UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

MARCOS ANTÔNIO ALMEIDA DE OLIVEIRA

# Heurística aplicada ao problema Árvore de Steiner Euclidiano com representação nó-profundidade-grau

Goiânia
2014

**TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR AS TESES E DISSERTAÇÕES ELETRÔNICAS (TEDE) NA BIBLIOTECA DIGITAL DA UFG**

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

**1. Identificação do material bibliográfico**: **[ X ] Dissertação** **[ ] Tese**

**2. Identificação da Tese ou Dissertação**

| | | | | | |
|---|---|---|---|---|---|
| Autor (a): | Marcos Antônio Almeida de Oliveira | | | | |
| E-mail: | marcosoliveira.comp@gmail.com | | | | |
| Seu e-mail pode ser disponibilizado na página? [ X ]Sim [ ] Não | | | | | |
| Vínculo empregatício do autor | Servidor público municipal | | | | |
| Agência de fomento: | Fundação de Amparo à Pesquisa do Estado de Goiás | | | Sigla: | FAPEG |
| País: | Brasil | UF: | GO | CNPJ: | 08.156.102/0001-02 |
| Título: | Heurística aplicada ao problema árvore de Steiner Euclidiano com representação nó-profundidade-grau | | | | |
| | | | | | |
| Palavras-chave: | Árvore de Steiner Euclidiana, Representação nó-profundidade-grau, Heurística | | | | |
| Título em outra língua: | Heuristic applied to the Euclidean Steiner tree problem with node-depth-degree encoding | | | | |
| | | | | | |
| Palavras-chave em outra língua: | Euclidean Steiner Tree Problem, Heuristic Algorithm, Node-Depth-Degree Encoding | | | | |
| | | | | | |
| Área de concentração: | CIÊNCIA DA COMPUTAÇÃO | | | | |
| Data defesa: (dd/mm/aaaa) | 03/09/2014 | | | | |
| Programa de Pós-Graduação: | Programa de Pós-Graduação em Ciência da Computação | | | | |
| Orientadora: | Telma Woerle Lima Soares | | | | |
| E-mail: | telma@inf.ufg.br | | | | |
| Co-orientador:* | Leslie Richards Foulds | | | | |
| E-mail: | lesfoulds@gmail.com | | | | |

*Necessita do CPF quando não constar no SisPG

**3. Informações de acesso ao documento:**

Concorda com a liberação total do documento [ X ] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF ou DOC da tese ou dissertação.

O sistema da Biblioteca Digital de Teses e Dissertações garante aos autores, que os arquivos contendo eletronicamente as teses e ou dissertações, antes de sua disponibilização, receberão procedimentos de segurança, criptografia (para não permitir cópia e extração de conteúdo, permitindo apenas impressão fraca) usando o padrão do Acrobat.

_____ Data: _____ / _____ / _____

Assinatura do (a) autor (a)

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

MARCOS ANTÔNIO ALMEIDA DE OLIVEIRA

# Heurística aplicada ao problema árvore de Steiner Euclidiano com representação nó-profundidade-grau

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Computação.

**Área de concentração:** CIÊNCIA DA COMPUTAÇÃO.

**Orientadora:** Profa. Telma Woerle Lima Soares

**Co-Orientador:** Prof. Leslie Richards Foulds

Goiânia
2014

**Dados Internacionais de Catalogação na Publicação na (CIP)**
**GPT/BC/UFG**

O48h

Oliveira, Marcos Antônio Almeida de.
Heurística aplicada ao problema árvore de Steiner Euclidiano com representação nó-profundidade-grau [manuscrito] / Marcos Antônio Almeida de Oliveira. - 2014.
71 f. : il.

Orientadora: Profª. Drª. Telma Woerle Lima Soares; Co-orientador: Leslie Richards Foulds.
Dissertação (Mestrado) – Universidade Federal de Goiás, Instituto de Informática, 2014.
Bibliografia.
Inclui lista de figuras, tabelas, algoritmos, símbolos, abreviaturas e siglas.

1. Algoritmos 2. Heurística. I. Título.

CDU 004.021/.023

Marcos Antônio Almeida de Oliveira

# Heurística Aplicada ao Problema Árvore de Steiner Euclidiano com Representação Nó-profundidade-grau

Dissertação defendida no Programa de Pós-Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Ciência da Computação, aprovada em 03 de setembro de 2014, pela Banca Examinadora constituída pelos professores:

**Profa. Dra. Telma Woerle de Lima Soares**
Instituto de Informática - UFG
Presidente da Banca

**Prof. Dr. Leslie Richard Foulds**
Instituto de Informática – UFG

**Prof. Dr. Alexandre Cláudio Botazzo Delbem**
ICMC – USP

**Profa. Dra. Érika Morais Martins Coelho**
Instituto de Informática - UFG

Dedico esse trabalho ao meu Pai, Sr. Alonço Ribeiro de Oliveira, sem ele esse trabalho não existiria.

# Agradecimentos

Agradeço principalmente a Deus pelas oportunidades de crescimento.

Agradeço aos meus pais, pessoas excepcionais, fundamentais pelo incentivo e apoio que tive.

À minha esposa pela compreensão e companheirismo em todos os momentos.

Ao meus irmãos pela contribuição e amizade.

À professora Dra. Telma Woerle Lima Soares pelo excelente trabalho e orientação e pela parceria construída.

Ao professor Dr. Leslie Richards Foulds pela contribuição no problema de estudo.

Ao professor Dr. Anderson da Silva Soares pelo acolhimento e contribuição como pesquisador.

Ao professor Dr. Alexandre Cláudio Botazzo Delbem pelo apoio e participação na banca.

À Prefeitura Municipal de Anápolis pelo apoio e compreensão durante o período desse trabalho.

À Fundação de Amparo à Pesquisa do Estado de Goiás (FAPEG) pelo apoio financeiro.

A todos meus sinceros agradecimentos.

A simplicidade é o último grau de sofisticação.

**Leonardo da Vinci**,

.

# Resumo

Oliveira, M. A. A.. **Heurística aplicada ao problema árvore de Steiner Euclidiano com representação nó-profundidade-grau**. Goiânia, 2014. 72p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

É apresentada uma variação do algoritmo de Beasley (1992) para o Problema árvore de Steiner Euclidiano. Essa variação utiliza a Representação Nó-Profundidade-Grau que requer, em média, tempo $O(\sqrt{n})$ em operações para gerar e manipular florestas geradoras. Para problemas de árvore geradora essa representação possui complexidade de tempo linear sendo aplicada em problemas de projeto de redes com algoritmos evolutivos. Resultados computacionais são dados para casos de teste envolvendo instâncias de até 500 vértices. Esses resultados demonstram a utilização da representação Nó-Profundidade-Grau em uma heurística exata, e isso sugere a possibilidade de utilização dessa representação em outras técnicas além de algoritmos evolutivos. Um comparativo empírico e da análise de complexidade entre o algoritmo proposto e uma representação convencional indica vantagens na eficiência da solução encontrada.

**Palavras–chave**

Árvore de Steiner Euclidiana, Representação nó-profundidade-grau, Heurística.

# Abstract

Oliveira, M. A. A.. **Heuristic applied to the Euclidean Steiner tree problem with node-depth-degree encoding**. Goiânia, 2014. 72p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

A variation of the Beasley (1992) algorithm for the Euclidean Steiner tree problem is presented. This variation uses the Node-Depth-Degree Encoding, which requires an average time of $O(\sqrt{n})$ in operations to generate and manipulate spanning forests. For spanning tree problems, this representation has linear time complexity when applied to network design problems with evolutionary algorithms. Computational results are given for test cases involving instances up to 500 vertices. These results demonstrate the use of the Node-Depth-Degree in an exact heuristic, and this suggests the possibility of using this representation in other techniques besides evolutionary algorithms. An empirical comparative and complexity analysis between the proposed algorithm and a conventional representation indicates the efficiency advantages of the solution found.

**Keywords**

Euclidean Steiner Tree Problem, Heuristic Algorithm, Node-Depth-Degree Encoding

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Algoritmos

# Lista de Símbolos

| | |
|---|---|
| **A** | Um ponto no plano ou vértice em exemplos do problema . . . . . 18 |
| **a** | Distância entre os pontos $A$ e $P$ . . . . . 18 |
| **AB** | Lado do triângulo que conecta os pontos $A$ e $B$ . . . . . 20 |
| **ABC** | Triângulo formado pelos pontos $A$, $B$ e $C$ . . . . . 18 |
| $\bar{A}$ | O ponto equilátero composto pelos pontos $B$ e $C$, oposto ao ponto $A$, do triângulo $ABC$. . . . . 20 |
| $a_{ij}$ | Elemento de uma matriz na linha $i$ e coluna $j$. . . . . 38 |
| **B** | Um ponto no plano ou vértice em exemplos do problema . . . . . 18 |
| **b** | Distância entre os pontos $B$ e $P$ . . . . . 18 |
| $\bar{B}$ | O ponto equilátero composto pelos pontos $A$ e $C$, oposto ao ponto $B$, do triângulo $ABC$. . . . . 20 |
| **C** | Um ponto no plano ou vértice em exemplos do problema . . . . . 46 |
| **c** | Distância entre os pontos $C$ e $P$ . . . . . 18 |
| **Ci** | Circunferência que circunscreve o triângulo $Eq$. . . . . 25 |
| **CD** | Lado do triângulo que conecta os pontos $C$ e $D$ . . . . . 20 |
| $\bar{C}$ | O ponto equilátero composto pelos pontos $A$ e $B$, oposto ao ponto $C$, do triângulo $ABC$. . . . . 20 |
| **D** | Um ponto no plano ou vértice em exemplos do problema . . . . . 20 |
| **Eq** | Triângulo equilátero composto pelos vértices $v_1$ e $v_4$ e $v_{eq}$. . . . . 24 |
| **E** | Exemplo de vértice em uma topologia . . . . . 22 |
| **F** | Exemplo de vértice em uma topologia . . . . . 22 |
| **G** | Grafo não direcionado. . . . . 28 |
| **g** | Índice auxiliar utilizado no algoritmo . . . . . 48 |
| **H** | Vértices do grafo $G$. . . . . 28 |
| **h** | Uma vértice do grafo $G$. . . . . 38 |
| **I** | Arestas do grafo $G$. . . . . 28 |
| **i** | Índice auxiliar utilizado no algoritmo . . . . . 48 |
| **J** | Uma vértice de Steiner no exemplo da $RNPG$. . . . . 33 |
| **j** | Índice auxiliar utilizado no algoritmo . . . . . 48 |
| **K** | Uma subárvore de quatro pontos. . . . . 33 |
| **k** | Inteiro indicando quantidade de vértices. . . . . 56 |
| **L** | Conjunto de pontos de Steiner contendo outros além dos que compõem a solução do problema. . . . . 28 |
| **l** | Comprimento do lado de um triângulo equilátero. . . . . 26 |
| **m** | Coeficiente angular da reta. . . . . 25 |
| $M_{ma}$ | Tempo médio de execução do $OCC$ com Matriz de Adjacências. . . . . 63 |
| $M_r$ | Tempo médio de execução do $OCC$ com RNPG. . . . . 63 |
| **N** | Conjunto de pontos originais de um problema . . . . . 21 |
| **n** | Número de pontos originais do problema . . . . . 15 |
| **P** | O ponto de Steiner procurado para os pontos $A$, $B$ e $C$ . . . . . 18 |

| | |
|---|---|
| $P_1$ | Um ponto qualquer incidido pela circunferência . . . . . 26 |
| $P_{AB}$ | O terceiro vértice do triângulo equilátero composto pelos pontos $A$ e $B$ . 20 |
| $P_{CD}$ | O terceiro vértice do triângulo equilátero composto pelos pontos $C$ e $D$ . 20 |
| **p** | Número inteiro que indica um índice em $K$ . . . . . 48 |
| **q** | Número inteiro que indica um índice em $K$ . . . . . 48 |
| $R_1$ | Reta que passa por $v_1$ e $v_4$. . . . . 24 |
| $R_2$ | Reta perpendicular a $R_1$ que passa por $v_{med}$. . . . . 25 |
| $R_3$ | Reta que cruza os pontos $v_{eq}$ e $v_0$. . . . . 27 |
| **r** | Raio da circunferência . . . . . 48 |
| **S** | Conjunto de pontos de Steiner na solução de um problema . . . . . 21 |
| **s** | Número de pontos de Steiner na solução de um problema . . . . . 20 |
| $s_1$ | Número inteiro que indica um índice em $K$ . . . . . 48 |
| **T** | Uma árvore que representa a solução de um problema . . . . . 21 |
| **T**($V_0$) | Custo da $MST$ . . . . . 58 |
| **T**($V_F$) | Custo da solução final . . . . . 58 |
| $tam$ | Variável inteira para delimitar grau . . . . . 48 |
| $tam_q$ | Tamanho da árvore enraizada em $q$ . . . . . 51 |
| $tam_p$ | Tamanho para armazenar a subárvore de $p$ . . . . . 51 |
| **V** | Conjunto de vértices originais $N$ mais vértices de Steiner sendo adicionados. 32 |
| **v** | Um vértice de um grafo $G$. . . . . 38 |
| $V_i$ | Um dos terminais da árvore para encontrar o ponto de Steiner . . . . . 31 |
| $V_j$ | Um dos terminais da árvore para encontrar o ponto de Steiner . . . . . 31 |
| $V_k$ | Um dos terminais da árvore para encontrar o ponto de Steiner . . . . . 31 |
| $v_{eq}$ | Terceiro vértice do triângulo equilátero composto por $v_1$ e $v_4$. . . . . 24 |
| $v_c$ | Ponto central da circunferência $Ci$. . . . . 26 |
| $v_{med}$ | Ponto médio entre $v_1$ e $v_4$. . . . . 24 |
| $v_0$ | Um dos vértices do exemplo real para encontrar o ponto de Steiner . . . . 24 |
| $v_1$ | Um dos vértices do exemplo real para encontrar o ponto de Steiner . . . . 24 |
| $v_4$ | Um dos vértices do exemplo real para encontrar o ponto de Steiner . . . . 24 |
| **X** | O terceiro vértice do triângulo equilátero composto pelos pontos $B$ e $C$ . 18 |
| **x** | Coordenada no plano euclidiano . . . . . 23 |
| **y** | Coordenada no plano euclidiano . . . . . 23 |
| **Θ** | Complexidade assintótica . . . . . 32 |

# Lista de Abreviaturas e Siglas

| | |
|---|---|
| $dc-MSTP$ | degree-constrained Minimum Spanning Tree Problem . . . . . 41 |
| $ESTP$ | Euclidean Steiner Tree Problem . . . . . 15 |
| $MST$ | Minimal Spanning Tree . . . . . 17 |
| $FST$ | Full Steiner Tree . . . . . 22 |
| $OCC$ | Operador de Consulta de Conjuntos . . . . . 16 |
| $ORN$ | Operador de Remoção de Nó . . . . . 16 |
| $RNPG$ | Representação nó-profundidade-grau . . . . . 16 |
| $RMST$ | Relatively Minimal Steiner Tree . . . . . 22 |
| $RSMT$ | Rectilinear Steiner Minimal Tree . . . . . 29 |
| $RST$ | Rectilinear Steiner Tree . . . . . 29 |
| $RSTP$ | Rectilinear Steiner Tree Problem . . . . . 15 |
| $SMT$ | Steiner Minimal Tree . . . . . 21 |
| $SPG$ | Steiner Problem in Graphs . . . . . 15 |
| $STP$ | Steiner Tree Problem . . . . . 21 |
| $VLSI$ | Very-Large-Scale Integration . . . . . 15 |

CAPÍTULO 1

# Introdução

O problema de árvore de Steiner tem por objetivo encontrar a rede mínima que interconecta $n$ pontos fixos dados em determinado espaço métrico. Para atingir tal objetivo é permitida a adição de pontos auxiliares que minimizem comprimento de ligação total da rede, que é a soma dos comprimentos da linhas conectando todos esses pontos[28][7][17]. Os pontos auxiliares adicionados são chamados de pontos de Steiner [28].

Diversas aplicações do mundo real consistem em minimizar o comprimento de ligação dos pontos de uma rede. Alguns exemplos dessas aplicações são as redes de transporte [50], as redes de televisão por cabo [33], o roteamento de integração em larga escala (*VLSI*, *Very-Large-Scale Integration*) [42][46][39], e as redes de roteamento [42]. Essas aplicações podem ser modeladas pelo problema da árvore de Steiner [37].

Diversos trabalhos tem estudado o problema de árvore de Steiner [28][8][67][63][10][65][18][7][36][6][17][74][3][44]. Um dos objetivos desses trabalhos é propor soluções mais eficientes para obter a árvore de Steiner mínima. No entanto sabe-se que obter tal resposta é um problema NP-difícil [41], portanto melhorias na eficiência computacional para redes de larga escala não são triviais.

Devido à complexidade do problema de árvore de Steiner algumas variações do mesmo têm sido propostas. Essas variações buscam limitar o problema em função da aplicação. Dentre elas o problema de Steiner em Grafos (*SPG*, *Steiner Problem in Graphs*) [21], o problema de Steiner Retilíneo (*RSTP*, *Retilinear Steiner Tree Problem*) [32][37], e o problema de árvore de Steiner Euclidiano (*ESTP*, *Euclidean Steiner Tree Problem*)[34][7][17][3].

O *ESTP* consiste em encontrar a árvore de comprimento Euclidiano mínimo, abrangendo um conjunto de pontos fixos no plano, com a adição de pontos Euclidianos auxiliares. O *ESTP* está relacionado com muitas aplicações de engenharia. Exemplos dessas são roteamento para *VLSI* [3], otimização de redes de transmissão, circuitos impressos e sistemas de ventilação, de aquecimento, de extintores de incêndio, de suprimento de água e de drenagem [23].

No entanto, as pesquisas mostram que mesmo as variações do problema árvore de Steiner são NP-difíceis [24][21][26]. No caso de problemas NP-difíceis, os algoritmos

exatos possuem em sua maioria complexidade de tempo exponencial [25].

Embora algoritmos exatos ofereçam soluções ótimas para o problema [51][10][37][72][66], devido à complexidade dos mesmos, é apropriado considerar alternativas para encontrar um algoritmo de tempo computacional aceitável para aplicações práticas [25]. Uma dessas alternativas é o uso de algoritmos heurísticos. Esses algoritmos buscam encontrar uma solução adequada para o problema em um tempo aceitável, sem se concentrar, exclusivamente, na busca pela solução ótima [25].

Dada a sua grande variedade de aplicações este trabalho terá como base o problema de árvore de Steiner Euclidiano (*ESTP*). Uma série de algoritmos heurísticos tem sido propostos para o *ESTP*. Dentre esses pode-se citar Chang (1972)[8], Thompson (1973)[67], Smith et al. (1981)[63], Du et al. (1992)[18], Beasley (1992)[7], Beasley e Goffinet (1994)[6], Dreyer e Overton (1998)[17][17], Zachariasen (1999)[74] e Laarhoven e Ohlmann (2010)[44]. Dessas heurísticas a descrita em [7] será a base do desenvolvimento desse trabalho.

Outro fator importante para o desempenho dos algoritmos é a estrutura de dados utilizada para representar as soluções. Vários problemas de otimização podem ser codificados por diferentes representações, com desempenho variando significativamente em função da codificação utilizada [59].

Uma das representações que tem mostrado resultados relevantes para codificar problemas modelados por grafos é a representação nó-profundidade-grau (*RNPG*) proposta em [16]. A *RNPG* tem sua eficiência computacional demonstrada no contexto de problemas de projeto de redes com algoritmos evolutivos. Algoritmos evolutivos são métodos de otimização inspirados na natureza que podem ser usados por muitos problemas de otimização e podem imitar princípios básicos da vida por meio da implementação computacional de mecanismos da genética como mutação, cruzamento, e seleção de uma sequência de alelos [59].

Esse trabalho propõe investigar o uso da *RNPG* na heurística de Beasley [7] para o *ESTP*. A melhoria usando a heurística de Beasley [7] possui melhor qualidade comparado a outras heurísticas como Chang (1972) [8] e Smith et al. (1981) [63][7]. Além de oferecer soluções de qualidade relativamente alta, foi avaliado que o tempo gasto para sua implementação seria menor que outras heurísticas mais recentes como Beasley e Goffinet (1994)[6] e Laarhoven e Ohlmann (2010)[44]. Por esse motivo a heurística de Beasley (1992)[7] foi considerada mais apropriada em função do principal objetivo desse trabalho, que é o de pesquisar por vantagens em termos de eficiência computacional a partir da utilização da RNPG em uma heurística não evolutiva.

Para viabilizar o uso da *RNPG* na heurística de Beasley (1992) [7] é necessário o desenvolvimento de operadores específicos. O Operador de Consulta de Conjuntos (OCC) e o Operador de Remoção de Nó (ORN) implementam as etapas principais

da heurística proposta para o *ESTP*. Como resultado, tem-se o algoritmo denominado *SMTBeasleyRNPG*. Esse algoritmo utiliza a *RNPG* para representar as árvores de Steiner Euclidianas e os operadores desenvolvidos para implementar uma variação da heurística apresentada em [7].

