

UNIVERSIDADE FEDERAL DA PARAÍBA  
CENTRO DE CIÊNCIAS EXATAS E DA NATUREZA  
DEPARTAMENTO DE INFORMÁTICA  
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

**MÉTODOS HEURÍSTICOS APLICADOS AO PROBLEMA DA  
ÁRVORE DE STEINER RECTILINEAR**

THIAGO GOUVEIA DA SILVA

João Pessoa-PB  
Agosto-2009

THIAGO GOUVEIA DA SILVA

**MÉTODOS HEURÍSTICOS APLICADOS AO PROBLEMA DA  
ÁRVORE DE STEINER RECTILINEAR**

Dissertação de Mestrado apresentada ao Centro de Ciências Exatas e da Natureza da Universidade Federal da Paraíba, como requisito parcial para obtenção do Título de Mestre em Informática (Sistemas de Computação).

**Orientador:** Prof. Dr. Lucídio dos Anjos Formiga Cabral

**Banca Examinadora:**

Prof. Dr. Lucídio dos Anjos Formiga Cabral (UFPB)

Prof. Dr. Antonio Carlos Cavalcanti (UFPB)

Prof. Dr. Júlio Francisco Barros Neto

João Pessoa-PB

Agosto-2009

S586m Silva, Thiago Gouveia da.  
Métodos heurísticos aplicados ao problema da árvore de  
Steiner rectilinear / Thiago Gouveia da Silva.- João Pessoa,  
2009.  
108f. : il.

Orientador: Lucídio dos Anjos Formiga Cabral  
Dissertação (Mestrado) – UFPB/CCEN  
1. Informática. 2. Árvores Retilíneas de Steiner. 3.  
Metaheurística. 4. Simulated Annealing. 5. Algoritmos  
Genéticos.

UFPB/BC

CDU: 004(043)

*Ata da Sessão Pública de Defesa de Dissertação de Mestrado do aluno Thiago Gouveia da Silva, candidato ao Título de Mestre em Informática na Área de Sistemas de Computação, realizada em 28 de agosto de 2009.*




1 Aos vinte e oito dias do mês de agosto do ano dois mil e nove, às nove horas, na Sala de  
2 Reuniões do Centro de Ciências Exatas e da Natureza da Universidade Federal da Paraíba,  
3 reuniram-se os membros da Banca Examinadora constituída para examinar o candidato ao  
4 grau de Mestre em Informática, na área de “Sistemas de Computação”, na linha de pesquisa  
5 “Computação Distribuída”, o Sr. Thiago Gouveia da Silva. A comissão examinadora foi  
6 composta pelos professores doutores: Lucídio dos Anjos Formiga Cabral (DI-UFPB),  
7 Orientador e Presidente da Banca Examinadora, Antonio Carlos Cavalcanti (DI-UFPB),  
8 como examinador interno e Júlio Francisco Barros Neto (UFC), como examinador externo.  
9 Dando início aos trabalhos, o Prof. Lucídio dos Anjos Formiga Cabral, cumprimentou os  
10 presentes, comunicou aos mesmos a finalidade da reunião e passou a palavra ao candidato  
11 para que o mesmo fizesse, oralmente, a exposição do trabalho de dissertação intitulado  
12 “MÉTODOS HEURÍSTICOS APLICADOS AO PROBLEMA DA ÁRVORE DE STEINER  
13 RECTILINEAR”. Concluída a exposição, o candidato foi argüido pela Banca Examinadora  
14 que emitiu o seguinte parecer: “aprovado”. Assim sendo, deve a Universidade Federal da  
15 Paraíba expedir o respectivo diploma de Mestre em Informática na forma da lei e, para  
16 constar, eu, professora Tatiana Aires Tavares, coordenadora do Programa de Pós-  
17 Graduação em Informática, servindo de secretária, lavrei a presente ata que vai assinada por  
18 mim mesmo e pelos membros da Banca Examinadora. João Pessoa, 28 de agosto de 2009.

19   
20 Tatiana Aires Tavares

21  
Prof. Dr. Lucídio dos Anjos Formiga Cabral  
Orientador (DI-UFPB)

Prof. Dr. Antonio Carlos Cavalcanti  
Examinador Interno (DI-UFPB)

Prof. Dr. Júlio Francisco Barros Neto  
Examinador Externo (UFC)

## RESUMO

### RESUMO

Este trabalho apresenta uma nova heurística, denominada Heurística 1, e a implementação das metaheurísticas GRASP, *Simulated Annealing* e Algoritmos Genéticos para o problema da árvore retilínea mínima de Steiner (RSMTP), discutindo sobre seus aspectos teóricos, como a complexidade computacional; e práticos, como pseudocódigos e estratégias de implementação. As novas abordagens para o RSMTP apresentadas, em especial os Algoritmos Genéticos, ostentam resultados computacionais de qualidade superior às apresentadas pelas melhores heurísticas da literatura atual.

**Palavras-chave:** Árvores Retilíneas de Steiner, Metaheurística, *Simulated Annealing*, GRASP, Algoritmos Genéticos.

### ABSTRACT

This work presents a new heuristic, called Heurística 1, and the implementations of the GRASP, Simulated Annealing and Genetic Algorithms metaheuristics for the rectilinear Steiner minimum tree problem (RSMTP), talking about its theoretical aspects, like computational complexity, and practical ones, like pseudo-codes and implementation strategies. The new techniques for RSMTP presented, especially the Genetic Algorithms, have computational results of superior quality in comparison to the best heuristics in present literature.

**Key words:** Rectilinear Steiner Trees, Metaheuristic, Simulated Annealing, GRASP, Genetic Algorithms.

## **AGRADECIMENTOS**

### Agradeço

Primeiramente a DEUS, por ter me feito companhia em cada madrugada fria de trabalho.

A meu pai – Antônio Gabriel, minha mãe – Josélia, minha irmã – Gaby e minha avó – Hilda, pelo apoio incondicional em cada etapa da construção deste trabalho.

Ao Prof. Dr. Lucídio, pelo laço de amizade construído durante estes anos e por acreditar, sempre, na conclusão deste trabalho.

Ao Professor Hélio, por sempre demonstrar confiança na minha capacidade.

Aos meus amigos Nailson, Pedoca, Curuma, Alysson, Alan e Niltinho, por compreenderem este longo período de ausências.

Aos amigos da CAGEPA: Helton, Márcio, Leonardo, Eduardo e Renatão.

Aos amigos da SEFAZ-PE: Rafael, Amaro, Paulo, Ricardo e Marcelo Rosas.

As empresas ATI-PE e SEFAZ-PE, pela dispensa de oito horas semanais para conclusão deste trabalho.

Especialmente a Thiago Curvelo, pela amizade, paciência e dedicação no apoio a este que vos escreve, não só em termos de ciência, mas em todos os âmbitos da vida.

# SUMÁRIO

RESUMO.....	IV
AGRADECIMENTOS .....	V
SUMÁRIO .....	VI
LISTA DE FIGURAS.....	IX
LISTA DE TABELAS .....	XI
LISTA DE ACRÔNIMOS.....	XII
CAPÍTULO 1 - INTRODUÇÃO.....	1
1.1 DELIMITAÇÃO DE OBJETO .....	3
1.1.1 <i>Objetivos Gerais</i> .....	3
1.1.2 <i>Objetivos Específicos</i> .....	3
1.2 METODOLOGIA .....	4
1.2.1 <i>Atividades</i> .....	4
1.2.2 <i>Ambiente de Desenvolvimento</i> .....	5
CAPÍTULO 2 - FUNDAMENTAÇÃO TEÓRICA .....	7
2.1 FUNDAMENTOS SOBRE HEURÍSTICAS E METAHEURÍSTICAS .....	7
2.2 A METAHEURÍSTICA GRASP .....	9
2.3 A METAHEURÍSTICA SIMULATED ANNEALING.....	12
2.4 ALGORITMOS GENÉTICOS .....	16
2.4.1 <i>Representação de Indivíduos</i> .....	18
2.4.2 <i>Processo de Geração da População Inicial</i> .....	19
2.4.3 <i>Etapa de Reprodução</i> .....	20
2.4.3.1 Operador Clássico de Recombinação.....	21
2.4.3.2 Operador Clássico de Mutação .....	21
2.4.4 <i>Critérios de Parada</i> .....	22
2.4.5 <i>Problemas de Convergência</i> .....	22
2.5 FUNDAMENTOS SOBRE ÁRVORE RETILÍNEA MÍNIMA DE STEINER.....	23
2.5.1 <i>Teorema da grade de Hanan</i> .....	24

2.5.2 Tipos de pontos de Steiner .....	25
2.5.3 Topologias de árvores de Steiner .....	26
2.5.4 Árvores Retilíneas Full-Steiner .....	27
2.5.5 Algoritmos de Árvore Geradora Retilínea Mínima .....	28
2.5.6 Half Perimeter Wirelength .....	30
2.5.7 Aresta e Distância Gargalo.....	31
2.5.8 Redução de Pontos.....	31
<b>CAPÍTULO 3 - O ESTADO DA ARTE .....</b>	<b>33</b>
3.1 BIIS .....	34
3.1.1 Independência entre pontos.....	36
3.1.2 Manutenção dinâmica da RMST .....	37
3.1.3 Outras melhorias no BIIS.....	38
3.2 BGA.....	38
3.3 FLUTE.....	42
3.3.1 Sequência Vertical .....	42
3.3.2 Distância Marginal.....	43
3.3.3 Vetores de Coeficientes Potencialmente Ótimos .....	44
3.3.4 FLUTE para Instâncias com Dez ou Mais Terminais .....	45
3.3.5 Complexidade Computacional e Resultados .....	47
3.4 GEOSTEINER.....	48
3.4.1 Algoritmo de Geração de RFST .....	48
3.4.2 Concatenação de RFSTs .....	49
3.4.3 Outras Utilidades do conjunto de RFSTs .....	49
<b>CAPÍTULO 4 - MÉTODOS PROPOSTOS .....</b>	<b>51</b>
4.1 NOTAÇÕES .....	51
4.2 HEURÍSTICA1 – H1 .....	52
4.2.1 Procedimento de Geração de RMST .....	53
4.2.2 Localização de Pontos de Steiner Candidatos .....	54
4.2.3 Complexidade Computacional .....	55
4.2.4 Exemplo Prático.....	55



4.3 GRASP .....	56
4.3.1 <i>H1 Multi-Start</i> .....	56
4.3.2 <i>GRASP-H1</i> .....	58
4.3.3 <i>Complexidade Computacional</i> .....	60
4.4 SIMULATED ANNEALING .....	61
4.4.1 <i>Movimento Edge_Swap</i> .....	64
4.4.2 <i>Movimento Add_Steiner_3</i> .....	65
4.4.3 <i>Movimento Del_Steiner</i> .....	66
4.4.4 <i>Movimento Graceful_Del_Steiner</i> .....	68
4.4.5 <i>Movimento Pierce_Steiner</i> .....	68
4.4.6 <i>Complexidade Computacional</i> .....	70
4.5 ALGORITMO GENÉTICO .....	70
4.5.1 <i>Representação Genética das Soluções</i> .....	71
4.5.2 <i>Função de Aptidão de um Indivíduo</i> .....	74
4.5.3 <i>Procedimentos de Geração Populacional</i> .....	74
4.5.3.1 <i>Procedimento de Recombinação Odd_Even_Crossover</i> .....	74
4.5.3.2 <i>Procedimento de Mutação hy-M</i> .....	75
4.5.3.3 <i>Ilustração da Etapa de Reprodução do GA</i> .....	76
4.5.4 <i>Definição da População Sobrevivente</i> .....	76
<b>CAPÍTULO 5 - RESULTADOS COMPUTACIONAIS .....</b>	<b>78</b>
5.1 <i>SIMULAÇÃO COM INSTÂNCIAS GERADAS ALEATORIAMENTE</i> .....	78
5.2 <i>SIMULAÇÃO COM INSTÂNCIAS BENCHMARK DA OR_LIB</i> .....	81
<b>CAPÍTULO 6 - CONSIDERAÇÕES FINAIS E PROPOSTA DE TRABALHOS</b>	
<b>FUTUROS .....</b>	<b>89</b>
<b>REFERÊNCIAS.....</b>	<b>91</b>

## LISTA DE FIGURAS

FIGURA 2.1: EXEMPLO DE VARIAÇÃO DO CUSTO PELO ESPAÇO DE SOLUÇÕES.....	9
FIGURA 2.2: PSEUDOCÓDIGO DA METAHEURÍSTICA GRASP .....	10
FIGURA 2.3: PROCEDIMENTO CONSTRUTIVO GRASP .....	11
FIGURA 2.4: COMPORTAMENTO DO FATOR DE BOLTZMANN EM RELAÇÃO À TEMPERATURA..	13
FIGURA 2.5: PSEUDOCÓDIGO DA METAHEURÍSTICA <i>SIMULATED ANNEALING</i> .....	15
FIGURA 2.6: PSEUDOCÓDIGO DOS ALGORITMOS GENÉTICOS .....	18
FIGURA 2.7: GERAÇÃO UNIFORME DA POPULAÇÃO INICIAL.....	19
FIGURA 2.8: GERAÇÃO DA POPULAÇÃO INICIAL POR INVERSÃO .....	20
FIGURA 2.9: (A) ÁRVORE EUCLIDIANA, (B) ÁRVORE RETILÍNEA E (C) ÁRVORE RETILÍNEA DE STEINER.....	23
FIGURA 2.10: TEOREMA DA GRADE DE HANAN .....	25
FIGURA 2.11: EXEMPLOS DE PONTOS DE STEINER.....	25
FIGURA 2.12: REMOÇÃO DE UM <i>CORNER-POINT</i> .....	26
FIGURA 2.13: DIFERENTES TOPOLOGIAS DE UMA MESMA ÁRVORE RETILÍNEA DE STEINER ....	27
FIGURA 2.14: TOPOLOGIAS DE HWANG PARA RFSTs .....	27
FIGURA 2.15: OCTANTES DE UM PONTO QUALQUER.....	29
FIGURA 2.16: TRÊS FASES DA REDUÇÃO DE PONTOS E ARESTAS DE WINTER .....	32
FIGURA 2.17: REGIÕES VAZIAS: A) DIAMANTE, B) RETÂNGULO, C) TRIÂNGULO E D) CÍRCULO	32
FIGURA 3.1: CLASSES DE TÉCNICAS PARA O RSMT .....	33
FIGURA 3.2: EXECUÇÃO SIMPLES DO IIS.....	35
FIGURA 3.3: QUADRANTES DEFINIDOS PELA PARTIÇÃO DIAGONAL DE UM PONTO .....	37
FIGURA 3.4: MANUTENÇÃO DINÂMICA DA RMST .....	38
FIGURA 3.5: ÁRVORE 3-RESTRITA .....	39
FIGURA 3.6: PSEUDOCÓDIGO DO HGP [ <i>KAHNG, MANDOIU E ZELIKOVSKY, 2003</i> ] .....	41
FIGURA 3.7: ROTINA DE COMPUTAÇÃO DA ARESTA DE GARGALO DO CAMINHO ENTRE DOIS TERMINAIS U E V .....	41
FIGURA 3.8: SEQUÊNCIA VERTICAL .....	43
FIGURA 3.9: DISTÂNCIA MARGINAL .....	44
FIGURA 3.10: VETORES DE COEFICIENTES.....	44

FIGURA 3.11: QUEBRA DE INSTÂNCIA .....	46
FIGURA 3.12: AUMENTANDO UMA RFST .....	48
FIGURA 4.1: PSEUDOCÓDIGO DA HEURÍSTICA 1 .....	52
FIGURA 4.2: PROCEDIMENTO DE GERAÇÃO DE RMST .....	53
FIGURA 4.3: GRADE DE HANAN $k$ -RESTRITA .....	54
FIGURA 4.4: EXEMPLO DE EXECUÇÃO DA H1 .....	56
FIGURA 4.5: DIFERENTES RMSTs PARA UM MESMO CONJUNTO DE TERMINAIS .....	57
FIGURA 4.6: PROCEDIMENTO H1 <i>MULTI-START</i> .....	58
FIGURA 4.7: PROCEDIMENTO GRASP DE GERAÇÃO DE ÁRVORES RETILÍNEAS .....	59
FIGURA 4.8: COMPARAÇÃO ENTRE OS PROCEDIMENTOS H1 <i>MULTI-START</i> E GRASP-H1 .....	60
FIGURA 4.9: METAHEURÍSTICA HÍBRIDA SA / <i>MULTI-START</i> .....	61
FIGURA 4.10: PSEUDOCÓDIGO DO <i>SIMULATED ANNEALING</i> .....	62
FIGURA 4.11: PSEUDOCÓDIGO DO MOVIMENTO <i>EDGE_SWAP</i> .....	64
FIGURA 4.12: EXEMPLO DE EXECUÇÃO DO MOVIMENTO <i>EDGE-SWAP</i> .....	65
FIGURA 4.13: PSEUDOCÓDIGO DO MOVIMENTO <i>ADD_STEINER_3</i> .....	65
FIGURA 4.14: EXEMPLO DE EXECUÇÃO DO MOVIMENTO <i>ADD_STEINER_3</i> .....	66
FIGURA 4.15: PSEUDOCÓDIGO DO MOVIMENTO <i>DEL_STEINER</i> .....	67
FIGURA 4.16: EXEMPLO DE EXECUÇÃO DO MOVIMENTO <i>DEL_STEINER</i> .....	68
FIGURA 4.17: EXEMPLO DE EXECUÇÃO DO MOVIMENTO <i>PIERCE-STEINER</i> .....	69
FIGURA 4.18: IMPLEMENTAÇÃO DO ALGORITMO GENÉTICO .....	71
FIGURA 4.19: REPRESENTAÇÃO GENÉTICA DE ÁRVORES DE STEINER .....	72
FIGURA 4.20: REPRESENTAÇÃO DE UMA POPULAÇÃO DE ÁRVORES DE STEINER .....	73
FIGURA 4.21: PROCEDIMENTO DE RECOMBINAÇÃO <i>ODD_EVEN_CROSSOVER</i> .....	75
FIGURA 4.22: EXECUÇÃO DO PROCEDIMENTO DE REPRODUÇÃO .....	76
FIGURA 5.1: RELAÇÃO ENTRE O GAP PARA O ÓTIMO E O NÚMERO DE PONTOS DAS INSTÂNCIAS .....	83
FIGURA 5.2: MELHORIA MÉDIA SOBRE A RMST .....	84

## LISTA DE TABELAS

TABELA 2.1: TEMPO DE EXECUÇÃO MÉDIO DOS ALGORITMOS DE GERAÇÃO DE RMST .....	30
TABELA 3.1: NÚMERO DE ITERAÇÕES E NÚMERO DE PONTOS ADICIONADOS POR ITERAÇÃO NO BIIS .....	36
TABELA 3.2: RELAÇÃO DE POWVs POR GRUPO .....	45
TABELA 3.3: PARÂMETRO $A$ .....	47
TABELA 5.1: DETALHAMENTO DO GRUPO DE INSTÂNCIAS <i>RAND_LIB</i> .....	79
TABELA 5.2: <i>GAP</i> PARA O ÓTIMO DAS TÉCNICAS H1, GRASP, SA E GA SOBRE A <i>RAND_LIB</i> .	79
TABELA 5.3: <i>GAP</i> PARA A RMST DAS TÉCNICAS H1, GRASP, SA E GA SOBRE A <i>RAND_LIB</i>	80
TABELA 5.4: COMPARATIVO DO <i>GAP</i> PARA O ÓTIMO ENTRE O GA E O SA .....	81
TABELA 5.5: : DETALHAMENTO DO GRUPO DE INSTÂNCIAS <i>BENCHMARK</i> DA <i>OR_LIB</i> .....	82
TABELA 5.6: TABELA-RESUMO DO <i>GAP</i> PARA O ÓTIMO .....	83
TABELA 5.7: TABELA-RESUMO DO <i>GAP</i> SOBRE A RMST .....	84
TABELA 5.8: RESULTADOS OBTIDOS PARA O GRUPO 100 DA <i>OR_LIB</i> .....	85
TABELA 5.9: RESULTADOS OBTIDOS PARA O GRUPO 250 DA <i>OR_LIB</i> .....	86
TABELA 5.10: RESULTADOS OBTIDOS PARA O GRUPO 500 DA <i>OR_LIB</i> .....	87
TABELA 5.11: RESULTADOS OBTIDOS PARA O GRUPO 1000 DA <i>OR_LIB</i> .....	88

## LISTA DE ACRÔNIMOS

BIIS	BATCHED ITERATED 1-STEINER
BGA	BATCHED GREEDY ALGORITHM
FLUTE	FAST LOOKUP TABLE BASED WIRELENGTH ESTIMATION TECHNIQUE
FLUTE-MR	FLUTE MERGE REDUNDANT
FLUTE-AM	FLUTE AGGRESSIVE MERGE
FPGA	FIELD PROGRAMMABLE GATE ARRAY
GA	GENETIC ALGORITHM
GTCA	GREEDY TRIPLE CONTRACTION ALGORITHM
GRASP	GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURES
H1	HEURÍSTICA 1
HGP	HIERARQUICAL GREEDY PREPROCESSING
HPWL	HALF PERIMETER WIRELENGTH
IIS	ITERATED 1-STEINER
MST	MINIMUM SPANNING TREE
MSTP	MINIMUM SPANNING TREE PROBLEM
POWV	POTENTIALLY OPTIMAL WIRELENGTH VECTOR
RFST	RECTILINEAR FULL STEINER TREE
RMST	RECTILINEAR MINIMUM SPANNING TREE
RSMT	RECTILINEAR STEINER MINIMUM TREE PROBLEM
SA	SIMULATED ANNEALING
SMT	STEINER MINIMUM TREE PROBLEM
VLSI	VERY-LARGE-SCALE INTEGRATION
VNS	VARIABLE NEIGHBORHOOD SEARCH

## CAPÍTULO 1 - INTRODUÇÃO

O problema da árvore geradora mínima de Steiner (*Steiner Minimum Tree Problem* - SMTP), de modo similar ao problema da árvore geradora mínima (*Minimum Spanning Tree Problem* - MSTP), consiste em: dado um conjunto  $P$  de pontos no plano cartesiano, interconectá-los através da árvore  $A$  de menor comprimento. O comprimento de uma árvore é dado pela soma dos custos de todas as arestas desta. A diferença entre o SMTP e o MSTP é que, no SMTP, pontos extras podem ser adicionados a  $P$  com o intuito de diminuir o comprimento de  $A$ . Estes pontos extras são denominados pontos de Steiner, enquanto a árvore resultante é chamada árvore de Steiner.

Apesar de o MSTP possuir famosas soluções em tempo polinomial - os algoritmos de Kruskal e de Prim, ambos apresentando complexidade  $O(n \log(n))$  [Cormen et al., 2001] - o SMTP é NP-difícil [Garey et al., 1977]. Esta classe de problemas caracteriza-se pela não existência de algoritmos de tempo polinomial, o que propicia espaço para a utilização de abordagens semi-exatas.

O objeto deste trabalho é o problema da árvore geradora mínima de Steiner em distância retilínea (*Rectilinear Steiner Minimum Tree Problem* - RSMTP), variante do SMTP na qual a distância entre os pontos é aferida mediante métrica L1, distância de Manhattan. O RSMTP possui várias aplicações no projeto de chips VLSI (*Very-large-scale integration*). Nas fases de síntese e posicionamento pode ser usado para estimar o comprimento final do circuito, o congestionamento e o atraso das interconexões; enquanto nas fases de roteamento global e detalhado, é usado para gerar a topologia de roteamento de cada rede.

Mesmo com as novas funções-objetivo para roteamento introduzidas pelos recentes avanços na tecnologia dos circuitos integrados (especialmente em escala *deep-submicron*), o RSMTP mantém sua relevância [Mandoiu et al., 1999]: para redes não-críticas, ou instâncias fisicamente pequenas, a minimização do

comprimento das interconexões significa a diminuição da capacitância e da área total do circuito.

Assim como o SMTP, o RSMTP é NP-difícil [Garey et al., 1977], e muito esforço tem sido dedicado ao desenvolvimento de algoritmos e heurísticas para este problema. Entre as abordagens semi-exatas, o *Batched Iterated First Steiner* (BI1S) [Kahng e Robins, 1992] ostentou os resultados mais precisos por muitos anos. Recentemente, porém, o BI1S foi superado pelo *Fast Lookup Table Based Wirelength Estimation Technique* (FLUTE) [Chu, 2004]. Entre as técnicas exatas, o GeoSteiner [Warme, Winter e Zachariasen, 1998] apresenta o menor tempo de execução médio para instâncias aleatórias.

Nesse contexto, combinaram-se alguns avanços anteriores a esta dissertação com abordagens heurísticas e metaheurísticas, concebendo um conjunto de novas estratégias para o RSMTP. Estas novas abordagens foram encapsuladas em um aplicativo denominado NeoSteiner.

Em um primeiro momento, desenvolveu-se uma heurística simples e rápida (denominada H1) que, assim como as primeiras heurísticas propostas para o RSMTP, é baseada no algoritmo da árvore geradora mínima de Prim.

Posteriormente, foram aplicadas as metaheurísticas GRASP (*Greedy Randomized Adaptive Search Procedures*), SA (*Simulated Annealing*) e GA (*Genetic Algorithms*) ao RSMTP. Diferentemente do SMTP, não se encontrou referência a abordagens metaheurísticas aplicadas ao RSMTP na literatura.

Os resultados obtidos pelo NeoSteiner mostraram-se bastante competitivos com o estado da arte, seja em tempo de execução (caso do H1 e do GRASP) ou em qualidade da solução (caso do SA e do GA).

Com o intuito de descrever da melhor maneira o trabalho realizado, dividiu-se esta dissertação em seis capítulos.

Esta introdução (primeiro capítulo) apresenta a delimitação do escopo e os objetivos gerais e específicos deste trabalho, assim como a metodologia utilizada para o seu desenvolvimento.

O segundo capítulo, responsável pela fundamentação teórica, clarifica os conceitos-chave necessários para a compreensão das estratégias aqui propostas.

O terceiro capítulo apresenta uma importante revisão das heurísticas (e do algoritmo GeoSteiner) que compõem o estado da arte para o RSMT. Algumas técnicas utilizadas pelo NeoSteiner, tais como a redução de pontos e a manutenção dinâmica da árvore geradora mínima (*Minimum Spanning Tree* - MST), são baseadas nestes trabalhos.

O quarto capítulo, espinha dorsal desta dissertação, apresenta o estudo, a implementação, o pseudocódigo e a análise da complexidade computacional da heurística H1 e das metaheurísticas GRASP, SA e GA aplicadas ao RSMT.

O quinto capítulo apresenta os resultados obtidos pelo NeoSteiner em comparação às abordagens descritas no capítulo três.

O sexto capítulo apresenta a conclusão, as considerações finais e as propostas de trabalhos futuros, sendo seguido pelas referências bibliográficas.

## **1.1 Delimitação de Objeto**

### **1.1.1 Objetivos Gerais**

Compreender o RSMT e suas aplicações, a fim de manter o foco no estudo e desenvolvimento de técnicas adequadas ao cenário atual.

Adquirir um conhecimento mais profundo nas áreas de inteligência computacional, com ênfase nas metaheurísticas SA, GRASP e GA.

Estudar, planejar e implementar novas técnicas para o RSMT, com o intuito de melhorar as abordagens existentes em custo computacional e/ou qualidade de solução.

### **1.1.2 Objetivos Específicos**

Os objetivos específicos deste trabalho descrevem as metas a serem alcançadas, tendo em vista o objeto de estudo desta dissertação. Para tanto, estão previstos os seguintes pontos:

- Estudar as melhores heurísticas e algoritmos da literatura e identificar o que de melhor possuem.



- Desenvolver uma heurística rápida e robusta o suficiente para ser usada como estimativa de roteamento em chips VLSI.
- Implementar e parametrizar as metaheurísticas SA, GRASP e GA para o RSMTTP.
- Realizar testes e simulações para comparar os resultados obtidos com o estado da arte, comprovando, assim, a eficácia da heurística desenvolvida e das metaheurísticas aplicadas.

## **1.2 Metodologia**

A realização deste trabalho foi executada mediante uma metodologia “*top-down*”, na qual é buscada, a princípio, uma compreensão holística do problema, seguida pela prospecção das soluções existentes, para, só então, realizar-se o projeto e a implementação de uma nova solução.

### **1.2.1 Atividades**

Para atingir os objetivos supracitados foram realizadas as seguintes atividades:

- Obtenção de subsídios teóricos sobre RSMTTP, visando melhor compreensão dos seus pilares matemáticos.
- Levantamento bibliográfico direcionado à identificação das melhores soluções da Literatura.
- Obtenção e estudo das melhores soluções do RSMTTP, tanto em código-fonte quanto em publicações nas quais são descritas.
- Estudo sobre algoritmos, heurísticas e metaheurísticas, objetivando a definição da estratégia mais eficiente de resolução do RSMTTP.
- Definição, estudo, projeto e implementação da abordagem heurística H1.
- Definição, estudo, projeto, implementação e parametrização das metaheurísticas GRASP, SA e GA aplicadas ao RSMTTP.

