.; LUCENA, A.; RIBEIRO, C. C. Solving Diameter Constrained Minimum Spanning Tree Problems in Dense Graphs. **Proceedings of the International Workshop on Experimental Algorithms**, v. 3059, p. 458–467, 2004.

37. SINGH, A.; GUPTA, A. K. Improved heuristics for the bounded-diameter minimum spanning tree problem. **Soft Computing**, v. 11, n. 10, p. 911–921, 10 maio 2007.