O objetivo principal da pesquisa desenvolvida nesse trabalho é mostrar a viabilidade do uso de uma representação eficiente para melhorar o tempo computacional de algoritmos heurísticos. Assim, além das análises empíricas da implementação proposta, este trabalho apresenta a análise da complexidade de tempo dos operadores desenvolvidos.

As análises empíricas do algoritmo *SMTBeasleyRNPG* são realizadas com os casos de teste disponíveis em [5]. Esses casos envolvem *15* instâncias de grafos Euclidianos para cada ordem, ou seja, o número de vértices. A ordem dos grafos varia de *10* a *1000*. Os resultados mostraram que o *SMTBeasleyRNPG* apresenta melhoria média aproximada de *2,92%*, bem próximo da melhoria média de Beasley (*2,95%*). Essa melhoria é calculada a partir da média aritmética do percentual de redução obtido pela solução oferecida em função da árvore mínima geradora (MST, Minimal Spanning Tree).

Com o objetivo de avaliar a eficiência da RNPG para o algoritmo *SMTBeasleyRNPG* foi realizada a análise de complexidade de tempo do OCC e do ORN. O OCC possui complexidade $O(n^2)$ no caso médio, o ORN, $O(kn)$. Nessa análise $n$ representa o número de vértices originais da árvore de entrada e $k$ o número de nós a remover no ORN. Um comparativo do OCC implementado com a representação convencional Matriz de Adjacências mostra que essa representação possui complexidade de tempo $O(n^4)$. Essa diferença de complexidade foi comprovada por meio de testes empíricos. Na melhor situação o OCC com Matriz de Adjecências é quase 500% mais lento do que o OCC com RNPG. Além disso, conforme aumenta o número de vértices dos grafos aumenta a diferença entre os algoritmos, com o OCC com Matriz de Adjacências sendo quase 2500% mais lento do que o OCC com RNPG.

O trabalho está estruturado da seguinte forma. O Capítulo 2 apresenta o problema da árvore de Steiner e algumas variações. No Capítulo 3 são descritas as principais heurísticas encontradas na literatura para o *ESTP*. O Capítulo 4 descreve algumas das principais representações existentes para problemas de projeto de redes, dentre elas a *RNPG*. No Capítulo 5 é apresentado o algoritmo *SMTBeasleyRNPG* como uma variação do algoritmo de Beasley(1992) [7], utilizando uma versão da *RNPG* definida especificamente para esse algoritmo. Apresenta também a análise de complexidade. O Capítulo 6 apresenta os resultados obtidos nos casos de teste. Considerações finais são apresentadas no Capítulo 7.

CAPÍTULO 2

# Problema de Árvore de Steiner

O problema de Steiner consiste em conectar um conjunto de pontos fixos no plano com custo mínimo. Para isso pontos extras podem ser adicionados de forma a minimizar o custo total da rede, calculado pela soma dos comprimentos da linhas conectando todos esses pontos. Esse problema surgiu no trabalho de Steiner, conforme descreve Courant e Robbins (1969) [12], que apresentaram o problema de unir três vilarejos com uma rede viária de comprimento total mínimo [67].

Formalmente, o problema pode ser descrito como segue. Três pontos $A$, $B$ e $C$ são dados em um plano, então, um quarto ponto $P$ no plano deve ser encontrado, de modo que a soma $a+b+c$ seja mínima, sendo $a$, $b$ e $c$ as distâncias entre $P$ e os pontos $A$, $B$ e $C$ respectivamente [12]. Esse ponto $P$ é único para o triângulo formado por $A$, $B$ e $C$ [51].

Para obter a resposta do problema, deve-se verificar se no triângulo $ABC$ todos os ângulos são menores do que $120°$, então $P$ é o ponto a partir do qual quaisquer dois vértices do triângulo $ABC$ formam em $P$ um ângulo de $120°$ (Figura 2.1). Caso contrário, o vértice no qual o ângulo for igual ou maior que $120°$ será o ponto $P$ [12][51].

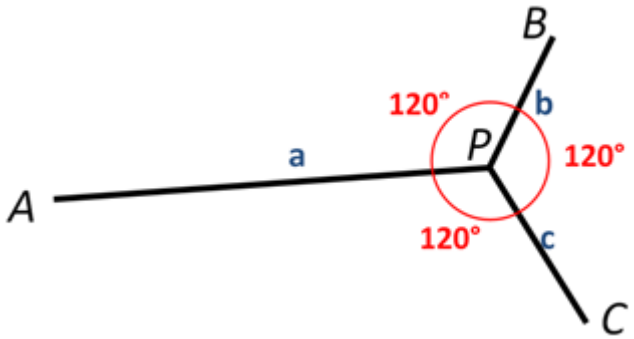


**Figura 2.1:** *Ponto P, que subtende um ângulo de* $120°$ *para cada um dos vértices.*

O ponto $P$ pode ser encontrado pela seguinte construção: escolher um lado no triângulo $ABC$. Considerar um terceiro vértice $X$ compondo um triângulo equilátero com o lado escolhido, cujo interior não se sobreponha ao triângulo formado por $ABC$. Depois disso, construir um círculo que perpasse os vértices desse triângulo equilátero. O ponto $P$ é a intersecção entre esse círculo e o segmento de reta que vai do vértice $X$ ao vértice que

não faz parte do triângulo equilátero [51]. A Figura 2.2 mostra essa construção a partir da escolha do lado $BC$.

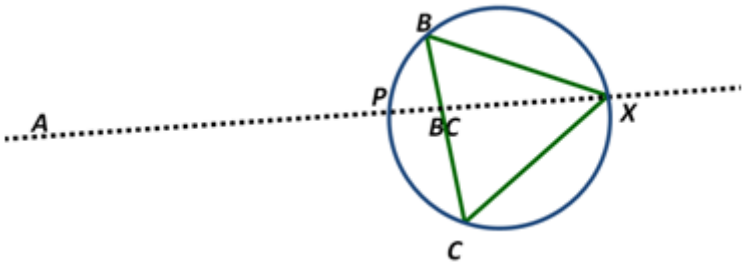


**Figura 2.2:** *Obtenção do ponto P a partir do lado BC.*

A origem do problema de Steiner é frequentemente investigada até Fermat (1601-1665)[51][28][37]. Fermat propôs o problema de encontrar no plano um ponto cuja soma das distâncias de três pontos dados é mínima. Torricelli propôs uma solução para esse problema por volta de 1640. Essa solução afirma que os três círculos que circunscrevem os triângulos equiláteros construídos a partir dos lados e para fora do triângulo formado pelos três pontos têm intersecção no ponto procurado [37].

A Figura 2.3 exemplifica a solução proposta por Torricelli. Nela os pontos $A$, $B$ e $C$ são os pontos dados e $P$ é o ponto procurado.

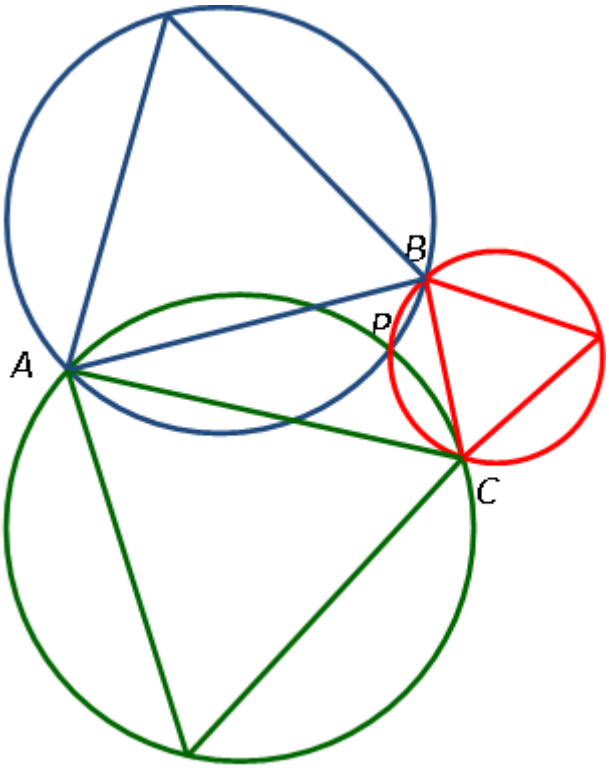


**Figura 2.3:** *Ponto P encontrado pela intersecção dos círculos dos triângulos.*

Cavalieri em 1647 mostrou que os segmentos de linha criados a partir dos três pontos dados para o ponto de Torricelli formam um ângulo de 120° entre si [37]. Simpson

em 1750 provou que os três segmentos de reta que ligam o vértice externo dos triângulos equiláteros aos respectivos vértices opostos do triângulo dado também se intersectam no ponto de Torricelli [37].

A Figura 2.4 exemplifica a solução proposta por Simpson. Nela os pontos $A$, $B$ e $C$ são os pontos dados, $\bar{A}$,$\bar{B}$ e $\bar{C}$ são os vértices externos dos triângulos equiláteros e $P$ é o ponto procurado.

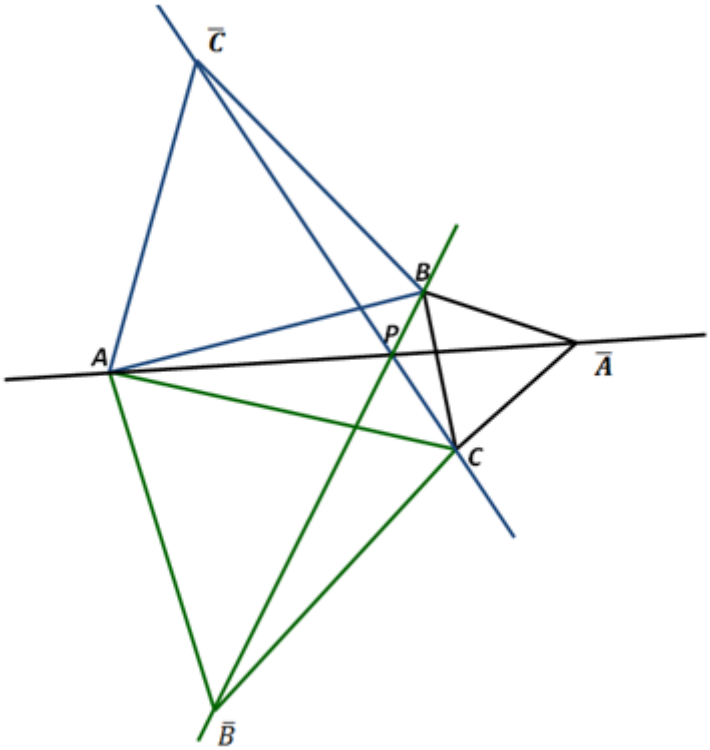


**Figura 2.4:** *Ponto P encontrado pela intersecção retas.*

Expandindo para $n = 4$, ou seja, uma rede composta por quatro pontos $A$, $B$, $C$ e $D$, a árvore de comprimento mínimo pode ter $s = 0$, $s = 1$ ou $s = 2$ pontos de Steiner [52]. No caso, $s = 0$ todos os vértices da árvore podem ser conectados por meio da árvore mínima geradora (*MST*, *Minimal Spanning Tree*). Árvore geradora é uma árvore que contém todos os vértices de um conjunto ou grafo. A MST é a rede de conexões diretas entre vértices (terminais) que tem o menor comprimento total possível, ou seja, a menor soma dos comprimentos das conexões [55]. Quando $s = 1$ três dos vértices são conectados a um ponto de Steiner e o vértice restante é conectado ao vértice mais próximo. No último caso, $s = 2$, exemplificado na Figura 2.5, são encontrados os dois pontos de Steiner $P_1$ e $P_2$ baseando-se nos triângulos equiláteros formados a partir de pares dos pontos da rede original.

Primeiro, localiza-se os pontos $P_{AB}$ e $P_{CD}$ a partir dos triângulos equiláteros dos pares $AB$ e $CD$, respectivamente, e esses pontos são conectados por um segmento de reta.

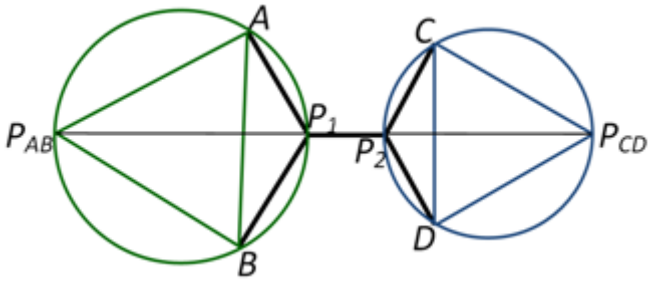


**Figura 2.5:** *Pontos $P_1$ e $P_2$ baseados nos triângulos equiláteros dos conjuntos de 2 pontos de Steiner.*

Então os pontos de Steiner $P_1$ e $P_2$ são localizados a partir da interseção entre o segmento de reta $P_{AB}$ $P_{CD}$ e os círculos apresentados na Figura 2.5 [52].

Para cada lado $AB$ e $CD$ existem dois triângulos equiláteros e, portanto, duas possibilidades para $P_{AB}$ e $P_{CD}$. O ponto é escolhido de modo que o equilátero seja projetado para o lado contrário aos pontos do conjunto inicial. Outra escolha necessária é a da topologia, que define as conexões entre os pontos. Conforme exemplificado na Figura 2.6 há duas possibilidades, e a escolha depende do comprimento final de cada uma delas.

O problema de Steiner inicial considerava que apenas três pontos são dados. No entanto, para encontrar uma extensão realmente significativa do problema, deve-se generalizar para o caso de $n$ pontos, onde $n$ é um número inteiro positivo fixo, geralmente significativamente maior que 3, e procurar a rede de conexão de comprimento total mais curto [12].

O problema de árvore de Steiner (*STP*, *Steiner Tree Problem*) busca encontrar a rede mínima que interconecta $n$ pontos dados, por meio da adição de pontos auxiliares para minimizar o comprimento total [28][7][17]. Um vértice extra que é adicionado para uma árvore para reduzir seu comprimento, ou seja reduzir seu custo total de conexão, que na Figura 2.2 corresponde ao ponto $P$, é chamado ponto de Steiner [28].

A rede mínima de conexão que é a solução do problema, composta pelo conjunto $N$ com os $n$ pontos dados e o conjunto $S$ com $s$ pontos de Steiner, é chamada árvore mínima de Steiner (*SMT*, *Steiner Minimal Tree*) [28]. Em uma *SMT*, todo ponto de Steiner deve ter exatamente três conexões. Se houver menos que três conexões esse ponto pode ser removido do conjunto $S$, e uma conexão pode ser feita diretamente entre os pontos adjacentes. Se houver mais que três conexões, há ângulos menores que $120^\circ$ incidentes nesse ponto, nesse caso podem ser adicionados pontos em $S$ adjacentes a esse ponto minimizando a rede ainda mais. Pelo mesmo motivo, qualquer ponto do conjunto $N$ terá no máximo 3 conexões. Devido a essas condições, o tamanho máximo do conjunto $S$ é $n$-2 pontos [51][28].

Uma rede de conexão $T$ é chamada árvore de Steiner se satisfaz três condições

[36]:

1. $T$ é uma árvore;
2. quaisquer duas arestas de $T$ se encontram em um ângulo de no mínimo 120°;
3. um ponto de Steiner não pode ser de grau 1 ou 2.

Independente do tamanho de $N$ existe uma sequência finita de construções Euclidianas produzindo todas as árvores mínimas de Steiner [51]. O algoritmo proposto por Melzak (1961) [51] oferece a solução ótima porque procura a melhor solução dentre todas essas árvores. A melhor solução depende da disposição dos pontos dados e das conexões entre esses pontos e os pontos de Steiner [28].

A topologia de uma árvore define a matriz de conexão dos pontos de $N$ e $S$. Uma árvore de Steiner mínima para determinada topologia é chamada árvore de Steiner relativamente mínima (*RMST*, *Relatively Minimal Steiner Tree*). Uma topologia é cheia se o número de pontos de Steiner é $s=n-2$ [28]. Uma árvore é nomeada árvore de Steiner cheia (*FST*, *Full Steiner Tree*) quando é a *RMST* para determinada topologia cheia [28].

Um exemplo frequente na literatura [12][28] mostra como a topologia pode impactar no comprimento total da rede. Esse exemplo é exibido na Figura 2.6 que contém duas *FSTs* possíveis para uma rede de conexão com $n=4$ pontos ($A$, $B$, $C$ e $D$) e $s=2$ pontos de Steiner ($E$ e $F$). Embora as árvores sejam relativamente mínimas (*RMSTs*), a árvore da Figura 2.6(b) tem comprimento mínimo total menor do que a árvore da Figura 2.6(a).

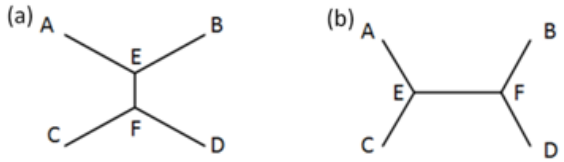


**Figura 2.6:** *Topologias de Steiner diferentes para os mesmos pontos.*

A árvore mínima de Steiner pode ser obtida pela construção de uma *RMST* para cada topologia [28]. No entanto, esse processo é inviável, visto que o número de topologias de Steiner cresce exponencialmente com os valores de $n$ e $s$ [28]. Assim, a formulação do problema de Steiner como um problema de otimização é NP-difícil [41][24][26][25]. Portanto, é necessário buscar alternativas para oferecer soluções adequadas em tempo computacional aceitável.

De maneira prática, é preciso definir questões como generalizações e representações a fim de limitar o problema e o tempo de solução necessário. Uma dessas questões são as diversas versões para o problema de Steiner. Das quais destaca-se o problema de Steiner em grafos (*SPG*, *Steiner Problem in Graphs*)[21], o problema de árvore de Steiner retilíneo (*RSTP*, *Rectilinear Steiner Tree Problem*) [32][37] e o problema de árvore de Steiner Euclidiano (*ESTP*, *Euclidean Steiner Tree Problem*) [34][7][17][3].

A seguir, é apresentada uma breve descrição dessas versões. Como o foco deste trabalho é o problema de árvore de Steiner Euclidiano, para essa versão será apresentada uma descrição mais detalhada.

## 2.1 Problema de árvore de Steiner Euclidiano

O problema de árvore de Steiner Euclidiano (*ESTP*, *Euclidean Steiner Tree Problem*) consiste em encontrar a árvore de comprimento Euclidiano mínimo, abrangendo um conjunto de pontos fixos no plano, com a adição de pontos de Steiner [17][7]. Esse problema é definido para o plano Euclidiano [37]. Dados dois pontos $a(x_1, y_1)$ e $b(x_2, y_2)$ a distância Euclidiana é:

$$d(a,b) = \sqrt{((x_1 - x_2)^2 + (y_1 - y_2)^2)}. \tag{2-1}$$

Considerando $N$ um conjunto de $n$ pontos no plano Euclidiano. A solução para o *ESTP* é obtida por meio da árvore geradora mínima (*MST*, *Minimal Spanning Tree*) de $N \cup S$, em que $S$ é um conjunto de pontos de Steiner [7]. Cada um dos pontos em $N \cup S$ é composto por suas coordenadas $p(x, y)$.

Formalmente o comprimento Euclidiano total de uma *MST* $T$ pode ser definido da seguinte forma. Seja $W$ uma matriz de comprimentos entre vértices, com $w_{i,j}$ sendo a distância Euclidiana entre os vértices $v_i$,$v_j$ quando existe aresta conectando esses vértices. O comprimento Euclidiano total de $T$ é:

$$\sum_{(i,j)\in T} w_{v_i,v_j} \tag{2-2}$$

### 2.1.1 Exemplo de *SMT* obtida para o *ESTP*

A seguir é demonstrado um exemplo dos pontos de Steiner obtidos para uma árvore mínima de Steiner (*SMT*, *Steiner Minimal Tree*) no *ESTP*. O exemplo corresponde ao segundo caso de teste de tamanho $n = 10$ disponível em [5]. A Tabela 2.1 apresenta as coordenadas dos dez pontos do grafo utilizado.

A Figura 2.7 apresenta a *MST* do grafo exemplo. O peso das arestas é o valor da distância Euclidiana conforme equação 2-1. A *MST* para conectar esses pontos tem custo linear total de $1,614569662$.

O custo linear total da *SMT* para conectar esses pontos é de $1,606868229$. Esse custo é obtido por meio da inclusão de 2 pontos de Steiner. A Tabela 2.2 apresenta as coordenadas dos pontos 10 e 11 que correspondem aos dois pontos de Steiner adicionados para obter a solução ótima.