- Encapsulamento das soluções implementadas em um único aplicativo, denominado NeoSteiner.
- Utilização do aplicativo gprof [Fenlason e Stallman, 2008] para identificar quais métodos do NeoSteiner consomem mais tempo de processamento, facilitando, assim, a redução do tempo total de execução do aplicativo.
- Utilização do aplicativo valgrind [Seward et al., 2008] para identificar e remover usos indevidos de memória, eventualmente presentes no NeoSteiner.
- Realização de testes para geração de tabelas e gráficos comparativos entre o NeoSteiner e o estado da arte.
- Avaliação dos resultados obtidos, levantamento dos pontos fortes e fracos do NeoSteiner e proposição de trabalhos futuros.
- Publicação dos resultados obtidos.
- Construção do manual de utilização do NeoSteiner.
- Redação desta dissertação.

### 1.2.2 Ambiente de Desenvolvimento

O NeoSteiner foi escrito em linguagem de programação “C” e compilado utilizando o gcc versão “Debian 4.3.3-10”, em modo de otimização de código “O3”. O computador no qual se deu o processo de projeto e desenvolvimento, assim como de realização dos testes comparativos, apresenta as seguintes características:

- Sistema operacional Debian Linux (kernel 2.6.18);
- Processador Intel® Pentium® 4 CPU 2.80GHz;
- Memória cache L1 de 1024 KB;
- 2048 KB de memória Ram;

A visualização das soluções é propiciada pelo o aplicativo gnuplot, interligado ao NeoSteiner através da biblioteca gnuplot\_i.

A execução de testes e a geração das tabelas e gráficos deste trabalho foram automatizadas através de scripts shell e awk.

## **CAPÍTULO 2 - FUNDAMENTAÇÃO TEÓRICA**

Este capítulo destina-se a elucidar os conceitos centrais deste trabalho, oferecendo, assim, os subsídios teóricos necessários à sua completa compreensão.

A primeira seção traz a definição de heurística e metaheurística, tal como uma breve explicação das metaheurísticas GRASP, SA e GA.

A seção subsequente, além definir formalmente o RSMTP, apresenta alguns avanços matemáticos e geométricos utilizados para uma solução mais eficiente deste problema.

### ***2.1 Fundamentos Sobre Heurísticas e Metaheurísticas***

Muitos problemas práticos existentes no dia a dia podem ser solucionados otimamente por meio da computação, i.e., é possível encontrar a melhor solução existente para tal problema.

Há problemas, porém, que por apresentar características combinatórias, não possuem algoritmos capazes de encontrar a sua solução ótima em tempo polinomial. Estes problemas constituem uma classe denominada NP-difícil.

Em problemas dessa natureza, onde métodos exatos tornam-se inviáveis, pode-se encontrar, todavia, boas soluções em tempo razoável. Em geral, tais soluções apresentam uma relação custo/benefício bastante vantajosa, visto que o esforço computacional exigido para encontrar a solução ótima pode demandar, em muitos casos, anos, décadas ou séculos de processamento (tempos impraticáveis computacionalmente).

O conceito de heurística é definido como uma técnica inspirada em processos intuitivos que procura uma solução de boa qualidade a um custo computacional aceitável, sem, no entanto, estar capacitada a garantir a solução ótima, bem como garantir quão próximo está desta [Souza, 2008].

O desafio é produzir, em tempo reduzido, soluções tão próximas quanto possível do ótimo. Muitos esforços têm sido feitos nesta direção e heurísticas muito eficientes foram desenvolvidas para diversos problemas. Entretanto, a maioria das heurísticas desenvolvidas é específica para um problema particular, não sendo eficiente, ou mesmo aplicável, à resolução de uma classe mais ampla de problemas.

As heurísticas são classificadas em duas classes: heurísticas construtivas e heurísticas de refinamento.

As heurísticas construtivas, como o nome sugere, são responsáveis pela construção, elemento por elemento, de uma solução. A seleção do elemento a ser inserido em cada passo, varia de acordo com a função de avaliação adotada, que, por sua vez, depende do problema abordado. Nas heurísticas clássicas, os elementos candidatos são geralmente ordenados segundo uma função gulosa, que estima o benefício da inserção de cada elemento, e somente o “melhor” elemento é inserido a cada passo desta.

As heurísticas de refinamento -- também conhecidas como técnicas de busca local -- partem de uma solução inicial qualquer, obtida através de uma heurística construtiva ou gerada aleatoriamente, e caminham iterativamente sobre a vizinhança desta solução a fim de encontrar melhores resultados.

[Souza, 2008] define o conceito de vizinhança como segue: seja  $S$  o espaço de pesquisa de um problema de otimização e  $f$  a função objetivo a minimizar, a função  $N$ , a qual depende da estrutura do problema tratado, associa a cada solução  $s \in S$ , sua vizinhança  $N(s) \subseteq S$ . Cada solução  $s' \in N(s)$  é chamada de vizinho de  $s$ . Denomina-se movimento a modificação que transforma uma solução  $s \in S$  em outra,  $s' \in N(s)$ .

A partir da década de 80 surgiu uma nova classe de heurísticas, reunindo conceitos das áreas de otimização e inteligência artificial, denominada metaheurística. Caracteriza-se como um conjunto de conceitos usados para definir métodos heurísticos que podem ser aplicados em vários problemas diferentes de otimização.

Uma característica comum às metaheurísticas é a capacidade de escapar de ótimos locais a fim de continuar a busca pelo ótimo global. Tal atributo não é compartilhado com os métodos heurísticos. A Figura 2.1 exemplifica a presença de diversos mínimos locais em um dado espaço de soluções hipotético.

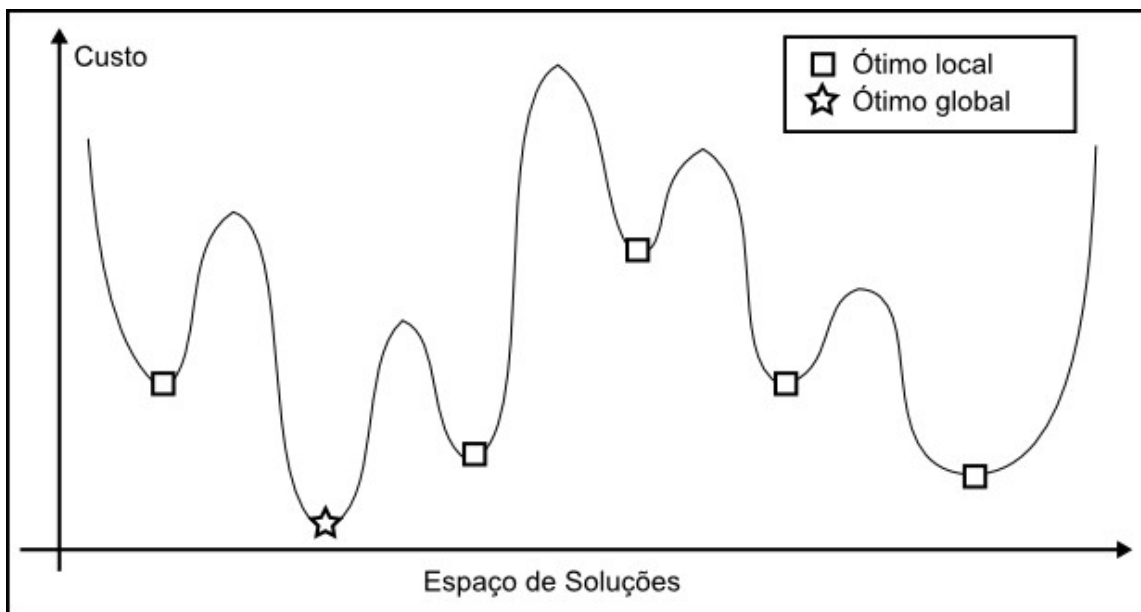


Figura 2.1: Exemplo de variação do custo pelo espaço de soluções

Define-se: metaheurísticas são procedimentos destinados a encontrar uma boa solução, eventualmente a ótima, consistindo na aplicação, em cada passo, de uma heurística subordinada, a qual tem que ser modelada para cada problema específico [Ribeiro, 1996].

Dentre os procedimentos enquadrados como metaheurísticas destacam-se: algoritmos genéticos, redes neurais, *Simulated Annealing*, busca tabu, GRASP, VNS (*Variable Neighborhood Search*) e colônia de formigas.

## 2.2 A Metaheurística GRASP

O GRASP (*Greedy Randomized Adaptive Search Procedure*), em português “Procedimento de Busca Gulosa Adaptativa Aleatória”, é uma metaheurística *multi-start* iterativa na qual cada iteração é composta por uma fase de construção e uma fase de busca local [Resende, 2003].

O GRASP foi proposto inicialmente por [Feo et al., 1994] e, atualmente, possui um grande destaque na literatura devido aos bons resultados obtidos em problemas de otimização [Rocha e Alvarenga, 2006]. A Figura 2.2 mostra o pseudocódigo do procedimento GRASP.

Procedimento GRASP

```
1 ENQUANTO (condição de parada não for satisfeita), FAÇA
2   solução = crie uma solução construtiva_ateatoria();
3   solução = busca_local(solução);
4   SE solução é a melhor solução até então conhecida ENTÃO
5     grave(solução);
6   FIM SE
7 FIM ENQUANTO
```

Figura 2.2: Pseudocódigo da metaheurística GRASP

No GRASP, a fase de construção é iterativa, adaptativa, semi-gulosa e randômica, gerando soluções viáveis para o problema através de um procedimento parcialmente guloso e parcialmente aleatório [Ferreira e Ochi, 2007].

A cada etapa da construção da solução, seleciona-se uma lista restrita, formada pelos  $a\%$  melhores candidatos, baseada num critério de ordenação pré-definido. Sobre esta lista é feita aleatoriamente a escolha do próximo elemento a compor a solução.

O parâmetro  $a$  (no intervalo  $[0,1]$ ) controla o nível de “gulosidade” e aleatoriedade do procedimento de construção. Um valor  $a = 0$  faz gerar soluções puramente gulosas, enquanto  $a = 1$  faz produzir soluções totalmente aleatórias.

A metaheurística GRASP é dita adaptativa pois os benefícios associados com a escolha de cada elemento são atualizados em cada iteração da fase de construção para refletir as mudanças oriundas da seleção do elemento anterior [Souza, 2008].

A aleatoriedade da escolha dos elementos permite a visita de uma porção maior do universo de soluções. Essa característica permite que o procedimento

GRASP escape de ótimos locais. O procedimento de construção da solução é ilustrado na Figura 2.3.

A fase de refinamento torna-se bastante importante devido às características aleatórias da fase de construção, que, assim como muitos procedimentos não determinísticos, não nos encaminham para soluções localmente ótimas.

Nesta fase podem ser utilizadas quaisquer heurísticas de busca local, que tem como objetivo melhorar uma solução inicial até que ela atinja um ótimo local.

Procedimento Construtivo GRASP

```
1 Inicialize o conjunto C de Candidatos;  
2 ENQUANTO C diferente de vazio FAÇA  
3   Ordene a lista de Candidatos;  
4   Monte a Lista_Restrita com os ALPHA melhores Candidatos;  
5   Selecione aleatoriamente um membro da Lista_Restrita, K;  
6   Adicione K à solução;  
7   Atualize o conjunto C;  
8 FIM ENQUANTO  
9 Retorne a solução obtida;
```

Figura 2.3: Procedimento construtivo GRASP

O parâmetro  $\alpha$  é basicamente o único parâmetro a ser ajustado na implementação de um procedimento GRASP. Valores de  $\alpha$  que levam a uma lista de candidatos restrita de tamanho muito limitado implicam em soluções finais de qualidade muito próxima àquela obtida de forma puramente gulosa, obtidas com um baixo esforço computacional. Em contrapartida, provocam uma baixa diversidade de soluções construídas. Já uma escolha de  $\alpha$  próxima da seleção puramente aleatória leva a uma grande diversidade de soluções construídas, por outro lado, muitas das soluções construídas são de qualidade inferior, tornando mais lento o processo de busca local.

O procedimento GRASP bem parametrizado e implementado agrega os bons aspectos dos algoritmos gulosos e dos algoritmos de construção aleatória, constituindo-se uma metaheurística rápida e poderosa.



As principais características do GRASP são:

- Facilidade de implementação;
- Facilidade de parametrização, devido ao único parâmetro  $\alpha$ ;
- Facilidade de implementação paralela, devido à relativa independência entre suas iterações.

## **2.3 A Metaheurística Simulated Annealing**

A metaheurística SA (*Simulated Annealing*), proposta originalmente por [Kirkpatrick et al., 1983], fundamenta-se em uma analogia com a termodinâmica, ao simular o arrefecimento de um conjunto de átomos aquecidos.

O nome recozimento (*annealing*) é dado ao processo de aquecimento de um sólido até o seu ponto de fusão, seguido de um resfriamento gradual e vagaroso, até que se alcance novamente o seu enrijecimento. Nesse processo, o lento resfriamento é essencial para se manter um equilíbrio térmico onde os átomos encontrarão tempo suficiente para se organizarem em uma estrutura uniforme e com energia mínima. Se o sólido é resfriado bruscamente, seus átomos formarão uma estrutura irregular e fraca [Mazzucco Junior, 1999].

O recozimento pode ser visto como um processo estocástico de determinação de uma organização dos átomos de um sólido que apresente energia mínima.

Em temperaturas altas, os átomos se movem livremente e, com grande probabilidade, podem mover-se para posições que incrementarão a energia total do sistema.

Quando se tem baixas temperaturas, os átomos gradualmente se movem em direção a uma estrutura regular e, somente com pequena probabilidade, incrementarão suas energias.

Em termos computacionais, a metaheurística SA é uma técnica de busca local probabilística [Souza, 2008]. A SA começa sua busca a partir de uma solução inicial qualquer. O procedimento principal consiste em um laço que gera aleatoriamente, em cada iteração, um único vizinho  $s'$  da solução corrente  $s$ .

Considerando um problema de minimização, seja  $\Delta$  a variação de valor da função objetivo ao mover-se para uma solução vizinha candidata, isto é,  $\Delta = f(s') - f(s)$ . Ocorrem as seguintes situações:

- $\Delta < 0$ : Houve uma redução da energia. O método aceita o movimento e a solução vizinha passa a ser a nova solução corrente.
- $\Delta = 0$ : Caso de estabilidade, pouco comum na prática. A aceitação do movimento é indiferente.
- $\Delta > 0$ : Houve um aumento da energia. A solução vizinha candidata também poderá ser aceita, mas neste caso, com uma probabilidade  $e^{-\Delta/T}$ , fator de Boltzmann, onde  $T$  é um parâmetro do método, chamado de temperatura e que regula a probabilidade de se aceitar soluções de pior custo. A Figura 2.4 demonstra, para  $\Delta = 1$ , a relação entre a temperatura e o fator de Boltzmann.

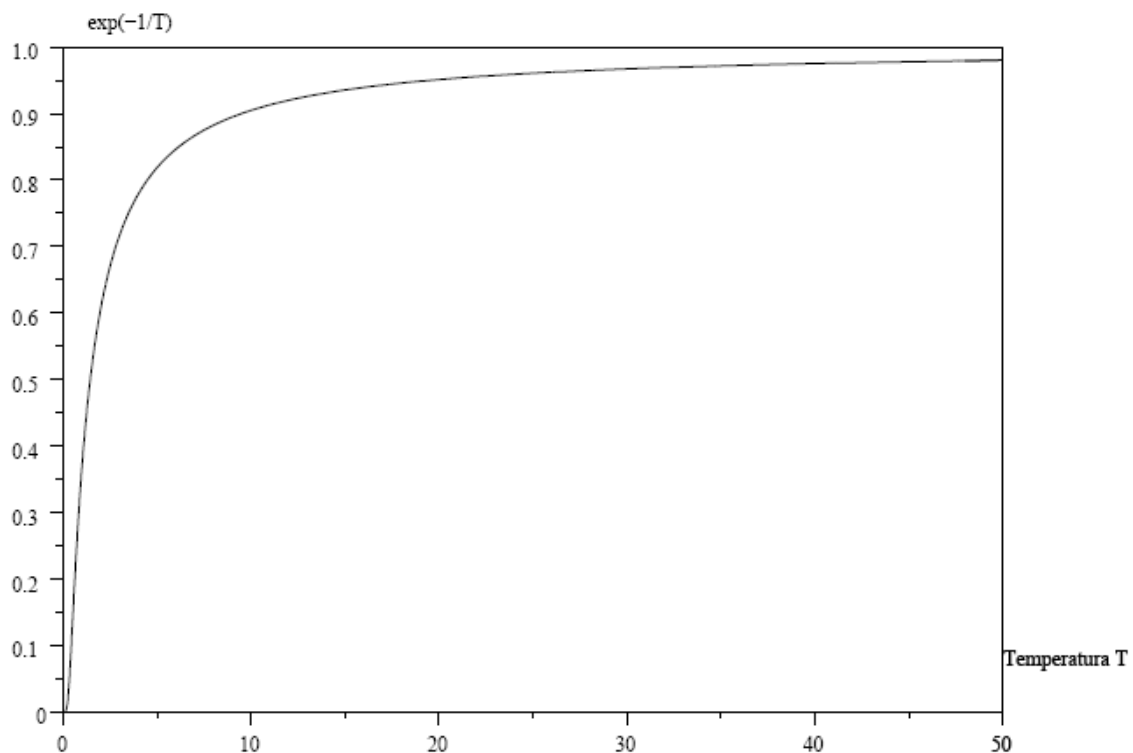


Figura 2.4: Comportamento do fator de Boltzmann em relação à temperatura

A temperatura  $T$  assume, inicialmente, um valor elevado  $T_0$ . Após um número fixo de iterações (o qual representa o número de iterações necessárias para o sistema atingir o equilíbrio térmico em uma dada temperatura), a temperatura é gradativamente diminuída por uma razão de resfriamento  $\alpha[0,1]$ , tal que  $T_k \leftarrow \alpha \times T_{k-1}$ .

Com esse procedimento, no início existe uma chance maior de escape de mínimos locais e, à medida que  $T$  aproxima-se de zero, o algoritmo comporta-se como o método de descida, uma vez que diminui a probabilidade de que movimentos de piora sejam aceitos.

O processo é finalizado quando a temperatura chega a um valor próximo a zero e nenhuma solução que piore o valor da solução corrente seja mais aceita. A solução obtida quando o sistema encontra-se nesta situação evidencia o encontro de um mínimo local.

Existem algumas variações entre os algoritmos baseados em SA, que geralmente incluem reaquecimento seguido de um novo processo de resfriamento, utilizado quando a quantidade de movimentos consecutivamente rejeitados é alta. Também é comum trabalhar nas temperaturas mais altas com taxa de resfriamento menor e aumentá-la quando a temperatura reduzir [Souza, 2008].

Os parâmetros de controle do SA são a razão de resfriamento  $\alpha$ , o número de iterações para cada temperatura ( $SA_{\max}$ ) e a temperatura inicial  $T_0$ .

Em teoria, a temperatura final deve ser zero. Entretanto, na prática é suficiente chegar a uma temperatura próxima de zero, devido à precisão limitada da implementação computacional [Torreão, 2008]. Um valor típico é tomar  $T_f = 0,001$ . Alternativamente, pode-se identificar o congelamento do sistema quando a taxa de aceitação de movimentos apresentar valores abaixo de um nível predeterminado.

Observa-se que, como regra geral, os parâmetros mais adequados para uma dada aplicação do algoritmo só podem ser estabelecidos por experimentação.

O procedimento SA pode ser modelado matematicamente por intermédio da teoria de cadeias de Markov. Existem vários resultados na literatura que, utilizando esse modelo, garantem que esta metaheurística converge para o ótimo global [Hajek, 1988]. O número de iterações necessário para que se o método SA convirja para o ótimo global, no entanto, na maioria dos casos, é computacionalmente proibitivo.

A Figura 2.5 traz o pseudocódigo de um procedimento SA básico.

```

Procedimento SA básico aplicado a um problema de minimização

1  s* <= s;      # Mantém a melhor solução
2  iterT <= 0;   # Numero de iterações para cada T
3  T <= T0;      # Temperatura inicial
4
5  ENQUANTO (T > 0) FAÇA
6      ENQUANTO (iterT < SAmáx) FAÇA
7          iterT <= iterT +1;
8          gere um vizinho s' de s;
9           $\Delta = f(s') - f(s)$ ;
10
11         SE ( $\Delta < 0$ )
12             s <= s' ;
13             SE ( $f(s') < f(s^*)$ ) s* <= s' ;
14         FIM SE
15
16         SE ( $\Delta \geq 0$ )
17             gere aleatoriamente x entre 0 e 1;
18             SE ( $x < e^{-\Delta/T}$ ) s <= s' ;
19         FIM SE
20     FIM ENQUANTO
21
22     T <=  $\alpha \times T$ ;
23     iterT <= 0;
24 FIM ENQUANTO
25
26 Retorne s*; # a melhor solução obtida;

```

Figura 2.5: Pseudocódigo da metaheurística *Simulated Annealing*

## 2.4 Algoritmos Genéticos

Os Algoritmos Genéticos (GA), do inglês *Genetic Algorithms*, são métodos metaheurísticos de otimização inspirados nos mecanismos de evolução das populações de seres vivos. Foram introduzidos por John Holland [Holland, 1975] e popularizados por um dos seus alunos, David Goldberg [Goldberg, 1989].

Os GA seguem os princípios da seleção natural e da sobrevivência do mais apto, declarados em 1859 por Charles Darwin no livro “A Origem das Espécies” [Lacerda e Carvalho, 1999]. Segundo Darwin, quanto melhor um indivíduo se adaptar ao seu meio ambiente, maior será sua chance de sobreviver e gerar descendentes.

Em outros termos: o meio ambiente seleciona, a cada geração, os seres vivos mais aptos de uma população para sobrevivência. Como resultado, uma vez que os menos adaptados ao ambiente são eliminados, geralmente, antes de se reproduzir, apenas os mais aptos conseguem gerar descendentes.

Durante a reprodução, ocorrem fenômenos como mutação e recombinação (*crossover*), entre outros, que atuam sobre o material genético armazenado nos cromossomos levando à variabilidade dos seres vivos da população.

Os GA são a metáfora desses fenômenos, herdando, por isso, muitos termos originários da biologia. Listamos a seguir os termos biológicos mais usados nos GA, assim como e sua correspondência computacional:

- **Cromossomo, genoma ou código genético:** Estrutura de dados que codifica uma solução para o problema, ou seja, um simples ponto no espaço de busca.
- **Gene:** Segmento do cromossomo que corresponde (codifica) um parâmetro qualquer do problema.
- **Alelo:** Possível valor que cada gene pode assumir.
- **Aptidão:** Função que avalia um cromossomo a fim de mensurar seu grau de adaptação ao meio. Geralmente a função objetivo do problema é utilizada como função de aptidão.

- **Indivíduo:** Um membro qualquer da população, formado por um cromossomo (codificado ou não) e sua aptidão.
- **População:** Conjunto de indivíduos que coexistem em determinada geração.
- **Clone:** Indivíduos que possuem códigos genéticos completamente iguais.
- **Geração:** Representação numérica de uma dada iteração do algoritmo genético.
- **Genótipo:** Representa a informação contida no cromossomo.
- **Mutação:** Alteração intencional e randômica de um ou mais genes a fim de aumentar a diversificação genética da população.
- **Fenótipo:** Representa o objeto, estrutura ou organismo construído a partir das informações do genótipo.

Os GA iniciam sua busca com a geração de uma população inicial  $\{s_1^0, s_2^0, \dots, s_n^0\}$ , denominada geração zero ou população em tempo zero.

O procedimento principal é um laço que cria a população  $\{s_1^{t+1}, s_2^{t+1}, \dots, s_n^{t+1}\}$  no tempo  $t+1$ , a partir de uma população no tempo  $t$ . Para tal, os indivíduos da geração  $t$  passam por um processo de reprodução, que consiste na seleção de indivíduos para recombinação e/ou mutação.

Gerada a nova população em  $t+1$ , define-se, baseado na função de aptidão, os indivíduos que devem sobreviver, isto é, as  $n$  soluções que irão compor a geração  $t+1$ . Os critérios comumente usados na definição dos sobreviventes são:

1. **Aleatório**, no qual os sobreviventes são escolhidos à sorte;
2. **Roleta**, onde a chance de sobrevivência de cada cromossomo é proporcional ao seu nível de aptidão ou
3. **Misto**, no qual se utiliza uma combinação dos dois métodos anteriores.

Vale salientar que em qualquer desses critérios admite-se a sobrevivência de indivíduos menos aptos tendo-se por objetivo tentar escapar de ótimos locais.

A Figura 2.6 traz o pseudocódigo de um algoritmo genético básico.

#### Algoritmo Genético Básico

```
1  $t \leftarrow 0$  ;  
2 Gere a população inicial  $P(t)$  ;  
3 Avalie  $P(t)$  ;  
4 ENQUANTO (Critérios de parada não satisfeitos) FAÇA  
5    $t \leftarrow t+1$  ;  
6   gere  $P(t)$  a partir de  $P(t-1)$  ;  
7   Avalie  $P(t)$  ;  
8   Defina a população sobrevivente ;  
9 FIM ENQUANTO
```

Figura 2.6: Pseudocódigo dos Algoritmos Genéticos

Os principais parâmetros de controle dos GA são: o tamanho  $n$  da população, o número de gerações, a probabilidade de mutação, a probabilidade de recombinação e o número de iterações sem melhora [Souza, 2008].

### 2.4.1 Representação de Indivíduos

Um cromossomo  $p$ , o qual representa uma solução para o problema, é representado na forma de um vetor com  $m$  posições:  $p = (x_1, x_2, \dots, x_m)$ , onde cada componente  $x_i$  representa um gene.

As representações mais conhecidas para os cromossomos são: a representação binária e a representação por inteiros. A característica que diferencia estas representações é o alelo: na primeira, o gene é limitado aos valores '0' ou '1', i.e.,  $x_i \in \{0,1\}$ ; enquanto na segunda, o gene pode apresentar qualquer valor inteiro:  $x_i \in \mathbb{Z}$ .

A representação binária é clássica nos GA, no entanto, existem problemas para os quais é mais conveniente utilizar a representação inteira, por exemplo, o problema do caixeiro viajante [Lacerda e Carvalho, 1999].

### 2.4.2 Processo de Geração da População Inicial

A escolha do processo de geração da população inicial deve ser tomada levando em consideração dois fatores: uma maior cobertura do espaço de soluções e a minimização de esforços computacionais redundantes.

A geração puramente aleatória tanto pode ocasionar a não representação de algumas regiões do espaço de busca, caso o tamanho escolhido para a população seja muito pequeno, quanto à existência de indivíduos iguais ou muito semelhantes, caso o tamanho da população seja grande demais.

Estes problemas podem ser minimizados através da geração uniforme da população, i.e., com pontos igualmente espaçados, como se preenchessem uma grade no espaço de soluções, vide Figura 2.7.

10000000	00001000
01000000	00000100
00100000	00000010
00010000	00000001

Figura 2.7: Geração uniforme da população inicial

Uma alternativa é gerar apenas a primeira metade da população de forma randômica e a segunda metade a partir da primeira, invertendo-se os bits. Isto garante que cada gene tenha pelo menos um representante na população com os valores '0' e '1'. A Figura 2.8 exemplifica esta abordagem.



1ª metade gerada aleatoriamente	2ª metade inverte os bits da 1ª metade
1011010	0100101
0111011	1000100
0001101	1110010
1100110	0011001

Figura 2.8: Geração da população inicial por inversão

Uma abordagem interessante seria utilizar uma população inicial grande, provendo diversificação do espaço de buscas, e, gradativamente, diminuir o tamanho da população para as gerações subseqüentes, economizando processamento.

Outra técnica comum é o *seeding*, que consiste em inserir na população inicial soluções encontradas por outros métodos, garantindo, assim, que os GA sejam tão ou mais preciso que estes [Lacerda e Carvalho, 1999].

### 2.4.3 Etapa de Reprodução

A geração da população em tempo  $t+1$  dá-se através de uma fase de reprodução, na qual são selecionados indivíduos da geração  $t$  para sofrerem operações de recombinação e/ou mutação.