**Tabela 2.1:** *Coordenadas (x,y) do ESTP para* $n = 10$.

| *Ponto* | x | y |
|---|---|---|
| *0* | 0.1470158 | 0.6131368 |
| *1* | 0.1251936 | 0.2125058 |
| *2* | 0.7411042 | 0.2036110 |
| *3* | 0.8044316 | 0.4252316 |
| *4* | 0.2116787 | 0.2721307 |
| *5* | 0.6624317 | 0.7419167 |
| *6* | 0.8043960 | 0.5698598 |
| *7* | 0.6684968 | 0.2824909 |
| *8* | 0.4043257 | 0.1895910 |
| *9* | 0.4978816 | 0.2117592 |

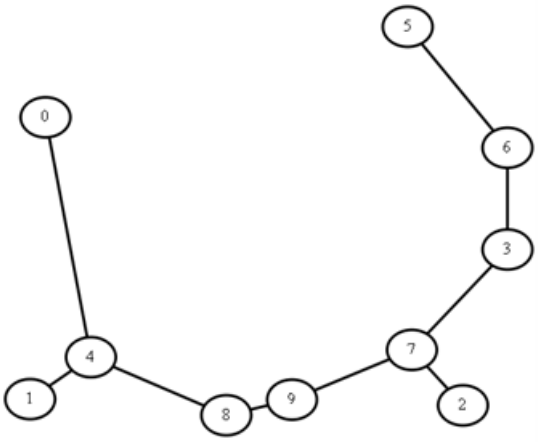


**Figura 2.7:** *MST para os 10 pontos da Tabela 2.1.*

A *SMT* é apresentada na Figura 2.8. Nessa Figura os pontos representados por triângulos são os dois pontos de Steiner adicionados.

Os pontos de Steiner adicionados podem ser obtidas pela aplicação do método proposto por Melzak (1961) [51]. Esse método tem como base a aplicação das equações de Torricelli e Simpson. O primeiro ponto de Steiner adicionado é o de número 10. Esse ponto é utilizado para conectar os pontos 0, 4, e 1. A seguir é demonstrado como o ponto 10 foi obtido, conforme mostra a Figura 2.2.

O primeiro passo é escolher aleatoriamente dois dos três vértices para os quais se deseja obter o ponto de Steiner. Nesse exemplo, foram escolhidos os vértices 4 e 1, aqui nomeados $v_4$ e $v_1$, respectivamente. As coordenadas que serão utilizadas para esses vértices são as apresentadas na Tabela 2.1.

A seguir, deve-se encontrar o terceiro ponto $v_{eq}$ de um triângulo equilátero $Eq$ formado pelos pontos escolhidos ($v_4$ e $v_1$). Dois pontos são possíveis para compor $Eq$. O ponto escolhido $v_{eq}$ deve ser o ponto oposto ao triângulo formado por $v_0$, $v_4$ e $v_1$. Considere $R_1$ a reta que passa por $v_1$ e $v_4$, e $v_{med}$ o ponto médio entre esses pontos. A Figura 2.9 mostra esses pontos e a reta $R_1$.

**Tabela 2.2:** *Coordenadas (x,y) dos dois pontos de Steiner adicionados para reduzir o custo de ligação.*

| *Ponto* | x | y |
|---|---|---|
| *10* | 0.2023310953 | 0.2756612762 |
| *11* | 0.7046908268 | 0.2796910899 |

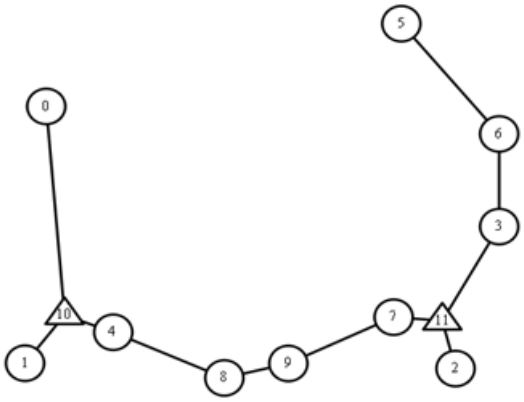


**Figura 2.8:** *SMT do ESTP com a adição de dois pontos de Steiner.*

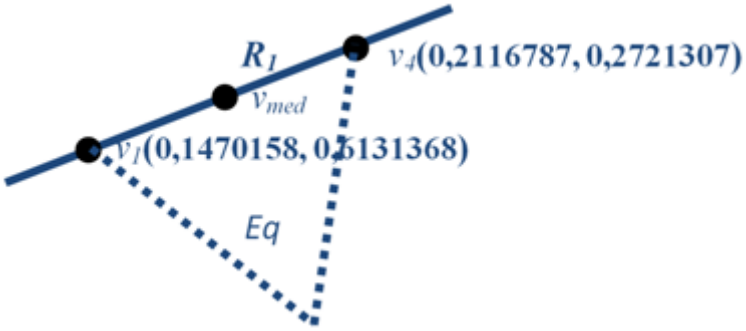


**Figura 2.9:** *A reta $R_1$ que cruza os pontos $v_1$ e $v_4$.*

Considere $R_2$ a reta perpendicular a $R_1$ que passa pelo ponto $v_{med}$. Para obter os possíveis pontos de *Eq*, utiliza-se a intersecção entre a circunferência *Ci* que circunscreve *Eq* e a reta $R_2$. A Figura 2.10 mostra as retas $R_1$ e $R_2$.

A Equação 2-3 apresenta como obter o coeficiente angular $m$ da reta a partir da equação geral de uma reta. Considere $(x_0, y_0)$ e $(x_1, y_1)$ são as coordenadas de dois pontos quaisquer cruzados pela reta.

$$m = \frac{y_0 - y_1}{x_0 - x_1} \tag{2-3}$$

Aplicando a Equação 2-3 aos pontos $v_1$ e $v_4$ obtem-se o valor de $m = 0,689423859$ para a reta $R_1$. As coordenadas de $v_{med} = (x_{med}, y_{med})$ são obtidas por $x_{med} = (x_4 + x_1)/2 = 0,16843614$ e $y_{med} = (y_4 + y_1)/2 = 0,242318243$.

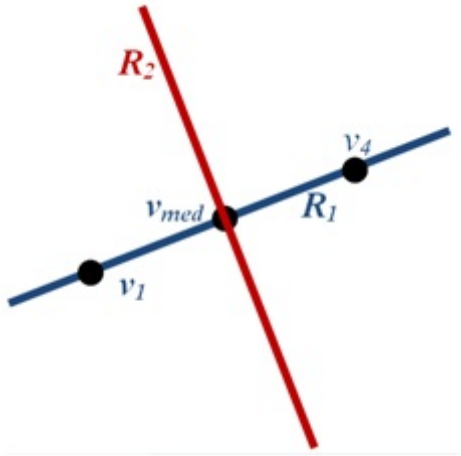


**Figura 2.10:** *A reta $R_2$ perpendicular a $R_1$ que passa pelo ponto $v_{med}$.*

A Equação 2-4 apresenta a equação da reta perpendicular $R_2$.

$$b = \frac{x}{m} + y \tag{2-4}$$

Para obter o valor de $b$ utiliza-se as coordenadas do ponto médio $v_{med}$, e o valor de $m$ calculado para $R1$, na equação 2-4. Assim, $b = 0,486632586$. Em seguida deve-se encontrar a circunferência que circunscreve $Eq$. Para tal utiliza-se a equação geral da circunferência descrita na Equação 2-5. Considere, $P_1 = (x_1, y_1)$ um ponto qualquer incidido pela circunferência e $v_c = (x_c, y_c)$ o ponto central da circunferência de raio $r$.

$$(x_1 - x_c)^2 + (y_1 - y_c)^2 = r^2 \tag{2-5}$$

O raio da circunferência desejada pode ser obtido em função do lado do triângulo equilátero circunscrito pela mesma, conforme mostra a equação 2-6. Considere $r$ o raio da circunferência e $l$ o comprimento do lado do triângulo.

$$r = \frac{\sqrt{3}}{3} l \tag{2-6}$$

O valor de $l$ para o triângulo $Eq$ é obtido pela distância Euclidiana entre os pontos $v_4$ e $v_1$, $l = 0,10504666$ e o valor de $r$ obtido em função de $l$ é $r = 0,060648717$. A seguir, subtituí-se na Equação 2-5 da circunferência o ponto $P_1$ por $v_4$ ou $v_1$. Assim, a reta $R_2$ intersecta $Ci$ no ponto $v_c$. Dois pontos são possíveis para $v_c$, cada um desses pontos corresponde a um dos triângulos equiláteros possíveis para $Eq$. Como resultado da intersecção entre $R_2$ e $Ci$ tem-se $x'_c = 0,185648412$ e $x''_c = 0,151223868$, os dois possíveis valores de $x_c$.

Novamente, a partir da equação da reta $R_2$, obtem-se os possíveis valores ($y'_c$ e

$y_c''$) de $y_c$ para $v_c$, esses valores são $y_c' = 0,217352077$ e $y_c'' = 0,267284393$. Esses pontos se referem aos pontos centrais de duas circunferências, cada uma delas circunscreve um dos dois triângulos equiláteros possíveis em conjunto com $v_4$ e $v_1$. A equação 2-7 mostra as coordenadas escolhidas para $v_c$, que deve ser o ponto oposto ao triângulo formado por $v_0$, $v_4$ e $v_1$, ou seja está oposto ao vértice $v_0$.

$$\begin{aligned} x_c &= 0,185648412 \\ y_c &= 0,217352077 \end{aligned} \tag{2-7}$$

A Figura 2.11 mostra a circunferência *Ci* baseada no raio *r*, no triângulo *Eq* e no ponto $v_c$ cruzado pela reta $R_2$.

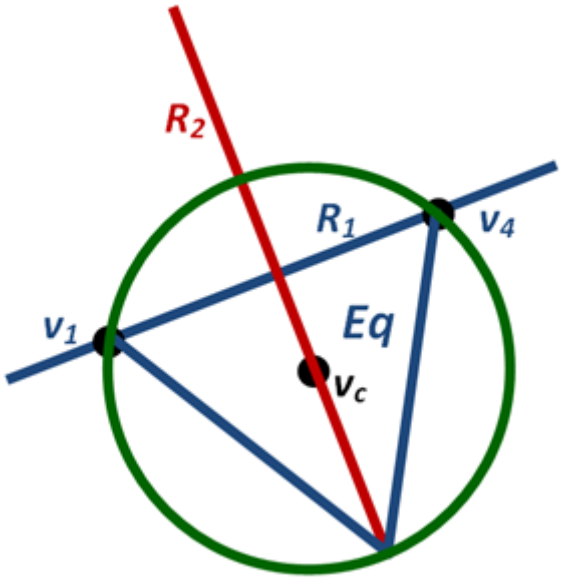


**Figura 2.11:** *A circunferência Ci, o triângulo Eq, e o ponto $v_c$.*

O mesmo processo de interseção entre $R_2$ e *Ci* pode ser reaplicado substituindo na equação da circunferência o ponto $v_c$. Como resultado obtém-se os 2 possíveis pontos para $v_{eq}$. No entanto, somente um desses pontos corresponde às coordenadas de $v_{eq}$. O ponto descartado coincide com o ponto central da circunferência que foi ignorada e está do mesmo lado do triângulo formado por $v_0$, $v_4$ e $v_1$. A equação 2-8 mostra o ponto $v_{eq}$.

$$\begin{aligned} x_{eq} &= 0,220072791 \\ y_{eq} &= 0,16741997 \end{aligned} \tag{2-8}$$

Por meio dessa técnica de intersecção de pontos entre uma reta e a circunferência é possível encontrar o ponto de Steiner $P = (x_{st}, y_{st})$. Nesse caso, conforme exemplificado na Figura 2.2, a intersecção entre a circunferência *Ci* e a reta $R_3$ que cruza os pontos $v_{eq}$ e $v_0$. A constante *m* da equação da reta $R_3$ é $m = -6,10094699$. O ponto de Steiner obtido

pela interseção entre a circunferência *Ci* e a reta $R_3$ é mostrado na Equação 2-9. O ponto 11, que conecta os pontos 7, 2 e 3, pode ser obtido da mesma forma que o ponto 10.

$$\begin{aligned} x_{st} &= 0,2023310953 \\ y_{st} &= 0,2756612762 \end{aligned} \tag{2-9}$$

O problema de Steiner Euclidiano está relacionado com muitas aplicações de engenharia, como redes ou roteamento para integração em larga escala (*VLSI*, *Very-Large-Scale Integration*) [3] e projetos de redes de transmissão [23]. Outras aplicações incluem circuitos impressos e sistemas de ventilação, de aquecimento, de extintores de incêndio, de suprimento de água e de drenagem [23].

## 2.2 Problema de Steiner em Grafos

Considere um grafo não direcionado *G = (H, I)* com pesos positivos associados às arestas e um subconjunto $J \subseteq H$ não vazio. $H$ é o conjunto de vértices e $I$ é o conjunto de arestas de $G$. O problema de Steiner em grafos (*SPG*, *Steiner Problem in Graphs*) consiste em encontrar um subgrafo de $G$ que perpasse todos os vértices do subconjunto $J$ e tem o mínimo peso total de arestas. A solução mínima para o problema é uma árvore $T$, que pode ter vértices em $H - J$, os pontos de Steiner [21].

No *SPG* os pontos de Steiner são escolhidos em um conjunto finito de vértices $L = H - J$ ou seja $L = \{X | x \in H \text{ e } x \notin J\}$. Os pontos de Steiner escolhidos compõem um conjunto $S \in L$. A árvore $T$, solução do *SPG*, é a *MST* dos vértices $J \cup S$ [37][21].

O *SPG* tem uma larga variedade de aplicações. Um exemplo é a localização de instalações tais como em centros de comutação telefônica [21]. Cada vértice representa um possível local para um dos centros e as arestas representam os possíveis cabos e seus custos associados. Os vértices especiais representam os locais que devem ser conectados da maneira mais econômica possível. Outras aplicações do SPG variam desde o layout de placas de circuito impresso até a construção de árvores filogenéticas [21][22].

## 2.3 Problema de Steiner Retilíneo

Assim como em outras versões, o problema de árvore de Steiner retilíneo (*RSTP*,*Retilinear Steiner Tree Problem*) básico tem $n$ pontos localizados no plano, e pontos de Steiner adicionais podem ser usados na construção da árvore de comprimento mínimo. A distância métrica é a retilínea, isto é, a distância entre $a$ para $b$ é dada pela

Equação 2-10.

$$d(a,b) = |x_a - x_b| + |y_a - y_b|. \tag{2-10}$$

O comprimento esperado da árvore de Steiner mínima retilínea pode ser delimitado por um limite próximo ao do comprimento da árvore geradora mínima retilínea [36].

Formalmente, um segmento retilíneo é um segmento horizontal ou vertical de reta conectando seus dois terminais no plano. Uma árvore retilínea é uma coleção de segmentos conectados acíclicos que intersectam apenas em seus terminais. Uma árvore de Steiner retilínea (*RST*, *Rectilinear Steiner Tree*) para um conjunto dado de terminais é uma árvore retilínea em que cada terminal é o ponto final de um segmento na árvore. O comprimento da árvore é a soma dos comprimentos dos segmentos e a árvore mínima de steiner retilínea (*RSMT*, *Rectilinear Steiner Minimal Tree*) é a árvore de comprimento mínimo [37].

Uma *RSMT* possui as seguintes propriedades: qualquer ponto tem grau menor ou igual a 4; se um ponto com grau 2 tem segmentos incidentes colineares então o ponto é um terminal; um ponto com grau maior ou igual a 3 que não é um terminal é chamado ponto de Steiner; qualquer outro ponto que não seja um terminal, nem um ponto de Steiner, é denominado ponto de canto, que tem grau 2 e conecta dois segmentos não colineares [37].

Os resultados do problema de Steiner para a métrica Euclidiana são intimamente relacionados à geometria e não se estendem à métrica retilínea. O *RSTP* está mais intimamente relacionado ao *SPG* do que ao *ESTP* [57]. Como explicado a seguir, o *RSTP* pode ser modelado a partir de um conjunto finito, incluindo outros tipos de pontos além dos pontos terminais originais.

Hanan (1966)[32] estabeleceu um resultado que Richards (1989) [57] chamou de "dimensão de redução". Considere $N$ o conjunto de $n$ terminais no plano. Perpasse linhas horizontais e verticais através de cada um destes pontos. Considere o grafo *G(H, I)*, em que $H$ é o conjunto compreendendo os $n$ terminais adicionados às intersecções entre essas linhas horizontais e verticais. A partir dessa construção, existirá uma aresta entre dois vértices se eles estiverem diretamente conectados por uma linha, horizontalmente ou verticalmente. Hanan (1966)[32] mostrou que a *RST* está contida em $G$, isto é, os segmentos da árvore são compostos por arestas de $G$ [32][57]. Devido ao teorema de Hanan segue-se que qualquer algoritmo para o *SPG* pode ser utilizado para resolver o *RSTP* [57].

A atenção no *RSTP* tem como uma das principais motivações o potencial em aplicações de projetos de circuitos e layout de fios em placas de circuito impresso [26][57][32]. O *RSTP* tem várias aplicações práticas como projeto *VLSI* [36], síntese física, planejamento e posicionamento de interconexões, estimativa de fios de carga, congestionamento de rotas e atrasos de sistemas de interconexão [71].

CAPÍTULO 3

# Heurísticas para o Problema de Steiner Euclidiano

Sabe-se que o Problema de árvore de Steiner Euclidiano (*ESTP*, *Euclidean Steiner Tree Problem*) é NP-difícil [24][25][37]. A intratabilidade inerente do *ESTP* encoraja o desenvolvimento de heurísticas, que são projetadas para encontrar boas conexões de pontos em tempos que são, geralmente, polinômios de baixa ordem em função do tamanho do problema [31].

Hwang (1992) [37] define uma heurística eficaz para o *ESTP* como um algoritmo que pode rodar em tempo proporcional a um baixo grau polinomial de $n$, o número de vértices da instância do problema. Essa heurística deve garantir que o comprimento da árvore de saída não exceda em grande quantidade o de uma árvore mínima de Steiner (*SMT*, *Steiner Minimal Tree*).

A árvore geradora mínima (*MST*, *Minimal Spanning Tree*) é uma solução heurística eficaz para o problema, pois existem algoritmos de complexidade computacional $O(n \log n)$ para obte-la. Além disso, essa árvore nunca excede em *15,5%* o comprimento da *SMT* [37][31]. A *MST* torna-se naturalmente o padrão de comparação de outras heurísticas. Muitas soluções heurísticas iniciam com uma *MST* dada e a partir dessa, buscam por soluções melhores, ou utilizam determinado algoritmo da *MST* durante sua execução [37].

Gilbert e Pollack (1968) [28] definiram *"Steiner Ratio"* como a relação de desempenho da *MST* e conjecturaram que essa relação é igual a $\frac{\sqrt{3}}{2}$ [37][28]. Embora o desempenho de heurísticas baseadas em *MST* sejam aproximamente relacionadas à *"Steiner Ratio"*, a maioria das heurísticas propostas possui relação de desempenho melhor [37].

A Seção 3.1 descreve algumas das principais heurísticas presentes na literatura baseadas na *MST*: as heurísticas de Chang (1972) [8], Thompson (1973) [67], Dreyer e Overton (1998) [17] e Beasley (1992) [7]. A Seção 3.2 apresenta os métodos Diagrama de Voronoi e Triangulação de Delaunay e alguns algoritmos que utilizam esses métodos: Shamos e Hoey (1975) [62], Smith et al. (1981) [63], Beasley e Goffinet (1994) [6]. A

Seção 3.3 menciona os trabalhos de Lundy (1985) [49], Barreiros (2003) [3] e Frommer e Golden (2007) [23] que utilizam outros métodos heurísticos para o *ESTP* como o *annealing* e algoritmos genéticos.

## 3.1 Heurísticas baseadas na *MST*

A maioria das heurísticas para o *ESTP* executa uma série de melhorias iterativas usando a *MST* como solução inicial [73]. Os primeiros algoritmos heurísticos para o *ESTP*, propostos por Chang (1972) [8] e Thompson (1973) [67], se baseiam na inserção iterativa de vértices de Steiner em uma *MST* [36][74][67][8].

### 3.1.1 Heurísticas de Chang, Thompson e Dreyer e Overton

Thompson (1973) [67] sugeriu um processo sequencial executado para conjuntos de três vértices com arestas da *MST* que estão em um ângulo menor que $120^\circ$. Enquanto houver arestas violando a restrição de ângulo executa-se os seguintes passos: 1) insere um vértice de Steiner para o triângulo formado pelos três vértices; 2) conecta os três vértices ao vértice de Steiner inserido; 3) por fim, remove as arestas que antes conectavam os vértices na *MST* [67][44].