Dentre as formas de seleção de indivíduos para o processo de reprodução, destacam-se a seleção puramente aleatória e a seleção por torneio. No torneio, cada pai é escolhido mediante o seguinte procedimento: são selecionados aleatoriamente  $k$  indivíduos da população e apenas aquele que possuir a melhor aptidão é escolhido como pai. Um valor usualmente utilizado é  $k=2$ , caracterizando o torneio binário (*binary tournament*).

Na operação de recombinação, a qual geralmente ocorre com probabilidade elevada (por exemplo, 80% [Souza, 2008]), os genes de dois (ou mais) cromossomos pais são combinados de forma a gerar filhos (comumente dois). Em cada cromossomo filho haverá um conjunto de genes de cada cromossomo pai.

A operação de mutação, a qual geralmente ocorre com baixa probabilidade (1 a 2%, em geral [Souza, 2008]), consiste em alterar aleatoriamente parte dos genes dos cromossomos.

#### 2.4.3.1 Operador Clássico de Recombinação

A idéia do operador clássico de recombinação (*crossover*) é efetuar cruzamentos entre dois ou mais cromossomos pais e formar cromossomos filhos (*offsprings*) a partir da união de segmentos de genes de cada pai.

Inicialmente são feitos cortes aleatórios nos pais. Por exemplo, considere dois pais e um ponto de corte, representado pelo caractere '|', realizado na parte central dos cromossomos pais:

$$p_1 = (1\ 0\ 0\ 1\ | \ 0\ 1\ 1\ 0) = (p_1^1\ | \ p_1^2)$$

$$p_2 = (1\ 1\ 1\ 1\ | \ 0\ 0\ 0\ 0) = (p_2^1\ | \ p_2^2)$$

A partir dos cortes, são gerados dois filhos, cada qual formado a partir da reunião de partes de cada um dos pais:

$$f_1 = (1\ 0\ 0\ 1\ | \ 0\ 0\ 0\ 0) = (p_1^1\ | \ p_2^2)$$

$$f_2 = (1\ 1\ 1\ 1\ | \ 0\ 1\ 1\ 0) = (p_2^1\ | \ p_1^2)$$

Observe que os cromossomos filhos, apesar de diferentes dos pais, carregam segmentos de informação genética de cada um deles.

#### 2.4.3.2 Operador Clássico de Mutação

O operador de mutação tem como objetivo o escape de ótimos globais. Classicamente, para uma representação binária, consiste em alterar o valor de um ou mais genes de '0' para '1' ou vice-versa. Segue um exemplo de mutação:

$$p = (1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 1) \Rightarrow \text{Mutação} \Rightarrow p' = (1\ 0\ 1\ 1\ 0\ 1\ 0\ 1\ 1\ 0\ 0)$$

Perceba que dois genes foram alterados: o valor de  $x_3$  mudou de '0' para '1' e o valor de  $x_{11}$  passou de '1' para '0'.

#### **2.4.4 Critérios de Parada**

Dos vários critérios de parada dos GA, os mais comuns são: chegada ao número limite de gerações, convergência da população ou homogeneização dos cromossomos.

A convergência populacional ocorre quando não acontece melhoria significativa da melhor solução por um dado número de gerações. Isto significa que a metaheurística encontrou um ótimo, quer seja local ou global, do qual não consegue escapar. Outra maneira de detectar a convergência populacional é computar, na fase de avaliação da população, o desvio padrão das aptidões dos indivíduos.

A homogeneização dos cromossomos se dá, geralmente, nas gerações mais avançadas dos GA, quando os indivíduos da população possuem entre 90 e 95% dos genes com o mesmo valor [Lacerda e Carvalho, 1999].

#### **2.4.5 Problemas de Convergência**

A convergência prematura é um conhecido problema dos GA. Pode ocorrer quando surgem cromossomos com alta aptidão em relação à população, sem que esta possua os cromossomos realmente ótimos; ou quando a população inicial não cobriu suficientemente o espaço de soluções.

Os cromossomos pseudo-ótimos, chamados super-indivíduos, acabam por gerar um excessivo número de filhos, que dominam a população, uma vez esta possui tamanho limitado. Tais cromossomos espalham seus genes por toda a população, enquanto outros genes desaparecem, fenômeno conhecido como *genetic drift*. Como consequência, o método converge para ótimos locais.

A convergência prematura pode ser combatida através da limitação do número de descendentes de cada cromossomo, do aumento da taxa de mutação ou da não inserção de clones na população.

## 2.5 Fundamentos Sobre Árvore Retilínea Mínima de Steiner

O problema da árvore retilínea mínima de Steiner (RSMTP) destaca-se como um dos problemas fundamentais na automação do roteamento de redes de circuitos eletrônicos, uma vez que uma interconexão de comprimento mínimo possui mínima capacitância total e requer uma mínima quantidade de área [Cabral, 2001].

Define-se árvore retilínea de Steiner como sendo um grafo acíclico e conexo, no qual os vértices (pontos no plano cartesiano) são interligados utilizando apenas segmentos verticais e horizontais, onde é permitida a adição de pontos extras, denominados pontos de Steiner, a fim de diminuir o seu comprimento.

A Figura 2.9 traz exemplos de árvore euclidiana, árvore retilínea e árvore retilínea de Steiner para o mesmo conjunto de pontos.

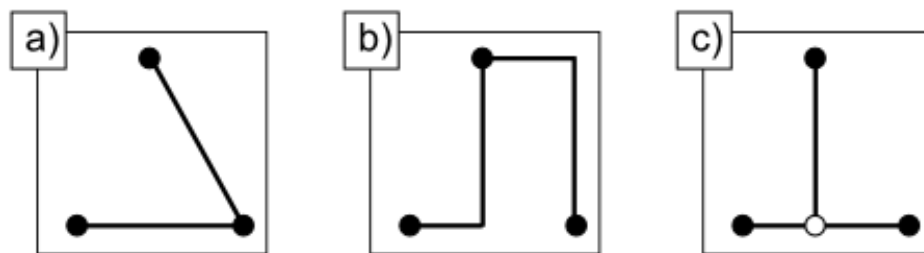


Figura 2.9: (a) Árvore Euclidiana, (b) Árvore Retilínea e (c) Árvore Retilínea de Steiner

Encontrar a árvore retilínea de Steiner de comprimento mínimo (RSMT) para um conjunto qualquer de pontos  $P = \{p_1, p_2, \dots, p_n\}$ , consiste em selecionar um conjunto de pontos de Steiner  $S = \{s_1, s_2, \dots, s_m\}$  tal que  $RMST(P \cup S)$  possua o menor custo.

No RSMTP, o custo (comprimento) de uma árvore  $A$ , representado por  $|A|$ , é dado pela soma dos custos das arestas que a compõem. O custo de uma

aresta que conecta os pontos  $p_i(x_i, y_i)$  e  $p_j(x_j, y_j)$  é aferido através da fórmula  $d_1(p_i, p_j) = |x_i - x_j| + |y_i - y_j|$ , conhecida como métrica L1 ou distância de Manhattan.

Muitos termos comuns no RSMT são oriundos do jargão do VLSI, tais como: pinos ou terminais, sinônimos de pontos; e *netlist* (ou *net*), que significa um conjunto de terminais.

O rápido avanço das tecnologias VLSI tornou possível o desenvolvimento de circuitos digitais com baixo custo, alto desempenho e elevado número de transistores, culminando na necessidade de realização de roteamento de *nets* cada vez maiores.

### 2.5.1 Teorema da grade de Hanan

O teorema da grade de Hanan [Hanan, 1966] constitui-se um resultado fundamental das pesquisas sobre o RSMT, seguido pela maioria (senão totalidade) das técnicas de construção de RSMT.

A grade de Hanan é obtida traçando-se linhas verticais e horizontais sobre cada ponto do conjunto de terminais, como demonstra a Figura 2.10. Hanan provou que existe ao menos uma solução ótima para o problema de geração da RSMT que utiliza apenas pontos situados nas intersecções dessas linhas. Posteriormente, Snyder generalizou os resultados de Hanan para qualquer dimensão [Snyder, 1992].

A grade de Hanan propicia uma redução importante no RSMT, permitindo-nos considerar apenas  $n^2$  pontos para a geração da RMST, sendo  $n$  terminais e  $n^2 - n$  pontos de Steiner.

Apesar da redução do espaço de soluções provida pelo teorema da grade de Hanan, o RSMT mantém-se NP-difícil [Garey e Johnson, 1977]. Zachariasen apresenta um catálogo de problemas que podem ser resolvidos utilizando a grade de Hanan [Zachariasen, 2000].

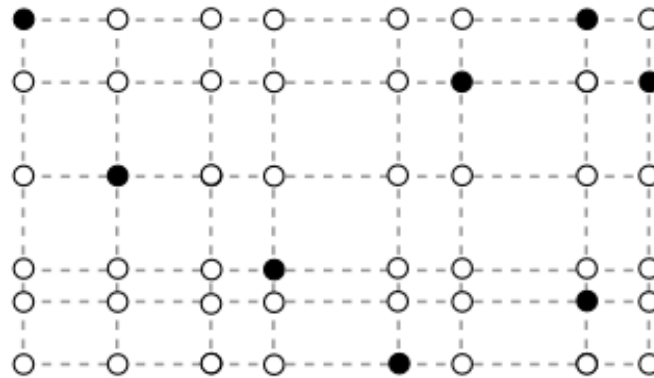


Figura 2.10: Teorema da grade de Hanan

## 2.5.2 Tipos de pontos de Steiner

Com base na grade de Hanan, os pontos de Steiner podem ser classificados de acordo com o seu grau: *corner-points*, *t-points*, ou *cross-points*, cujos graus são 2,3 e 4 respectivamente. A Figura 2.11 apresenta uma árvore de Steiner composta por terminais, *corner-points* ('a', 'b' e 'd'), *t-points* ('c') e *cross-points* ('e').

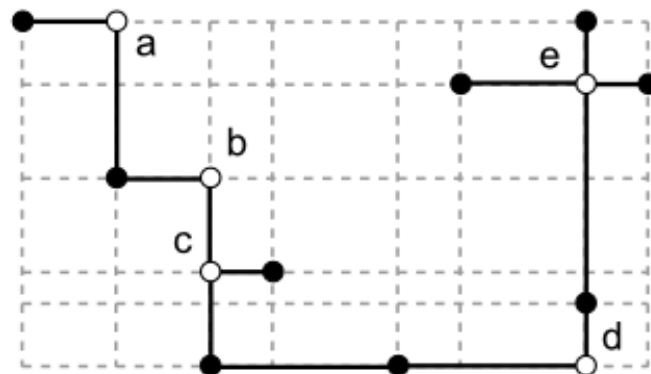


Figura 2.11: Exemplos de pontos de Steiner

Os *corner-points* podem ser removidos sem prejuízo de qualquer árvore de Steiner, pois, por possuírem grau 2, não oferecem nenhuma diminuição no custo desta. A Figura 2.12 demonstra a remoção de um *corner-point* sem impacto no custo total da árvore, visto que  $|A| = d_1(a, s) + d_1(s, b) = |B| = d_1(a, b)$ .

Por consequência da definição, a RSMT é composta apenas por terminais, *t-points* e *cross-points*. Apesar disto, algumas técnicas permitem o uso de *corner-*

*points* em fases intermediárias, pois, posteriormente, estes podem se transformar em *t-points* ou *cross-points*.

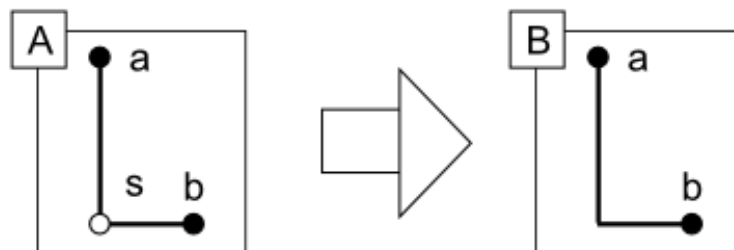


Figura 2.12: Remoção de um *corner-point*

### 2.5.3 Topologias de árvores de Steiner

Por conta da geometria do RSMT e da grade de Hanan, uma mesma árvore de Steiner pode apresentar diferentes formatos, denominados topologias. Alterar a topologia de uma árvore de Steiner consiste em mudar sua forma mantendo seus terminais, pontos de Steiner e arestas.

A técnica utilizada para alterar a topologia de uma árvore de Steiner é o *flip*: mudança do *corner* utilizado para conectar dois pontos. A Figura 2.13 demonstra quatro topologias diferentes, obtidas através de *flips*, para uma mesma árvore de Steiner.

Em consequência da grande quantidade de topologias existente para uma mesma árvore, o espaço de soluções para o RSMT é composto por muitas regiões planas: vários mínimos locais de mesmo custo e alguns mínimos globais. Esta peculiaridade tem duas implicações:

1. É relativamente fácil encontrar soluções de boa qualidade para o RSMT;
2. Uma vez encontrado um mínimo local, a homogeneidade da vizinhança torna o processo refinamento bastante árduo;

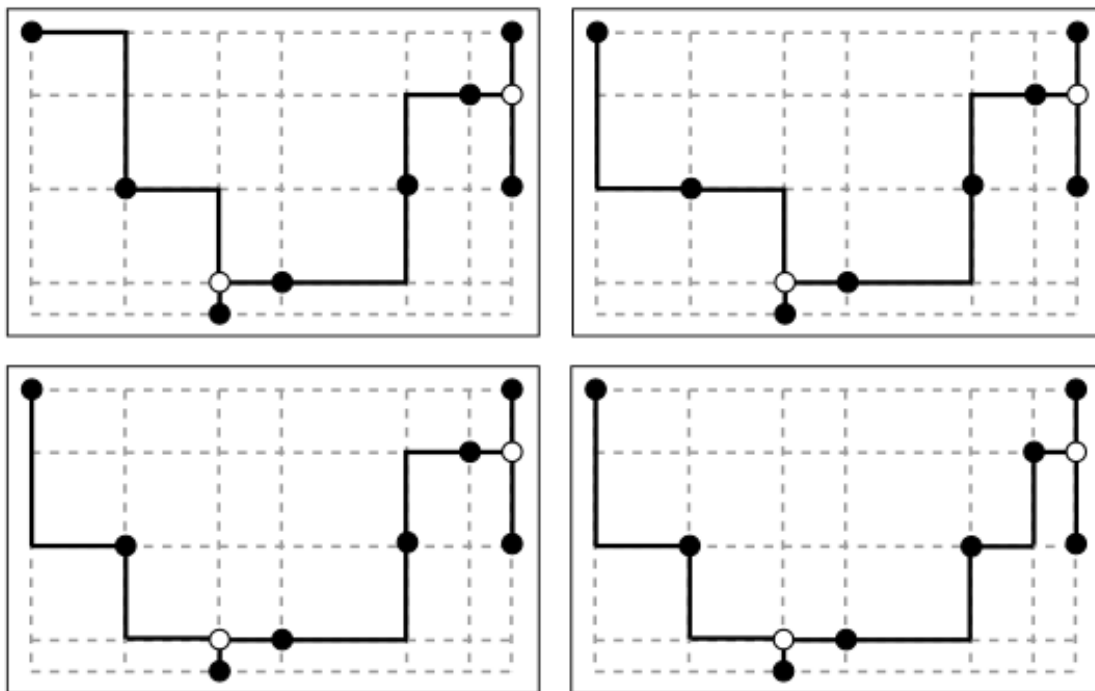


Figura 2.13: Diferentes topologias de uma mesma árvore retilínea de Steiner

### 2.5.4 Árvores Retilíneas *Full-Steiner*

Uma árvore retilínea *Full-Steiner* (*Rectilinear Full Steiner Tree* - RFST) é uma árvore de Steiner em métrica L1 na qual todo terminal é folha (e toda folha é terminal) e todos os outros vértices – pontos de Steiner – possuem grau três ou quatro [Zachariasen, 2000].

Uma RSMT é uma união de RFSTs e sempre existe uma RSMT na qual cada RFST que a compõe possui uma das duas topologias de Hwang [Hwang, 1976], exibidas na Figura 2.14.

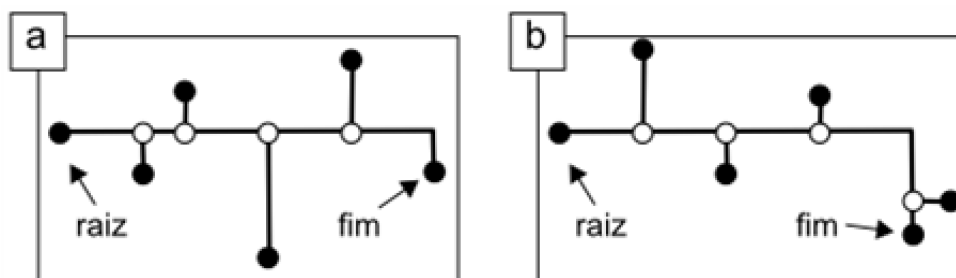


Figura 2.14: Topologias de Hwang para RFSTs



Uma RFST conectando  $k$  terminais consiste em um 'L' composto por um ponto inicial (denominado raiz) e um ponto final (denominado fim). A raiz incide sobre a "perna longa" e o fim sobre a "perna curta" do 'L'. A RFST de Hwang tipo (a) possui  $k - 2$  segmentos alternados incidentes à perna longa e nenhum segmento incidente à perna curta, enquanto a tipo (b) possui  $k - 3$  segmentos alternados incidentes à perna longa e um segmento incidente à perna curta.

Vale salientar que os conceitos de perna longa e perna curta não estão relacionados ao comprimento do segmento, mas sim à quantidade de pontos que incidem sobre este.

As definições de RFST e das topologias de Hwang constituem a base do melhor algoritmo de geração de RSMT da atualidade [Zachariasen, 2000], o GeoSteiner.

### 2.5.5 Algoritmos de Árvore Geradora Retilínea Mínima

A íntima relação entre o RSMT e a árvore geradora retilínea mínima (RMST, do inglês: *Rectilinear Minimum Spanning Tree*) acarretou que muitas heurísticas de construção de RSMT utilizam a RMST como base.

As primeiras heurísticas de RSMT eram simples melhorias dos algoritmos de RMST, seja através de alterações na fase de geração da RMST ou refinamentos a partir desta.

No entanto, a tarefa de gerar a árvore de cobertura mínima de um grafo completo não é tão simples quanto parece. A complexidade dos algoritmos de Kruskal e Prim é  $O(e \log v)$  para um grafo qualquer composto por  $v$  vértices e  $e$  arestas [Cormen et al., 2001]. Todavia, o número de arestas de um grafo completo é dado por  $e = v^2$ , aumentando a complexidade para  $O(v^2 \log v)$ . Na prática essa complexidade pode ser diminuída para  $O(v^2)$ .

[Hwang, 1976] propôs um algoritmo de geração de RMST em  $O(v \log v)$ , baseada na construção do diagrama de Voronoi seguida da triangulação de Delaunay. Contudo, por conta da complexidade da geometria utilizada, a implementação dessa abordagem é bastante complexa.

Uma alternativa, de programação mais simples, é realizar uma redução de grau no grafo. [Guibas e Stolfi, 1983] provam que para encontrar uma RMST, basta considerar o grafo no qual cada terminal liga-se apenas com o seu vizinho mais próximo em cada octante (Figura 2.15). Este novo grafo, de grau oito, pode ser construído em  $O(v \log v)$  e a computação da RMST deste grafo se dá em  $O(v \log v)$  por Kruskal ou Prim.

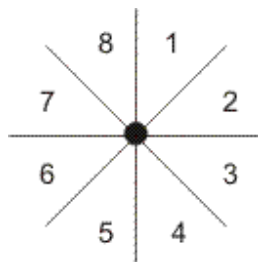


Figura 2.15: Octantes de um ponto qualquer

[Kahng e Mandoiu, 2001] apresenta um estudo comparativo entre três procedimentos de geração de RMST:

1. Uma eficiente implementação do algoritmo de Prim em  $O(v^2)$ .
2. Uma implementação em  $O(v \log v)$  da redução de grau de Guibas e Stolfi seguida do algoritmo de Prim.
3. Uma implementação do algoritmo de Prim com computação dinâmica do vizinho mais próximo em cada octante, proposta por Dr. Lou Scheffer.

A Tabela 2.1 exibe os tempos de execução médios – em segundos – em uma CPU “195MHz SGI Origin 2000”), para instâncias de até 500.000 vértices, de cada um dos algoritmos de RMST estudados em [Kahng e Mandoiu, 2001].

# de Terminais	$O(n^2)$ Prim	Scheffer	Guibas-Stolfi
5	<b>0.000006</b>	0.000400	0.000053
10	<b>0.000024</b>	0.000685	0.000138
50	<b>0.000567</b>	0.003601	0.001118
100	<b>0.002269</b>	0.007959	0.002613
500	0.055323	0.051744	<b>0.017536</b>
1000	0.223079	0.118234	<b>0.038819</b>
5000	5.774400	0.889230	<b>0.243520</b>
10000	23.641100	2.477000	<b>0.531860</b>
50000	N/A	22.685750	<b>3.461500</b>
100000	N/A	75.654200	<b>9.253000</b>
500000	N/A	486.621200	<b>91.634800</b>

Tabela 2.1: Tempo de execução médio dos algoritmos de geração de RMST

Contudo, [Hwang, 1976] provou que as heurísticas de geração de RSMT baseadas em melhorias sobre a RMST têm a qualidade da solução limitada a  $\frac{2}{3}|RMST|$  (proporção de Hwang). Visando a superação deste limite, os novos métodos que surgiram utilizam outras abordagens – tal como preceitos geométricos – para resolver o RSMT.

Os algoritmos de RMST, porém, mantêm sua importância, pois, mesmo as técnicas recentes de geração de RSMT se valem do cálculo da RMST. O BI1S (*Batched Iterated First Steiner* [Griffith et al., 1994]), por exemplo, utiliza a RMST para aferir o ganho da inserção de cada ponto de Steiner na solução final.

### 2.5.6 Half Perimeter Wirelength

O *half perimeter wirelength* (HPWL), metade do perímetro do menor retângulo que contenha todos os pontos de uma *net*, é utilizado comumente pelos posicionadores (*placers*) como estimativa rápida do custo de roteamento final no circuito.

O HPWL de um conjunto de pontos  $P$  ordenado por suas coordenadas  $x$  e  $y$  dá-se em  $O(1)$  pela fórmula:  $HPWL(P) = (x_{\max} - x_{\min}) + (y_{\max} - y_{\min})$ . Caso  $P$  não esteja ordenado, localizar  $x_{\max}$ ,  $x_{\min}$ ,  $y_{\max}$  e  $y_{\min}$  é feito em  $O(n)$ .

O FLUTE [Chu, 2008-A] utiliza o HPWL em sua heurística para *nets* com 10 ou mais terminais.

### 2.5.7 Aresta e Distância Gargalo

Seja  $A$  uma árvore (ou um grafo) composta por um conjunto  $P$  de terminais tal que  $p_i, p_j \in P$ . A aresta de gargalo (*bottleneck edge*) entre  $p_i$  e  $p_j$  é a aresta de maior custo no(s) caminho(s) entre  $p_i$  e  $p_j$  em  $A$ . A distância de gargalo (*bottleneck distance*) é o custo da aresta de gargalo.

Note que, seja  $b(p_i, p_j)$  a distância de gargalo entre  $p_i$  e  $p_j$  na  $RMST(P)$ , nenhuma aresta do caminho entre  $p_i$  e  $p_j$  na  $RSMT(P)$  terá comprimento maior que  $b(p_i, p_j)$  [Zachariasen, 1997].

O cálculo da aresta de gargalo é importante para a remoção de ciclos em uma árvore. Algumas abordagens para o RSMT – o BGA, por exemplo – são baseadas na adição de pontos de Steiner à RMST e manutenção do comprimento mínimo pela remoção da aresta de gargalo de cada ciclo formado.

### 2.5.8 Redução de Pontos

As técnicas de redução de pontos constituem uma fase de pré-processamento importante no RSMT. Consistem em remoções de pontos de Steiner da grade de Hanan, tal que exista ao menos uma árvore de Steiner ótima construída com os pontos remanescentes.

[Winter, 1995] discute um conjunto de reduções de pontos e arestas baseado no grau dos vértices e nas distâncias de gargalo do grafo da grade de Hanan. Em média, as reduções propostas por Winter diminuem em 70-80% o número de pontos de Steiner que devem ser considerados.

A Figura 2.16 traz um exemplo das três fases das reduções de Winter. Para uma instância contendo inicialmente oito terminais e cinquenta e seis pontos de Steiner. Aplicadas as reduções, restaram apenas dezesseis pontos de Steiner (diminuição de aproximadamente 72%).

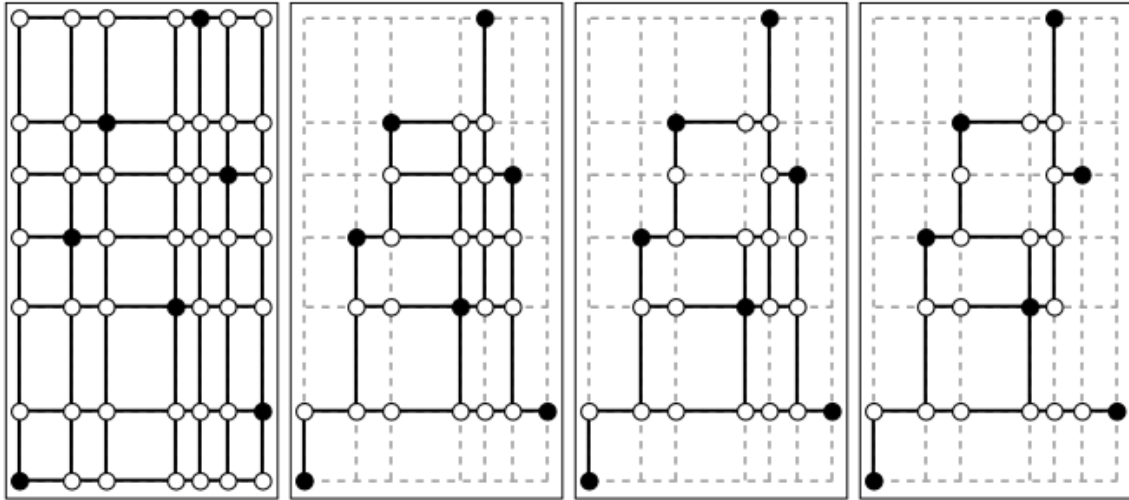


Figura 2.16: Três fases da redução de pontos e arestas de Winter

Outra maneira de reduzir os pontos da árvore de Hanan é através de testes de regiões vazias. O teorema das regiões vazias define que se o segmento  $uv$  pertence à RMST, não existe nenhum outro ponto (Steiner ou terminal) dentro das regiões vazias de  $uv$ .

[Zachariasen, 1997] demonstra quatro regiões vazias para o RSMT, representadas na Figura 2.17: (a) o diamante, (b) o retângulo, (c) o triângulo e (d) o círculo vazios.

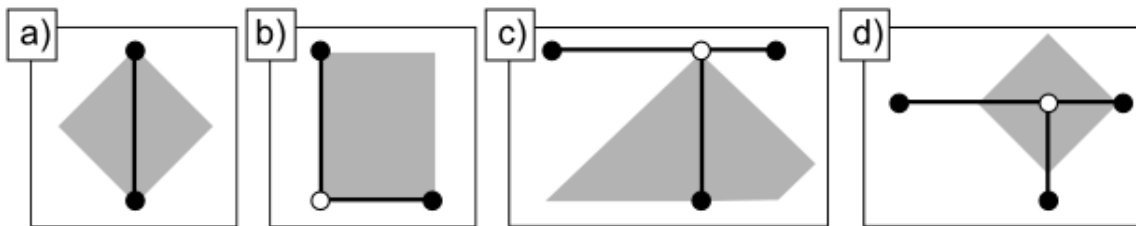


Figura 2.17: Regiões vazias: a) diamante, b) retângulo, c) triângulo e d) círculo

O GeoSteiner [Warme, Winter e Zachariasen, 2009] se vale de testes de diamante e retângulo vazio na fase de geração das RFSTs para eliminar RFSTs que não satisfaçam estas condições e, por consequência, não fazem parte da solução ótima.

## CAPÍTULO 3 - O ESTADO DA ARTE

Neste capítulo serão discutidas as técnicas de RSMTTP mais bem sucedidas da atualidade, objetivando a compreensão das suas melhores características para possível utilização no desenvolvimento de novas abordagens.

Existem três classes de técnicas utilizadas para solucionar o problema da menor árvore retilínea de Steiner:

- Abordagens exatas: técnicas utilizadas para resolver instâncias que contêm número reduzido de pontos.
- Heurísticas semi-ótimas: técnicas utilizadas para resolver instâncias maiores do problema, com ênfase à aplicação na fase de roteamento de chips da classe FPGA (*Field Programmable Gate Array*).
- Heurísticas rápidas: técnicas utilizadas nas fases iniciais do projeto de *chips* – como a síntese ou o posicionamento -- para estimar o comprimento final do roteamento.

A Figura 3.1 exibe a classificação das abordagens estudadas neste capítulo segundo as classes apresentadas.

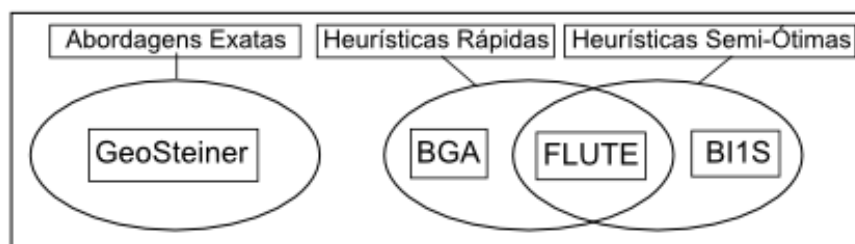


Figura 3.1: Classes de Técnicas para o RSMTTP

São utilizados dois métodos para qualificar uma heurística  $H$  de RSMTTP:

- A melhoria média propiciada pelas soluções de  $H$  sobre as respectivas RMSTs, dada por  $\delta(H) = \frac{(|RMST| - |H|)}{|RMST|}$ ;

- A diferença média entre as soluções de  $H$  e as respectivas soluções ótimas, dada por  $FFO(H) = (|H| - |\text{Ótimo}|) / |H|$ ;

### 3.1 BI1S

O *Batched Iterated 1-Steiner* (BI1S) consiste em uma implementação prática da heurística *Iterated 1-Steiner* (I1S), utilizando um novo esquema de manutenção dinâmica da RMST e o conceito de liberdade entre pontos de Steiner.

O I1S [Kahng e Robins, 1992] foi proposto como alternativa às heurísticas baseadas na construção de RMST, apresentando desempenho que supera a proporção de Hwang.

A idéia central do I1S é encontrar, iterativamente, o ponto de Steiner que cause o maior decréscimo no custo da solução, dito 1-Steiner, e adicioná-lo até que nenhuma melhoria seja possível. Sejam  $P$  e  $S$  dois conjuntos de pontos, a melhoria de  $S$  sobre  $P$  é definida como:  $\Delta RMST(P, S) = RMST(P) - RMST(P \cup S)$ . Então, o ponto 1-Steiner  $s \in S$  maximiza  $\Delta RMST(P, \{s\}) > 0$ .

Apesar de uma RSMT conter no máximo  $n - 2$  pontos de Steiner [Kahng e Robins, 1992], o I1S pode adicionar mais do que  $n - 2$  pontos. Por isso, a cada iteração são eliminados os pontos de Steiner de grau dois ou menos. A Figura 3.2 ilustra uma execução simples do I1S para uma instância composta por quatro terminais.

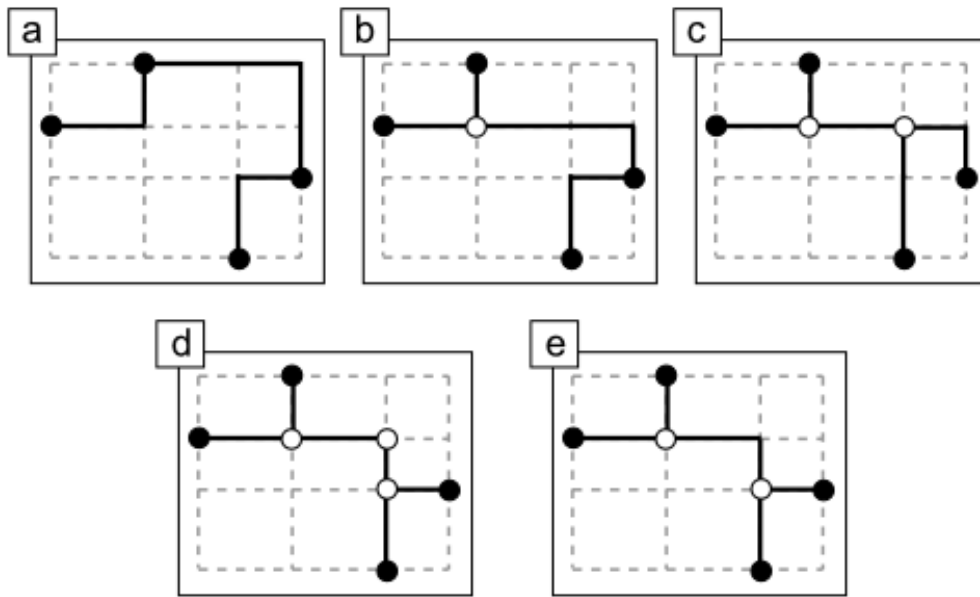


Figura 3.2: Execução simples do I1S

Atente que na Figura 3.2(d) existem três pontos de Steiner, ultrapassando o limite de  $n - 2 = 2$  permitido por esta instância. Por isso, na Figura 3.2(e) há a remoção de um ponto de Steiner de grau dois.

Apesar de apresentar uma qualidade interessante:  $\delta(I1S) \cong 11\%$  e  $FPO(I1S) \cong 0.5\%$ , o I1S possui uma complexidade computacional muito alta:  $O(n^4 \log n)$ .

O BI1S [Griffith et al., 1994], consiste em uma melhoria na complexidade computacional do I1S, sem afetar sua qualidade, utilizando o conceito de independência entre pontos e um esquema de manutenção dinâmica da árvore geradora mínima em distância retilínea.

As inovações apresentadas pelo BI1S, reduzindo a complexidade do I1S para  $O(n^3)$ , fizeram esta heurística ostentar a melhor qualidade média entre as técnicas de RSMT por muitos anos, sendo superada apenas em 2004 pelo FLUTE [Chu, 2008-A].



### 3.1.1 Independência entre pontos

Dois pontos  $a$  e  $b$  são ditos independentes se a inserção do ponto  $a$  na  $RMST(P \cup \{b\})$  não diminuir o  $\Delta RMST(P, \{b\})$ . A idéia principal contida no BI1S consiste em, a cada interação, ao invés de adicionar o 1-Steiner, adicionar o conjunto  $S$  de pontos independentes de Steiner que maximize  $\Delta RMST(P, S)$ .

Formalmente, um conjunto  $S$  de pontos de Steiner é dito independente se satisfaz a condição:  $\Delta RMST(P, S) \geq \sum_{x \in S} \Delta RMST(P, \{x\})$ .

A tarefa de encontrar o melhor conjunto de pontos de Steiner independentes herda a complexidade NP-difícil do RSMTP. Contudo, uma implementação gulosa apresenta bons resultados na prática [Griffith et al., 1994].

Uma vez selecionado o (aproximadamente) melhor conjunto de pontos independentes de Steiner, este é inserido na solução e inicia-se uma nova iteração do BI1S. Este processo é repetido até que uma iteração falhe em adicionar pontos de Steiner.

O número de iterações executadas pelo BI1S – detalhadas na Tabela 3.1 – é uma constante que independe do tamanho da instância e apresenta um valor pequeno (em média menor que três).

n	Pontos de Steiner			Iterações		
	min	média	max	min	média	max
3	0	0,90	1	1	1,90	2
4	0	1,53	2	1	2,18	3
5	1	2,25	3	2	2,33	4
7	1	3,63	6	2	2,60	5
10	2	5,63	8	2	2,92	7
14	5	8,22	11	2	3,18	6
20	9	12,14	15	2	3,26	5
30	15	18,99	22	3	3,53	6
50	29	32,86	35	3	4,14	6

Tabela 3.1: Número de iterações e número de pontos adicionados por iteração no BI1S

### 3.1.2 Manutenção dinâmica da RMST

[Griffith et al., 1994] propõe um novo esquema de manutenção dinâmica da RMST com o intuito de agilizar o cálculo da melhoria que a adição de um conjunto de pontos causa sobre esta (base do BI1S).

Uma vez calculada a árvore geradora mínima para um conjunto de pontos  $P$ , o acréscimo de um único ponto  $x$  a  $P$  produz apenas um pequeno e constante número de alterações entre  $RMST(P)$  e  $RMST(P \cup \{x\})$ .

Parte-se da observação de que para a manutenção dinâmica da RMST, é suficiente considerar apenas quatro vizinhos de cada novo ponto de Steiner adicionado, o mais próximo para cada região definida por duas retas orientadas a  $45^\circ$  e  $-45^\circ$  (Figura 3.3), dita partição diagonal de  $x$ .

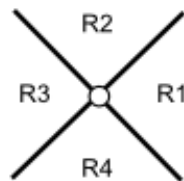


Figura 3.3: Quadrantes definidos pela partição diagonal de um ponto

Então, para manter a RMST em consequência da adição de um novo ponto  $x$ , basta conectar  $x$  ao seu vizinho mais próximo em cada região de sua partição diagonal e remover a aresta de maior comprimento de cada ciclo formado. A Figura 3.4 demonstra a inclusão de um novo ponto em uma árvore com manutenção dinâmica da RMST.

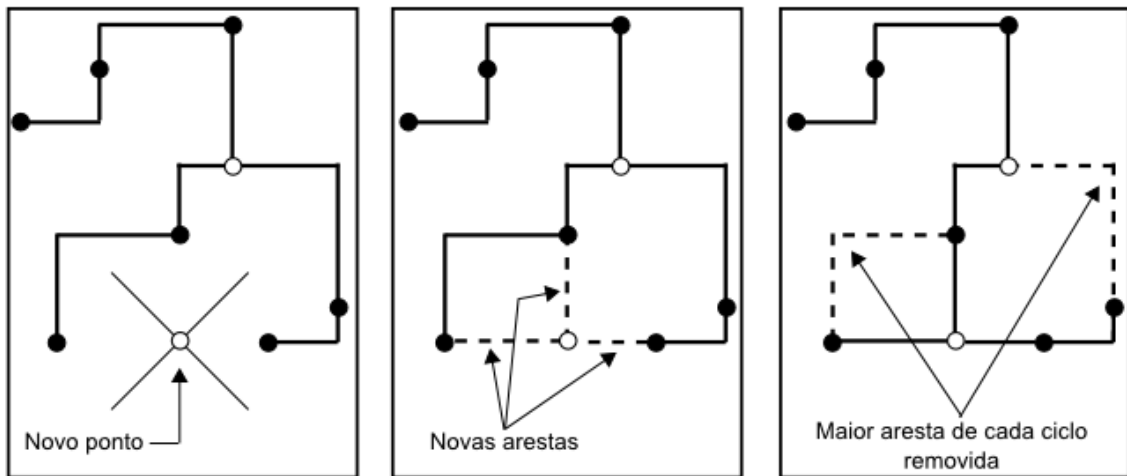


Figura 3.4: Manutenção dinâmica da RMST

A inclusão da técnica de manutenção dinâmica de RMST reduz a complexidade do BI1S para  $O(Kn^3)$ , onde  $K$  é o número de iterações realizadas pela heurística. Como o número de iterações é constante, tem-se que a complexidade do BI1S é  $O(n^3)$ .

### 3.1.3 Outras melhorias no BI1S

Outras melhorias na velocidade do BI1S são advindas de técnicas de redução de pontos. Madoiu propõe uma melhoria significativa no tempo de execução do BI1S baseada no teste do retângulo vazio [Madoiu, 1999].

## 3.2 BGA

O BGA (*Batched Greedy Algorithm*) [Kahng, Madoiu e Zelikovsky, 2003] foi proposto com o intuito de resolver instâncias muito grandes do RSMT em tempo hábil e sem perda de qualidade. Para tal, apresenta uma heurística em  $O(n \log^2 n)$  que utiliza  $O(n)$  memória. Este tempo de execução caracteriza o BGA como primeira heurística sub-quadrática para o RSMT.

A eficiência do BGA é proveniente de três conceitos chave:

- Uma combinação da implementação do GTCA (*Greedy Triple Contraction Algorithm*) [Zelikovsky, 1992] com a idéia do conjunto de pontos independentes B1S.
- Um novo método de divisão-e-conquista para calcular o conjunto de triplas requerido pelo GTCA.
- Uma estrutura de dados linear que possibilita a localização da aresta de gargalo em  $O(\log n)$ , após um pré-processamento em  $O(n \log n)$ .

O GTCA consiste em computar a (aproximadamente) menor árvore 3-restrita para um conjunto de pontos, sendo uma árvore  $k$ -restrita composta por RFSTs (denominadas componentes) contendo no máximo  $k$  terminais. A Figura 3.5 demonstra uma árvore 3-restrita.

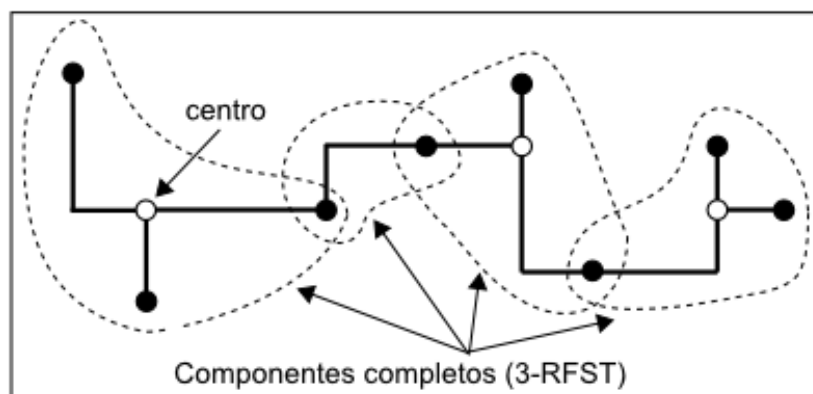


Figura 3.5: Árvore 3-restrita

A cada iteração, o GTCA seleciona gulosamente uma tripla  $\tau$ , onde tripla é uma RSMT de uma net de três pontos – i.e. um componente 3-restrito – que reduza o custo da RMST, e adiciona-a à solução. Para tal, é necessária a computação de todas as triplas possíveis e seus ganhos, tal que o ganho de uma tripla  $\tau$ , denotado  $\Delta RMST(P, \tau)$ , é calculado de maneira análoga ao cálculo do ganho na inserção de um ponto de Steiner no I1S.

O número de triplas necessário, porém, é pequeno, pois se prescindem as triplas que não diminuem o custo da RMST ( $\Delta RMST(P, \tau) \leq 0$ ) e as que não são vazias [Fößmeier, Kaufmann e Zelikovsky, 1997]. Uma tripla  $\tau$  é vazia se o fecho convexo desta não contiver nenhum outro terminal.

O BGA utiliza-se de uma abordagem de divisão-e-conquista que computa em  $O(n \log n)$  um conjunto contendo todas as  $O(n \log n)$  triplas vazias.

Uma vez obtido o conjunto de todas as triplas vazias, o BGA seleciona, a cada iteração, o (aproximadamente) melhor conjunto de triplas independentes e adiciona à solução. O conceito de independência entre triplas é análogo ao de independência de pontos do B1S.

É fácil perceber que a computação do ganho de cada tripla implica em encontrar e eliminar a aresta gargalo de cada ciclo formado sua inserção. O *hierarchical greedy preprocessing* (HGP), computa, para uma dada árvore com  $n$  terminais, dois vetores auxiliares, *parent* e *edge*, com no máximo  $2n - 1$  elementos cada. Usando estes vetores, a aresta de gargalo entre dois terminais quaisquer é localizada em  $O(\log n)$ .

Dado um conjunto de arestas ordenado ascendentemente de acordo com seu custo, o procedimento HGP – descrito na Figura 3.6 – calcula os vetores *edge* e *parent* em  $O(n)$ . A complexidade do HGP é dominada, portanto, pela ordenação  $O(n \log n)$  das arestas.

<b>Entrada:</b>	Árvore $T(V=\text{vértices}, E=\text{arestas})$ com $n$ vértices
<b>Saída:</b>	Arrays $\text{edge}(i)$ e $\text{parent}(i)$ , $i=1, \dots, 2n-1$
1.	Ordene as arestas em ordem ascendente de custo
2.	Inicialização $\text{next} \leftarrow n$ <b>PARA</b> cada $i=1, 2, \dots, 2n-1$ <b>FAÇA</b> $\text{parent}(i) \leftarrow \text{NULL}$ $\text{edge}(i) \leftarrow \text{NULL}$
3.	<b>PARA</b> cada aresta $e(i) = (u, v)$ , $i=1, \dots, n-1$ , <b>FAÇA</b> <b>ENQUANTO</b> $u \neq v$ <b>E</b> $\text{parent}(u) \neq \text{NULL}$ <b>E</b> $\text{parent}(v) \neq \text{NULL}$ <b>FAÇA</b> $u \leftarrow \text{parent}(u)$ $v \leftarrow \text{parent}(v)$  <b>SE</b> $\text{parent}(u) = \text{parent}(v) = \text{NULL}$ , <b>ENTÃO</b> $\text{next} \leftarrow \text{next} + 1$ $\text{parent}(u) \leftarrow \text{parent}(v) \leftarrow \text{next}$ $\text{edge}(u) \leftarrow \text{edge}(v) \leftarrow i$  <b>SE</b> $\text{parent}(u) = \text{NULL}$ <b>E</b> $\text{parent}(v) \neq \text{NULL}$ , <b>ENTÃO</b> $\text{parent}(u) \leftarrow \text{parent}(v)$ $\text{edge}(u) \leftarrow i$  <b>SE</b> $\text{parent}(u) \neq \text{NULL}$ <b>E</b> $\text{parent}(v) = \text{NULL}$ , <b>ENTÃO</b> $\text{parent}(v) \leftarrow \text{parent}(u)$ $\text{edge}(v) \leftarrow i$
4.	Retorne os arrays $\text{parent}$ e $\text{edge}$

Figura 3.6: Pseudocódigo do HGP [Kahng, Mandoiu e Zelikovsky, 2003]

Computados os vetores  $\text{edge}$  e  $\text{parent}$ , o procedimento da Figura 3.7 encontra a aresta de gargalo no caminho entre dois terminais  $O(\log n)$ .

<b>Entrada:</b>	Árvore conjunto $E$ de arestas ordenado; arrays $\text{edge}$ e $\text{parent}$ ; e os terminais $u$ e $v$
<b>Saída:</b>	Aresta de maior custo no caminho entre $u$ e $v$
1.	$\text{index} \leftarrow -\infty$
2.	<b>ENQUANTO</b> $u \neq v$ <b>FAÇA</b> $\text{index} \leftarrow \max\{\text{index}, \text{edge}(u), \text{edge}(v)\}$ $u \leftarrow \text{parent}(u)$ $v \leftarrow \text{parent}(v)$
3.	Retorne $e(\text{index})$

Figura 3.7: Rotina de computação da aresta de gargalo do caminho entre dois terminais  $u$  e  $v$

A inclusão do HGP e do conceito do conjunto de triplas independentes fazem cada iteração do BGA possuir uma complexidade computacional de  $O(n \log^2 n)$ . Como, na prática, assim como no BI1S, o número de iterações é pequeno, esta é a complexidade do BGA.

Apesar de apresentar, em média, soluções de qualidade um pouco inferior se comparado ao BI1S ( $\delta(BGA) \cong 11\%$  e  $FFO(BGA) \cong 0.6\%$ ), o BGA caracteriza-se, por conta de seu tempo de execução e da utilização de estrutura de memória linear, como a heurística mais escalonável para o RSMT, devido ao seu tempo de execução e pela utilização de estrutura de memória linear. Ela é capaz de encontrar boas soluções para instâncias compostas por milhares de terminais.

### 3.3 FLUTE

O *Fast Lookup Table Based Wirelength Estimation Technique* (FLUTE), proposta inicialmente por Chris Chu em 2004, consiste em uma heurística de geração de RSMT rápida e de excelentes resultados para instâncias com pequeno número de pontos. Estudos empíricos comprovam o comportamento ótimo do FLUTE para *nets* com nove ou menos terminais [Chu, 2008-A].

A heurística baseia-se na pré-computação de topologias potencialmente ótimas, relacionadas à posição relativa entre os pontos da instância. Estas topologias são guardadas em tabelas (*Lookup Tables*), e a computação da RSMT reduz-se à minimização do custo destas topologias pré-tabeladas.

O cálculo das topologias possíveis para cada instância depende dos conceitos de sequência vertical, distância marginal e vetor de coeficientes.

#### 3.3.1 Sequência Vertical

Dada uma instância  $P$  qualquer, composta por  $n$  pontos  $(x_i, y_i)$ , para  $1 \leq i \leq n$ , ordenados ascendentemente em respeito à sua coordenada  $x$ , assumamos  $x_1 \leq x_2 \leq \dots \leq x_n$ . Define-se sequência vertical  $s_1 s_2 \dots s_n$  como a lista de índices dos

pontos de  $P$  ordenados de acordo com a coordenada  $y$ . A Figura 3.8 exibe uma instância composta por quatro terminais cuja sequência vertical é 3142.

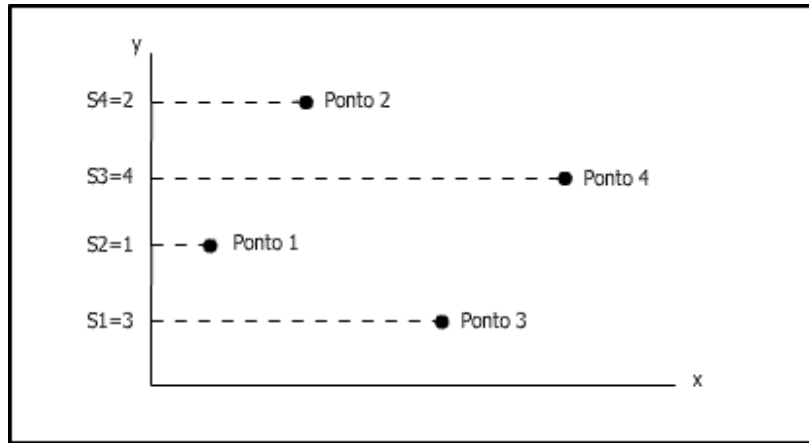


Figura 3.8: Sequência Vertical

Em outros termos, a sequência vertical de uma instância classifica-a de acordo com a posição relativa entre seus pontos. Instâncias que compartilham a mesma sequência vertical compõem um grupo cujas RSMT podem ser construídas obedecendo a um pequeno número de topologias pré-definidas.

### 3.3.2 Distância Marginal

Define-se como distância marginal horizontal (vertical), a distância entre duas linhas horizontais (verticais) adjacentes na grade de Hanan. Denotam-se todas as distâncias entre margens horizontais  $h_i = x_{i+1} - x_i$  e verticais  $v_i = y_{s_{i+1}} - y_{s_i}$ , para  $1 \leq i \leq n$ . Estas definições podem ser observadas na Figura 3.9.



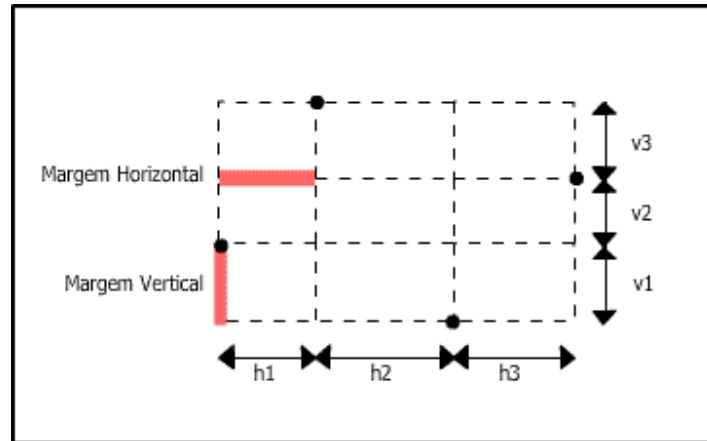


Figura 3.9: Distância marginal

### 3.3.3 Vetores de Coeficientes Potencialmente Ótimos

Note que qualquer solução para o RSMTTP pode ser escrita como uma combinação linear de distâncias marginais cujos coeficientes são números inteiros positivos. Por exemplo, para a instância da Figura 3.8, as três possíveis soluções mostradas na Figura 3.10 (a), (b) e (c) podem ser escritas nas formas  $h_1 + 2h_2 + h_3 + v_1 + v_2 + 2v_3$ ,  $h_1 + h_2 + h_3 + v_1 + 2v_2 + 3v_3$  e  $h_1 + 2h_2 + h_3 + v_1 + v_2 + v_3$ , respectivamente. Por simplicidade, é utilizado o vetor dos coeficientes das distâncias marginais como notação. Para a Figura 3.10 (a) temos (1,2,1,1,1,2).

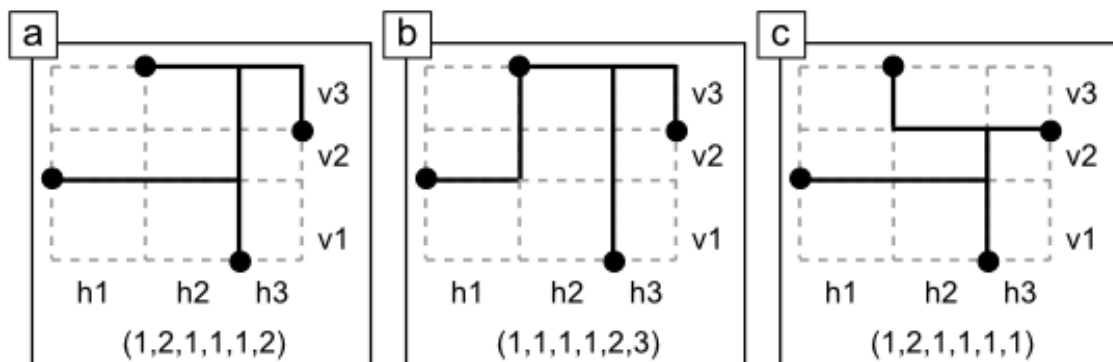


Figura 3.10: Vetores de Coeficientes

Para cada grupo – instâncias que compartilham mesma sequência vertical – é calculado o conjunto de vetores de coeficientes que podem levar à solução ótima (POWV, do inglês *Potentially Optimal Wirelength Vector*). Muitos vetores são

redundantes, pois apresentam valores maiores ou iguais do que outro vetor em todos os coeficientes. Por exemplo, o vetor de coeficientes da Figura 3.10 (a) (1,2,1,1,1,2) pode ser ignorado, pois o vetor de coeficientes da Figura 3.10 (c) (1,2,1,1,1,1) é sempre menor do que este.

Uma instância contendo  $n$  terminais possui exatamente  $n!$  grupos (sequências verticais), e cada grupo apresenta um pequeno número de POWVs, como demonstra a Tabela 3.2. Por conta disto, é possível pré-calculer todos os POWVs possíveis para pequenas instâncias.

n	n!	# POWVs por grupo		
		min	média	max
2	2	1	1,000	1
3	6	1	1,000	1
4	24	1	1,667	2
5	120	1	2,467	3
6	720	1	4,433	8
7	5040	1	7,932	15
8	40320	1	15,803	34
9	362880	1	30,039	79

Tabela 3.2: Relação de POWVs por grupo

Uma vez de posse do conjunto de POWVs e dos custos das distâncias marginais para a instância apresentada, a computação da RSMT consiste apenas em algumas somas e multiplicações com o objetivo de encontrar o POWV de menor custo.

Utilizando técnicas complexas, Chu conseguiu gerar todos os POWVs para instâncias compostas por nove ou menos terminais, garantindo o comportamento ótimo do FLUTE para estas. A partir daí, a persistência é fator proibitivo para computação, tabelamento e acesso aos POWVs.

### 3.3.4 FLUTE para Instâncias com Dez ou Mais Terminais

Para instâncias compostas por dez ou mais pontos, o tempo necessário para gerar as tabelas de POWVs e a quantidade de memória necessária para guardá-las tornam o FLUTE impraticável. Para estes problemas, é utilizada uma

técnica de quebra da instância, dividindo-a em sub-instâncias de dois a nove terminais.

Para quebrar uma instância, escolhe-se um ponto, dito ponto de quebra, e uma direção (horizontal ou vertical), separando as instâncias em duas, de acordo com as coordenadas do ponto de quebra e a direção escolhida. O ponto de quebra é incluído em ambas as sub-instâncias, que são solucionadas independentemente. A quebra é, então, realizada recursivamente até que nenhuma sub-instância possua mais que nove pontos. A Figura 3.11 (a) demonstra uma quebra de instância em direção horizontal, enquanto a Figura 3.11 (b) exibe a RSMT gerada para cada sub-instância.