Chang (1972) [8] sugeriu um esquema parecido, porém com um método diferente e mais complexo de encontrar conjuntos de três vértices. No início $T$ é a *MST* e no fim é a árvore heurística de saída. $T$ é composta por subárvores com topologias cheias (FSTs), chamados componentes cheios. A inserção de vértices de Steiner é feita em cada passo de modo a maximizar a diferença de comprimento entre a árvore atual e a árvore modificada. Para isso os três vértices envolvidos ($V_i$, $V_j$ e $V_k$) devem ser selecionados em uma das seguintes possíveis formas [8][37]:

- $V_i$, $V_j$ e $V_k$ são todos terminais e pertencem a diferentes componentes cheios da árvore;
- $V_i$ e $V_j$ são terminais e pertencem ao mesmo componente cheio e $V_k$ é um terminal em um componente cheio diferente;
- $V_i$ e $V_j$ são vértices de Steiner em diferentes componentes cheios e $V_k$ é um terminal adjacente para $V_i$ e $V_j$.

A heurística proposta por Chang (1972) [8] tem complexidade de tempo $O(n^4)$. Beasley (1992) [7] e Dreyer e Overton (1998) [17] também desenvolveram heurísticas baseadas em inserir vértices e melhorar a *MST* efetuando otimizações locais.

Dreyer e Overton (1998) [17] sugeriram duas heurísticas, a primeira, denominada *"Steiner Insertion Heuristic"*, é uma extensão da heurística de Thompson (1973)

[67] que insere vértices de Steiner entre os menores ângulos de arestas da *MST* e depois obtém a árvore relativamente mínima, encontrando a posição ideal dos vértices de Steiner [17]. No pior caso essa heurística possui complexidade de tempo de $O(n^3)$ [17]. A segunda inicia encontrando a árvore de Steiner para três vértices escolhidos e em seguida executa iterações para conectar cada vértice fixo nessa árvore a partir de um novo vértice de Steiner e encontra a árvore mínima relativa [17]. Essa heurística possui complexidade $\Theta(n^2)$.

### 3.1.2 Heurística de Beasley

Beasley (1992) [7] apresenta uma heurística para o *ESTP* e para o problema de árvore de Steiner Retilíneo (*RSTP*, *Rectilinear Steiner Tree Problem*) baseada em encontrar soluções ótimas de Steiner para subgrafos da *MST*. A heurística considera todos subgrafos da *MST* que contêm quatro vértices. Para cada um deles obtém a *SMT* e adiciona ao conjunto de vértices $V$ os vértices de Steiner desses subgrafos que resultam na maior redução de custo [7].

Essa heurística consiste basicamente nos passos descritos no Algoritmo 3.1. Considere $V$ um conjunto de vértices mantido pelo algoritmo. $V$ é inicializado a partir dos vértices dados e, ao final da heurística contém os vértices iniciais mais os vértices de Steiner adicionados. A *MST* do conjunto $V$ é a solução oferecida pela heurística.

**Algoritmo 3.1:** Algoritmo de Beasley

**Entrada**: O conjunto de vértices iniciais $V$

**Saída**: O novo conjunto $V$ contendo os vértices iniciais mais os vértices de Steiner $S$ adicionados, e a *MST* que é a solução do problema.

1 Considere: $T(K)$ o custo de conectar $K$ vértices por meio de sua *MST*; $T_s(K)$ o custo de conectar os mesmos $K$ vértices, por meio de sua *SMT*; $V_0$ o conjunto inicial de vértices e $t$ o contador de iteração. $V \leftarrow V_0$; $t \leftarrow 0$;

2 Encontre a *MST* de $V$. $t \leftarrow t+1$;

3 Enumere o conjunto $L$ com todas as subárvores de quatro vértices $K$ na *MST*;

4 Para cada $K \in L$ calcule a redução de custo $R(K)$ que ocorre quando os vértices de $K$ são conectados via *SMT*;

5 Se $max[R(K)|K \in L] = 0$ pare;

6 Seja $M$ um conjunto de vértices vazio a cada nova iteração. Ordene de forma decrescente em função de $R(K)$ os conjuntos de vértices $K \in L$. Neste conjunto ordenado, para cada $K$, se $|M \cap K| = 0$ e $R(K) > 0$ então adicione os vértices de Steiner associados para $V$ e atribua $M \leftarrow M \cup K$;

7 Encontre a *MST* de $V$ e remova qualquer vértice de $V$ que represente um vértice de Steiner com grau menor ou igual a dois;

8 Para cada vértice de Steiner com grau 3, mova esse vértice para sua localização ótima. Repita esse processo até que não seja possível nenhuma redução de custo;

9 Encontre a *MST* de $V$ e para cada subárvore ($V$* de custo $C$*) que tem topologia apropriada para uma árvore de Steiner cheia (*FST*, *Full Steiner Tree*): aplique o algoritmo de Hwang [35] para encontrar a *FST* de $V$*; Se o custo da *FST* é menor que $C$* mova os vértices de Steiner para as localizações dadas pela *FST*; Se o custo da *FST* é maior que $C$* tente uma redução de custo removendo vértices de Steiner que estejam aumentando o custo de $V$*;

10 Volte ao passo (2);

Beasley (1992) [7] exibe resultados computacionais que indicam que sua heurística oferece soluções de melhor qualidade comparado a outras heurísticas como Chang (1972) [8] e Smith et al. (1981) [63][7]. A melhoria usando essa heurística é normalmente de 3 a 4% [36]. Beasley (1992) [7] sugere que a complexidade de tempo dessa heurística seja $O(n^{1,317})$, calculada empiricamente pela média dos tempos computacionais requeridos.

### 3.1.3 Deficiências em Estruturas Globais

As heurísticas de Chang (1972) [8], Thompson (1973) [67] e Beasley(1992) [7] têm a ideia de converter uma *MST* em árvores de Steiner relativamente mínimas (*RMST*, *Relatively Minimal Steiner Tree*) por meio de uma sequência de *"local Steinerizations"*. Esse é o nome dado ao processo de adicionar vértices de Steiner para conectar três vértices e remover arestas formadoras de ciclos na *MST* [36]. Esse processo, conforme exemplificado por Laarhoven (2010) [44], é geralmente alheio a qualquer estrutura global e, por isso, pode ter graves deficiências para determinadas instâncias. Isso porque a inserção de vértices de Steiner é feita para vértices próximos uns dos outros, e alguns vértices que podem ser críticos para obter a melhoria significativa na rede podem ser ignorados.

O exemplo da Figura 3.1(A) ilustra uma *MST* para uma instância do problema com nove vértices em que heurísticas de inserção, que essencialmente efetuam *"local Steinerizations"*, falham em obter a solução ideal [44]. A Figura 3.1(B) mostra que apenas os vértices de Steiner contidos nos triângulos menores são encontrados por tais heurísticas, que são incapazes de detectar o vértice de Steiner necessário para obter a solução ótima [44].

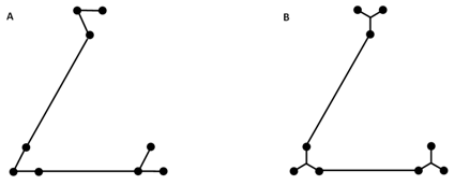


**Figura 3.1:** *A MST (A), e uma árvore de Steiner subótima obtida por heurísticas de inserção (B) para uma instância de 9 vértices.*

A Figura 3.2 mostra a solução ótima para o problema, que deve ter um vértice de Steiner no centro conectando as três subárvores formadas pelos triângulos menores [44].

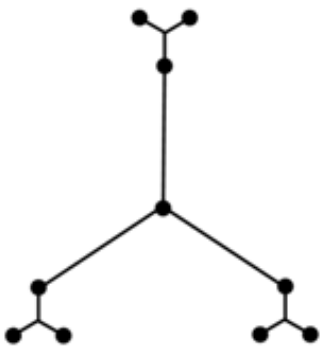

**Figura 3.2:** *A solução ótima para a instância de 9 vértices do exemplo*

Para contornar esse problema, algumas heurísticas utilizam as propriedades geométricas existentes em problemas Euclidianos. A Seção seguinte descreve algumas dessas heurísticas.

## 3.2 Diagrama de Voronoi e Triangulação de Delaunay

O Diagrama de Voronoi de um conjunto de vértices particiona o espaço em regiões, uma por vértice. A região para um vértice *s* consiste de todos os vértices mais próximos de *s* do que de qualquer outro vértice. Shamos e Hoey (1975) [62] introduziram o Diagrama de Voronoi como uma estrutura geométrica que produz algoritmos ótimos para problemas de vértices mais próximos no plano [62].

Fortune (1997) [20] descreve matematicamente uma relação de dualidade e correspondência óbvia de um para um entre o Diagrama de Voronoi e Triangulação de Delaunay. A Triangulação de Delaunay consiste na única triangulação dos vértices dados na qual a circunferência de qualquer triângulo não contém nenhum vértice em seu interior [20].

Shamos e Hoey (1975) [62] oferecem um algoritmo de complexidade média de tempo *O(n log n)* para construir o Diagrama de Voronoi. Além disso, mostraram que uma *MST* é um subgrafo da Triangulação de Delaunay. Uma *MST* pode ser construída a partir de um Diagrama de Voronoi em tempo *O(n)* pelo algoritmo de Cheriton e Tarjan (1976) [9] [37].

Beasley e Goffinet (1994) [6] apresentam uma heurística para o *ESTP* baseada na Triangulação de Delaunay para gerar vértices de Steiner candidatos e então remover vértices de Steiner redundantes via *MST* [6]. Essa heurística reúne o poder da Triangulação de Delaunay e a abordagem da heurística de Beasley (1992) [7] para oferecer um método de alta qualidade com um requisito de tempo, calculado empiricamente, de $O(n^{2,19})$ [6].

Beasley e Goffinet (1994) [6] compararam sua heurística com a heurística de Beasley (1992) [7] e mostram que a heurística utilizando Triangulação de Delaunay apresenta maior percentual de redução em nove dos dez casos de teste com a ordem variando de 10 a 100. A heurística de Beasley e Goffinet (1994) [6] alcança uma média percentual de redução de $3,025\%$.

Smith et al. (1981) [63] oferecem uma heurística *O(nlogn)* para o *ESTP*. O algoritmo é baseado em uma abordagem de decomposição em que primeiro particiona o conjunto de vértices em triângulos por meio da Triangulação de Delaunay. Em seguida, recompõe a árvore relativamente mínima a partir da utilização de Diagrama de Voronoi e da *MST* do conjunto de vértices [63].

A heurística de Smith et al. (1981) [63] escolhe conjuntos de três e quatro vértices a partir dos vértices de triângulos ou dois triângulos adjacentes da Triangulação

de Delaunay e então constrói árvores de Steiner para esses conjuntos [37]. Em geral o percentual de redução do algoritmo de Smith et al. (1981) [63] é de $3,137\%$.

## 3.3 Outras Heurísticas

Smith e Shor (1992) [65] sugeriram uma variação da *MST* que possui relação de desempenho melhor que a *"Steiner Ratio"*. Eles nomearam essa variação de Árvores Gulosa. Algumas propriedades de Árvores Gananciosas em espaços Euclidianos são: a Árvore Gulosa é a *MST* de seus vértices; nenhum ângulo na Árvore Gulosa é menor que $90^\circ$; qualquer segmento de linha entre vértices numa Árvore Gulosa também é uma conexão da *MST*; o comprimento da Árvore Gulosa não é maior que o da *MST* [65].

Outro método heurístico encontrado em alguns trabalhos é denominado *annealing*. Um algoritmo *annealing* é um procedimento de busca estocástica que procura o mínimo de uma função objetivo determinista. O método aplica pequenas perturbações em uma solução vigente para chegar a uma nova solução [49].

Lundy (1985) [49] propôs um algoritmo *annealing* como uma heurística *SMT*. Nesse algoritmo uma perturbação consiste na troca de topologias cheias. Após cada perturbação é realizado o reposicionamento dos vértices de Steiner para obter uma solução que torna-se a nova solução vigente, caso seja melhor que a anterior [37]. Grimwood (1994) [31] demonstra que o *annealing* aplicado ao *ESTP* oferece soluções de boa qualidade.

Em trabalhos mais recentes Barreiros (2003) [3] e Frommer e Golden (2007) [23] apresentam algoritmos genéticos para o *ESTP*. O algoritmo de Barreiros (2003) [3] tem duas camadas, no primeiro nível o algoritmo toma decisiões de particionamento e o nível mais baixo resolve o *ESTP* para cada partição dada. O percentual médio de redução obtido por esse algoritmo é de aproximadamente $3,73\%$ para instâncias com 10 vértices e $2,81\%$ para instâncias com 100 vértices. Para uma instância com 1000 vértices o percentual de redução obtido foi de $2,64\%$ [3].

O algoritmo de Frommer e Golden (2007) [23] foi construído a partir da experimentação de diferentes algoritmos genéticos, usando uma variedade de operadores e estratégias heurísticas de busca como diversidade de população e divisão e conquista. Foi escolhido o algorítmo genético que melhor se adaptou ao problema. O algoritmo de Frommer e Golden (2007) [23] possui complexidade de tempo estimada em $O(n^2)$, e para problemas de médio porte apresenta melhoria média aproximada de $1\%$.

CAPÍTULO 4

# Representações para Problemas Modelados por Grafos

Muitos problemas de otimização podem ser codificados por grafos, assim como o problema de Steiner, e possuem uma variedade de diferentes representações. Pesquisas mostram que o desempenho, principalmente de algoritmos evolutivos, entre outras heurísticas, pode variar muito em função da representação utilizada [59]. Uma representação adequada deve pelo menos ser capaz de codificar todas as soluções possíveis de um problema de otimização [59].

Centenas de problemas computacionais são expressos em termos de grafos. Muitas representações possuem métodos para representar e pesquisar um grafo. Pesquisar um grafo significa acompanhar sistematicamente suas arestas de modo a alcançar seus vértices. Técnicas para pesquisar um grafo estão o núcleo de algoritmos para grafos [11]. Os algoritmos para grafos são implementados em função da representação. As representações para grafos podem ser divididas nas codificações convencionais, tais como matriz de adjacências e lista de adjacências e não-convencionais, como as desenvolvidas para heurísticas específicas. A Seção 4.1 apresenta as representações convencionais e a Seção 4.2 apresenta algumas das representações não-convencionais.

## 4.1 Representações Convencionais de Grafos

Lista de Adjacências 4.1.1 e Matriz de Adjacências 4.1.2 são as duas estruturas de dados usuais para representar um grafo $G(H,I)$, em que $H$ é o conjunto de vértices e $I$ é o conjunto de arestas que forma o grafo. A representação Lista de Adjacências, em geral, é preferida, porque fornece um modo compacto de representar grafos esparsos. Mas, a representação Matriz de Adjacências pode ser preferível quando o grafo é denso, ou quando há necessidade de saber com rapidez se existe uma aresta conectando dois vértices do grafo [11]. A seguir são descritas essas duas formas de representação e suas características.

### 4.1.1 Representação Lista de Adjacências

A representação Lista de Adjacências de um grafo *G(H,I)* consiste em um arranjo de $|H|$ listas, uma para cada vértice em $H$. Para cada $h \in H$, a Lista de Adjacências contém todos os vértices $v$, tais que, existe uma aresta $(h,v) \in I$. Ou seja, todos os vértices de $G$ adjacentes a $h$. Como outra alternativa, pode conter ponteiros para os vértices adjacentes a $h$ [11].

As Listas de Adjacências podem facilmente ser adaptadas para grafos ponderados armazenando o peso da aresta juntamente com o vértice $v$ contido na lista de adjacências de $h$. Essa representação exige a quantidade $O(|H|+|I|)$ de memória, para grafos orientados ou não. Uma desvantagem potencial é que não existe um modo mais rápido para determinar se uma aresta *(h, v)* está no grafo do que procurar por $v$ na lista de adjacências de $h$ [11]. Essa desvantagem pode ser contornada pela representação Matriz de Adjacências, descrita a seguir.

### 4.1.2 Representação Matriz de Adjacências

A representação Matriz de Adjacências de um grafo $G(H,I)$ consiste em uma matriz $|H| \times |H|$ na qual cada elemento $a_{ij}$ pode conter o valor 1 ou 0, indicando respectivamente se existe ou não uma aresta entre os vértices $i$ e $j$. No caso de um grafo ponderado, a matriz de adjacência pode conter em cada aresta $a_{ij}$ o peso da aresta ou outro valor que indicaria a não existência de aresta (nulo, por exemplo) [11].

Embora, a Lista de Adjacências seja assintoticamente pelo menos tão eficiente quanto a Matriz de Adjacências, essa última pode ser preferível pela sua simplicidade e por poder representar uma aresta usando apenas um bit de entrada, no caso de grafos não ponderados [11].

## 4.2 Representações não Convencionais para Árvores de Grafos

Várias representações têm sido utilizadas para representar árvores de grafos. Essas representações apresentam desempenho melhor que as representações convencionais de grafos descritas em 4.1. Dentre elas destaca-se Número de Prufer [75][1], Chaves Aleatórias para Redes [61], Conjunto de Arestas [56], Nó-Profundidade [15], e Nó-Profundidade-Grau [16]. A seguir essas representações são descritas e a representação Nó-Profundidade-Grau é detalhada.

### 4.2.1 Representação Número de Prufer

O número de Prufer é uma representação para árvore que se baseia na propriedade de correspondência única entre uma string de comprimento $n-2$ e uma árvore geradora. O número de Prufer para uma árvore $T$ com $n$ vértices é um número com *n-2* dígitos, que possuem valores entre 1 e $n$. Os algoritmos de codificação e decodificação do número de Prufer podem ser implementados com complexidade de tempo *O(nlogn)* [30].

O número de Prufer possui características importantes tais como: qualquer árvore pode ser representada por um número de Prufer; qualquer número de Prufer representa uma árvore, mesmo que seja construído aleatoriamente; todo número de Prufer representa apenas uma árvore; todas as árvores são representadas uniformemente, ou seja, números de Prufer diferentes representam árvores diferentes.

Existem $n^{n-2}$ árvores para um grafo completo de tamanho $n$, exatamente a mesma quantidade de números de Prufer. O número de Prufer não é o único mapeamento de correspondência *1-1* entre uma string de comprimento *n-2* e uma árvore. Picciotto (1999) propôs outros mapeamentos com essa característica como o *Blob Code* e *Dandelion Code*.

Apesar da eficiência dos algoritmos de codificação e decodificação, muitos pesquisadores apontam deficiências do número de Prufer que o tornam inadequado para pesquisas evolucionárias [53][60][56]. Gottlieb et al. (2001) [30] apresentam um comparativo do uso da representação número de Prufer em algoritmos evolutivos com várias outras representações, dentre elas Chaves Aleatórias para Redes 4.2.2 e Conjunto de Arestas 4.2.3. Resultados empíricos em quatro problemas NP-difíceis envolvendo árvores geradoras confirmaram que o número de Prufer é uma má escolha para algoritmos evolutivos. Em todos os casos os algoritmos com número de prufer retornaram resultados piores além de seu desempenho ter piorado mais rapidamente em instâncias maiores [30].

### 4.2.2 Representação Chaves Aleatórias para Redes

A representação Chaves Aleatórias introduzida por Bean (1994)[4] é um eficiente método para codificar problemas de ordenação e agendamento [61]. Rothlauf et al. (2002)[61] estendeu o uso de Chaves Aleatórias para codificar árvores em problemas representados por grafos.

A representação Chaves Aleatórias codifica a solução com números aleatórios, que são usados como chaves de ordenação para decodificar a solução [4]. A representação Chaves Aleatórias para Redes representa a árvore por meio de um vetor em que cada índice é associado a uma aresta. Valores reais entre 0 e 1 são atribuídos para cada uma dessas arestas. Esse valores podem ser usados para representar a importância da aresta [61].

Rothlauf et al. (2002) [61] aplicou a representação Chaves Aleatórias para Redes para problemas de projeto de redes representadas por grafos. Os resultados experimentais foram obtidos com algoritmos evolutivos para o problema proposto indicando um desempenho satisfatório em comparação com outras representações. Chaves Aleatórias para Redes utiliza um vetor de tamanho $m$, o número de arestas do grafo, para codificar as soluções. O algoritmo de decodificação da solução requer tempo $O(m\ log(m))$.

### 4.2.3 Representação Conjunto de Arestas

A representação Conjunto de Arestas proposta por Raidl e Julstrom (2003)[56] consiste em representar árvores geradoras diretamente como conjuntos de suas arestas. Essa representação pode ser implementada em um vetor ou em tabela *hash* cujas entradas são os pares de vértices que definem cada aresta [56].

A representação Conjunto de Arestas possui complexidade de espaço $O(n)$, em que $n$ é o número de vértices do grafo. Raidl e Julstrom (2003)[56] apresentam resultados experimentais para algoritmos evolutivos aplicados a problemas modelados por grafos que indicam um desempenho superior da representação Conjunto de Arestas em comparação com outras representações como *Blob Code* e Chaves Aleatórias para Redes [56]. Tzschoppe et al. (2004)[69], porém, mostram que a representação Conjunto de Arestas com operadores evolutivos utilizando heurísticas possuem tendência, gerando mais comumente árvores próximas à árvore geradora mínima [69].