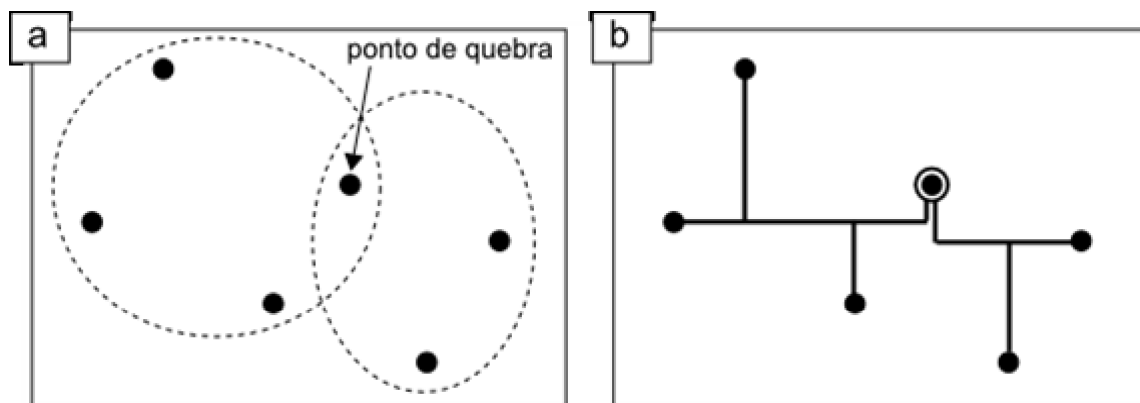


Figura 3.11: Quebra de Instância

A escolha da direção e do ponto de quebra afeta diretamente a qualidade da solução, ocasionando a perda do comportamento ótimo do FLUTE para instâncias com dez ou mais terminais.

A primeira versão da heurística (FLUTE 1.0 [Chu, 2004]), utiliza o HPWL como heurística subordinada para estimar o melhor ponto de quebra em cada direção. A técnica é, então, executada para cada ponto/direção selecionados, retornando apenas o melhor resultado.

No FLUTE 2.0 [Chu e Wong, 2005], são apresentadas três heurísticas de quebra de instâncias e o parâmetro  $A$ , definido como número de maneiras em que uma instância será quebrada. Apesar de aumentar substancialmente o tempo de execução do algoritmo, propicia ao usuário a escolha entre qualidade e tempo de

execução do programa. A Tabela 3.3 exibe um comparativo entre qualidade da solução e o valor do parâmetro  $A$ , onde se constatou o melhor custo-benefício para  $A=3$ . Nas versões mais recentes do FLUTE, são recomendados  $A=6$  e  $A=12$  para uma maior qualidade na solução.

Parâmetro $A$	% FFO	Tempo de Execução	
		(s)	Normalizado
$A=1$	0,330	5,04	0,61
$A=2$	0,151	6,16	0,74
<b><math>A=3</math></b>	<b>0,070</b>	<b>8,31</b>	<b>1,00</b>
$A=4$	0,039	12,10	1,46
$A=5$	0,026	18,36	2,21
$A=6$	0,020	29,60	3,56
$A=7$	0,016	51,38	6,18
$A=8$	0,013	96,35	11,59
$A=9$	0,012	190,67	22,94

Tabela 3.3: Parâmetro  $A$

Recentemente, Chu apresentou o FLUTE 3.0 [Chu, 2008-B], com melhorias significativas em relação às versões anteriores. As maiores contribuições da nova versão são a técnica de quebra das instâncias baseada em RMST e as heurísticas de junção FLUTE-MR (*Merge Redundant*) e FLUTE-AM (*Aggressive Merge*).

### 3.3.5 Complexidade Computacional e Resultados

Por possuir os POWVs já pré-computados, o FLUTE possui um tempo de execução sub-quadrático, no qual se pode, através do parâmetro  $A$ , optar por uma solução melhor em troca de um maior custo computacional. A sua complexidade computacional é  $O(A!n \log n)$ .

Dentre as heurísticas de RSMT contemporâneas, o FLUTE 3.0 com  $A=12$  é a técnica que apresenta as soluções mais próximas do ótimo ( $FFO(FLUTE12) \cong 0.3\%$ ).

### 3.4 GeoSteiner

O algoritmo exato mais rápido para o RSMT é o GeoSteiner [Warme, Winter e Zachariasen, 2009]. Esta abordagem utiliza um esquema em duas fases: Primeiramente um pequeno, mas suficiente, conjunto de RFSTs é gerado e então, uma RSMT é gerada a partir deste conjunto utilizando *backtrack*, técnicas de programação dinâmica ou programação inteira.

#### 3.4.1 Algoritmo de Geração de RFST

Baseado nas topologias de Hwang existe no máximo  $n!$  RFSTs para uma dada instância contendo  $n$  pontos. O GeoSteiner, porém, exclui uma grande quantidade de RFSTs que, notoriamente, não fazem parte da árvore ótima.

O algoritmo de geração de RFST baseia-se no gradual crescimento de RFSTs. Para um dado terminal  $p_0$  e uma direção  $\alpha$ , tenta-se aumentar uma RFST incidindo-se pontos na “perna longa” (denotada  $l_0$ ) formada pelo segmento de direção  $\alpha$  originado em  $p_0$ .

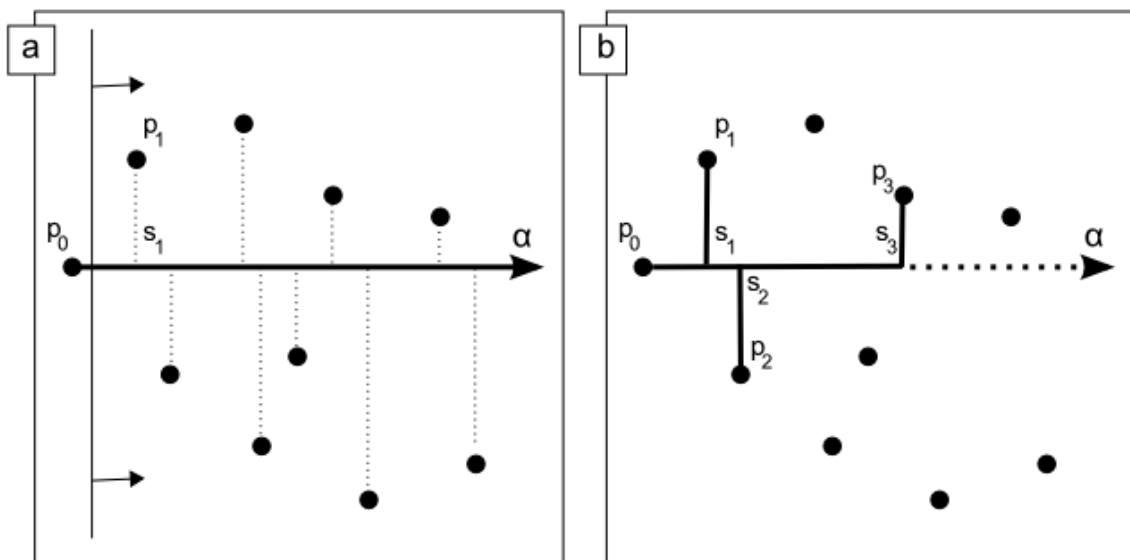


Figura 3.12: Aumentando uma RFST

O algoritmo pode ser mais bem visualizado pela passagem de uma linha perpendicular à  $l_0$  em ambos os sentidos de  $\alpha$  (Figura 3.12 (a)). Seja  $\{p_1, \dots, p_{i-1}\}$  a atual lista de terminais ligados à “perna longa” de  $p_0$  através dos pontos de Steiner  $\{s_1, \dots, s_{i-1}\}$ , tal que os segmentos  $\{p_1s_1, \dots, p_{i-1}s_{i-1}\}$  alternem sobre  $l_0$ . Este conjunto, denotado  $T_{i-1}$ , é uma RFST parcial (Figura 3.12 (b)). Note que, caso  $T_{i-1} \neq \{\}$ ,  $T_{i-1}$  é uma RFST na primeira topologia de Hwang e que  $p_{i-1}s_{i-1}$  é a sua “perna curta”.

Em seguida, procura-se um terminal  $p_i$  no lado oposto de  $l_0$  em relação à  $p_{i-1}$ . Se a nova árvore  $T_i = T_{i-1} \cup \{s_{i-1}s_i\} \cup \{p_is_i\}$  passar pelos testes do retângulo vazio, do diamante vazio e das arestas de gargalo da RMST, ela é adicionada ao conjunto  $F$  das RFSTs candidatas a participar da solução ótima.

### 3.4.2 Concatenação de RFSTs

Uma vez computado o conjunto  $F$  que contenha pelo menos uma Árvore de Steiner ótima, a fase de concatenação de RFSTs deve encontrar um conjunto  $F^* \subseteq F$  cujo comprimento seja mínimo e que interconecte todos os terminais da instância.

[Warne, 1998] demonstrou que a fase de concatenação de RFSTs reduz-se ao problema da árvore geradora mínima do hipergrafo  $H = (P, F)$  e formulou este problema como programação inteira. Warne solucionou este problema usando técnicas de *branch-and-cut*, possibilitando a computação da solução exata de instâncias compostas por até mil terminais.

### 3.4.3 Outras Utilidades do conjunto de RFSTs

O conjunto de RFSTs gerado pelo GeoSteiner pode ser aplicado, além de em algoritmos exatos para o RSMT, em heurísticas de RSMT que utilizem uma alguma abordagem semi-ótima de concatenação de RFSTs, ou em reduções de pontos.

O algoritmo  $O(n^2)$  de geração do conjunto  $F$  de RFSTs do GeoSteiner promove uma redução na grade de Hanan – que possui  $O(n^2)$  pontos – para aproximadamente  $O(3n)$  pontos. Este aspecto representa uma relação custo-benefício muito boa para heurísticas que tomam vantagem de reduções de pontos.

## CAPÍTULO 4 - MÉTODOS PROPOSTOS

Neste capítulo serão apresentadas e discutidas as novas abordagens para o RSMT propostas por este trabalho, focando seus aspectos teóricos, como a complexidade computacional; e práticos, como seus pseudocódigos e estratégias de implementação.

Primeiramente, serão definidas as notações referentes ao RSMT adotadas. Em seguida, serão detalhadas as implementações da Heurística1 e das metaheurísticas GRASP, SA e GA.

### 4.1 Notações

Para prover uma melhor descrição das abordagens construídas, os próximos capítulos obedecem às notações que seguem:

Os terminais que devem ser interligados pelas técnicas de construção de RSMT são representados pelo conjunto  $P$  de pontos no plano cartesiano  $Z^2$ , tal que  $P = \{p_1, \dots, p_n \mid p_i = (x_i, y_i)\}$ . O conjunto  $S$  de pontos de Steiner, de modo análogo a  $P$ , é denotado por  $S = \{s_1, \dots, s_m \mid s_i = (x_i, y_i)\}$ .

Define-se  $U = \{s_1, \dots, s_k \mid s_i = (x_i, y_i)\}$  como o conjunto de todos os pontos de Steiner possíveis para uma dada instância, tal que  $S \subseteq U$ . Inicialmente, o conjunto  $U$  é composto pelos pontos de Steiner da grade de Hanan, contudo, as técnicas de redução de pontos atuam sobre  $U$  excluindo alguns de seus pontos.

Cada técnica abordada neste capítulo retorna ao menos uma árvore de Steiner na seguinte forma:  $A = \{P \cup S, E\}$ , onde  $E = \{e_1, \dots, e_{n+m-1} \mid e_i = p_i p_j\}$  representa o conjunto de  $n + m - 1$  arestas de  $A$ .

Por fim, o custo/comprimento de uma árvore  $A$  é dado pela soma dos comprimentos de cada aresta que a compõe. Formalmente,  $|A| = \sum_{i=1}^{n+m-1} |e_i|$  para  $|e_i| = |x_i - x_j| + |y_i - y_j|$ .



## 4.2 Heurística1 – H1

A “Heurística 1”, por simplicidade H1, foi desenvolvida com o objetivo de prover solução inicial para qualquer técnica de busca-local – neste trabalho, para o *Simulated Annealing* – SA. Deve, para tal, apresentar soluções de qualidade razoável com um baixo custo computacional.

A H1 é composta por um esquema em quatro fases, como detalha o pseudocódigo da Figura 4.1.

<b>Entrada:</b>	Conjunto P de terminais
<b>Saída:</b>	Árvore Retilínea de Steiner
1.	Construção de uma árvore base $A \leftarrow \text{RMST}(P)$
2.	Localização de pontos de Steiner candidatos $S \leftarrow \text{LocalizaInter}(A)$
3.	Construção da Árvore Final $A \leftarrow \text{RMST}(P \cup S)$
4.	Remoção de pontos de Steiner desnecessários $A \leftarrow \text{Limpa}(A)$

Figura 4.1: Pseudocódigo da Heurística 1

Na primeira fase da H1 é construída uma RMST, denominada  $A_{base}$ , para o conjunto de terminais. Na segunda fase é realizado um processamento sobre as arestas de  $A_{base}$  no qual é selecionado um conjunto de pontos de Steiner que pode vir a diminuir o custo desta. A terceira fase constrói uma nova RMST, chamada  $A_{final}$ , que interconecte os conjuntos de terminais e de pontos de Steiner encontrados na fase dois. Por fim, na fase quatro, são removidos de  $A_{final}$  os pontos de Steiner que possuem grau um ou dois.

A fase dois – na qual são localizados os pontos de Steiner candidatos – configura-se a mais importante do procedimento H1, devido à simplicidade das fases restantes.

### 4.2.1 Procedimento de Geração de RMST

O procedimento de geração de RMST utilizado neste trabalho é uma implementação simples do algoritmo de Prim, como demonstra a Figura 4.2, com complexidade  $O(n^2)$ . Como observado na Tabela 2.1, por conta de grandes constantes ocultas, a versão em  $O(n \log n)$  do algoritmo de Prim apresenta tempo de execução inferior ao da versão em  $O(n^2)$  apenas para instâncias compostas por quinhentos ou mais pontos.

<b>Entrada:</b>	Conjunto de terminais: $P = \{ p_1, \dots, p_n \}$
<b>Saída:</b>	Lista de adjacências: L Vetor de graus: G
<b>1.</b>	<b>Inicialização</b> $P \leftarrow P - \{ p_1 \}$ <b>PARA</b> $i=1$ <b>ATÉ</b> $n$ $G[i] \leftarrow 0$ $L[i] \leftarrow \text{NULL}$
<b>2.</b>	<b>Laço Principal</b> <b>ENQUANTO</b> $P \neq \{ \}$ $\text{melhor\_ligacao} \leftarrow -1$ $\text{custo\_melhor\_ligacao} \leftarrow \infty$ <b>PARA</b> $i=1$ <b>ATÉ</b> $\# \text{Elementos}(P)$ <b>SE</b> $\text{custo}(p_i) < \text{custo\_melhor\_ligacao}$ <b>ENTÃO</b> $\text{melhor\_ligacao} \leftarrow p_i$ $\text{custo\_melhor\_ligacao} \leftarrow \text{custo}(p_i)$ $P \leftarrow P - \{ \text{melhor\_ligacao} \}$ $\text{Atualiza}(G[i], p_i)$ $\text{Atualiza}(L[i], p_i)$
<b>3.</b>	<b>RETORNE</b> os vetores G e L

Figura 4.2: Procedimento de Geração de RMST

O procedimento que atualiza o vetor G incrementa os graus de  $p_i$  e do ponto no qual  $p_i$  se ligará ( $\text{link}(p_i)$ ), enquanto o procedimento de atualização da lista de adjacências L adiciona o ponto  $p_i$  à lista de  $\text{link}(p_i)$  e vice-versa.

#### 4.2.2 Localização de Pontos de Steiner Candidatos

A compreensão da fase de localização de pontos de Steiner candidatos da H1 está ligada ao conceito de grade de Hanan  $k$ -restrita.

A grade de Hanan  $k$ -restrita de uma árvore retilínea  $A$  qualquer é obtida pela concatenação das grades de Hanan do conjunto de todas sub-árvores de  $A$  que contenham exatamente  $k$  terminais. A Figura 4.3 exibe as grades de Hanan 2, 3 e  $n$ -restritas, respectivamente (b), (c) e (d), da árvore apresentada na Figura 4.3 (a). Perceba que a grade  $n$ -restrita de qualquer árvore equivale exatamente à sua grade de Hanan.

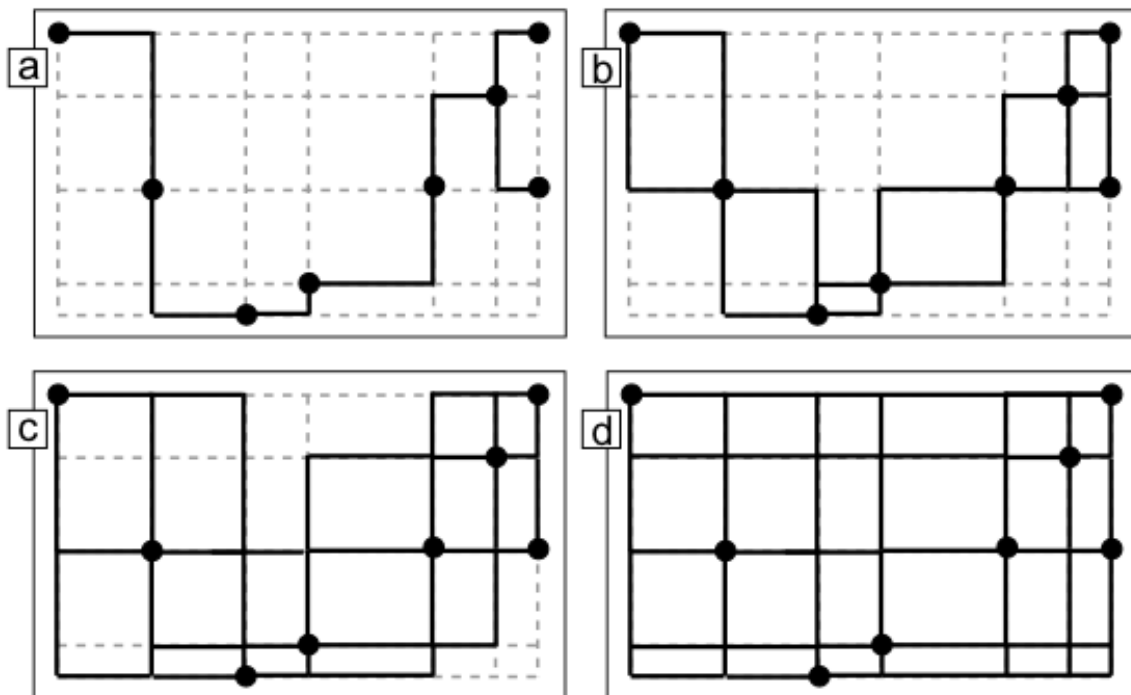


Figura 4.3: Grade de Hanan  $k$ -restrita

A partir de uma árvore retilínea  $A$  qualquer (no caso da H1 uma RMST), o procedimento de localização de pontos de Steiner seleciona todos os pontos da grade 2-restrita de  $A$  que são cruzados por duas ou mais arestas e adiciona-os ao conjunto de pontos de Steiner candidatos  $S$ . Visualmente, os pontos de Steiner candidatos selecionados são aqueles que intersectam dois retângulos da grade 2-restrita de  $A$ .

O algoritmo é simples: para cada sub-árvore de  $A$  composta por exatamente três terminais, localize o ponto  $p_{medio} = (x_{medio}, y_{medio})$  que satisfaça  $x_{\min} \leq x_{medio} \leq x_{\max}$  e  $y_{\min} \leq y_{medio} \leq y_{\max}$  e, caso  $p_{medio}$  não faça parte desta sub-árvore, adicione-o ao conjunto de pontos de Steiner candidatos.

### 4.2.3 Complexidade Computacional

As fases um e três da H1 apenas computam uma RMST, apresentando, portanto, como demonstrado em 4.2.1, complexidade computacional  $O(n^2)$ . Estas fases podem ser implementadas, contudo, em uma versão  $O(n \log n)$ . No entanto, esta é vantajosa apenas para grandes instâncias.

A fase de localização de pontos de Steiner candidatos possui complexidade  $O(n)$ . Como cada ponto possui no máximo grau quatro, a geração de todas as sub-árvores compostas por três pontos requer  $O(16n) \rightarrow O(n)$  e a localização do ponto candidato de cada sub-árvore se dá em  $O(1)$ .

A remoção de pontos de Steiner de graus um ou dois é dada por uma varredura simples sobre o vetor de graus dos pontos de Steiner, portanto  $O(n)$ .

Logo, a complexidade computacional da H1 é dominada pelo procedimento de geração de RMST, que neste trabalho é implementado em  $O(n^2)$ , mas pode ter a complexidade reduzida à  $O(n \log n)$ .

### 4.2.4 Exemplo Prático

A Figura 4.4 apresenta uma execução da H1 para uma instância composta por oito terminais: em (a) é exibida a grade 2-restrita criada a partir de uma RMST; em (b) são localizados os pontos de Steiner candidatos; (c) demonstra uma nova RMST, composta por terminais e pontos de Steiner candidatos; e (d) é a árvore final, da qual foram removidos os pontos de Steiner de graus um e dois.

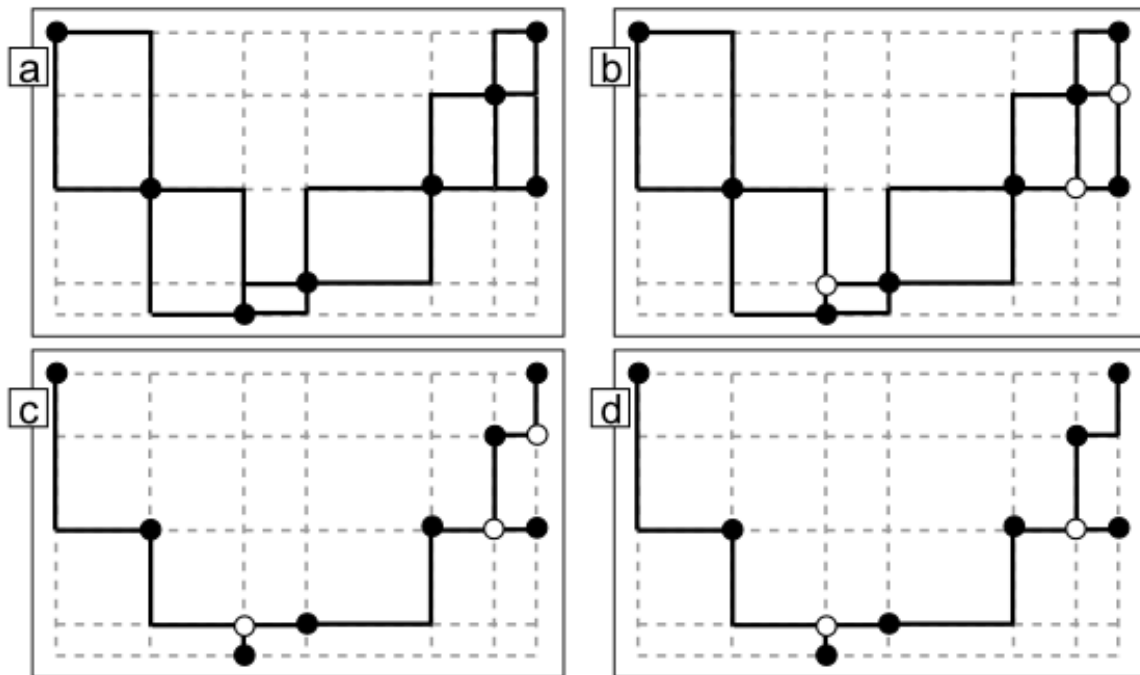


Figura 4.4: Exemplo de execução da H1

### 4.3 GRASP

Por ser baseada em melhorias sobre uma RMST, a H1 está sujeita ao limite de qualidade imposto pela proporção de Hwang (ver 2.5.5). Objetivando escapar desta barreira, foram implementadas algumas melhorias sobre a H1, culminando em um procedimento GRASP.

#### 4.3.1 H1 *Multi-Start*

Uma das grandes dificuldades enfrentadas por heurísticas de RSMT é a homogeneidade do espaço de soluções, o que implica na possível existência de diferentes RMSTs para um mesmo conjunto de terminais. A Figura 4.5 exibe três diferentes RMSTs para a mesma *netlist*.

Vale salientar que, dependendo da RMST construída na primeira fase do H1, o procedimento de localização de pontos de Steiner candidatos pode gerar conjuntos diferentes, o que acaba por afetar o comprimento da árvore final. Os conjuntos de pontos Steiner encontrados a partir das árvores (a), (b) e (c) da

Figura 4.5 são distintos e apenas o conjunto exibido em (b) é capaz de prover a solução ótima para esta *net*.

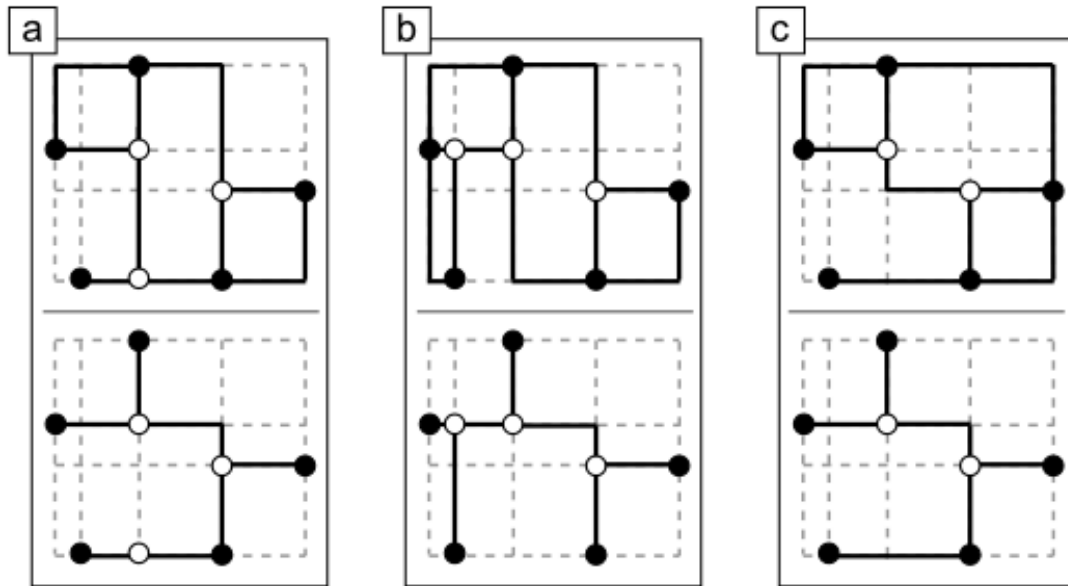


Figura 4.5: Diferentes RMSTs para um mesmo conjunto de terminais

A RMST que será construída pelo procedimento de geração de RMST utilizado neste trabalho pode variar de acordo com o ponto inicial escolhido e com o método de quebra de empates adotado. Por simplicidade, o ponto de partida escolhido é sempre o primeiro do conjunto de terminais, e, na quebra de empates, é privilegiado o ponto que primeiro apresentar o menor custo de interconexão.

Uma maneira de varrer um pouco mais o espaço de soluções, e, por conseguinte, diminuir o comprimento das árvores obtidas, é tentar gerar um número maior de RMSTs. Com este intuito, foi implementado uma versão *multi-start* da H1, no qual esta heurística é executada  $M$  vezes e, antes de cada execução, é feita uma alteração na ordem do conjunto de terminais para que sejam geradas diferentes RMSTs. A Figura 4.6 traz o pseudocódigo da H1 *multi-start*.

Uma vez que o rearranjo do conjunto de terminais é executado em  $O(n)$ , a complexidade computacional da H1 *multi-start* é  $O(M \times O(H1)) \rightarrow O(Mn^2)$ .

<b>Entrada:</b> Conjunto de terminais: $P = \{ p_1, \dots, p_n \}$ Número de execuções Multi-Start: $M$	
<b>Saída:</b> Lista de adjacências: $L$ Vetor de graus: $G$	
1.	Laço <i>Multi-Start</i> <b>PARA</b> $m=1$ <b>ATÉ</b> $M$
2.	Reordenamento do conjunto de Terminais <b>PARA</b> $i=1$ <b>ATÉ</b> $n/2$ $i \leftarrow \text{Aleatorio}() \mid 0 < i \leq n$ $j \leftarrow \text{Aleatorio}() \mid 0 < j \leq n$ TrocaPosição( $p_i, p_j$ )
3.	Execução da H1 $L[m], G[m] \leftarrow H1( P )$
4.	<b>RETORNE</b> a melhor árvore encontrada: $G[\text{melhor}]$ e $L[\text{melhor}]$

Figura 4.6: Procedimento H1 *Multi-Start*

Apesar da melhoria proporcionada pela versão *multi-start* da H1, esta ainda não consegue escapar da proporção de Hwang, pois, mesmo visitando uma fração maior do espaço de soluções, este procedimento ainda atua apenas melhorando uma RMST.