### 4.2.4 Representação Nó-Profundidade

Segundo Delbem e Carvalho (2004) [15] algoritmos evolutivos usando codificações convencionais, tais como a string binária de algoritmos genéticos, produzem muitos componentes desconectados ou grafos cíclicos, quando aplicados a grandes sistemas de problemas modelados por grafos. Com o objetivo de reduzir esses problemas Delbem e Carvalho (2004)[15] propuseram a representação Nó-Profundidade. Essa representação utiliza os conceitos de vértice e profundidade de uma vértice para codificar as soluções. Uma das vantagens da representação Nó-Profundidade é que somente são produzidos componentes conectados e grafos acíclicos ao utilizar os operadores propostos para a mesma.

A representação nó-profundidade consiste basicamente de listas lineares contendo os vértices e suas profundidades. Nessa representação, um vértice que é a raiz da árvore é o primeiro elemento da lista e tem profundidade 0. A ordem dos pares (vértice, profundidade) na lista pode ser determinada por uma busca em profundidade. Esse processo de codificação de uma árvore em grafos auxília no processo de decodificação das soluções e permite o fácil mapeamento das arestas presentes na solução.

Delbem e Carvalho (2004) [15] propuseram um algoritmo evolutivo usando a representação Nó-Profundidade para o problema da árvore geradora mínima com restrição de grau (*dc-MSTP*, degree-constrained Minimum Spanning Tree Problem). O *dc-MSTP* é o problema de encontrar a árvore geradora em que os graus de vértices são menores que ou iguais a uma constante positiva, e que o custo total das arestas é mínimo.

O algoritmo proposto por Delbem e Carvalho (2004) [15] foi comparado com outro algoritmo genético para o *dc-MSTP* usando a representação Número de Prufer. Os resultados mostraram que o algoritmo proposto por Delbem e Carvalho (2004) [15] é capaz de trabalhar com grandes gráficos usando relativamente pequeno tempo de execução. Com restrição de grau variando de 3 a 5, o algoritmo de Delbem e Carvalho (2004) [15] gastou aproximadamente $1,5\%$ para gráficos com 500 vértices e $1,1\%$ para gráficos com 1000 vértices do tempo gasto pelo algoritmo usando representação Número de Prufer e obteve aproximadamente 42% para gráficos com 500 vértices e 58% para gráficos com 1000 vértices dos melhores custos (média dos melhores indivíduos) obtidos pelo algoritmo usando representação Número de Prufer.

### 4.2.5 Representação Nó-Profundidade-Grau

Uma das codificações que tem mostrado resultados relevantes para codificar árvores em grafos é a representação nó-profundidade-grau (RNPG) proposta por Delbem, Lima e Telles (2012) [16]. A RNPG é uma extensão da representação Nó-Profundidade proposta em [15].

A RNPG foi proposta para atender as exigências de algoritmos evolutivos para problemas de larga escala modelados por grafos, que requerem a geração de milhares de florestas [1] geradoras até encontrar um projeto de rede apropriado. A RNPG é uma representação devidamente codificada para florestas geradoras e oferece operações para manipular e gerar novas florestas [16].

A RNPG tem como base os conceitos de vértice, profundidade e grau de um vértice em um grafo. A forma de representar a floresta geradora é utilizado um conjunto de listas lineares de trincas (vértice, profundidade, grau), uma para cada árvore da floresta. A ordem em que as trincas estão dispostas na lista depende da posição do vértice em relação ao vértice inicial chamado de raiz da árvore [16]. O primeiro elemento da lista é a raiz da árvore, a partir da raiz qualquer elemento da lista aparece depois do seu elemento pai (o primeiro elemento mais próximo da raiz que o conecta). Essa ordem pode ser obtida por meio de uma busca em profundidade na árvore a partir do vértice escolhido como raiz [16].

[1] Uma floresta geradora em grafos consiste em um conjunto de árvores disconexas que contém todos os vértices.

A Figura 4.1 ilustra o exemplo de uma árvore geradora de um grafo com 9 vértices. Essa árvore geradora é representada pela notação RNPG na Tabela 4.1.

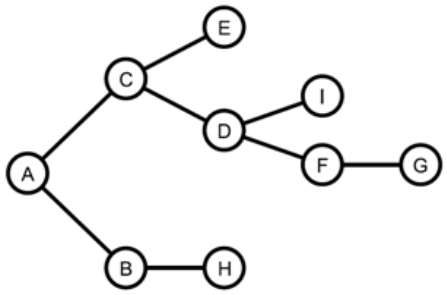


**Figura 4.1:** *Exemplo de árvore na notação RNPG.*

**Tabela 4.1:** *Lista linear contendo em cada coluna da RNPG a tripla representando a árvore geradora da Figura 4.1.*

| *Posição* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| *Vértice* | A | B | H | C | D | F | G | I | E |
| *Profundidade* | 0 | 1 | 2 | 1 | 2 | 3 | 4 | 3 | 2 |
| *Grau* | 2 | 2 | 1 | 3 | 3 | 2 | 1 | 1 | 1 |

A RNPG quando utilizada em algoritmos evolutivos possui duas operações para gerar novas soluções à partir de soluções já existentes [16]. O efeito dessas operações é o de transferir uma subárvore de uma árvore para outra na floresta. A primeira operação produz mudanças simples e pequenas na floresta, enquanto que a segunda operação gera mudanças maiores e mais complexas. Embora sejam diferentes, essas operações compartilham as seguintes características: preservar a consecutividade de vértices na RNPG; preservar a precedência da raiz de uma subárvore em relação aos seus descendentes; não introduzir ciclos nas árvores; preservar o número de árvores na floresta [16].

Para ilustrar as vantagens da RNPG, a mesma foi aplicada em um algoritmo evolutivo para o *dc-MSTP*. O algoritmo desenvolvido foi comparado com outro algoritmo evolutivo para o *dc-MSTP* usando a representação Conjunto de Arestas proposto em [56]. Resultados experimentais mostraram que o algoritmo com a RNPG é mais rápido e produziu soluções melhores ou equivalentes as obtidas com conjunto de arestas [16].

Os resultados recentes demonstram que a RNPG tem tido destaque para melhorar o desempenho de algoritmos evolutivos para problemas modelados por grafos. Esses resultados incentivam a investigação do uso da RNPG em outros contextos, como é o caso deste trabalho. No próximo capítulo será apresentada a proposta de utilização da RNPG em uma heurística para o problema de árvore de Steiner Euclidiano baseada no algoritmo de Beasley (1992) [7].

CAPÍTULO 5

# Algoritmo SMTBeasleyRNPG

Este capítulo apresenta a proposta deste trabalho, a heurística SMTBeasleyRNPG baseado em [7] para o problema de árvore de Steiner Euclidiano (ESTP). O principal objetivo desse trabalho é demonstrar vantagens na eficiência computacional a partir da utilização da *RNPG* em um heurístico não evolutivo, cujas soluções sejam as mesmas para as mesmas entradas. Foi feita uma pesquisa na literatura para verificar um algoritmo heurístico não evolutivo, que oferece soluções de boa qualidade. O algoritmo proposto em [7] foi julgado mais apropriado em função dessas exigências, conforme apresenta a Seção 5.1.

Para obter uma melhoria na eficiência computacional, essa heurística será associada a uma representação adequada para problemas de modelados por grafos de árvores geradoras, que é o caso do ESTP. No algoritmo de Beasley (1992) [7], a solução atual em cada iteração é uma única MST do conjunto atual de vértices. Não existe a necessidade de representar florestas, nem operadores para florestas. Assim, optou-se por utilizar a *RNPG* com uma única árvore geradora. No entanto, foi necessário o desenvolvimento de operadores específicos para algumas operações do Algoritmo 3.1. A *RNPG* é utilizada na SMTBeasleyRNPG para representar a topologia da árvore. Uma estrutura de grafo representa as localizações dos vértices do conjunto $V$ definido para o Algoritmo 3.1.

No desenvolvimento da heurística SMTBeasleyRNPG, não serão utilizados os operadores atuais da *RNPG*, pois a heurística base não propõem operações de poda e enxerto de subárvores. Por outro lado, será necessário o desenvolvimento de operadores específicos na *RNPG* para executar alguns passos da heurística utilizada.

A descrição da proposta deste trabalho está estruturada como segue. A Seção 5.1 justifica a escolha do algoritmo proposto em [7]. A Seção 5.2 descreve algumas modificações que foram necessárias na *RNPG* para permitir a manipulação dos dados de forma eficiente. O operador de consulta de conjuntos, responsável por recuperar as subárvores de 4 vértices é apresentado na Seção 5.3. Por fim a Seção 5.4 descreve o operador para remoção dos vértices de Steiner adicionados que não possuem mais as características necessárias dessa função.

## 5.1 Escolha do algoritmo Beasley (1992)

Dentre os algoritmos pesquisados, alguns utilizam Diagrama de Voronoi e Triangulação de Delaunay [20]. É o caso dos contidos em [62], [6], [63], [44]. Esses dois últimos inclusive são mais recentes que o algoritmo escolhido. A Tabela 5.1 mostra que esses algoritmos, [6] e [44], oferecem, em média, soluções de melhor qualidade comparado a Beasley (1992) [7].

**Tabela 5.1:** *Comparativo dos percentuais de redução em relação à MST dos algoritmos de Beasley (1992) [7], Beasley e Goffinet (1994) [6] e Laarhoven (2010) [44].*

| $n$ | Beasley (1992) [7] | Beasley e Goffinet (1994) [6] | Laarhoven (2010) [44] |
|---|---|---|---|
| 10 | 3,138 | 3,223 | 3,25 |
| 20 | 3,015 | 3,123 | 3,15 |
| 30 | 2,868 | 2,948 | 3,07 |
| 40 | 3,024 | 2,972 | 3,14 |
| 50 | 2,841 | 2,921 | 3,03 |
| 60 | 2,946 | 3,178 | 3,27 |
| 70 | 2,844 | 2,945 | 3,11 |
| 80 | 2,817 | 2,917 | 3,03 |
| 90 | 2,935 | 2,953 | 3,11 |
| 100 | 2,952 | 3,073 | 3,26 |

Apesar de serem mais recentes e apresentar soluções de melhor qualidade, os algoritmos propostos em [6] e [44] não foram escolhidos em função do tempo de implementação. Para essa implementação, haveria a necessidade do aprendizado do Diagrama de Voronoi e Triangulação de Delaunay [20], que levaria um tempo maior que o tempo gasto para a implementação de [7], que não depende do conhecimento de outra técnica.

A melhoria em relação à *MST* usando a heurística de Beasley [7] é normalmente de 3 a 4% [36], essa melhoria possui melhor qualidade comparado a outras heurísticas como Chang (1972) [8] e Smith et al. (1981) [63][7]. As Tabelas 5.2 e 5.3 comprovam que o percentual de redução de [7] é, em média, melhor que [8] e [63].

**Tabela 5.2:** *Comparativo dos percentuais de redução em relação à MST dos algoritmos de Beasley (1992) [7] e Chang (1972) [8].*

| $n$ | Beasley (1992) [7] | Chang (1972) [8] |
|---|---|---|
| 10 | 3,138 | 2,2 |
| 20 | 3,015 | 3,012 |
| 30 | 2,868 | 3,087 |

**Tabela 5.3:** *Comparativo dos percentuais de redução em relação à MST dos algoritmos de Beasley (1992) [7] e Smith et al. (1981) [63].*

| $n$ | Beasley (1992) [7] | Smith et al. (1981) [63] |
|---|---|---|
| 10 | 3,138 | 3,173 |
| 20 | 3,015 | 2,333 |
| 30 | 2,868 | 2,769 |
| 40 | 3,024 | 2,663 |
| 50 | 2,841 | 2,568 |

Alguns algoritmos heurísticos não oferecem resultados computacionais que tornem possível esse comparativo de qualidade de soluções, é o caso de Thompson (1973) [67], Lundy (1985) [49] e Dreyer e Overton (1998) [17]. Os algoritmos de Barreiros (2003) [3] e Frommer e Golden (2007) [23] não foram considerados para essa proposta em função do objetivo do trabalho que é investigar a *RNPG* em outro contexto além de algoritmos evolutivos.

Em resumo, o algoritmo de Beasley (1992) [7] foi considerado adequado por oferecer soluções de boa qualidade e atender os objetivos do trabalho. Considera-se que essa investigação é válida para outras heurísticas não evolutivas, como é o caso de Beasley e Goffinet (1994) [6] que é baseada em [7]. Essa comprovação é sugerida para trabalhos futuros.

## 5.2 Versão da RNPG para o SMTBeasleyRNPG

Com o objetivo de melhorar a eficiência computacional de acesso e busca a determinadas posições na *RNPG* foram adicionados três dados para cada vértice. A primeira informação necessária é um *flag* para indicar se o vértice é um ponto de Steiner ou não, respectivamente com os valores 1 e 0, chamada de *steiner*. Além disso, foi adicionada a posição do pai do vértice atual, chamado de *indicePai*. Esse dado possibilita o acesso direto ao vértice ancestral do ponto atual da árvore. O terceiro dado adicionado, chamado de *indiceFilho*, indica a posição do último filho do vértice atual na hierarquia da árvore. A Tabela 5.4 mostra a *RNPG* da árvore da Figura 4.1 com as novas informações inseridas. Para os dados de *indicePai* e $indiceFilho$ quando o valor é $-1$ isso indica que o vértice não possui ancestral ou descendente.

A Tabela 5.5 apresenta um exemplo da nova *RNPG* quando ocorre a adição de vértices de Steiner. Para que o ponto de Steiner seja inserido na posição correta é necessário deslocar os vértices posteriores a ele na *RNPG* em um posição. Lembrando que a posição na *RNPG* é importante para o processo de decodificação da árvore representada. Um vértice de profundidade $x$ é considerado ligado ao primeiro vértice anterior a ele de

**Tabela 5.4:** *Exemplo da RNPG com colunas acrescentadas para o algoritmo SMTBeasleyRNPG.*

| *Posição* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| *Nó* | A | B | H | C | D | F | G | I | E |
| *Profundidade* | 0 | 1 | 2 | 1 | 2 | 3 | 4 | 3 | 2 |
| *Grau* | 2 | 2 | 1 | 3 | 3 | 2 | 1 | 1 | 1 |
| *steiner* | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *indicePai* | -1 | 0 | 1 | 0 | 3 | 4 | 5 | 4 | 3 |
| *indiceFilho* | 3 | 2 | -1 | 8 | 7 | 6 | -1 | -1 | -1 |

profundidade $x-1$, representado também pelo *indicePai*. Também é necessário ajustar os valores de *indicePai* e *indiceFilho* de todos os vértices e, para os que estão ligados ao vértice de Steiner, é preciso corrigir as informações de grau e profundidade, quando necessário. No exemplo da Tabela 5.5, foi adicionado o vértice de Steiner $J$, ligado aos vértices $A$, $B$ e $H$. Nesse caso, a profundidade e grau de $B$ e $H$ precisam ser atualizados e os valores de *indicePai* e *indiceFilho* de toda lista a partir da posição um. A Figura 5.1 mostra a árvore com a adição do vértice de Steiner $J$.

**Tabela 5.5:** *Exemplo da versão da RNPG com ponto de Steiner J adicionado para os pontos A, B e H*

| *Posição* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| *Nó* | A | J | B | H | C | D | F | G | I | E |
| *Profundidade* | 0 | 1 | 2 | 2 | 1 | 2 | 3 | 4 | 3 | 2 |
| *Grau* | 2 | 3 | 1 | 1 | 3 | 3 | 2 | 1 | 1 | 1 |
| *steiner* | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| *indicePai* | -1 | 0 | 1 | 1 | 0 | 4 | 5 | 6 | 5 | 4 |
| *indiceFilho* | 4 | 3 | -1 | -1 | 9 | 8 | 7 | -1 | -1 | -1 |

O *flag steiner* é utilizado para realizar algumas operações restritas a pontos de Steiner. Os valores de *indicePai* e *indiceFilho* contribuem para identificar as arestas de determinado vértice de forma mais eficiente. A aresta que liga o vértice atual ao seu ancestral é dada diretamente pelo *indicePai*. Já o *indiceFilho* é utilizado para delimitar o intervalo da lista da *RNPG* que deve ser percorrido para encontrar os descendentes do vértice atual. Por exemplo, em um ponto de Steiner, seu grau deve ser 3 e as arestas que conectam esse vértice devem estar nas posições *indicePai*, (índice do vértice)+1, e *indiceFilho*. Dessa forma, essas conexões podem ser descobertas sem a necessidade de iterar na *RNPG*.

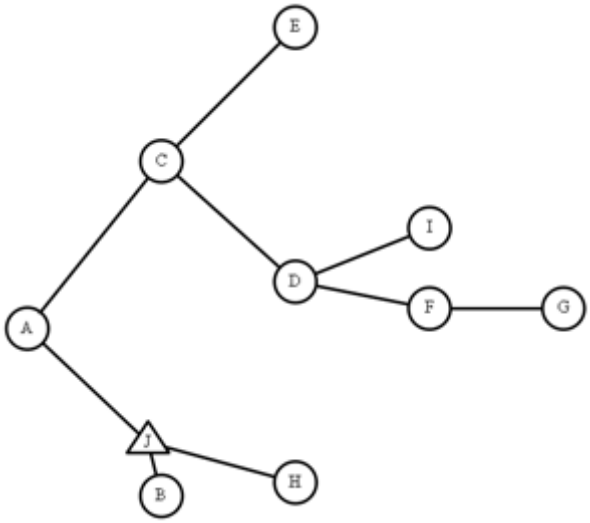


**Figura 5.1:** *Exemplo de árvore na notação RNPG com vértice de Steiner J adicionado.*

## 5.3 Operador de Consulta de Conjuntos

O Operador de Consulta de Conjuntos (OCC) é proposto para implementar o passo 3 do Algoritmo 3.1 da heurística base deste trabalho. O OCC percorre a *RNPG* que representa a *MST* para buscar os subgrafos de quatro vértices. Os subgrafos válidos para compor o conjunto *L* resultante do OCC devem representar um dos seguintes casos:

1. Os vértices do subgrafo formam um caminho na *MST*;
2. Uma configuração de estrela na MST centrada em um dos vértices.

O OCC obtém os subgrafos de quatro vértices na *RNPG* de três formas: conexões seqüenciais, i.e. cada um dos vértices está em uma profundidade diferente da árvore; subgrafos com raiz de grau 2; subgrafos com topologia estrela, ou seja o vértice raiz de grau maior ou igual a 3, nesse caso são enumeradas todas as permutações de três elementos do conjunto de vértices da topologia estrela. Cada uma das permutações corresponde a um novo *K*, que deve ser adicionado a *L*. A Figura 5.2 ilustra um exemplo de cada uma dessas três possíveis formas de se obter uma subárvore de quatro vértices.

O Algoritmo 5.1 descreve os passos para execução do OCC. A entrada do algoritmo é a MST representada pela *RNPG*, e a saída o conjunto *L* de subgrafos de quatro vértices. Para facilitar a compreensão do Algoritmo 5.1, que implementa o OCC, foi feita a divisão em três trechos que implementa cada uma das três possíveis formas de se obter um conjunto de quatro vértices, exemplificadas na Figura 5.2. Esses trechos são descritos pelos Algoritmos 5.2, 5.3 e 5.4.

Considere na descrição dos Algoritmos 5.1, 5.2, 5.3 e 5.4: *solucaoatual* a MST atual dada como entrada do algoritmo representada na *RNPG*; *N* o número de vértices da

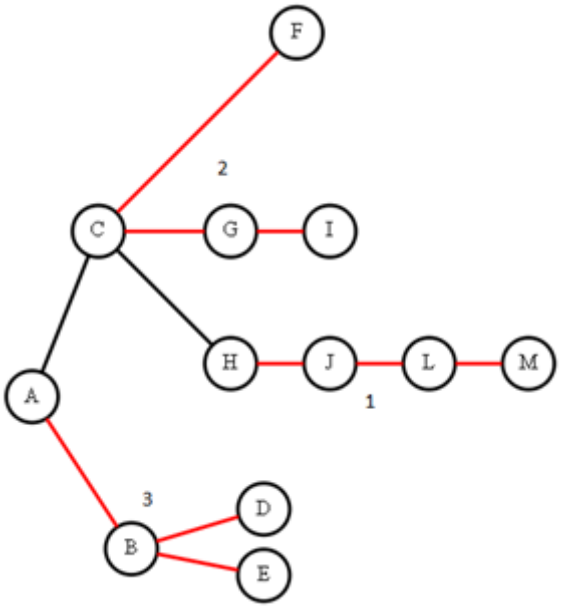


**Figura 5.2:** *Exemplo das três possíveis formas de se obter uma subárvore de 4 vértices em uma MST.*

*solucaoatual*; $p$, $q$, $r$ e $s_1$ números inteiros que identificão os índices na *solucaoatual* dos vértices que compõem o subgrafo de quatro vértices; $K$ um conjunto de quatro vértices correspondendo a um subgrafo da *solucaoatual*; $L$ um conjunto de todos possíveis $K$'s que ao final do algoritmo conterá todos os subgrafos de quatro vértices da *solucaoatual*; *aux* um vetor de inteiros auxiliar para armazenar posições de vértices adjacente a um vértice; *tam*, $g$, $h$, $i$, $j$ inteiros auxiliares, representando índices na *RNPG*.

**Algoritmo 5.1:** Operador de Consulta de Conjuntos

**Entrada**: A MST de entrada representada pela *RNPG* (*solucaoatual*).
**Saída**: Os conjuntos de quatro pontos representando subárvores da *solucaoatual* ($K$).

```
1 para g ← 0;g < N faça
2     p ← g;
3     Algoritmo 5.2;
4     Algoritmo 5.3;
5     Algoritmo 5.4;
6 fim
```

**Algoritmo 5.2:** Trecho do OCC que obtém a subárvore por meio de conexões sequenciais, conforme ítem 1 da Figura 5.2.

**Entrada**: A mesma do Algoritmo 5.1.
**Saída**: As subárvores da *solucaoatual* formadas por conexões sequenciais de quatro pontos.

```
1  se solucaoatual[p].profundidade ≥ 3 então
2      q ← solucaoatual[p].indicePai;
3      r ← solucaoatual[q].indicePai;
4      s1 ← solucaoatual[r].indicePai;
5      crie um novo K com os nós p, q, r e s1;
6      adicione K em L;
7  fim
```

**Algoritmo 5.3:** Trecho do OCC que obtém a subárvore por meio de subgrafos com raiz de grau 2, conforme ítem 2 da Figura 5.2.

**Entrada**: A mesma do Algoritmo 5.1.
**Saída**: As subárvores da *solucaoatual* formadas por subgrafos com raiz de grau 2.

```
1  se solucaoatual[p].profundidade > 0 então
2      q ← solucaoatual[p].indicePai;
3      i ← q + 1;
4      enquanto i ≤ solucaoatual[q].indiceFilho faça
5          se (solucaoatual[i].profundidade = solucaoatual[q].profundidade+1) e i ≠ p então
6              para j ← i + 1; j ≤ solucaoatual[i].indiceFilho faça
7                  se solucaoatual[j].profundidade = solucaoatual[i].profundidade+1 então
8                      r ← i; s1 ← j;
9                      crie um novo K com os nós p, q, r e s1;
10                     adicione K em L;
11                 fim
12             fim
13             i ← j;
14         fim
15     fim
16 fim
```

**Algoritmo 5.4:** Trecho do OCC que obtém a subárvore por meio de subgrafos com topologia estrela, conforme ítem 3 da Figura 5.2.

**Entrada**: A mesma do Algoritmo 5.1.
**Saída**: As subárvores da *solucaoatual* formadas por subgrafos com topologia estrela.

```
1  se solucaoatual[p].grau ≥ 3 então
2      aux[solucaoatual[p].grau];
3      tam ← 0;
4      para i ← p+1;i ≤ solucaoatual[p].indiceFilho faça
5          se (solucaoatual[i].profundidade = solucaoatual[p].profundidade+1) então
6              aux[tam] ← i; tam ← tam+1;
7          fim
8      fim
9      se (solucaoatual[p].profundidade >0) então
10         aux[tam] ← solucaoatual[p].indicePai;
11         tam ← tam+1;
12     fim
13     para h ← 0;h < tam−2 faça
14         para i ← h+1;i < tam−1 faça
15             para j ← i+1; j < tam faça
16                 q ← aux[h]; r ← aux[i]; s1 ← aux[j];
17                 crie um novo K com os nós p, q, r e s1;
18                 adicione K em L;
19             fim
20         fim
21     fim
22 fim
```

Nesta seção também será apresentada a análise de complexidade de tempo média do OCC. Além disso, uma versão do OCC que faz uso da representação convencional Matriz de Adjacência [11], descrita na seção 4.1.2. Ambas as versões serão comparadas com relação a complexidade de tempo computacional.

### 5.3.1 Complexidade do OCC

A análise de complexidade de tempo do Algoritmo 5.1 do operador OCC será primeiramente para cada forma de identificação dos conjunto de quatro vértices. Depois, será realizada a análise da complexidade total do algoritmo proposto. No entanto, primeiro é necessário fazer a análise da complexidade de espaço, ou seja, do tamanho em memória ocupado pelo vetor da *RNPG* que contém a MST.

Na primeira execução do OCC, o conjunto $V$ contém somente os vértices do grafo e, portanto, tem tamanho $n$. No decorrer dos passos da heurística são adicionados os vértices de Steiner para reduzir o custo da MST. No pior caso, a solução ótima é uma árvore de Steiner completa. Assim, o conjunto $V$ será composto pelos $n$ vértices iniciais e os $n-2$ vértices de Steiner de uma árvore completa. Portanto, tem-se que no pior caso o tamanho da *RNPG* será de $n+n-2$, ou seja, $2n-2$, ou ainda, $O(n)$. Nos demais casos o conjunto $V$ será composto por um número inferior de vértices, então a análise de complexidade de tempo será conduzida considerando uma entrada de tamanho $n$.

A primeira maneira descrita para obter um conjunto de quatro vértices é por meio de conexões sequenciais na *RNPG*. Essa situação é considerada no condicional da linha 1 do Algoritmo 5.2. Os passos internos, executados nesse condicional, são de complexidade de tempo constante, pois consistem de atribuições diretas de valor com base no *indicePai* dos vértices considerados. Assim, pode-se afirmar que a identificação de um conjunto $K$ válido por conexões sequenciais possui complexidade de tempo de melhor, média e pior caso de $O(1)$.

A segunda situação descrita, considera vértices raiz de grau 2 e é expressa pelo condicional da linha 1 do Algoritmo 5.3. Nesse caso, tem-se dois *loop* aninhados que precisam ser analisados. O *loop* da linha 4 tem como limite o *indiceFilho* do vértice definido em $q$, esse índice indica a posição do último vértice diretamente ligado a $q$ na *RNPG*. Assim, o maior valor possível do *indiceFilho* é o tamanho da subárvore enraizada em $q$, $tam_q$. O segundo *loop* da linha 6 tem o seu limite inserido no intervalo do *loop* externo, pois a subárvore enraizada no vértice analisado faz parte da subárvore do vértice $q$. Na linha 13 o valor do índice $i$ do *loop* da linha 4 é atualizado com o valor de $j$, índice do *loop* interno. Como os *loops* são complementares, pode-se dizer que essa etapa possui complexidade de tempo $O(tam_q)$. No melhor caso, $tam_q = 0$, com isso os *loops* não são executados. No pior caso, $tam_q = n$ e a etapa possui complexidade de tempo $O(n)$. O caso médio depende do tamanho de $tam_q$, para o qual não é simples realizar uma análise.

A terceira, e última forma descrita para se obter um conjunto $K$ de quatro vértices ocorre quando exite uma formação de topologia estrela com três vértices ou mais centrada no vértice $p$. Esse caso é analisado no condicional da linha 1 do Algoritmo 5.4. Os conjunto de passos a serem realizados, nesse caso, envolve um *loop* para localizar todos os adjacentes do vértice $p$. No pior caso, esse *loop* necessitará de percorrer toda a *RNPG* para recuperar esses vértices e armazená-los no *array* auxiliar e, portanto, possui complexidade de tempo $O(n)$ no pior caso. Na média e melhor caso, é necessário percorrer toda a subárvore de $p$ para encontrar os adjacentes. Então, considerando $tam_p$, o tamanho para armazenar a subárvore de $p$ na *RNPG*, tem-se que a complexidade média de tempo do *loop* da linha 4 é $O(tam_p)$. A seguir, é realizado um condicional para adição do vértice pai de $p$ à lista de adjacentes que é realizado em tempo constante $O(1)$. Depois, o algoritmo obtém a permutação de elementos adjacentes a $p$ a partir dos *loops* aninhados 13, 14 e 15. Esses *loops* tem como limite o valor *tam*, que corresponde ao número de adjacentes de $p$, ou seja, ao grau de $p$. O pior caso, ocorre quando a topologia da MST de entrada do algoritmo é uma estrela centrada no vértice $p$. Nesse caso, tem-se que o grau de $p$ é $n-1$ e, portanto, o processo de permutação dos adjacentes em grupos de três vértices possui complexidade de tempo de pior caso $O(n^3)$. Nos casos médio e melhor, tem-se que a complexidade de tempo da permutação será de $O(tam^3)$, onde *tam* é delimitado

pelo grau de $p$. A complexidade total desse passo, no pior caso, é de $O(n) + O(n^3)$, portanto, $O(n^3)$. Nos casos médio e melhor, a complexidade é dada por $O(tam_p + tam^3)$, para obter-se a função assintótica que domina essa complexidade é necessário analisar os possíveis valores que cada um desses limitantes pode assumir, o que é complexo devido a combinação de situações disponíveis.

Para concluir é preciso analisar a complexidade de tempo total do Algoritmo 5.1. As três situações definidas são internas ao *loop* da linha 1. Esse *loop* percorre toda lista da *RNPG* que possui tamanho $n$, e, portanto, é $O(n)$. A cada passagem desse *loop* principal podem ocorrer três possibilidades, (i) executar todas as situações previstas nos condicionais, (ii) executar somente um condicional e (iii) executar dois condicionais. O caso (i) só poderá ocorrer quando o indice $g >= 3$. Nessa possibilidade, não é possível obter o pior caso do terceiro condicional, mas é possível que ocorra o pior caso do segundo condicional, portanto, a complexidade de tempo total é $O(n) \times (O(1) + O(n) + O(tam_p + tam^3)) = O(n^2)$. Na situação (ii), para o índice $g = 0$, só é possível executar a terceira situação, para $g = 1$ e $g = 2$, pode ocorrer a segunda ou a terceira situação, somente a partir de $g >= 3$, existe a possibilidade de ocorrer todas as situações. Então, se para $g = 0$ ou $g = 1$ ocorrer, o pior caso da terceira situação, a complexidade de tempo pode ser escrita como $O(n^3) + (n-1) \times O(1) = O(n^3)$, pois, esse pior caso só pode ocorrer em uma das iterações do *loop* principal e os demais vértices não corresponderam a nenhum dos outros casos. Esse caso não ocorre para os outros valores de $g$ devido as características de construção da *RNPG*. Nas demais situações apresentadas, tem-se que elas serão dominadas assintoticamente por $O(n)$ e a complexidade de tempo do algoritmo pode ser dita como $O(n^2)$. Quando as características permitirem que a cada iteração sejam executadas duas situações, o domínio assintótico também será de $O(n)$, e a complexidade de tempo média do algoritmo de $O(n) * (O(n) + O(n) = O(n^2)$. Resumindo, tem-se que a complexidade de tempo do operador OCC implementado com a *RNPG*, pode ser no pior caso $O(n^3)$, na média e no melhor caso $O(n^2)$. Na prática, esse limitante pode ser inferior, mas não é possível teoricamente afirmar com facilidade qual seria esse valor, pois as operações dependem fortemente do número de vértices das subárvores e das topologias locais das mesmas.

### 5.3.2 Versão com Matriz de Adjacências

Com o objetivo de verificar se o uso da *RNPG* resultado em uma vantagem de tempo computacional em relação ao uso de uma representação convencional, o operador de identificação de todos os subgrafos de quatro vértices foi implementado com a representação Matriz de Adjacências. Essa operação foi escolhida para esse propósito, pois pode ser considerada uma das de maior complexidade da heurística proposta em [7].

O Algoritmo 5.5 descreve os passos da versão do Algoritmo 5.1 para essa modificação. O algoritmo tem como entrada a MST representada por uma Matriz de Adjacências e, assim como o Algoritmo 5.1, retorna o conjunto $L$.

Para o Algoritmo 5.5 considere: *solucaoatual* a MST atual dada como entrada e representada pela Matriz de Adjacências; $N$ o número de vértices da *solucaoatual*; $p$, $q$, $r$ e $s_1$ números inteiros correspondendo aos índices na matriz e vértices do grafo; $K$ um conjunto de 4 pontos correspondendo a uma subárvore; $L$ um conjunto de conjuntos $K$ que ao final do algoritmo conterá todos os subgrafos de quatro vértices da *solucaoatual*.

O OCC implementado com Matriz de Adjacências opera buscando subgrafos das mesmas três maneiras que a versão com *RNPG*. As conexões seqüenciais na árvore, obtidas no condicional da linha 1 do Algoritmo 5.1 correspondem ao *loop* da linha 24 da versão com Matriz de Adjacências; subgrafos com raiz de grau 2, obtidas no condicional da linha 1 do Algoritmo 5.1 correspondem ao *loop* da linha 14; subgrafos com topologia estrela com raiz de grau maior ou igual a 3, obtidas conforme condicional da linha 1 do Algoritmo 5.1 correspondem ao *loop* da linha 4.

A análise de complexidade de tempo do do Algoritmo 5.5 é mais simples de ser realizada, pois para todas situações propostas o algoritmo necessita de executar um *loop* percorrendo todos os vértices do conjunto $V$. Além disso, como o limitante de todos os *loops* especificados é $N$ o número de vértices na *solucaoatual*, então não existe diferença de melhor, médio e pior caso. No Algoritmo 5.5 existem três *loops* aninhados principais e internamente a eles o algoritmo possui mais três laços não aninhados, um para cada situação possível de subgrafo. Como resultado, tem-se que a complexidade de tempo é de $O(N^3) \times (O(N) + O(N) + O(N)) = O(N^4)$. Como já analisado na Seção de complexidade do Algoritmo 5.1 o tamanho da entrada $N$ é $O(n)$, então a complexidade de tempo do Algoritmo 5.5 pode ser escrita em função de $n$ como $O(n^4)$. A Seção 6.3 apresenta um comparativo empírico do tempo computacional dos Algoritmos 5.1 e 5.5, que comprova o resultado teórico descrito neste Capítulo.

**Algoritmo 5.5:** Operador de Consulta de Conjuntos com Matriz de Adjacências.

**Entrada**: A MST de entrada representada pela Matriz de Adjacência (*solucaoatual*).
**Saída**: Os conjuntos de quatro pontos representando subárvores da *solucaoatual* (*L*).

```
1  para p ← 0;p < N faça
2      para q ← 0;q < N faça
3          se solucaoatual[p][q] > 0 então
4              para r ← q + 1;r < N faça
5                  se solucaoatual[p][r] > 0 então
6                      para s ← r + 1;s < N faça
7                          se solucaoatual[p][s] > 0 então
8                              crie um novo K com os nós p, q, r e s;
9                              adicione K em L;
10                         fim
11                     fim
12                 fim
13             fim
14             para r ← 0;r < N faça
15                 se solucaoatual[p][r] > 0 e r ≠ q então
16                     para s ← 0;s < N faça
17                         se solucaoatual[r][s] > 0 e s ≠ q e s ≠ p então
18                             crie um novo K com os nós p, q, r e s;
19                             adicione K em L;
20                         fim
21                     fim
22                 fim
23             fim
24             para r ← 0;r < N faça
25                 se solucaoatual[q][r] > 0 e r ≠ p então
26                     para s ← 0;s < N faça
27                         se solucaoatual[r][s] > 0 e s ≠ q e s ≠ p então
28                             crie um novo L com os vértices p, q, r e s;
29                             adicione L em K;
30                         fim
31                     fim
32                 fim
33             fim
34         fim
35     fim
36 fim
```

## 5.4 Operador de Remoção de Nó

O Operador de Remoção de Nó (ORN) tem como finalidade a remoção dos vértices de Steiner adicionados que venham a perder as suas características obrigatórias após a construção da MST. A necessidade dessa remoção é descrita no passo 7 do Algoritmo 3.1. O Algoritmo 5.6 descreve os passos para executar o processo de remoção na MST representada pela *RNPG*. As entradas do algoritmo são a MST representada pela *RNPG* e o conjunto *V* de vértices do grafo, como resultado retorna a nova MST.

Para o algoritmo 5.6 considere: *V* o conjunto de vértices atual do grafo do problema composto pelo par *(x,y)*. Esse conjunto contém os vértices originais e os vértices

de Steiner adicionados; *solucaoatual* a MST representada pela *RNPG*; *N* o número de nós da *solucaoatual*; *NV* é a norma de *V*.

**Algoritmo 5.6:** Operador de Remoção de Nó.

**Entrada**: A MST de entrada representada pela *RNPG* (*solucaoatual*), e o grafo (*V*) atual com os pontos originais e os pontos de Steiner já adicionados.

**Saída**: A nova *solucaoatual* e o novo grafo *V*, sem os pontos de Steiner.

```
1  noremov ← true;
2  enquanto noremov faça
3      noremov ← false;
4      para i ← 0;i < N faça
5          se solucaoatual[i].steiner e solucaoatual[i].grau < 3 então
6              rem ← solucaoatual[i].n;
7              V[rem].status ← removido;
8              profundidade ← solucaoatual[i].profundidade;
9              indiceFilho ← solucaoatual[i].indiceFilho;
10             indicePai ← solucaoatual[i].indicePai;
11             se indiceFilho = −1 então
12                 solucaoatual[indicePai].grau ← solucaoatual[indicePai].grau − 1;
13             senão
14                 j ← i + 1;
15                 enquanto (j<N) e (solucaoatual[j].profundidade > profundidade) faça
16                     solucaoatual[j].profundidade ← solucaoatual[j].profundidade − 1;
17                     se solucaoatual[j].indicePai == i então
18                         solucaoatual[j].indicePai ← indicePai;
19                     fim
20                     j ← j + 1;
21                 fim
22             fim
23             j ← i + 1;
24             enquanto j<N faça
25                 se solucaoatual[j].indicePai > i então
26                     solucaoatual[j].indicePai ← solucaoatual[j].indicePai − 1;
27                 fim
28                 solucaoatual[j].indiceFilho ← solucaoatual[j].indiceFilho − 1;
29                 solucaoatual[j − 1] ← solucaoatual[j];
30                 j ← j + 1;
31             fim
32             //reduz o número de vértices presentes na solucaoatual
33             N ← N − 1;
34             para j ← 0;j < i faça
35                 se solucaoatual[j].indiceFilho > i então
36                     solucaoatual[j].indiceFilho ← solucaoatual[j].indiceFilho − 1;
37                 fim
38             fim
39             noremov ← true;
40         fim
41     fim
42 fim
```

Como a *RNPG* é estruturada em uma lista, ao remover um vértice, ORN deve mover todos os elementos que estão depois da posição do vértice removido para a posição imediatamente anterior. Além disso, ORN deve manter a consistência da representação, ou seja, atualizar as profundidades dos vértices da subárvore enraizada no elemento

removido e atualizar os valores dos *indicePai* e *indiceFilho* que correspondem a posições posteriores à do elemento removido; caso o vértice removido seja folha, atualizar o grau do *indicePai* do mesmo.

### 5.4.1 Complexidade do ORN

Esta Seção apresenta a análise de complexidade de tempo do Algoritmo 5.6 do ORN. Assim como, para o OCC, o tamanho da entrada do ORN é da $O(n)$. O melhor caso do Algoritmo 5.6 ocorre quando não é necessário remover nenhum vértice de Steiner. Nesse caso, o $loop$ da linha 2 possui uma única passagem que executa todas as iterações do $loop$ da linha 4. Esse $loop$, percorre todas as posições da *RNPG* de entrada e executa a verificação do condicional da linha 5 que terá resultado falso e seus comando internos não serão executados. Assim, no melhor caso a complexidade de tempo do algoritmo é $O(n)$.

No pior caso, a solução de entrada possui $n-2$ vértices de Steiner e todos eles precisarão ser removidos e cada um em uma passagem. Assim, o número de iterações do $loop$ da linha 2 do Algoritmo 5.6 é $O(n)$. Cada iteração busca na árvore pelos vértices de Steiner com grau menor que 3 no $loop$ da linha 4 com complexidade de tempo $O(n)$. Como, considerou-se que a cada iteração do $loop$ da linha 2 será removido um vértice de Steiner, o condicional da linha 5 será verdadeiro em somente uma das iterações do $loop$ de busca e nesse caso, terá complexidade de tempo $O(n)$, pois é preciso deslocar e atualizar os valores de todos os vértices na *RNPG*. As demais iterações do $loop$ de busca tem complexidade de tempo $O(1)$. Portanto, a complexidade total do ORN, no pior caso é $O(n) * (O(n) + O(n-1) * O(1)) = O(n^2)$. No entanto, esse caso é raro de ocorrer na prática.

O caso médio é complexo de ser analisado, pois não existe uma métrica para definir o número de vértices de Steiner que precisarão ser removidos. De forma simplificada, pode-se afirmar que se $k$ vértices precisarem ser removidos a complexidade média do ORN é $O(kn)$. Essa análise é similar a realizada para o pior caso com $k$ substituindo o valor de $n$ para o número de vértices a serem removidos.