### 4.3.2 GRASP-H1

O comportamento guloso apresentado pelo procedimento de construção de RMST (ver 4.2.1), no qual cada ponto adicionado à árvore é selecionado por apresentar – dentre todos os pontos remanescentes – o menor custo, configura um ambiente propício a uma implementação da metaheurística GRASP.

A nova versão da H1 substitui o método de construção da árvore base, que na heurística original gerava uma RMST, por uma variante GRASP deste procedimento. Ao invés de escolher o ponto de menor custo e conectá-lo à árvore, é construído um conjunto  $E$  (chamado conjunto elite) composto pelos  $\alpha \times n$  (para  $0 \leq \alpha \leq 1$ ) pontos que possuem o menor custo de interconexão. Então, o ponto que será adicionado à árvore é escolhido equiprovavelmente dentre os que pertencem a  $E$ . A Figura 4.7 exibe o pseudocódigo deste procedimento.

<b>Entrada:</b>	Conjunto de terminais: $P = \{ p_1, \dots, p_n \}$ Parâmetro $\alpha$
<b>Saída:</b>	Lista de adjacências: $L$ Vetor de graus: $G$

1.	Inicialização idêntica ao procedimento de geração de RMST (...)
2.	Laço Principal <b>ENQUANTO</b> $P \neq \{ \}$
3.	Inicialização do conjunto Elite  $e \leftarrow \alpha \times n$ <b>PARA</b> $i=1$ <b>ATÉ</b> $e$ $\text{custo}(E[i]) \leftarrow \infty$  <b>PARA</b> $i=1$ <b>ATÉ</b> $\#Elementos(P)$ <b>SE</b> $\text{custo}(p_i) < \text{custo\_pior\_elemento}(E)$ <b>ENTÃO</b> $E \leftarrow E - \{ \text{pior\_elemento}(E) \}$ $E \leftarrow E \cup \{ p_i \}$
4.	Escolha do ponto a ser conectado  $i \leftarrow \text{Aleatorio}() \mid 0 < i \leq e$ $P \leftarrow P - \{ p_i \}$ $\text{Atualiza}(G[i], p_i)$ $\text{Atualiza}(L[i], p_i)$
3.	<b>RETORNE</b> os vetores $G$ e $L$

Figura 4.7: Procedimento GRASP de geração de árvores retilíneas

O parâmetro  $\alpha$  controla o grau de aleatoriedade do procedimento GRASP de geração de árvores retilíneas. Para  $\alpha = 0$ , o conjunto  $E$  possui apenas um elemento, ocasionando um comportamento puramente guloso, exatamente igual ao do procedimento de geração de RMST. Em contrapartida, quanto maior o valor de  $\alpha$ , mais randômica será a árvore resultante.

Observe que, apesar da árvore gerada pelo procedimento GRASP raramente apresentar custo equivalente – ou até mesmo equiparável – ao da RMST, a geração da RMST de  $P \cup S$  (fase três da H1) e a remoção dos pontos de graus inferiores a três atuam como busca-local, garantido a qualidade das soluções obtidas.



A técnica GRASP-H1 não é limitada pela proporção de Hwang, pois constrói soluções a partir de diferentes árvores retilíneas de Steiner e não necessariamente a partir de uma RMST.

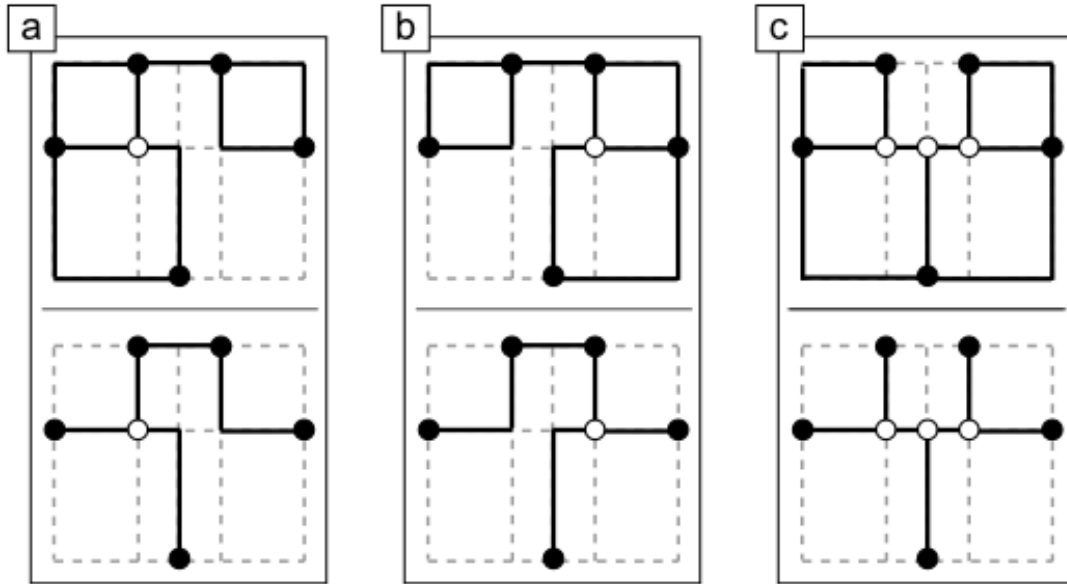


Figura 4.8: Comparação entre os procedimentos H1 *Multi-Start* e GRASP-H1

A Figura 4.8 exibe uma *net* para qual não é possível chegar à RSMT a partir de melhorias sobre sua RMST. (a) e (b) apresentam as duas RMSTs existentes para esta instância e a árvore computada pela H1; e (c) exibe a solução ótima obtida pelo procedimento GRASP-H1.

### 4.3.3 Complexidade Computacional

A variante GRASP da H1 apresenta complexidade computacional um pouco superior à da H1 original, pois cada iteração do procedimento de geração de árvores retilíneas requer a manutenção do ordenamento do conjunto elite para que seu pior elemento seja substituído. Visto que o número de elementos do conjunto  $E$  é calculado através da fórmula  $e = \alpha \times n$ , a complexidade do GRASP-H1 pode ser dada em função do parâmetro  $\alpha$ .

A inserção de um novo elemento em um conjunto composto por  $n$  elementos ordenados mantendo sua classificação pode ser implementada em  $O(\log n)$  com auxílio de um *heap* (de modo similar ao método *insert-sort*).

Portanto, a complexidade computacional apresentada pela metaheurística GRASP-H1 é  $O(Mn^2 \log(\alpha \times n))$ , onde o parâmetro  $M$  representa o número de iterações *multi-start* e  $\alpha$  indica o grau de aleatoriedade do procedimento de construção de árvores retilíneas. Perceba que para os valores  $M = 1$  e  $\alpha = 0$  o GRASP-H1 reduz-se à H1 original.

#### 4.4 Simulated Annealing

Nesta seção são apresentados cinco movimentos – funções que geram uma solução vizinha  $s'$  a partir de uma solução  $s$  (árvore de entrada) qualquer – para o problema da árvore retilínea mínima de Steiner: *Edge\_Swap*, *Add\_Steiner\_3*, *Del\_Steiner*, *Graceful\_Del\_Steiner* e *Pierce\_Steiner*. Este conjunto de movimentos fornece subsídios suficientes à implementação de qualquer metaheurística baseada no conceito de vizinhança.

Para validar a eficácia dos movimentos propostos, estes foram encapsulados em uma metaheurística híbrida SA / *multi-start*.

<b>Entrada:</b>	Conjunto de terminais: $P = \{ p_1, \dots, p_n \}$ Número de execuções Multi-Start: $M$ Constante de Arrefecimento: $\alpha$ Temperatura inicial: $T$ Número de iterações por Temperatura: $SA\_MAX$
<b>Saída:</b>	Árvore de Steiner $Z = (P \cup S, E)$
<b>1.</b>	<b>PARA</b> $m = 0$ <b>ATÉ</b> $M$ $A \leftarrow \text{GRASP-H1}(\alpha=0.04, M=1)$ $Z \leftarrow \text{SA}(A, \alpha, T, SA\_MAX)$ <b>SE</b> $ Z  <  Z^* $ <b>ENTÃO</b> $Z^* \leftarrow Z_{\text{atual}}$
<b>2.</b>	<b>RETORNE</b> $Z^*$

Figura 4.9: Metaheurística híbrida SA / *Multi-Start*

A Figura 4.9 exibe o pseudocódigo da SA / *multi-start* (doravante, por conveniência, identificada apenas por SA) no qual é detalhado o processo de hibridização escolhido: a cada iteração do *multi-start* é construída, utilizando a metaheurística GRASP-H1 com parâmetros  $M_{GRASP} = 1$  e  $\alpha_{GRASP} = 0,04$ , uma solução inicial que é refinada por um procedimento *simulated annealing*.

A Figura 4.10 detalha a implementação do procedimento de refinamento SA, onde a função  $Boltzmann(\Delta, T)$  calcula  $e^{-\Delta/T}$  (ver 2.3).

<b>Entrada:</b>	Solução Inicial: $A = (P \cup S, E)$ Constante de Arrefecimento: $\alpha$ Temperatura inicial: $T$ Número de Iterações por Temperatura: $SA\_MAX$
<b>Saída:</b>	Árvore de Steiner $Z = (P \cup S, E)$

1.	// Inicialização $Z^* \leftarrow Z_{atual} \leftarrow A$ // Melhor solução e solução corrente $prepare\_HGP(A)$
2.	// Laço Principal <b>ENQUANTO</b> $T > 0.001$ $T \leftarrow T \times \alpha$ // A partir da segunda iteração <b>PARA</b> $Iter = 0$ <b>ATÉ</b> $SA\_MAX$ $V \leftarrow$ Movimento selecionado aleatoriamente $Z_{tmp} \leftarrow V(Z_{atual})$ $\Delta \leftarrow  Z_{tmp}  -  Z_{atual} $
3.	// Vizinho gerado <b>MELHORA</b> a solução corrente <b>SE</b> $\Delta \leq 0$ <b>ENTAO</b> $Z_{atual} \leftarrow Z_{tmp}$ <b>SE</b> $ Z_{atual}  <  Z^* $ <b>ENTAO</b> $Z^* \leftarrow Z_{atual}$
4.	// Vizinho gerado <b>PIORA</b> a solução corrente $x \leftarrow Aleatorio() \mid 0 < x < 1$ <b>SE</b> $(\Delta > 0)$ <b>E</b> $(x < Boltzmann(\Delta, T))$ <b>ENTAO</b> $Z_{atual} \leftarrow Z_{tmp}$
5.	<b>RETORNE</b> $Z^*$

Figura 4.10: Pseudocódigo do *Simulated Annealing*

O GRASP-H1( $M_{GRASP} = 1$ ,  $\alpha_{GRASP} = 0,04$ ) foi escolhido como técnica de geração de soluções iniciais por possuir tempo de execução reduzido e propiciar várias topologias de árvores diferentes.

Os movimentos *Edge\_Swap*, *Add\_Steiner\_3* e *Pierce\_Steiner* fazem uso da computação da aresta de gargalo entre dois pontos. Para suprir esta demanda de modo menos oneroso computacionalmente foi desenvolvido um esquema com três funções baseado no procedimento HGP (descrito em 3.2):

1. Função *prepare\_HGP(A)*: Implementação exata do procedimento HGP, no qual primeiramente é classificado o vetor de arestas ( $O(n \log n)$ ) e depois são computados os vetores *edge* e *parent* ( $O(n)$ ).
2. Função *mantenha\_HGP(A)*: Implementação do procedimento HGP para um conjunto de arestas semi-classificado, resultante de  $K$  alterações em arestas de um conjunto, tais como deleção, substituição ou adição de uma nova aresta. Este conjunto é, então, reordenado ( $O(Kn) \rightarrow O(n)$ , pois  $K$  é pequeno e constante) e são computados os vetores *edge* e *parent* ( $O(n)$ ).
3. Função *maior\_aresta(A, p<sub>i</sub>, p<sub>j</sub>)*: Utiliza os vetores *edge* e *parent* pré-computados para localizar, em  $O(\log n)$ , a maior aresta do caminho entre dois pontos de uma árvore  $A$ , exatamente como demonstra o pseudocódigo da Figura 3.7.

Utilizando este esquema, o SA precisa executar apenas uma vez a função *prepare\_HGP*, que possui complexidade  $O(n \log n)$ , tornando possível que os movimentos encontrem qualquer aresta de gargalo em  $O(\log n)$ . Por sua vez, caso o movimento altere alguma aresta, deve manter os vetores *edge* e *parent* através da função *mantenha\_HGP*, em  $O(n)$ .

Os tópicos que seguem detalham cada movimento desenvolvido, discutindo sua implementação e seu papel nesta estrutura de vizinhança.

#### 4.4.1 Movimento *Edge\_Swap*

O movimento de substituição de aresta (batizado de *Edge\_Swap*) consiste em sortear dois pontos  $p_i, p_j \in P \cup S$ , remover a maior aresta do caminho entre eles e adicionar a aresta  $p_i p_j$  ao conjunto de arestas de  $A$ . A Figura 4.11 exhibe este procedimento em detalhes e a Figura 4.12 demonstra um exemplo de execução deste movimento.

O *Edge\_Swap* causa, na maioria das vezes, um aumento no comprimento da árvore corrente, contudo, a aceitação desta nova árvore pelo SA está condicionada ao valor atual da temperatura do sistema. As árvores que resultam deste movimento são importantes, pois permitem que os movimentos *Add\_Steiner\_3* e *Pierce\_Steiner* visitem novas regiões do espaço de soluções, encontrando diferentes pontos de Steiner.

Uma alternativa para minimizar o comportamento aleatório deste movimento é, ao invés de sortear um par de pontos, sortear um número pré-definido de pares e executar o restante do procedimento para o par que produzir a aresta de menor custo.

<b>Entrada:</b>	Árvore: $A = (P \cup S, E)$
<b>Saída:</b>	Árvore Vizinha de $A = (P \cup S, E)$
1.	$p_i \leftarrow \text{ponto\_Aleatorio}(P \cup S)$ $p_j \leftarrow \text{ponto\_Aleatorio}(P \cup S)$
2.	// Remoção da aresta de gargalo e adição da nova aresta $g \leftarrow \text{maior\_Aresta}(A, p_i, p_j)$ $a \leftarrow \text{nova\_Aresta}(p_i, p_j)$ $\text{substitui\_Aresta}(E, g, a)$
3.	// Manutenção dos vetores <i>edge</i> e <i>parent</i> $\text{mantenha\_HGP}(A)$
4.	RETORNE $A$

Figura 4.11: Pseudocódigo do movimento *Edge\_swap*

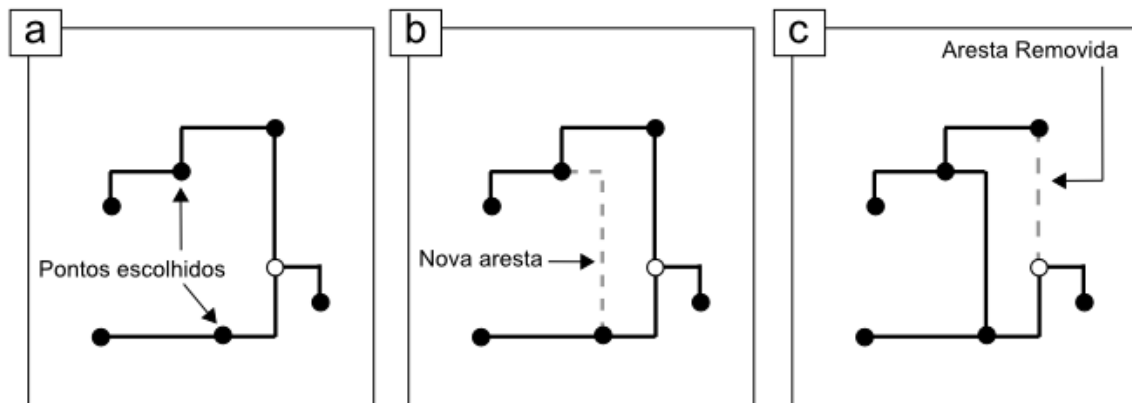


Figura 4.12: Exemplo de execução do movimento *Edge-Swap*

#### 4.4.2 Movimento *Add\_Steiner\_3*

O movimento de adição de um ponto de Steiner em uma sub-árvore de comprimento três (batizado de *Add\_Steiner\_3*) consiste em sortear uma tripla qualquer da árvore  $A$  e adicionar a esta um ponto de Steiner que diminua seu comprimento. O pseudocódigo do movimento *Add\_Steiner\_3* é exibido na Figura 4.13 em forma de pseudocódigo e graficamente na Figura 4.14.

<b>Entrada:</b>	Árvore: $A = (P \cup S, E)$
<b>Saída:</b>	Árvore Vizinha de $A = (P \cup S, E)$
1.	$A_{tmp} \leftarrow \text{sub-árvore\_Aleatoria}(A) \mid A_{tmp} \text{ possua exatamente 3 pontos}$ $pm \leftarrow \text{ponto\_medio}(A_{tmp}) \mid pm = (x_{medio}(A_{tmp}), y_{medio}(A_{tmp}))$
2.	SE $pm$ não PERTENCE a $A_{tmp}$ ENTAO // Criação de um novo ponto de Steiner $S \leftarrow S \cup \{pm\}$
3.	// Remoção das Arestas de $A_{tmp}$ $E \leftarrow E - \text{conjunto das Arestas}(A_{tmp})$
4.	// Ligação dos pontos de $A_{tmp}$ com o novo ponto de Steiner <b>PARA</b> $i=0$ <b>ATÉ</b> $\#Pontos$ de $A_{tmp}$ $E \leftarrow E \cup \text{nova\_Aresta}(p_i, pm)$
5.	// Manutenção dos vetores <i>edge</i> e <i>parent</i> mantenha_HGP( $A$ )
6.	<b>RETORNE</b> $A$

Figura 4.13: Pseudocódigo do movimento *Add\_Steiner\_3*

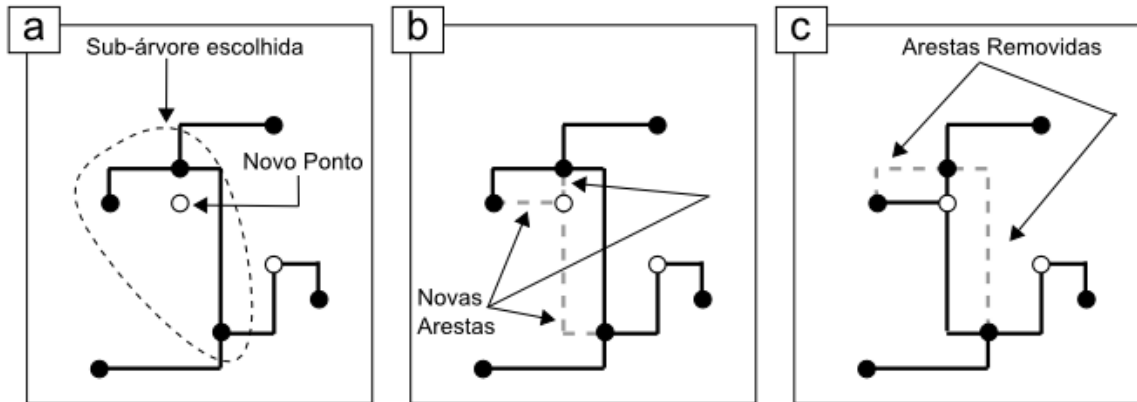


Figura 4.14: Exemplo de execução do movimento *Add\_Steiner\_3*

O processo de encontrar o ponto de Steiner que melhore uma tripla é simples: uma vez escolhida a sub-árvore  $A_3 = (\{p_1, p_2, p_3\}, \{e_1, e_2\})$ , composta por exatamente três terminais, localize o ponto  $p_{medio} = (x_{medio}, y_{medio})$  que satisfaça  $x_{min} \leq x_{medio} \leq x_{max}$  e  $y_{min} \leq y_{medio} \leq y_{max}$  e, caso  $p_{medio} \neq p_1, p_2, p_3$ , interconecte-o a cada ponto de  $A_3$  e remova as arestas  $e_1$  e  $e_2$  que compunham esta árvore.

Este movimento pode ser implementado de modo a realizar apenas três alterações sobre conjunto ordenado de arestas (duas substituições e uma adição). Sendo assim, a manutenção das estruturas de dados do HGP através da função *mantenha\_HGP* se dá em  $O(Kn) \rightarrow O(3n) \rightarrow O(n)$ .

O *Add\_Steiner\_3* exerce um papel fundamental dentre o conjunto de movimentos proposto neste trabalho, corrigindo (na maioria dos casos) as piores que foram aceitas pelo SA quando em valores mais altos de  $T$ . Perceba que este movimento sempre provê melhorias sobre a solução atual, ou, no pior dos casos, não faz nenhuma alteração sobre esta. Visitar toda a vizinhança gerada através do *Add\_Steiner\_3* comporta-se, pois, como uma boa heurística de refinamento.

#### 4.4.3 Movimento *Del\_Steiner*

O movimento de remoção de pontos de Steiner (batizado *Del\_Steiner*) consiste em selecionar aleatoriamente um ponto  $s \in S$  e removê-lo da árvore  $A$ , fazendo que todos os pontos diretamente conectados a  $s$  sejam ligados a um

novo ponto, dito *pivô*. A escolha do ponto *pivô* é facultativa entre aqueles que se interligavam a *s*. A Figura 4.15 detalha o *Del\_Steiner* enquanto a Figura 4.16 demonstra a deleção de um ponto de Steiner de grau três.

O número de alterações sobre o conjunto ordenado de arestas realizado pelo *Del\_Steiner* varia de acordo com o grau do ponto *s*, pois são executadas exatamente  $\text{grau}(s) - 1$  substituições e uma deleção. A manutenção das estruturas de dados do HGP se dá em  $O(Kn) \rightarrow O(\text{grau}(s) \times n)$ . Como todo ponto de Steiner possui no máximo grau quatro (ver 2.5.2), a complexidade computacional da função *mantenha\_HGP* executada por este movimento mantém-se em  $O(n)$ .

Este movimento produz mudanças muito abruptas sobre a topologia da árvore de entrada, propiciando, sempre que exclui pontos de Steiner de grau três ou quatro, uma significativa piora no custo da solução atual. A sua inclusão no conjunto de movimentos selecionáveis é importante, no entanto, pois viabiliza que o SA escape de difíceis ótimos locais.

<b>Entrada:</b>	Árvore: $A = (P \cup S, E)$
<b>Saída:</b>	Árvore Vizinha de $A = (P \cup S, E)$
1.	$ps \leftarrow \text{ponto\_Aleatorio}() \mid ps \text{ pertence a } S$
2.	// Localização dos pontos ligados a ps <b>PARA</b> $i=0$ <b>ATÉ</b> #Elementos de E <b>SE</b> aresta $E[i]$ <b>contem</b> ps <b>ENTÃO</b> $Etmp \leftarrow Etmp \cup \{E[i]\}$
3.	// Remoção da Primeira aresta de Etmp $pivo \leftarrow \text{ponto com o qual ps se interconecta na aresta } Etmp[1]$ $E \leftarrow E - \{Etmp[1]\}$ $Etmp \leftarrow Etmp - \{Etmp[1]\}$
4.	// Ligação dos pontos restantes ao pivo <b>ENQUANTO</b> $Etmp \neq \{\}$ faça $Etmp[i]$ ligar-se ao pivo $Etmp \leftarrow Etmp - \{Etmp[1]\}$ <i>mantenha_HGP</i> (A)
4.	<b>RETORNE</b> A

Figura 4.15: Pseudocódigo do movimento *Del\_Steiner*



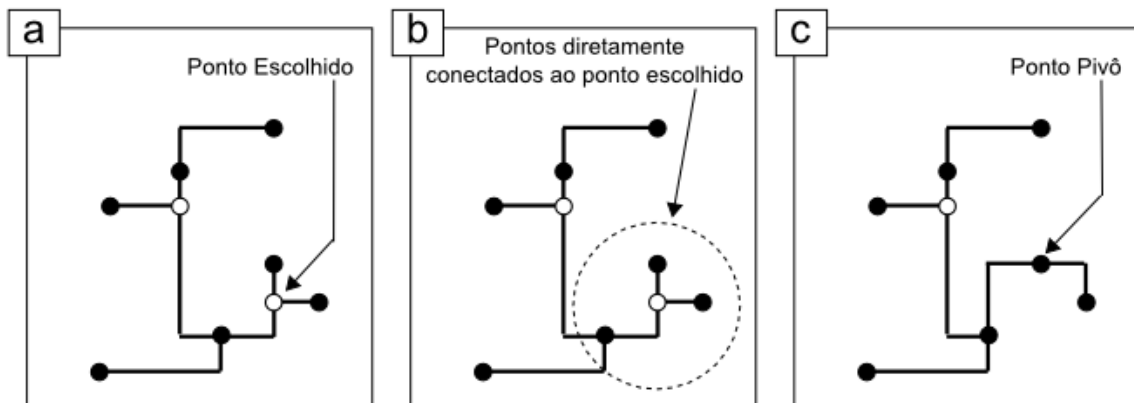


Figura 4.16: Exemplo de execução do movimento *Del\_Steiner*

#### 4.4.4 Movimento *Graceful\_Del\_Steiner*

O movimento de deleção educada de pontos de Steiner, batizado de *Graceful\_Del\_Steiner*, é uma versão do movimento *Del\_Steiner* na qual apenas é efetuada a remoção caso o ponto sorteado possua grau menor ou igual a dois.

O papel do *Graceful\_Del\_Steiner* é reduzir o número de pontos de Steiner que não influenciam o comprimento da árvore, aumentando, assim, a eficiência dos outros movimentos.

Por não afetar negativamente o custo da árvore de entrada, e retornar em  $O(1)$  sempre que o ponto sorteado tenha grau maior que dois, o *Graceful\_Del\_Steiner* deve ter probabilidade maior de ser escolhido para geração de vizinhos.

#### 4.4.5 Movimento *Pierce\_Steiner*

O movimento de inserção de pontos de Steiner por perfuração de arestas (batizado *Pierce-Steiner*) consiste em: a partir de um ponto  $p$  qualquer, sorteado dentre os pertencentes ao conjunto  $P \cup S$ , são traçadas duas retas paralelas aos eixos  $x$  e  $y$  que, eventualmente, cruzam (perfuram) outras arestas da árvore; por sobre a aresta perfurada mais próxima, exatamente no ponto de perfuração, é adicionado um ponto de Steiner que é ligado a  $p$ ; por fim, a maior aresta do ciclo formado é removida.

A Figura 4.17 ilustra uma execução do *Pierce-Steiner*. (a) demonstra o ponto inicial escolhido; (b) exibe as retas perpendiculares traçadas a partir do ponto inicial e as arestas que são perfuradas por estas e (c) exibe o novo ponto de Steiner criado e a aresta de gargalo do ciclo, que é removida.

O *Pierce-Steiner* foi escrito de modo a realizar as mesmas três alterações sobre conjunto ordenado de arestas do *Add\_Steiner\_3*. Por conseguinte, a manutenção das estruturas de dados do HGP possui o mesmo comportamento  $O(Kn) \rightarrow O(3n) \rightarrow O(n)$ .

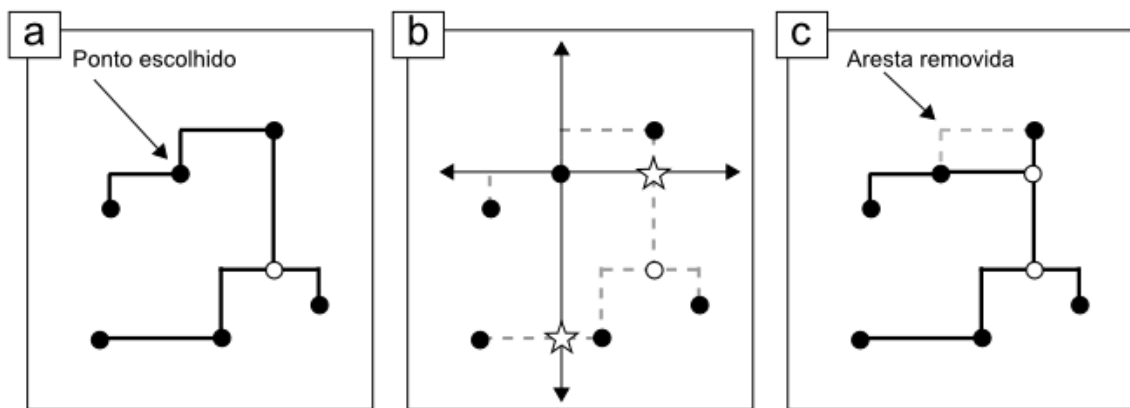


Figura 4.17: Exemplo de execução do movimento *Pierce-Steiner*

Uma melhoria implementada sobre o *Pierce\_Steiner* foi considerar, ao invés do conjunto de arestas da árvore de entrada, a grade 2-restrita desta durante o processo de perfurações, adicionando um novo ponto de Steiner sobre o ponto mais próximo de  $p$  dentre todas as grades (arestas) perfuradas.