CAPÍTULO 6

# Testes Empíricos e Resultados

Este Capítulo descreve os testes realizados, os casos de teste utilizados e os resultados obtidas. A heurística deste trabalho, denominada *SMTBeasleyRNPG*, implementa uma versão da solução proposta por Beasley em [7]. A principal diferença das duas propostas é que em *SMTBeasleyRNPG* é utilizada a representação nó-profundidade-grau (RNPG). Outra diferença é que não foi implementado na proposta deste trabalho a etapa da heurística de Beasley que analisa os subgrafos da solução obtida que são árvores de Steiner completas (FSTs), devido a complexidade dessa etapa.

A heurística *SMTBeasleyRNPG* foi desenvolvida na linguagem C. Os testes empíricos foram executados em um computador pessoal *Intel (R) Core (TM) i3-2120 CPU @ 3.30 GHz* com *4GB* de memória RAM e Sistema Operacional Windows 7. Com o objetivo de comparar os resultdos obtidos com a heurística original, foram utilizados os mesmos casos de teste de [7]. Esses casos de teste encontram-se disponíveis em [5].

Este capítulo está estruturado como segue. Os casos de teste são descritos na Seção 6.1. A Seção 6.2 apresenta como foram realizados os testes da heurística e os resultados obtidos em comparação com os apresentados em [7] e com a solução ótima obtida para alguns casos. A Seção 6.3 descreve o teste realizado e os resultados obtidos para comparação do desempenho prático do Operador de Consulta de Conjuntos (OCC) com as representações RNPG e Matriz de Adjacências. Por fim, a Seção 6.4 sumariza os resultados obtidos neste trabalho.

## 6.1 Casos de teste

Os casos de teste disponíveis em [5] são compostos por um conjunto de grafos euclidianos com número de vértices ($n$) de $10 \ldots 1000$. Os dados estão dispostos em arquivos de extensão *txt*, para cada número de vértices de grafo. Cada arquivo *txt* contém 15 instâncias para cada valor de $n$. Neste trabalho foram experimentados os 12 primeiros grafos com $n = 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 250, 500$. Os vértices que compõem cada um dos grafos especificados estão dispostas em um plano Cartesiano de duas dimensões de tamanho $1.0 \times 1.0cm$.

Cada arquivo correspondente ao grafo de ordem $n$ contem na primeira linha o número de instâncias (15). Cada uma das instâncias possui o padrão a seguir: uma linha com o número de vértices $n$; seguido de $n$ linhas com o par ordenado $(x,y)$ em números reais com 7 casas decimais, correspondente ao ponto do vértice no plano Cartesiano. Os grafos são considerados completos com o custo das arestas dado pela distância Euclidiana entre os vértices.

Para os casos de teste com $n$ de 10 a 100 também está disponível em [5] a solução ótima de cada instância em um arquivo *txt* para cada $n$. Cada arquivo contém na primeira linha o número de instâncias (15), e para cada instância o seguinte padrão: a primeira linha o custo total da árvore mínima de steiner (SMT); na próxima linha o custo total da árvore geradora mínima (MST); o número $s$ de vértices de Steiner adicionados na SMT está na linha seguinte; e as próximas $s$ linhas contém o par ordenado $(x,y)$ das coordenadas cartesianas dos vértices de Steiner com 10 casas decimais.

## 6.2 Comparativo entre *SMTBeasleyRNPG* e Algoritmo de Beasley

A heurística *SMTBeasleyRNPG* foi aplicada as 15 instâncias dos 12 números de vértices dos grafos descritos na Seção 6.1. Para cada instância analisada é cálculado o percentual de redução do custo da solução obtida pela heurística em relação ao custo da MST da instância. Esse cálculo é dado pela Equação 6-1, onde $T(V_0)$ é o custo da MST e $T(V_F)$ é o custo da solução final. O percentual de redução representa a qualidade da solução, visto que *MST*, como mostrado no Capítulo 3, é um padrão natural de comparação da eficácia das heurísticas para o problema de Steiner Euclidiano. Além disso, é computado o número de iterações executadas pela heurística e o número de vértices de Steiner presentes na solução.

$$100(T(V_0)-T(V_F))/T(V_0) \tag{6-1}$$

Os resultados obtidos pela *SMTBeasleyRNPG* são comparados com aqueles obtidos pela heurística original e relatados em [7]. Os parâmetros analisados para comparação das heurísticas são apresentados em termos de valores mínimo (Min.), menor valor entre das 15 instâncias, médio (Med.), média do resultado de todas as instâncias e máximo (Max.), maior valor obtido nas 15 instâncias, para cada grafo de $n$ vértices.

**Tabela 6.1:** *Número de iterações utilizadas pelas heurísticas Beasley e SMTBeasleyRNPG. Os valores da heurística Beasley foram obtidos em [7].*

| *Número de iterações* | | | | | | |
|---|---|---|---|---|---|---|
| | Beasley | | | *SMTBeasleyRNPG* | | |
| *n* | Min. | Med. | Max. | Min. | Med. | Max. |
| *10* | 2.000 | 3.000 | 4.000 | 2.000 | 6.933 | 40.000 |
| *20* | 3.000 | 3.600 | 5.000 | 3.000 | 14.200 | 35.000 |
| *30* | 3.000 | 3.533 | 5.000 | 3.000 | 10.000 | 21.000 |
| *40* | 3.000 | 5.133 | 21.000 | 4.000 | 12.600 | 35.000 |
| *50* | 3.000 | 4.067 | 7.000 | 3.000 | 12.333 | 23.000 |
| *60* | 3.000 | 4.533 | 11.000 | 3.000 | 14.667 | 29.000 |
| *70* | 3.000 | 4.133 | 6.000 | 4.000 | 12.067 | 26.000 |
| *80* | 3.000 | 4.467 | 9.000 | 4.000 | 11.800 | 33.000 |
| *90* | 3.000 | 4.733 | 8.000 | 4.000 | 14.533 | 26.000 |
| *100* | 4.000 | 5.333 | 9.000 | 7.000 | 16.800 | 32.000 |
| *250* | 4.000 | 5.867 | 15.000 | 8.000 | 15.867 | 25.000 |
| *500* | 5.000 | 6.400 | 9.000 | 13.000 | 17.067 | 22.000 |

A Tabela 6.1 apresenta o número de iterações executadas até que o critério de parada da heurística fosse atingido, ou seja, não houve redução do custo. Observa-se nos resultados obtidos que a heurística *SMTBeasleyRNPG* executou um número maior de iterações do que a proposta de Beasley até atingir o critério de parada. Visto que, o critério de parada é a diferença entre o custo da solução no inicio da iteração e ao final dela, ou seja, houve redução do custo total da solução, um dos motivos para número maior de iterações para *SMTBeasleyRNPG* é a inexistência do passo de ajuste da solução para os subgrafos que representam uma FST (passo 9 da heurística de Beasley).

A Tabela 6.2 mostra o percentual de redução alcançado calculado por meio da equação 6-1. Nessa Tabela o percentual obtido pela *SMTBeasleyRNPG* é comparado a redução obtida em [7] e a calculada para a árvore mínima de Steiner (SMT) ótima. Vale ressaltar que para as instâncias com 250 e 500 vértices não é reportado no repositório dos casos de teste SMT ótima. Por isso, para essas linhas não é apresentado o percentual de redução da solução ótima. Ao analisar os resultados obtidos, nota-se que as soluções de ambas as heurísticas possuem percentual de redução muito próximo do ótimo em média. Além desse comportamento médio em alguns casos os resultados mínimos e máximos foram iguais ou muito próximos aos da SMT. Pode-se observar, também, que a *SMTBeasleyRNPG* oferece soluções muito próximas das soluções obtidas pela heurística Beasley. Novamente, a diferença dos resultados deve ter como origem a ausência do ajuste das FSTs que ocorre na heurística Beasley.

**Tabela 6.2:** *Percentual de redução em relação a MST para as heurísticas* SMTBeasleyRNPG*, Beasley e a SMT. Os valores da heurística Beasley foram obtidos em [7]*

| *Redução (%)* | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Beasley | | | *SMTBeasleyRNPG* | | | Redução Ótima com SMT | | |
| *n* | Min. | Med. | Max. | Min. | Med. | Max. | Min. | Med. | Max. |
| *10* | 0.477 | 3.138 | 6.168 | 0.477 | 3.140 | 6.168 | 0.477 | 3.251 | 6.168 |
| *20* | 1.389 | 3.015 | 4.737 | 1.384 | 2.958 | 4.367 | 1.551 | 3.156 | 4.758 |
| *30* | 2.059 | 2.868 | 4.752 | 2.059 | 2.841 | 4.639 | 2.083 | 3.067 | 4.964 |
| *40* | 1.669 | 3.024 | 4.174 | 1.643 | 3.023 | 4.135 | 1.856 | 3.139 | 4.407 |
| *50* | 2.138 | 2.841 | 3.620 | 2.053 | 2.806 | 3.687 | 2.475 | 3.003 | 3.786 |
| *60* | 2.347 | 2.946 | 3.576 | 2.283 | 2.880 | 3.502 | 2.569 | 3.275 | 3.872 |
| *70* | 2.142 | 2.844 | 3.592 | 2.110 | 2.808 | 3.591 | 2.368 | 3.110 | 3.823 |
| *80* | 1.973 | 2.817 | 4.288 | 1.973 | 2.792 | 4.147 | 2.041 | 3.039 | 4.406 |
| *90* | 1.989 | 2.935 | 3.687 | 1.990 | 2.897 | 3.671 | 2.091 | 3.120 | 4.055 |
| *100* | 2.286 | 2.952 | 3.467 | 2.265 | 2.928 | 3.460 | 2.663 | 3.269 | 3.977 |
| *250* | 2.611 | 2.950 | 3.279 | 2.566 | 2.922 | 3.215 | | | |
| *500* | 2.668 | 3.052 | 3.316 | 2.661 | 3.020 | 3.280 | | | |

**Tabela 6.3:** *Número de vértices de Steiner adicionados por Beasley e SMTBeasleyRNPG. Os valores de Beasley foram obtidos em [7]*

| *Número de Pontos de Steiner* | | | | | | |
|---|---|---|---|---|---|---|
| | Beasley | | | *SMTBeasleyRNPG* | | |
| *n* | Min. | Med. | Max. | Min. | Med. | Max. |
| *10* | 2.000 | 3.400 | 6.000 | 2.000 | 3.400 | 6.000 |
| *20* | 6.000 | 7.667 | 10.000 | 6.000 | 7.933 | 10.000 |
| *30* | 9.000 | 11.667 | 16.000 | 9.000 | 11.800 | 15.000 |
| *40* | 13.000 | 15.733 | 18.000 | 13.000 | 15.733 | 18.000 |
| *50* | 16.000 | 19.333 | 22.000 | 16.000 | 19.533 | 22.000 |
| *60* | 18.000 | 23.733 | 28.000 | 19.000 | 24.133 | 27.000 |
| *70* | 22.000 | 26.467 | 31.000 | 23.000 | 27.133 | 33.000 |
| *80* | 28.000 | 31.667 | 35.000 | 28.000 | 31.733 | 37.000 |
| *90* | 32.000 | 36.733 | 41.000 | 31.000 | 37.467 | 44.000 |
| *100* | 32.000 | 39.333 | 47.000 | 34.000 | 39.933 | 46.000 |
| *250* | 92.000 | 98.467 | 106.000 | 93.000 | 99.933 | 105.000 |
| *500* | 190.000 | 200.533 | 209.000 | 189.000 | 202.133 | 212.000 |

A Tabela 6.3 apresenta o número de vértices de Steiner adicionados pelas heurísticas. Pode-se observar que ambas adicionam o mesmo número de vértices de Steiner na maioria dos casos de teste. Em alguns valores médios nota-se que a heurística *SMTBeasleyRNPG* adicionou pontos a mais do que a heurística Beasley. Esses resultados evidenciam que a inexistência do passo 9 da heurística de Beasley não tem grande

influência na quantidade de vértices de Steiner adicionados. Comparando os dados dessa Tabela juntamente com os das Tabelas 6.2 e 6.1 é possível considerar que a principal contribuição do passo 9, de ajuste das FSTs é no posicionamento ótimo dos vértices de Steiner para obter melhores reduções e consequentemente, reduzir o número de iterações.

**Tabela 6.4:** *Custo total da MST; custo total, percentual de redução e número de vértices de Steiner para a SMT e a heurística SMTBeasleyRNPG de todas as instâncias de grafo com 10 vértices.*

| $n = 10$ | *MST* | *SMT* | | | *SMTBeasleyRNPG* | | |
|---|---|---|---|---|---|---|---|
| Instância | Custo | Custo | Red. | Núm. Vért. | Custo | Red. | Núm. Vért. |
| *1* | 2.111465623 | 2.020673795 | 4.30% | 4 | 2.020674 | 4.30% | 4 |
| *2* | 1.614569662 | 1.606868229 | 0.48% | 2 | 1.606868 | 0.48% | 2 |
| *3* | 2.330090542 | 2.228074322 | 4.38% | 2 | 2.228115 | 4.38% | 2 |
| *4* | 1.819524699 | 1.798596253 | 1.15% | 2 | 1.798596 | 1.15% | 2 |
| *5* | 1.737172643 | 1.694433309 | 2.46% | 3 | 1.694433 | 2.46% | 3 |
| *6* | 2.421164591 | 2.309602568 | 4.61% | 4 | 2.332072 | 3.68% | 3 |
| *7* | 2.337311041 | 2.233858599 | 4.43% | 4 | 2.233859 | 4.43% | 4 |
| *8* | 2.212775434 | 2.177682904 | 1.59% | 3 | 2.177683 | 1.59% | 3 |
| *9* | 2.018842093 | 1.968478249 | 2.49% | 5 | 1.980363 | 1.91% | 4 |
| *10* | 2.100914567 | 2.059331690 | 1.98% | 4 | 2.059332 | 1.98% | 4 |
| *11* | 2.060383637 | 1.947322109 | 5.49% | 4 | 1.949903 | 5.36% | 5 |
| *12* | 1.763325148 | 1.753123664 | 0.58% | 2 | 1.753124 | 0.58% | 2 |
| *13* | 1.826538973 | 1.713886731 | 6.17% | 3 | 1.713887 | 6.17% | 3 |
| *14* | 2.065341690 | 1.949652208 | 5.60% | 6 | 1.94974 | 5.60% | 6 |
| *15* | 1.724564481 | 1.671645607 | 3.07% | 4 | 1.671675 | 3.07% | 4 |

A Tabela 6.4 apresenta os resultados para as 15 instâncias utilizadas para o caso de teste de ordem 10. Os dados apresentados são compostos pelos valores de custo total da MST, da SMT ótima e da solução obtida pela heurística *SMTBeasleyRNPG*. Além disso, para a SMT e a heurística também apresenta o percentual de redução em função da *MST* e o número de vértices de Steiner adicionados. O propósito dessa Tabela é o de verificar o desempenho da heurística proposta para uma mesma ordem de grafo com diferentes coordenadas cartesianas para os vértices em comparação com a solução ótima conhecida.

Os dados da Tabela 6.4 mostram que, para os grafos de 10 vértices, a heurística *SMTBeasleyRNPG* oferece a solução ótima para a maioria das instâncias. Para uma das instância de número 11 a heurística adicionou mais vértices de Steiner do que a solução ótima, no entanto, isso não resultou em uma redução maior do custo total da solução. Esse resultado evidencia que não é suficiente adicionar vértices de Steiner para reduzir o custo, mas que eles devem estar na posição correta para gerar reduções. Nas instâncias 6 e 9 a heurística adicionou menos vértices do Steiner do que o necessário para obter a solução ótima. Nesses casos, também, o custo das soluções heurísticas foi maior do que o custo da

solução ótima. No entanto, por se tratar de uma heurística não é esperado que o resultado seja a solução ótima em todos os casos, mas sim soluções próximas do ótimo, o qual foi o comportamento geral, tanto da heurística Beasley, quando da versão desenvolvida neste trabalho.

## 6.3 Análise do tempo de execução real do OCC com RNPG e Matriz de Adjacências

Com o objetivo de avaliar empiricamente o tempo de execução do operador de consulta de conjuntos (OCC) em suas versões presentes nos Algoritmos 5.1 e e 5.5, com RNPG e Matriz de Adjacências, respectivamente. Ambas as versões foram implementadas em linguagem C e serão avaliadas isoladamente do contexto geral da heurística. Vale lembrar que nos resultados apresentados anteriormente a heurística *SMTBeasleyRNPG* faz uso do operador com RNPG. Os testes foram desenvolvidos com o mesmo conjunto de grafos utilizado anteriormente acrescido do grafo de $n = 1000$ vértices. O tempo de execucação foi medido na execução do OCC com a entrada da primeira iteração da heurística, que corresponde a MST do grafo analisado.

**Tabela 6.5:** *Tempo de execução do OCC implementado utilizando a RNPG e a representação convencional de Matriz de Adjacências.*

| | RNPG | | | Matriz de Adjacências | | |
|---|---|---|---|---|---|---|
| *n* | Min. | Med. | Max. | Min. | Med. | Max. |
| *10* | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00033 | 0.00100 |
| *20* | 0.00000 | 0.00000 | 0.00000 | 0.00000 | 0.00026 | 0.00100 |
| *30* | 0.00000 | 0.00013 | 0.00100 | 0.00000 | 0.00040 | 0.00200 |
| *40* | 0.00000 | 0.00007 | 0.00100 | 0.00000 | 0.00033 | 0.00100 |
| *50* | 0.00000 | 0.00013 | 0.00100 | 0.00000 | 0.00053 | 0.00100 |
| *60* | 0.00000 | 0.00013 | 0.00100 | 0.00000 | 0.00033 | 0.00100 |
| *70* | 0.00000 | 0.00007 | 0.00100 | 0.00000 | 0.00040 | 0.00100 |
| *80* | 0.00000 | 0.00020 | 0.00100 | 0.00000 | 0.00073 | 0.00100 |
| *90* | 0.00000 | 0.00007 | 0.00100 | 0.00000 | 0.00060 | 0.00100 |
| *100* | 0.00000 | 0.00007 | 0.00100 | 0.00000 | 0.00073 | 0.00100 |
| *250* | 0.00000 | 0.00033 | 0.00100 | 0.00200 | 0.00273 | 0.00300 |
| *500* | 0.00000 | 0.00060 | 0.00100 | 0.00800 | 0.00893 | 0.00900 |
| *1000* | 0.00100 | 0.00160 | 0.00300 | 0.03200 | 0.03420 | 0.05200 |

A Tabela 6.5 mostra o tempo de execução da implementação dos Algoritmos 5.1 e 5.5 do OCC. A primeira coluna contém o tamanho da entrada, expressa pelo número de vértices. Para cada algoritmo é apresentado os tempos mínimo, médio e máximo, pois para

cada tamanho de entrada existem 15 instâncias com pesos e características topológicas diferentes para a MST de entrada. O tempo de execução é medido em segundos.

Os resultados apresentados na Tabela 6.5 comprovam o resultado teórico da análise de complexidade de tempo, de que a implementação com a RNPG é mais eficiente do que a implementação com Matriz de Adjcências. Ou seja, ao utilizar a RNPG o OCC demanda menor tempo computacional. Em todos os casos de teste os tempos de execução mínimo, máximo e médio do Algoritmo 5.1 foram inferiores ou iguais aos do Algoritmo 5.5.

A Figura 6.1 apresenta o gráfico com os valores médios do tempo de execução em segundos do OCC com RNPG e com Matriz de Adjacências de acordo com a Tabela 6.5. Observa-se, pelo comportamento assintótico apresentado na Figura 6.1, que o tempo médio de execução do Algoritmo 5.1, com RNPG, possui uma tendência de crescimento linear comparado ao Algoritmo 5.5 que apresenta uma tendência de crescimento quadrático. Os resultados empíricos do tempo de execução sugerem que na prática os algoritmos demandam um esforço computacional inferior ao determinado pela análise de complexidade.