O *Pierce-Steiner* constitui-se um movimento que, por si só, poderia ser usado como uma técnica de refinamento para o RSMT, devido à sua baixa complexidade computacional  $O(n)$  e capacidade de localizar bons pontos de Steiner distantes do ponto inicial, levando em conta toda a topologia da árvore de entrada.

#### 4.4.6 Complexidade Computacional

A complexidade computacional de procedimentos *simulated annealing* é dada tomando por base a complexidade apresentada pelo conjunto de movimentos adotado e pelos valores escolhidos para os parâmetros  $T_0$ ,  $\alpha$  e  $SA_{\max}$ . Como o SA implementado neste trabalho é envolvido por um laço *multi-start*, faz-se necessário considerar, adicionalmente, o parâmetro  $M$  e a complexidade da heurística utilizada para gerar soluções iniciais.

A técnica escolhida para gerar soluções iniciais foi o GRASP-H1 com parâmetros  $M_{GRASP} = 1$  e  $\alpha_{GRASP} = 0,04$ . Partindo do estudo feito em 4.3.3, chegamos à complexidade  $O(Mn^2 \log(\alpha \times n)) \rightarrow (n^2 \log(0,04 \times n)) \rightarrow (n^2)$ .

Por conta da nova técnica de manutenção dinâmica das estruturas de dados do HGP, todos os movimentos executados pelo SA possuem complexidade  $O(n)$ .

Como o laço *multi-start* engloba apenas a geração de soluções iniciais e o procedimento SA de refinamento, a ordem de complexidade da metaheurística híbrida SA / *multi-start* é dada por  $O(M \times O(SA))$ , pois a complexidade do procedimento de refinamento SA domina a complexidade do GRASP-H1 ( $M_{GRASP} = 1$ ,  $\alpha_{GRASP} = 0,04$ ).

#### 4.5 Algoritmo Genético

Tomando vantagem da redução de pontos de Steiner propiciada pelo procedimento de geração RFSTs do GeoSteiner (descrito em 3.4.1), que diminui de  $O(n^2)$  para aproximadamente  $O(4n) \rightarrow O(n)$  o número de pontos de Steiner do conjunto  $U$ , foi implementado um algoritmo genético para o RSMTTP, como detalhado na Figura 4.18.

<b>Entrada:</b>	Conjunto de terminais: P Número de Eras : Max_age Tamanho da população: Pop_size
<b>Saída:</b>	Árvore de Steiner A = ( P U S, E )
1.	// Redução de Pontos $U \leftarrow \text{GeoSteiner\_RFST}(P)$
2.	// População Inicial $\text{Pop}(0) \leftarrow \text{GeraPopulaçãoInicial}(U)$
3.	// Laço Principal <b>PARA</b> t=1 <b>ATÉ</b> Max_age $\text{Pop}(t) \leftarrow \text{GeraPopulação}(\text{Pop}(t-1))$
4.	// Laço de avaliação da população <b>PARA</b> i=1 <b>ATÉ</b> Pop_Size $\text{Avalie\_indivíduo}(\text{Pop}(t), i)$
5.	// Define a População sobrevivente $\text{Pop}(t) \leftarrow \text{Sobreviventes}(\text{Pop}(t))$
6.	<b>RETORNE</b> Melhor árvore A encontrada

Figura 4.18: Implementação do Algoritmo Genético

As seções que seguem são redigidas de modo a detalhar os métodos utilizados em cada etapa deste algoritmo genético; partindo da representação de soluções em forma de cromossomo, passando pelos operadores de recombinação e mutação escolhidos e finalizando com o procedimento de avaliação de populações baseado na manutenção dinâmica de RMSTs.

#### 4.5.1 Representação Genética das Soluções

Partindo da definição de  $U = \{s_1, \dots, s_m \mid s_i = (x_i, y_i)\}$  como o conjunto de todos os pontos de Steiner possíveis para uma dada instância e da representação de uma solução qualquer na forma  $A = \{P \cup S, E\}$  tal que  $S \subseteq U$ , é possível utilizar uma representação binária – na qual cada gene apenas pode apresentar valor ‘0’ ou ‘1’ – para codificar uma solução na forma de um vetor (dito cromossomo) com  $m$  posições:  $C = (c_1, c_2, \dots, c_m)$ , onde cada componente  $c_i$  indica se o ponto  $s_i \in U$  faz ( $c_i = 1$ ) ou não ( $c_i = 0$ ) parte da árvore em questão.

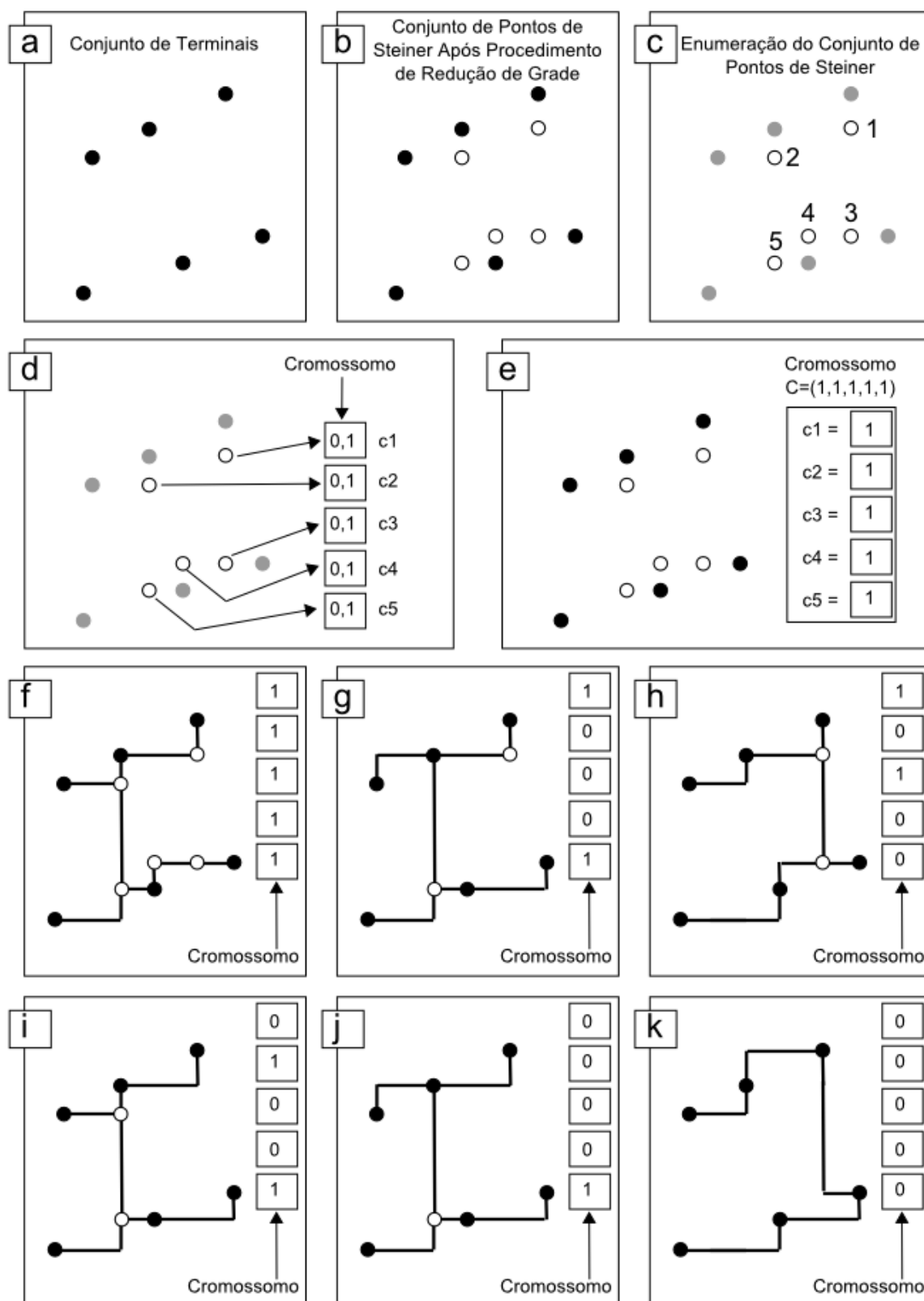


Figura 4.19: Representação genética de árvores de Steiner

A Figura 4.19 explica passo a passo o sistema de representação de árvores de Steiner por meio de cromossomos binários: (a) exibe o conjunto de terminais para o qual se deve construir uma RSMT; em (b), após a execução do procedimento de redução de pontos do GeoSteiner, resta um conjunto  $U$  composto por apenas cinco pontos; (c) enumera os pontos de  $U$  para que, como exibido em (d), cada ponto deste conjunto seja associado de maneira exclusiva ao mesmo gene em todos os cromossomos; em (e) é mostrado um cromossomo cujos todos os genes possuem valor '1' e, por conseguinte, todo o conjunto  $U$  é adicionado a esta árvore construída através da  $RMST(P \cup S)$ , como pode se observar em (f); (g), (h), (i), (j) e (k) exibem outros exemplos cromossomos e as respectivas árvores de Steiner por estes representadas.

Os algoritmos genéticos, diferentemente de todas outras abordagens discutidas neste trabalho, efetuam busca populacional, trabalhando concomitantemente com um conjunto de soluções, denominado população. Uma população nada mais é que um conjunto de indivíduos (cromossomos), descrito na forma  $Pop = \{C_1, \dots, C_{popsize}\}$ . A Figura 4.20 exibe a população formada pelas árvores (f), (g), (h), (i), (j) e (k) da Figura 4.19.

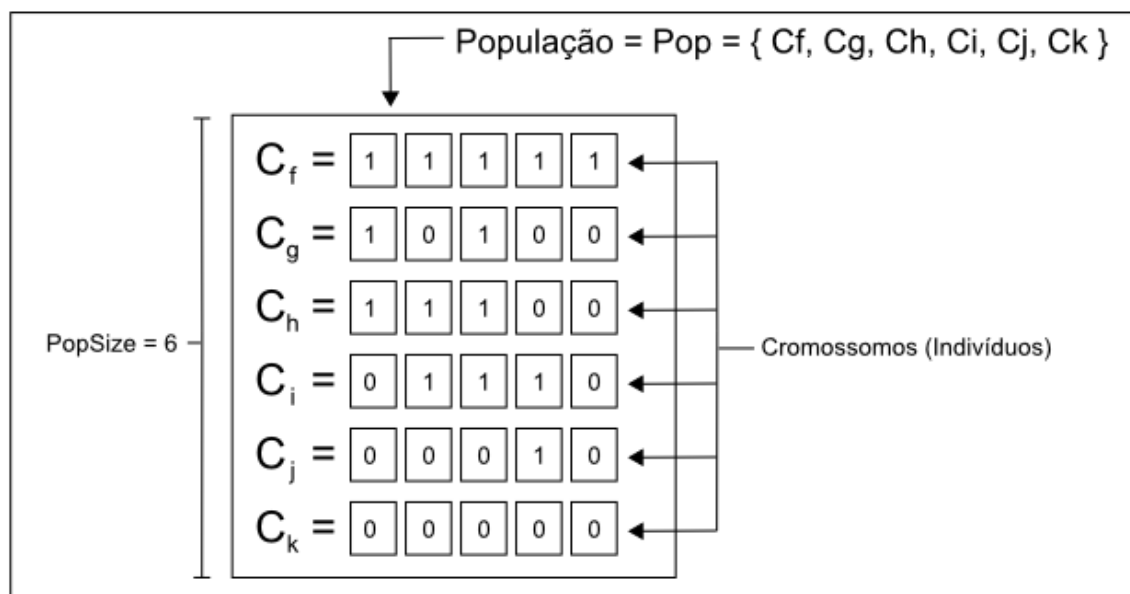


Figura 4.20: Representação de uma população de árvores de Steiner

#### 4.5.2 Função de Aptidão de um Indivíduo

Nesta implementação de GA, o cálculo da aptidão de cada indivíduo herda a função objetivo do RSMTP: o custo da RMST do conjunto  $P \cup S$ . Como cada cromossomo  $C$  representa um conjunto  $S \subseteq U$ , o cálculo de quão apto é o indivíduo representado por  $C$  reduz-se a  $|RMST(S \cup P)|$ , sendo  $S$  o conjunto dos pontos de  $U$  cujo índice em  $C$  é '1'.

#### 4.5.3 Procedimentos de Geração Populacional

Os GA necessitam dois procedimentos de geração de população: o algoritmo de geração de população inicial e a etapa de reprodução, que produz uma nova população a partir da população anterior.

A geração da população inicial foi implementada de forma puramente randômica, de modo que cada gene possui a mesma probabilidade de apresentar valor '1' ou '0'. Como o valor escolhido para o tamanho da população geralmente é maior que  $m$ , é difícil que algum gene não apareça em suas duas formas na população inicial.

Foi experimentado, também, um processo de *seeding* na população inicial com uma solução proveniente do GRASP-H1. Porém, não se detectou nenhuma alteração no comportamento do GA.

A etapa de reprodução deste GA foi implementada com base no esquema clássico de torneio binário e nos procedimentos de recombinação par/ímpar (*Odd\_Even\_Crossover*) e de mutação denominado hy-M. As seções que seguem detalham estes procedimentos.

##### 4.5.3.1 Procedimento de Recombinação *Odd\_Even\_Crossover*

O procedimento de recombinação par/ímpar (*Odd\_Even\_Crossover*), em síntese, consiste na geração duas proles a partir de dois pais. Em detalhes, primeiramente são selecionados dois cromossomos pais,  $C_{p_1}$  e  $C_{p_2}$ , através de um torneio binário clássico (já discutido em 2.4.3), parametrizado com  $k = 2$ ; uma

vez de posse de  $Cp_1$  e  $Cp_2$ , são gerados dois cromossomos filhos  $Cf_1$  e  $Cf_2$  através do algoritmo descrito em Figura 4.21. Cada filho passa, então, pelo procedimento de mutação hy-M.

<b>Entrada:</b>	Par de Cromossomos Pais: $Cp_1, Cp_2 = \{c_1, \dots, c_m\}$
<b>Saída:</b>	Par de Cromossomos Filhos: $Cf_1, Cf_2 = \{c_1, \dots, c_m\}$
1.	// Inicialização $aux \leftarrow PAR;$
2.	// Laço Principal <b>PARA</b> $i=1$ <b>ATÉ</b> $m$
3.	<b>SE</b> ( $Cp_1[i] \neq Cp_2[i]$ ) <b>E</b> ( $aux = PAR$ ) <b>ENTÃO</b> $Cf_1[i] \leftarrow Cp_2[i]$ $Cf_2[i] \leftarrow Cp_1[i]$ $aux \leftarrow IMPAR$
4.	<b>SENÃO SE</b> ( $Cp_1[i] \neq Cp_2[i]$ ) <b>E</b> ( $aux = IMPAR$ ) <b>ENTÃO</b> $Cf_1[i] \leftarrow Cp_1[i]$ $Cf_2[i] \leftarrow Cp_2[i]$ $aux \leftarrow PAR$
5.	<b>SENÃO</b> $Cf_1[i] \leftarrow Cp_1[i]$ $Cf_2[i] \leftarrow Cp_2[i]$
6.	// Mutação $Cf_1 \leftarrow hy-M(Cf_1)$ $Cf_2 \leftarrow hy-M(Cf_2)$

Figura 4.21: Procedimento de Recombinação *Odd\_Even\_Crossover*

#### 4.5.3.2 Procedimento de Mutação hy-M

Este procedimento de mutação foi batizado de hy-M (*hyper mutation*) por conta da alta probabilidade de serem gerados filhos mutantes, o que contrasta com a maioria dos GA implementados. O procedimento é simples: cada indivíduo que passa por hy-M possui 20% de chance de ter um gene, escolhido aleatoriamente, invertido, i.e. ter o valor alterado de '0' para '1' ou de '1' para '0'.

Outras estratégias, mais conservadoras foram experimentadas, contudo, as simulações que utilizaram o procedimento hy-M encontraram, em média, soluções de maior qualidade.



#### 4.5.3.3 Ilustração da Etapa de Reprodução do GA

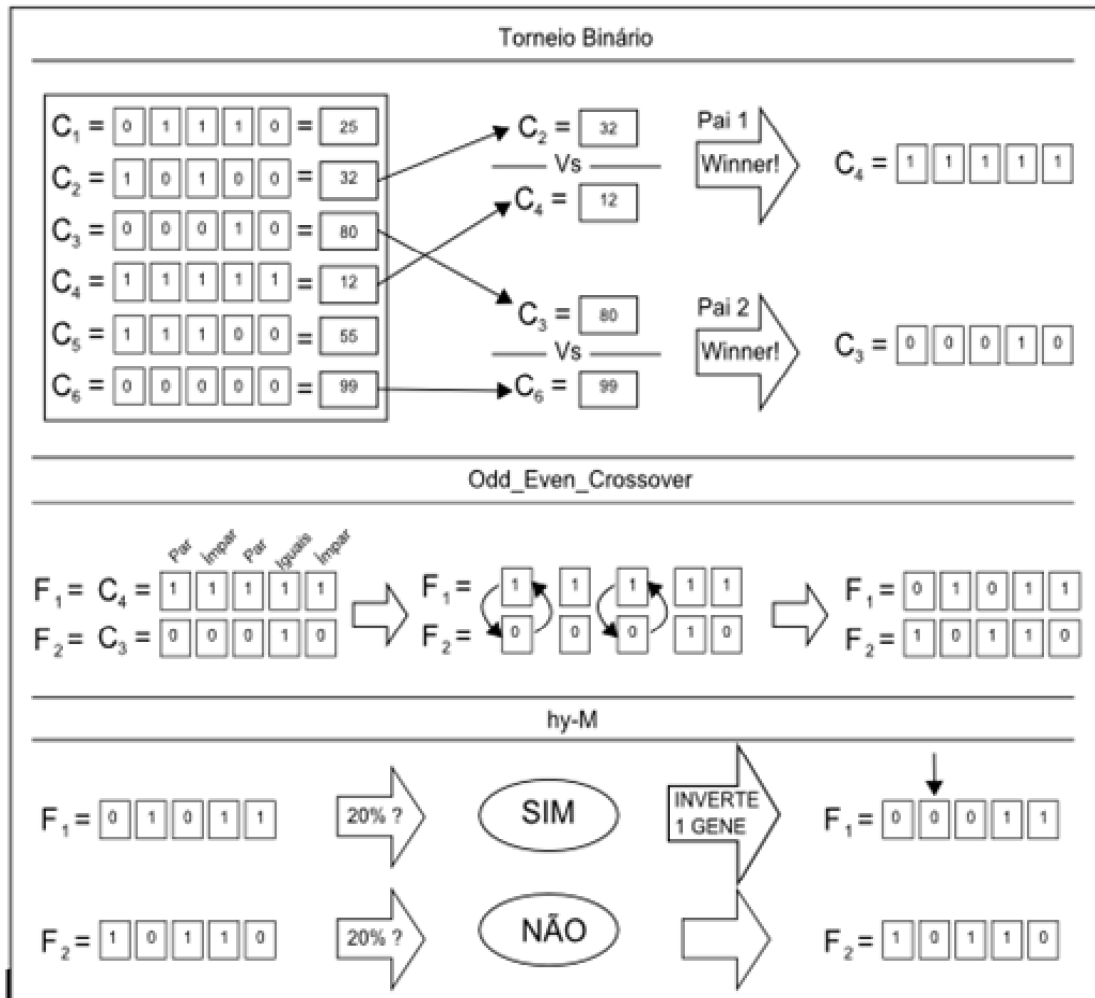


Figura 4.22: Execução do Procedimento de Reprodução

#### 4.5.4 Definição da População Sobrevivente

Para se definir a população remanescente, foi utilizado um sistema de elitismo: após as fases de reprodução e de avaliação da nova população, é gerado o conjunto  $Pop_{temp} = Pop(t) \cup Pop(t+1)$  que é classificado em ordem descendente de aptidão; Os  $m/2$  mais aptos sobrevivem automaticamente e os outros  $m/2$  sobreviventes são selecionados aleatoriamente entre os indivíduos restantes.

Este procedimento permite a melhora da aptidão dos indivíduos a cada geração sem diminuir a diversidade populacional, pois indivíduos não tão aptos podem ser sorteados e sobreviverem à próxima era.

## CAPÍTULO 5 - RESULTADOS COMPUTACIONAIS

Este capítulo pretende demonstrar os resultados obtidos pelas abordagens desenvolvidas neste trabalho frente ao estado da arte do RSMTP. Para tal, foram executadas várias simulações utilizando a princípio um conjunto de instâncias geradas aleatoriamente (*Rand\_lib*) com intuito de identificar a técnica mais promissora dentre as desenvolvidas, para, em seguida, compará-la com o FLUTE e o BGA.

A mensuração da qualidade de cada abordagem, assim como é feito na literatura, é dada de duas formas: pelo cálculo do *gap* sobre a RMST da instância:

$Gap(H, RMST) = \frac{(|RMST| - |H|)}{|RMST|}$ , que indica quão melhor é a solução

obtida frente à RMST (maiores valores indicam melhores soluções); ou, para instâncias nas quais é possível, calculando o *gap* sobre a solução ótima, que indica quão próximo a solução obtida ficou da solução ótima (menores valores

indicam melhores soluções):  $Gap(Ótimo, H) = \frac{(|H| - |Ótimo|)}{|H|}$ ;

Uma vez identificada a melhor técnica produzida por este trabalho, esta foi testada frente melhores heurísticas da literatura utilizando os *benchmarks* da *Or-Lib*.

### 5.1 Simulação com Instâncias Geradas Aleatoriamente

Com objetivo de identificar as qualidades e deficiências das abordagens implementadas neste trabalho e melhor parametrizá-las, antes de arriscar comparações com o estado da arte, foram executadas algumas simulações utilizando um conjunto de instâncias geradas de modo pseudo-aleatório denominado *Rand\_lib*.

Como detalhado na Tabela 5.1, a *Rand\_lib* é composta por doze grupos formados por cem instâncias cada, onde cada ponto possui coordenadas inteiras

$x$  e  $y$  obedecendo  $0 \leq x, y \leq 10.000$ . Estes grupos foram nomeados na forma *Rand\_N*, de acordo com o número de terminais que compõem cada instância, por exemplo, *Rand\_100* é o grupo cujos problemas são compostos por cem terminais.

Grupo	Nome	# de Instâncias	# de pontos por Instância	Valor máximo de coordenada
1	Rand_10	100	10	10.000
2	Rand_20	100	20	10.000
3	Rand_30	100	30	10.000
4	Rand_40	100	40	10.000
5	Rand_50	100	50	10.000
6	Rand_60	100	60	10.000
7	Rand_70	100	70	10.000
8	Rand_80	100	80	10.000
9	Rand_90	100	90	10.000
10	Rand_100	100	100	10.000
11	Rand_150	100	150	10.000
12	Rand_200	100	200	10.000

Tabela 5.1: Detalhamento do grupo de instâncias *Rand\_lib*

Para cada grupo de instâncias da *Rand\_lib*, foram executadas a heurística H1 e as metaheurísticas GRASP, SA e GA desenvolvidas neste trabalho, que tiveram os resultados comparados com o ótimo – obtido pelo GeoSteiner. Em uma análise rápida, logo se percebeu que a H1 e o GRASP não proporcionavam soluções competitivas frente ao SA e ao GA, como deixa claro o seu *gap* médio para o ótimo (Tabela 5.2) e para a RMST (Tabela 5.3).

	H1	GRASP	SA	GA
rand_10	2.11%	1.98%	0.00%	0.01%
rand_20	2.19%	1.99%	0.10%	0.06%
rand_30	2.76%	2.62%	0.20%	0.09%
rand_40	2.41%	2.27%	0.25%	0.12%
rand_50	2.67%	2.52%	0.24%	0.12%
rand_60	2.86%	2.72%	0.29%	0.11%
rand_70	2.68%	2.49%	0.29%	0.10%
rand_80	2.84%	2.68%	0.26%	0.15%
rand_90	2.82%	2.67%	0.27%	0.11%
rand_100	2.76%	2.11%	0.23%	0.12%
rand_150	2.91%	2.43%	0.45%	0.16%
rand_200	2.94%	2.62%	0.49%	0.18%

Tabela 5.2: *Gap* para o ótimo das técnicas H1, GRASP, SA e GA sobre a *Rand\_lib*

	H1	GRASP	SA	GA
rand_10	9.17%	9.28%	11.05%	11.04%
rand_20	9.10%	9.28%	10.95%	10.99%
rand_30	8.75%	8.88%	11.03%	11.12%
rand_40	8.85%	8.98%	10.77%	10.89%
rand_50	8.77%	8.90%	10.93%	11.04%
rand_60	8.84%	8.96%	11.12%	11.27%
rand_70	8.92%	9.08%	10.76%	11.20%
rand_80	8.74%	8.88%	11.04%	11.13%
rand_90	8.62%	8.75%	10.88%	11.02%
rand_100	8.89%	9.46%	11.13%	11.23%
rand_150	9.02%	9.44%	11.20%	11.45%
rand_200	8.97%	9.25%	11.14%	11.41%

Tabela 5.3: *Gap* para a RMST das técnicas H1, GRASP, SA e GA sobre a *Rand\_lib*

Todavia, mesmo não alcançando soluções de qualidade comparável às do SA e do GA, o baixo tempo de execução apresentado pela H1 e pelo GRASP credencia estas técnicas para que atuem construindo soluções iniciais, uma vez que apresentam árvores em média 9% menores que a RMST requerendo o mesmo esforço computacional.

Em se tratando de geração de soluções iniciais, a metaheurística híbrida GRASP/*Multi-Start* discutida em 4.3 merece destaque, pois oferece árvores de razoável qualidade, construídas a partir de diferentes regiões do espaço de soluções.

Para definir a técnica que melhor representaria esta dissertação na comparação com as heurísticas da literatura foi realizada uma nova sequência de simulações que levou em conta o tempo de execução médio do SA e do GA (na máquina descrita em 1.2.2).

A Tabela 5.4 exhibe um comparativo entre a qualidade das soluções do estado da arte e das metaheurísticas SA e GA deste trabalho, no qual também é levado em consideração o tempo de execução médio apresentado por estas. Observe que, mesmo necessitando de um esforço computacional maior, o SA não consegue superar as soluções do GA em termos de *gap* para o ótimo.

	BI1S	FLUTE 12	BGA	SA	SA (s)	GA	GA (s)
rand_10	0.35%	0.00%	0.48%	0.00%	0.56	0.01%	0.00
rand_20	0.40%	0.09%	0.51%	0.10%	1.00	0.06%	0.00
rand_30	0.56%	0.31%	0.58%	0.20%	1.60	0.09%	0.00
rand_40	0.57%	0.34%	0.62%	0.25%	2.75	0.12%	0.01
rand_50	0.53%	0.35%	0.61%	0.24%	3.31	0.12%	0.02
rand_60	0.55%	0.36%	0.66%	0.29%	4.22	0.11%	0.05
rand_70	0.57%	0.43%	0.66%	0.29%	5.75	0.10%	0.07
rand_80	0.52%	0.42%	0.56%	0.26%	7.79	0.15%	0.12
rand_90	0.53%	0.45%	0.65%	0.27%	9.58	0.11%	0.21
rand_100	0.58%	0.45%	0.60%	0.23%	11.06	0.12%	0.28
rand_150	0.61%	0.40%	0.68%	0.45%	26.85	0.16%	0.92
rand_200	0.63%	0.41%	0.68%	0.49%	49.58	0.18%	2.31

Tabela 5.4: Comparativo do *gap* para o ótimo entre o GA e o SA

Inferindo o comportamento do SA e do GA a partir dos resultados das simulações sobre a *Rand\_lib*, elegeu-se o GA para os testes sobre as instâncias da *Or\_Lib* frente às heurísticas da literatura.

## 5.2 Simulação com Instâncias Benchmark da *Or\_Lib*

O repositório de *benchmarks* para problemas de otimização *Or\_Library* (por simplicidade *Or\_Lib*) oferece um conjunto de instâncias para o SMTP que, por se tratarem apenas de conjuntos de pontos no plano cartesiano, também são amplamente utilizadas na literatura do RSMTP. Este conjunto é composto por treze subgrupos, denominados *Or\_Lib\_n*, cada um contendo quinze instâncias compostas por  $n$  terminais, para  $n = (10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 250, 500, 1000)$ .

A Tabela 5.5 detalha os grupos de instâncias que compõem o *benchmark* da *Or\_Lib*. As diferenças entre as instâncias da *Rand\_lib* e da *Or\_lib* são:

- Número de instâncias por grupo: cem na *Rand\_lib*, quinze na *Or\_lib*;
- Valor máximo de coordenada: na *Rand\_lib*  $0 \leq x, y \leq 10.000$ , na *Or\_lib*  $0 \leq x, y \leq 10.000.000$ ;
- Número máximo de pontos por instância: duzentos na *Rand\_lib*, mil na *Or\_lib*.

Grupo	Nome	# de Instâncias	# de pontos por Instância	Valor máximo de coordenada
1	Or_lib_10	15	10	10.000.000
2	Or_lib_20	15	20	10.000.000
3	Or_lib_30	15	30	10.000.000
4	Or_lib_40	15	40	10.000.000
5	Or_lib_50	15	50	10.000.000
6	Or_lib_60	15	60	10.000.000
7	Or_lib_70	15	70	10.000.000
8	Or_lib_80	15	80	10.000.000
9	Or_lib_90	15	90	10.000.000
10	Or_lib_100	15	100	10.000.000
11	Or_lib_250	15	250	10.000.000
12	Or_lib_500	15	500	10.000.000
13	Or_lib_1000	15	1000	10.000.000

Tabela 5.5: : Detalhamento do grupo de instâncias *benchmark* da *Or\_lib*

As simulações foram realizadas para cada instância de cada grupo da *Or\_lib* e a qualidade da solução foi aferida mediante comparação com a RMST e com a árvore ótima obtida pelo GeoSteiner. Vale salientar que o GeoSteiner não foi capaz de solucionar nove das quinze instâncias do grupo *Or\_Lib\_1000*.

O resultado utilizado para comparação com o estado da arte é a média aritmética do custo das árvores obtidas em dez execuções do GA.

Para cada grupo de instâncias da *Or\_Lib*, classificadas conforme o número de terminais, foi gerada uma tabela com os resultados obtidos pelo GeoSteiner (Ótimo), pelo método de geração de RMSTs proposto neste trabalho, pelo GA e pelas heurísticas do estado da arte: FLUTE (parametrizado para  $A=12$ ) e BGA. Cada tabela apresenta, adicionalmente, os *gaps* médios para o ótimo e para a RMST.

A Tabela 5.6 e o gráfico da Figura 5.1 exibem o *gap* para o ótimo obtido para cada grupo de instâncias da *Or\_Lib*. O GA conseguiu os melhores resultados para todos os grupos, exceto *Or\_Lib\_10* e *Or\_Lib\_30*. Para o grupo *Or\_Lib\_500*, formado por *nets* de quinhentos terminais, o GA apresenta árvores de comprimento em média 0,20% maior que a RMST, enquanto o FLUTE e o BGA obtêm resultados de 0,45 e 0,71% respectivamente.

	GA	BGA	FLUTE
10	0,010%	0,222%	<b>0,005%</b>
20	<b>0,029%</b>	0,345%	0,129%
30	0,127%	0,705%	<b>0,111%</b>
40	<b>0,091%</b>	0,675%	0,188%
50	<b>0,178%</b>	0,554%	0,194%
60	<b>0,113%</b>	0,570%	0,504%
70	<b>0,172%</b>	0,667%	0,508%
80	<b>0,105%</b>	0,570%	0,462%
90	<b>0,195%</b>	0,804%	0,280%
100	<b>0,134%</b>	0,444%	0,519%
250	<b>0,138%</b>	0,663%	0,380%
500	<b>0,206%</b>	0,711%	0,451%

Tabela 5.6: Tabela-Resumo do gap para o ótimo

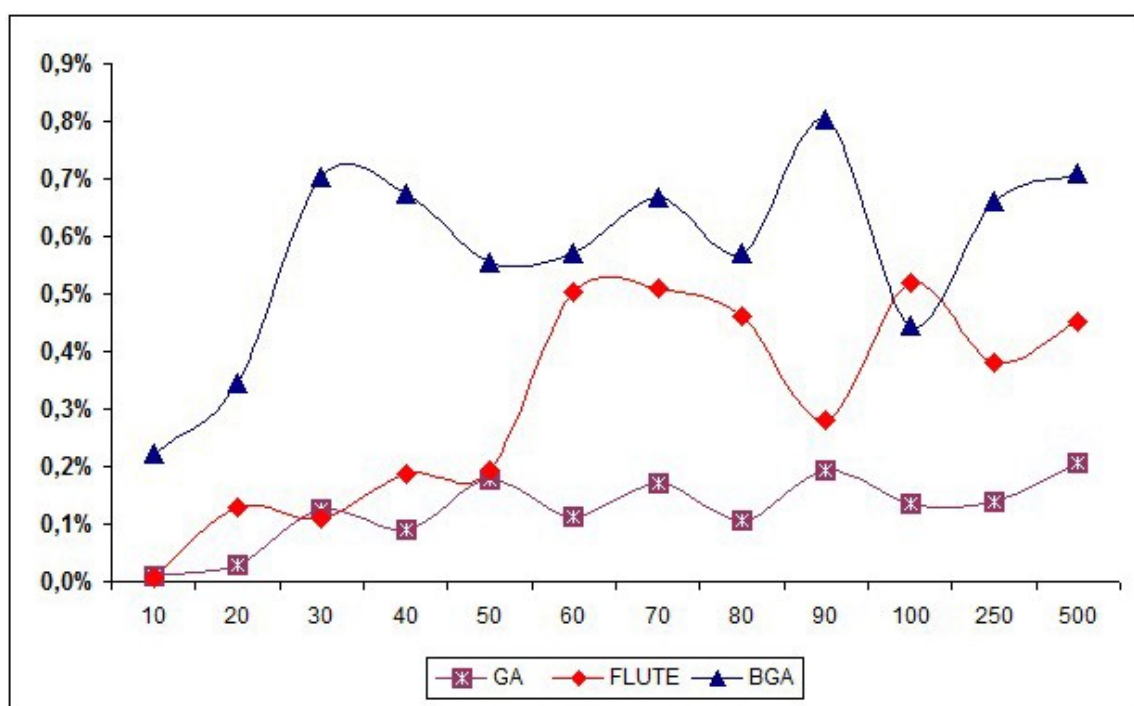


Figura 5.1: Relação entre o gap para o ótimo e o número de pontos das instâncias

A Tabela 5.7 e o gráfico da Figura 5.2, que demonstram o *gap* médio entre os resultados obtidos pelo GA, FLUTE e BGA sobre a RMST, corroboram a metaheurística GA como detentora das árvores de menor comprimento.



	GA	BGA	FLUTE
10	10,915%	10,726%	<b>10,919%</b>
20	<b>11,869%</b>	11,591%	11,781%
30	11,494%	10,984%	<b>11,508%</b>
40	<b>10,909%</b>	10,390%	10,824%
50	<b>10,715%</b>	10,380%	10,701%
60	<b>11,780%</b>	11,377%	11,436%
70	<b>11,240%</b>	10,802%	10,942%
80	<b>11,233%</b>	10,820%	10,916%
90	<b>11,286%</b>	10,746%	11,210%
100	<b>11,612%</b>	11,338%	11,273%
250	<b>11,529%</b>	11,065%	11,315%
500	<b>12,607%</b>	12,167%	12,394%
1000	<b>11,469%</b>	11,082%	11,247%

Tabela 5.7: Tabela-Resumo do *gap* sobre a RMST

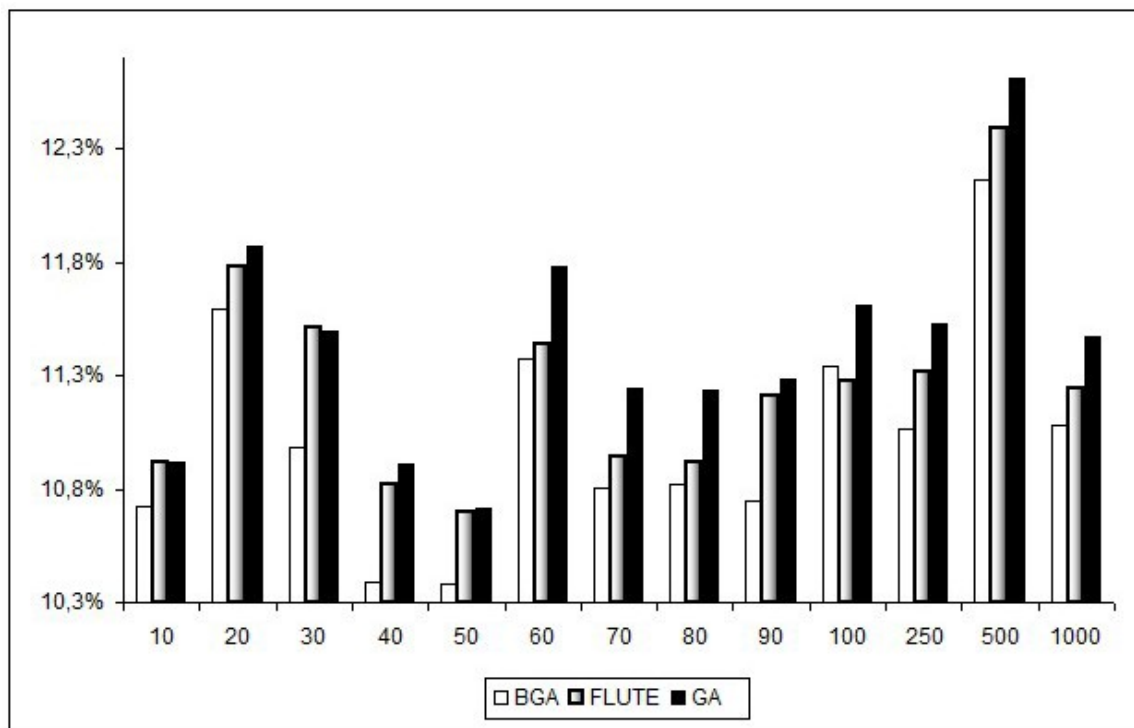


Figura 5.2: Melhoria média sobre a RMST

Os melhores resultados obtidos pelo GA foram para os grupos *Or\_Lib\_100*, *Or\_Lib\_250*, *Or\_Lib\_500* e *Or\_Lib\_1000*, detalhados, respectivamente, na Tabela 5.8, Tabela 5.9, Tabela 5.10 e Tabela 5.11, para os quais apenas não se obteve a árvore de menor comprimento para três instâncias, de um total de sessenta.

NOME	N	Instância	RMST	ÓTIMO	ALGORITMO GENÉTICO			BGA	FLUTE12
					MIN	MED	MAX		
OR-Lib--100-1.in	100	1	82516784	72522165	72522165	72614922	72698376	73146846	73155367
OR-Lib--100-2.in	100	2	85867210	75176630	75303154	75362003	75424094	75852545	75737355
OR-Lib--100-3.in	100	3	82320478	72746006	72783567	72783770	72784377	72784377	72746816
OR-Lib--100-4.in	100	4	83346063	74342392	74342392	74342410	74342465	74438767	74859621
OR-Lib--100-5.in	100	5	84275388	75670198	75670198	75670198	75670198	75675434	75983599
OR-Lib--100-6.in	100	6	85427448	74414990	74445875	74633821	74820451	74722697	74874833
OR-Lib--100-7.in	100	7	89036106	77740576	77911201	78069937	78136665	78143518	78790506
OR-Lib--100-8.in	100	8	82313369	73033178	73038516	73076491	73117135	73443123	73352432
OR-Lib--100-9.in	100	9	90184676	77952027	78039079	78123214	78199088	78277124	78433727
OR-Lib--100-10.in	100	10	85725695	75952202	75984703	76002999	76039487	76003103	76113751
OR-Lib--100-11.in	100	11	87384410	78674859	78684470	78698287	78739737	79048226	78814848
OR-Lib--100-12.in	100	12	86995616	76131099	76229364	76258069	76326812	76546220	76229364
OR-Lib--100-13.in	100	13	85036280	74604990	74619136	74662454	74716685	75041065	75047747
OR-Lib--100-14.in	100	14	87985079	78632795	78654052	78654052	78654052	79153368	78825706
OR-Lib--100-15.in	100	15	79537321	70446493	70573355	70603214	70661616	70775631	70927139
<b>SOMA</b>			1277951923	1128040600	1128801227	1129555839	1130331238	1133052044	1133892811

GAP( ÓTIMO )				
760627	1515239	2290638	5011444	5852211
0,067%	0,134%	0,203%	0,444%	0,519%

GAP( RMST )				
149150696	148396085	147620685	144899879	144059112
11,671%	11,612%	11,551%	11,338%	11,273%

Tabela 5.8: Resultados Obtidos para o grupo 100 da *Or\_lib*

NOME	N	Instância	RMST	ÓTIMO	ALGORITMO GENÉTICO			BGA	FLUTE12
					MIN	MED	MAX		
OR-Lib--250-1.in	250	1	130822271	116609813	116655265	116774069	116831226	117436700	117100164
OR-Lib--250-2.in	250	2	130344539	115150079	115254778	115256345	115257911	115868226	115785313
OR-Lib--250-3.in	250	3	129157105	114650399	114658840	114737381	114856253	115104746	114840444
OR-Lib--250-4.in	250	4	132907026	117819530	117906314	117964251	118081025	118727707	118207533
OR-Lib--250-5.in	250	5	132872202	116927089	117059087	117083942	117117578	117654953	117256761
OR-Lib--250-6.in	250	6	130005555	116256250	116256250	116318245	116411035	116883687	116675372
OR-Lib--250-7.in	250	7	128946271	115277351	115366713	115412615	115477642	115756145	115465547
OR-Lib--250-8.in	250	8	133832693	116833323	116896779	117053362	117229283	118040456	117363831
OR-Lib--250-9.in	250	9	132510403	116821988	116845389	116871372	116932911	117297858	117350728
OR-Lib--250-10.in	250	10	133565461	116857628	116937593	116963767	117013314	117550511	117452201
OR-Lib--250-11.in	250	11	128332519	112889613	113151452	113203319	113222273	113683651	113491191
OR-Lib--250-12.in	250	12	134362218	119035256	119164050	119206273	119237132	119817384	119366548
OR-Lib--250-13.in	250	13	132002881	116049496	116174530	116279883	116389954	117099325	116456131
OR-Lib--250-14.in	250	14	131652998	116188791	116259823	116406060	116661937	116969290	116665456
OR-Lib--250-15.in	250	15	131449213	115558198	115659949	115799584	115959869	116587336	116068650
<b>SOMA</b>			1972763355	1742924804	1744246812	1745330465	1746679343	1754477975	1749545870

GAP( ÓTIMO )				
1322008	2405661	3754539	11553171	6621066
0,076%	0,138%	0,215%	0,663%	0,380%

GAP( RMST )				
228516543	227432890	226084012	218285380	223217485
11,584%	11,529%	11,460%	11,065%	11,315%

Tabela 5.9: Resultados Obtidos para o grupo 250 da *Or\_lib*



NOME	N	Instância	RMST	ÓTIMO	ALGORITMO GENÉTICO			BGA	FLUTE12
					MIN	MED	MAX		
OR-Lib--500-1.in	500	1	184189997	162978810	163215198	163276912	163298965	164131706	163603274
OR-Lib--500-2.in	500	2	184354068	160756854	160964422	161065330	161158201	161989295	161938069
OR-Lib--500-3.in	500	3	185683216	162664661	162890685	162975639	163044601	164203258	163603709
OR-Lib--500-4.in	500	4	184943980	164110997	164466031	164504338	164564161	165271779	164962802
OR-Lib--500-5.in	500	5	180605717	160586161	160645338	160752657	160839012	161611277	161206584
OR-Lib--500-6.in	500	6	186488358	164685074	165146360	165199875	165271165	165744335	165330549
OR-Lib--500-7.in	500	7	186488358	160124233	160400519	160498404	160584534	161410381	160547399
OR-Lib--500-8.in	500	8	186488358	161248138	161422847	161481660	161548601	162556823	161961465
OR-Lib--500-9.in	500	9	186488358	162100435	162362175	162482523	162570356	163175619	163170721
OR-Lib--500-10.in	500	10	186488358	155581203	155795371	155872320	155975409	156770203	156247520
OR-Lib--500-11.in	500	11	186488358	161674316	161892073	161933271	161965815	162708192	162357898
OR-Lib--500-12.in	500	12	186488358	164009591	164159202	164231355	164273245	165367230	164592479
OR-Lib--500-13.in	500	13	186488358	161324201	161578346	161719236	161849023	162028045	161689118
OR-Lib--500-14.in	500	14	186488358	165984329	166296601	166471466	166635781	167329576	166973895
OR-Lib--500-15.in	500	15	186488358	160758467	161041439	161129376	161215397	161557319	161354683
SOMA			2784660558	2428587470	2432276607	2433594362	2434794266	2445855038	2439540165

GAP( ÓTIMO )				
3689137	5006892	6206796	17267568	10952695
0,152%	0,206%	0,256%	0,711%	0,451%

GAP( RMST )				
352383951	351066196	349866292	338805520	345120393
12,654%	12,607%	12,564%	12,167%	12,394%

Tabela 5.10: Resultados Obtidos para o grupo 500 da *Or\_lib*

NOME	N	Instância	RMST	ÓTIMO	ALGORITMO GENÉTICO			BGA	FLUTE12
					MIN	MED	MAX		
OR-Lib--1000-1.in	1000	1	261505815	230535806	230900149	231036896	231131162	232010078	231446549
OR-Lib--1000-2.in	1000	2	257303781	227886471	228478458	228520475	228557503	229434203	228776507
OR-Lib--1000-3.in	1000	3	256431483	X	228512073	228556665	228585712	229114768	229184602
OR-Lib--1000-4.in	1000	4	260424558	230200846	230464630	230587081	230673100	232009817	231225023
OR-Lib--1000-5.in	1000	5	257545964	X	228695565	228849847	229066023	229899174	229167520
OR-Lib--1000-6.in	1000	6	261265315	X	231587378	231652191	231760537	232444278	232253636
OR-Lib--1000-7.in	1000	7	260464728	X	231330801	231448320	231577736	232275215	231622433
OR-Lib--1000-8.in	1000	8	261495834	X	230986613	231069024	231118669	232212089	232182219
OR-Lib--1000-9.in	1000	9	259091016	227745838	228281767	228353152	228466041	229503167	228939283
OR-Lib--1000-10.in	1000	10	259966756	X	229514785	229651615	229724814	230591318	230131708
OR-Lib--1000-11.in	1000	11	261274503	231605619	232034303	232097922	232170278	233297524	232736865
OR-Lib--1000-12.in	1000	12	264526629	X	231527448	231628902	231787493	232303062	232086443
OR-Lib--1000-13.in	1000	13	257465011	X	228283319	228354315	228420476	229217397	228831644
OR-Lib--1000-14.in	1000	14	265492151	X	234573362	234608752	234651395	235715094	235547577
OR-Lib--1000-15.in	1000	15	260088797	229965775	230122654	230143949	230168961	231621419	231090821
SOMA			3904342341	1377940355	3455293305	3456559104	3457859900	3471648603	3465222830

GAP( ÓTIMO )				
2077352950	2078618749	2079919545	2093708248	2087282475
X	X	X	X	X

GAP( RMST )				
449049036	447783238	446482441	432693738	439119511
11,501%	11,469%	11,436%	11,082%	11,247%

Tabela 5.11: Resultados Obtidos para o grupo 1000 da *Or\_lib*

## CAPÍTULO 6 - CONSIDERAÇÕES FINAIS E PROPOSTA DE TRABALHOS FUTUROS

Neste trabalho foi realizado um estudo abrangente, profundo e detalhado sobre o problema da árvore retilínea mínima de Steiner e concebido um conjunto de novas abordagens heurísticas e metaheurísticas para este problema.

O estudo realizado sobre o estado da arte para o RSMTTP propicia uma preciosa fonte de pesquisa àqueles que desejarem embarcar neste tema, pois centraliza em um único trabalho as melhores características das mais bem sucedidas técnicas de RSMT encontradas na literatura, abordando-as de forma sucinta e direta.

Os resultados obtidos reiteram a qualidade das abordagens desenvolvidas neste trabalho frente às melhores heurísticas da literatura. Contudo, ainda se faz necessário uma equiparação em relação ao tempo de execução. Para tal, em respeito a trabalhos posteriores, proponho um processo de paralelização do algoritmo genético proposto em 4.5.

O algoritmo genético apresentado nesta dissertação representa, até a presente data, a técnica que provê as árvores retilíneas de Steiner de menor comprimento dentre todos os algoritmos aproximativos já publicados. O GA implementado demonstra-se uma técnica bastante robusta para instâncias de qualquer tamanho, produzindo, em especial para instâncias compostas por 200 terminais, soluções de custo apenas 0,16% maior que a solução ótima, frente os 0,41% alcançados pelo FLUTE com  $A = 12$ .

As contribuições desta dissertação não se limitam apenas às abordagens propostas por esta. Novas técnicas de geração de RSMTs podem se valer de alguns dos conceitos-chave definidos neste, como o novo esquema de manutenção das estruturas de dados do HGP, a nova técnica de manutenção dinâmica de MSTs ou a estrutura de movimentos e de vizinhança criada para o *Simulated Annealing*.

Foram idealizados, ainda, alguns possíveis melhoramentos sobre as técnicas desenvolvidas, como a adoção de um esquema de refinamento por reconexão de caminhos (*Path-Relinking*) ou a utilização do conceito de independência entre pontos de Steiner do BI1S como etapa de pré-processamento do algoritmo genético objetivando reduzir seu tempo de execução. Ficam como propostas de trabalho futuro.

## REFERÊNCIAS

- Cabral, L. A. F., Paralelizando a Fase de Roteamento de Circuitos Baseados em FPGAs. Tese de Doutorado, Rio de Janeiro, 2001.
- Chu, C., FLUTE: Fast Lookup Table Based Wirelength Estimation Technique. In Proc. International Conference on Computer Aided Design, pages 696-701, 2004.
- Chu, C., Wong, Y., Fast and Accurate Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. In Proc. International Symposium on Physical Design, pages 28-35, 2005.
- Chu, C., Wong, Y., FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design. In IEEE Transactions on Computer-Aided Design, vol. 27, no. 1, pages 70-83, 2008-A.
- Chu, C., Wong, Y., A Scalable and Accurate Rectilinear Steiner Minimal Tree Algorithm. In Proc. International Symposium on VLSI Design, Automation and Test, 2008-B.
- Cormen, T. H., et. al., Introduction to Algorithms, MIT Press, 1184 pp, 2001.
- Fenlason, J., Stallman, R., GNU gprof - The GNU Profiler, disponível em [http://www.cs.utah.edu/dept/old/texinfo/as/gprof\\_toc.html](http://www.cs.utah.edu/dept/old/texinfo/as/gprof_toc.html), Acessado em 2008.
- Feo, T., Resende, M., Smith, S., A greedy randomized adaptive search procedure for maximum independent set. Operations Research, 42:860–878, 1994.
- Ferreira, C.M.S., Ochi, L.S., Metaheurística GRASP com Memória Adaptativa para a solução do Problema da Árvore Geradora Mínima Generalizado, Proc. of the VI Encontro Nacional de Inteligência Artificial (VI ENIA), realizado em conjunto com o XXVII Congresso da SBC, pp: 1202-1211 - 2007.



- Fößmeier, U., Kaufmann, M., Zelikovsky, A. Z., Faster approximation algorithms for the rectilinear Steiner tree problem. In Proceedings of the 4th Annual International Symposium on Algorithms and Computation, Springer-Verlag, 1993.
- Garey, M., Johnson, D. S., The rectilinear Steiner problem is NPcomplete, SIAM J Applied Math., vol. 32, pp. 826-834, 1977.
- Garey, M. R., Graham, R. L., Johnson, D. S., The complexity of computing Steiner minimal trees, SIAM J. Appl. Math. 32, 835-839, 1977.
- Gilbert, E. N., Polak, H. O., Steiner Minimal Trees, SIAM J. Applied Math, 16, pp. 1-29, 1968.
- Goldberg D. E., Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Berkeley, 1989.
- Griffith, J., Robins, G., Salowe, J. S., et. al., Closing the gap: Near-optimal steiner trees in polynomial time. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 13(11):1351--1365, November 1994.
- Guibas, L. J., Stolfi, J., On computing all northeast nearest neighbors in the L1 metric, Information Processing Letters, 17, pp. 219-223, 1983.
- Hajek, B., Cooling Schedules for Optimal Annealing, Mathematics of Operations Research, 13, pp. 311-329, 1988.
- Hanan, M., "On Steiner's problem with rectilinear distance," SIAM, Applied Math., vol. 14, pp. 255-265, 1966.
- Holland, J. H., Adaptation in Natural and Artificial Systems. MIT Press, 1975.
- Hwang, F., On Steiner Minimal Trees with Rectilinear Distance, SIAM Journal of Applied Mathematics, Vol. 30, No. 1, pp. 104-114, 1976.

- Kahng, A. B., Robins, G., A new class of iterative Steiner tree heuristics with good performance. IEEE Transactions on Computer-Aided Design, 11:893-902, 1992.
- Kahng, A., Mandoiu, I., GSRC Bookshelf RMST-Pack: Rectilinear Minimum Spanning Tree Algorithms,. <http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/RMST/>. 2001.
- Kahng, A. B., Mandoiu, I., Zelikovsky, A., Highly scalable algorithms for rectilinear and octilinear Steiner trees, Proc. Asia and South Pacific Design Automation Conference (ASP-DAC), pp. 827-833, 2003.
- Lacerda, E. G. M., Carvalho, A. C. P. L., Introdução aos Algoritmos Genéticos. In: Galvão, C. O., Valença, M. J. S., Sistemas Inteligentes: aplicações a recursos hídricos e ciências ambientais. Porto Alegre: Ed. Universidade/UFRGS: Associação Brasileira de Recursos Hídricos. pp. 99-150, 1999.
- Mandoiu, I. I., Vazirani, V. V., Ganley, J. L., A New Heuristic for Rectilinear Steiner Trees, 1999, iccad, pp.157.
- Mazzucco Junior, J., Uma abordagem híbrida do problema da programação da produção através dos algoritmos Simulated Annealing e Genético. Tese de Doutorado, UFSC, Florianópolis, 1999.
- Resende, M. G. C., Ribeiro, C. C., Greedy randomized adaptive search procedures, in Handbook of Metaheuristics, Kluwer Academic Publishers, pp. 219-249, 2003.
- Ribeiro, C. C., Metaheuristics and Applications. In Advanced School on Artificial Intelligence, Estoril, Portugal, 1996.
- Rocha, M. L., Alvarenga, F. V., Uma Metaheurística GRASP Para o Problema da Árvore Geradora de Custo Mínimo com Grupamentos Utilizando Grafos Fuzzy. INFOCOMP (UFLA), v. 5, n. 1, p. 66-75, 2006.

- Seward, J., et. al., página do projeto Valgrind na Internet, disponível em <http://valgrind.org/>, Acessado em 2008.
- Snyder, T. L., Worst-Case Minimum Rectilinear Steiner Trees in All Dimensions. *Discrete & Computational Geometry* 8: 73-92, 1992.
- Souza, M. J. F., Notas de aula sobre inteligência computacional, disponível em <http://www.decom.ufop.br/prof/marcone/Disciplinas/InteligenciaComputacional/InteligenciaComputacional.htm>, Acessado em 2008.
- Torreão J. R. A., Inteligência Computacional. Notas de aula, Universidade Federal Fluminense, Niterói, 2008.
- Warne, D. M., Winter, P., Zachariasen, M., Exact Algorithms for Plane Steiner Tree. Problems: A Computational Study, University of Copenhagen, 1998.
- Warne, D. M., Spanning Trees in Hypergraphs with Applications to Steiner Trees. Ph.D. Thesis, Computer Science Dept., The University of Virginia, 1998.
- Warne, D. M., Winter, P., Zachariasen, M. GeoSteiner – Software for Computing Steiner Trees, <http://www.diku.dk/hjemmesider/ansatte/martinz/geosteiner/>, Acessado em Junho de 2009.
- Winter P., Reductions for the rectilinear Steiner tree problem, *Networks* 26 187-198. 25, 1995.
- Zachariasen, M., Rectilinear Full Steiner Tree Generation. Technical Report 97/29, DIKU, Department of Computer Science, University of Copenhagen, 1997.
- Zachariasen, M., A Catalog of Hanan Grid Problems. Technical Report 00892, Institute for Discrete Mathematics, University of Bonn, 2000.
- Zachariasen, M., Rohe, A., Rectilinear Group Steiner Trees and Applications in VLSI Design. Technical Report 00906, Institute for Discrete Mathematics, University of Bonn, 2000.

Zelikovsky, A. Z., An  $11/6$ -approximation algorithm for the Steiner problem on graphs. In Proceedings of the Fourth Czechoslovakian Symposium on Combinatorics, Graphs, and Complexity, pages 351-354, 1992.