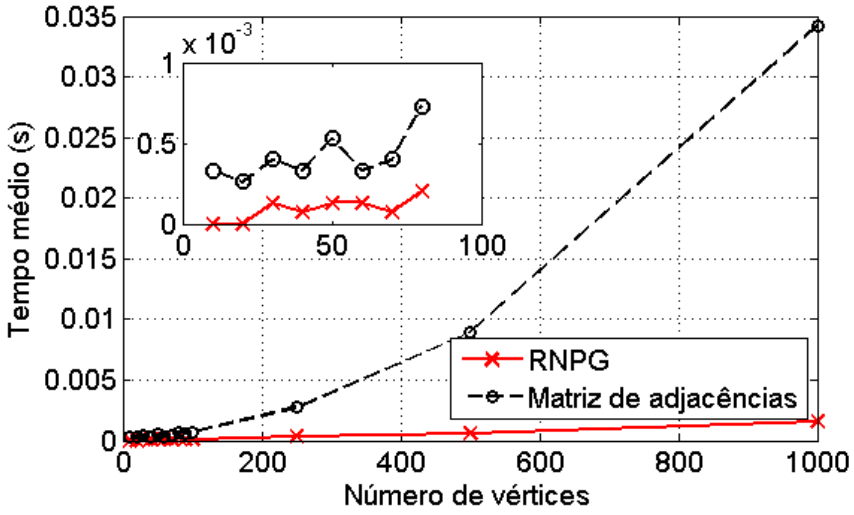


**Figura 6.1:** *Gráfico comparativo entre RNPG e Matriz de Adjacências*

Na Figura 6.2 é apresentado o gráfico do percentual do tempo de execução do OCC com Matriz de Adjacências em função do OCC com RNPG. Esse percentual é calculado da seguinte forma $100 \times (\frac{M_{ma}}{M_r})$, onde $M_{ma}$ é o tempo de execução médio do OCC com Matriz de Adjacências e $M_r$ o tempo de execução médio do OCC com RNPG. O gráfico não apresenta os pontos para os grafos de $n = 10$ e $n = 20$, pois o tempo de execução do OCC com RNPG foi zero.

Observa-se que na melhor situação o OCC com Matriz de Adjecências é quase 500% mais lento do que o OCC com RNPG. Além disso, conforme aumenta o número de

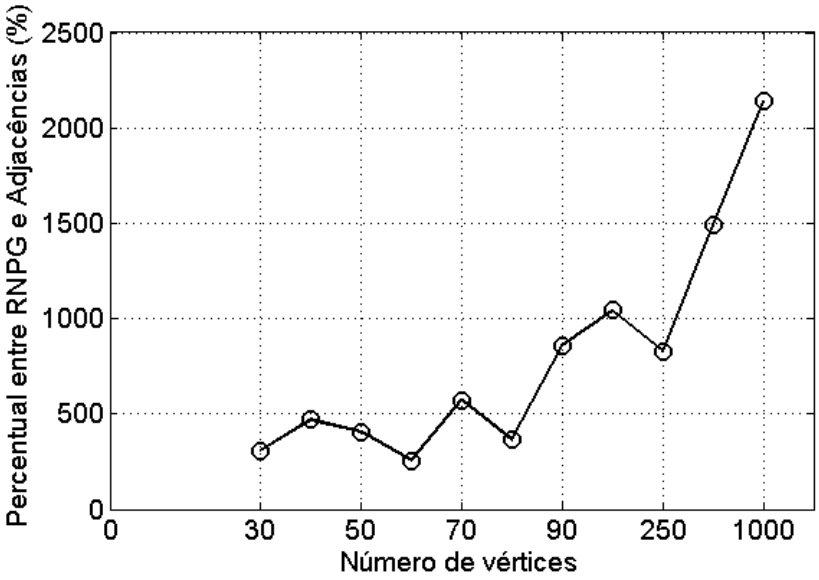


**Figura 6.2:** *Comparativo percentual de tempo gasto pelo OCC com Matriz de Adjacências em função do OCC com RNPG.*

vértices dos grafos aumenta a diferença entre os algoritmos, com o OCC com Matriz de Adjacências sendo quase 2500% mais lento do que o OCC com RNPG.

## 6.4 Considerações Finais

Os testes realizados e descritos neste Capítulo tem por propósito avaliar a heurística desenvolvida neste trabalho para o problema de árvore de Steiner euclidiano com a RNPG. Os resultados obtidos mostram que é viável utilizar a RNPG como estrutura de dados em uma heurística exata de forma satisfatória para o problema em foco. Além disso, evidenciam que foi possível reduzir o custo operacional da busca pelos conjuntos de quatro vértices.

CAPÍTULO 7

# Considerações Finais

O problema de árvore de Steiner possui diversas aplicações no mundo real em problemas onde é necessário reduzir o custo de ligação da rede. No entanto, trata-se de um problema da classe NP-Difícil para a qual, até o momento, não é fácil obter um algortimo exato em tempo polinomial determinístico. No intuito de reduzir a complexidade desse problema diversas reduções e variações foram desenvolvidas, dentre elas o problema de árvore de Steiner euclidiano. Nesse problema, os vértices são pontos no plano cartesiano e o peso das arestas é a distância euclidiana entre esses pontos. Porém, essas variações também foram mostradas pertencer a classe de problemas NP-Difícil.

Assim, como esses problemas possuem uma série de aplicações reais, as pesquisas tem desenvolvidos heurísticas para obter soluções aceitáveis em tempo computacional viável. Como já apresentado neste trabalho, o uso de representações específicas para grafos, mais diretamente para árvores em grafos tem contribuído para melhorar a eficiência computacional das heurísticas. Dentre as representações que tem sido desenvolvidas, destaca-se a representação nó-profundidade-grau (RNPG).

Este trabalho propõem a verificação da viabilidade de aplicar a (RNPG) para desenvolver uma heurística para o problema de árvore de Steiner euclidiano. Como o foco do trabalho é o uso de uma representação para melhorar a eficiência de uma heurística escolheu-se uma heurística existente para desenvolver a solução proposta. A heurística escolhida foi proposta por Beasley em [7] com resultados satisfatórios e próximos a solução ótima. Em alguns casos é possível obter a solução ótima com essa heurística. Além disso, a heurística de Beasley está entre as citadas nas pesquisas posteriores.

Para viabilizar o desenvolvimento da heurística de Beasley com a RNPG, foi necessária a proposta de operadores específicos para realizar alguns dos passos importantes da heurística. Esses operadores são OCC, responsável por identificar os subgrafos de quatro vértices e ORN, para remoção dos vértices de Steiner que ficarem com grau menor do que três. No entanto, devido a complexidade não foi possível desenvolver o operador de ajuste da posição dos vértices de Steiner nas subárvores de Steiner completas. A heurística resultante foi denominada *SMTBeasleyRNPG*.

Uma das contribuições para a eficiência da heurística atribuídas à utilização da

RNPG é no operador OCC. As subárvores de quatro vértices são encontradas de três formas possíveis. A primeira encontra facilmente as subárvores a partir da navegação pelo dado *indicePai*, adicionado a RNPG. No segundo e terceiro casos a RNPG limita a busca na lista por subárvores até o índice indicado pelo *indiceFilho* do vértice. A RNPG é essencial para identificar quando é possível executar cada uma dessas operações. A análise de complexidade mostrou que o OCC com a RNPG possui tempo de execução $O(n^2)$. Foi também desenvolvida uma versão do OCC com a representação de matriz de adjacências e essa possui complexidade de tempo de execução $O(n^4)$.

No operador ORN um vértice é facilmente removido, arrastando os elementos para a posição anterior a partir do primeiro elemento subseqüente ao removido. A RNPG mantém sua estrutura apenas com três ressalvas: caso o elemento removido seja uma folha, o vértice pai deve ter seu grau diminuído em uma unidade; o elemento que compor uma subárvore do vértice removido deve ter suas profundidade decrescidas em uma unidade para se tornar subárvore do pai do vértice removido; os índices *indicePai* e *indiceFilho* devem ser atualizados.

Os resultados descritos neste trabalho mostram que foi possível aplicar a RNPG para o problema de árvore de Steiner Euclidiano com resultados satisfatórios onde as soluções aproximam-se da solução ótima e das soluções obtidas por Beasley. Os operadores OCC e ORN já indicam algumas vantagens na utilização da RNPG em relação com outras representações convencionais de grafos, como eficiência na consulta de subárvores e facilidade de remoção de nó. Além dessa nova classe de problemas, os resultados indicam que é viável utilizar a RNPG em outras técnicas heurísticas.

Como trabalhos futuros sugere-se:

- Implementar o passo (9) da heurística de Beasley.
- Melhorar a eficiência computacional do ORN;
- Estudar as outras heurísticas para o problema de Steiner Euclidiano e verificar a viabilidade de uso da RNPG;
- Estudar aplicações reais do problema de Steiner Euclidiano e desenvolver soluções heurísticas para as mesmas.

# Referências Bibliográficas


[1] ABUALI, F. N.; SCHOENEFELD, D. A.; WAINWRIGHT, R. L. **Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees**. In: *Proceedings of the 1994 ACM symposium on Applied computing - SAC '94*, p. 242–246, New York, New York, USA, 1994. ACM Press.

[2] A.E. CALDWELL.; KAHNG, A.; MANTIK, S.; MARKOV, I.; ZELIKOVSKY, A. **On wirelength estimations for row-based placement**. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(9):1265–1278, 1999.

[3] BARREIROS, J. **An hierarchic genetic algorithm for computing (near) optimal Euclidean Steiner trees**. In: *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, p. 8, 2003.

[4] BEAN, J. **Genetic algorithms and random keys for sequencing and optimization**. *ORSA journal on computing*, 6(2):154–160, 1994.

[5] BEASLEY, J. E. **OR-library: Distributing test problems by electronic mail**. *Journal of the Operational Research Society*, 41(11):1069–1072, 1990.

[6] BEASLEY, J. E.; GOFFINET, F. **A delaunay triangulation-based heuristic for the euclidean steiner problem**. *Networks*, 24(4):215–224, 1994.

[7] BEASLEY, J. **A heuristic for Euclidean and rectilinear Steiner problems**. *European Journal of Operational Research*, 58:284– 292, 1992.

[8] CHANG, S. **The generation of minimal trees with a Steiner topology**. *Journal of the ACM (JACM)*, 19(4):699–711, 1972.

[9] CHERITON, D.; TARJAN, R. E. **Finding Minimum Spanning Trees**. *SIAM Journal on Computing*, 5(4):724–742, Dec. 1976.

[10] COCKAYNE, E.; HEWGILL, D. **Exact computation of Steiner minimal trees in the plane**. *Information Processing Letters*, 22(March):151–156, 1986.

[11] CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. **Introduction to Algorithms, Third Edition**. MIT Press, 2009.

[12] COURANT, R.; ROBBINS, H. **What Is Mathematics?** Oxford University Press, USA, 1969.

[13] CURRENT, J.; MARSH, M. **Multiobjective transportation network design and routing problems: Taxonomy and annotation**. *European Journal of Operational Research*, 65(1):4–19, 1993.

[14] DELBEM, A. C. B.; DE CARVALHO, A.; BRETAS, N. G. **Main chain representation for evolutionary algorithms applied to distribution system reconfiguration**. *Power Systems, IEEE Transactions on*, 20(1):425–436, 2005.

[15] DELBEM, A.; CARVALHO, A. D. **Node-depth encoding for evolutionary algorithms applied to network design**. In: *Genetic and Evolutionary Computation Conference (GECCO 2004)*, p. 678–687, 2004.

[16] DELBEM, A. C. B.; LIMA, T. W. D.; TELLES, G. P. **Efficient Forest Data Structure for Evolutionary Algorithms Applied to Network Design**. *IEEE Transactions on Evolutionary Computation*, 16(6):829–846, 2012.

[17] DREYER, D.; OVERTON, M. **Two heuristics for the Euclidean Steiner tree problem**. *Journal of Global Optimization*, p. 1–14, 1998.

[18] DU, D.-Z.; ZHANG, Y. **On better heuristics for Steiner minimum trees**. *Mathematical Programming*, 57(1-3):193–202, May 1992.

[19] ESBENSEN, H. **Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm**. *Networks*, 26(4):173–185, 1995.

[20] FORTUNE, S. **Voronoi diagrams and Delaunay triangulations**. In: Goodman, J. E.; O'Rourke, J., editors, *Handbook of discrete and computational geometry*, chapter Voronoi di, p. 377–388. CRC Press, Inc., Boca Raton, FL, USA, second edition, 1997.

[21] FOULDS, L. R.; RAYWARD-SMITH, V. J. **Steiner Problems in Graphs: Algorithms and Applications**. *Engineering Optimization*, 7(1):7–16, Jan. 1983.

[22] FOULDS, L.; GRAHAM, R. **The Steiner problem in phylogeny is NP-complete**. *Advances in Applied Mathematics*, 3:43–49, 1982.

[23] FROMMER, I.; GOLDEN, B. **A Genetic Algorithm for Solving the Euclidean Non-Uniform Steiner Tree Problem**. *Extending the Horizons: Advances in Computing, Optimization, and Decision Technologies - Operations Research/Computer Science Interfaces Series*, 37:31–48, 2007.

[24] GAREY, M. R.; GRAHAM, R. L.; JOHNSON, D. S. **The Complexity of Computing Steiner Minimal Trees**. *SIAM Journal on Applied Mathematics*, 32(4):835–859, 1977.

[25] GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability; A Guide to the Theory of NP-Completeness**. W. H. Freeman & Co., New York, NY, USA, 1990.

[26] GAREY, M.; JOHNSON, D. **The rectilinear Steiner tree problem is NP-complete**. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[27] GILBERT, E. N. **Minimum Cost Communication Networks**. *Bell System Technical Journal*, 46(9):2209–2227, 1967.

[28] GILBERT, E. N.; POLLAK, H. O. **Steiner Minimal Trees**. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[29] GORDEEV, E. N.; TARASTSOV, O. G. **The Steiner problem: A survey**. *Discrete Math. Appl.*, 3(4):339–364, 1993.

[30] GOTTLIEB, J.; JULSTROM, B. A.; RAIDL, G. R.; ROTHLAUF, F. **Prüfer numbers: A poor representation of spanning trees for evolutionary search**. In: *Genetic and Evolutionary Computation Conference (GECCO 2001)*, p. 343–350, San Francisco, California, USA, 2001.

[31] GRIMWOOD, G. **The Euclidean Steiner tree problem: Simulated annealing and other heuristics**. PhD thesis, 1994.

[32] HANAN, M. **On Steiner's problem with rectilinear distance**. *SIAM Journal on Applied Mathematics*, 14(2):255–265, 1966.

[33] HAOUARI, M.; CHAOUACHI SIALA, J. **A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem**. *Computers & Operations Research*, 33(5):1274–1288, May 2006.

[34] HWANG, F. K. **On steiner minimal trees with rectilinear distance**. *SIAM Journal of Applied Mathematics*, 30:104–114, 1976.

[35] HWANG, F. **A linear time algorithm for full Steiner trees**. *Operations Research Letters*, 4(5):235–237, 1986.

[36] HWANG, F.; RICHARDS, D. **Steiner tree problems**. *Networks*, 22:55–89, 1992.

[37] HWANG, F. K.; RICHARDS, D. S.; WINTER, P. **The Steiner Tree Problem**. Elsevier Science Publishers B. V., North Holland, 1992.

[38] Jünger, M.; Liebling, T. M.; Naddef, D.; Nemhauser, G. L.; Pulleyblank, W. R.; Reinelt, G.; Rinaldi, G.; Wolsey, L. A., editors. **50 Years of Integer Programming**. Springer, 2010.

[39] KAHNG, A.; ROBINS, G. **On optimal interconnections for VLSI**. 1994.

[40] KAPSALIS, A.; RAYWARD-SMITH, V. J.; SMITH, G. D. **Solving the graphical Steiner tree problem using genetic algorithms**. *Journal of the Operational Research Society*, 44(4):397–406, 1993.

[41] KARP, R. M. **Reducibility among combinatorial problems**. In: *Complexity of Computer Computations, Miller,Tatcher, Plenum*. 1974.

[42] KORTE, B.; PRÖMEL, H. J.; STEGER, A. **Steiner trees in VLSI-layout**. Report. Sonderforschungsbereich 303 Information und die Koordination Wirtschaftlicher Aktivit{ä}ten, Universit{ä}t Bonn. Sonderforschungsber., Univ., 1989.

[43] KRUSKAL, J. **On the shortest spanning subtree of a graph and the traveling salesman problem**. *Proceedings of the American Mathematical society*, 5:1955–1957, 1956.

[44] LAARHOVEN, J. W.; OHLMANN, J. W. **A randomized Delaunay triangulation heuristic for the Euclidean Steiner tree problem in Rd**. *Journal of Heuristics*, 17(4):353–372, July 2010.

[45] LAARHOVEN, J. V. **Exact and heuristic algorithms for the Euclidean Steiner tree problem**. PhD thesis, 2010.

[46] LENGAUER, T. **Combinatorial algorithms for integrated circuit layout**. John Wiley \& Sons, Inc., New York, NY, USA, 1 edition, 1990.

[47] LIMA, T. W. D. **Estruturas de Dados Eficientes para Algoritmos Evolutivos Aplicados a Projeto de Redes**. 2009.

[48] LU, C. L.; TANG, C. Y.; LEE, R. C.-T. **The full Steiner tree problem**. *Theoretical Computer Science*, 306(1-3):55–67, Sept. 2003.

[49] LUNDY, M. **Applications of the annealing algorithm to combinatorial problems in statistics**. *Biometrika*, 72(1):191–198, 1985.

[50] MAGNANTI, T. L.; WONG, R. T. **Network design and transportation planning: Models and algorithms**. *Transportation Science*, 18:1–56, 1984.

[51] MELZAK, Z. A. **On the problem of Steiner**. *Canadian Mathematical Bulletin*, 4(2):143–148, Jan. 1961.

[52] MELZAK, Z. **Companion to concrete mathematics**. John Wiley & Sons, Inc, 2012.

[53] PALMER, C. C.; KERSHENBAUM, A. **Representing trees in genetic algorithms**. In: *Proceedings of the First IEEE Conference on Evolutionary Computation*, p. 379–384. IEEE Press, 1994.

[54] PICCIOTTO, S. **How to encode a tree**. PhD thesis, 1999.

[55] PRIM, R. C. **Shortest Connection Networks And Some Generalizations**. *The Bell System Technical Journal*, p. 13, 1957.

[56] RAIDL, G.; JULSTROM, B. **Edge sets: an effective evolutionary coding of spanning trees**. *IEEE Transactions on Evolutionary Computation*, 7(3):225 – 239, 2003.

[57] RICHARDS, D. **Fast heuristic algorithms for rectilinear Steiner trees**. *Algorithmica*, p. 191–207, 1989.

[58] ROBINS, G.; ZELIKOVSKY, A. **Improved Steiner tree approximation in graphs**. In: *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, p. 770–779, 2000.

[59] ROTHLAUF, F. **Representations for Genetic and Evolutionary Algorithms**. Springer-Verlag, Netherlands, 2 edition, 2006.

[60] ROTHLAUF, F.; GOLDBERG, D. E. **Pruefer Numbers and Genetic Algorithms: A Lesson on How the Low Locality of an Encoding Can Harm the Performance of GAs**. In: *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, p. 395–404, London, UK, UK, 2000. Springer-Verlag.

[61] ROTHLAUF, F.; GOLDBERG, D. E.; HEINZL, A. **Network random keys: A tree representation scheme for genetic and evolutionary algorithms**. *Evol. Comput.*, 10(1):75–97, Mar. 2002.

[62] SHAMOS, M.; HOEY, D. **Closest-point problems**. In: *Foundations of Computer Science, 1975., 16th Annual Symposium on*, p. 151–162, 1975.

[63] SMITH, J. M.; LEE, D. T.; LIEBMAN, J. S. **An O(n log n) heuristic for steiner minimal tree problems on the euclidean metric**. *Networks*, 11(1):23–39, 1981.

[64] SMITH, J. **Generalized Steiner network problems in engineering design**. In: *Design optimization*, p. 312. Academic Press, Inc, 1985.

[65] SMITH, W. D.; SHOR, P. W. **Steiner tree problems**. *Algorithmica*, 7(1-6):329–332, June 1992.

[66] STANOJEVIĆ, M.; VUJOŠEVIĆ, M. **An exact algorithm for steiner tree problem on graphs**. *International Journal of Computers, Communications & Control*, I(1):41–46, June 2006.

[67] THOMPSON, E. A. **The method of minimum evolution.** *Annals of human genetics*, 36(3):333–40, Jan. 1973.

[68] Toth, P.; Vigo, D., editors. **The vehicle routing problem**. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[69] TZSCHOPPE, C.; ROTHLAUF, F.; PESCH, H. **The edge-set encoding revisited: On the bias of a direct representation for trees**. In: *Genetic and Evolutionary Computation Conference (GECCO 2004)*, p. 258–270, 2004.

[70] VINGRON, M.; STOYE, J.; LUZ, H.; BÖCKER, S. **Algorithms for phylogenetic reconstructions**. *Lecture, Notes and Exercises*.

[71] WANG, X. **Exact algorithms for Steiner tree problem**. PhD thesis, University of Twente, Enschede, The Netherlands, June 2008.

[72] WARME, D. M.; WINTER, P.; ZACHARIASEN, M. **Exact Algorithms for Plane Steiner Tree Problems: A Computational Study**. In: *Advances in Steiner Trees*, número April, p. 81–116, 1998.

[73] ZACHARIASEN, M. **Algorithms for plane steiner tree problems**. Technical report, 1998.

[74] ZACHARIASEN, M. **Local search for the Steiner tree problem in the Euclidean plane**. *European Journal of Operational Research*, 119:282–300, 1999.

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