Problemas de Otimização NP, Aproximabilidade e Computação Evolutiva: Da Prática à Teoria

Pablo Moscato

Departamento de Engenharia de Sistemas
Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas
Brasil

Tese de Doutorado, Março de 2001

Banca Examinadora

Paulo Morelato França - FEEC - UNICAMP (Presidente, orientador)
Carlos Cotta-Porras - Universidad de Málaga, España
Christiano Lyra Filho - FEEC - UNICAMP
Fernando José Von Zuben - FEEC - UNICAMP
Mauricio G.C. Resende - AT&T Labs-Research, New Jersey, USA
Paulo Augusto Valente Ferreira - FEEC - UNICAMP
Rafael Santos Mendes - FEEC - UNICAMP

Tese defendida na Faculdade de Engenharia Elétrica e de Computação
Universidade Estadual de Campinas, SP, Campinas, Brasil. 26 Março de 2001

© Pablo Moscato 2001
MMI

Resumo

Este trabalho estuda problemas de Otimização Combinatória e a sua resolução aproximada com métodos derivados da Computação Evolutiva. O objetivo é tentar identificar as características dos problemas que permitiriam decidir *"a priori"* se metaheurísticas evolutivas são recomendadas ou não para o problema de interesse. São estudados os problemas do Caixeiro Viajante e o problema de partição de um conjunto de números inteiros. Conclui-se que as classes de complexidade computacional derivadas dos algoritmos de aproximação existentes para os problemas estudados não estão correlacionadas com a dificuldade prática em resolvê-los com estas metaheurísticas. Logo, propõem-se novas classes de complexidade computacional para estudar a dificuldade de recombinar soluções. Discute-se a utilização da Teoria da Complexidade Parametrizada e alguns problemas de otimização da Teoria de Grafos são estudados como exemplos. Uma proposta de atividades futuras na área de metaheurísticas é feita, como a hibridização com métodos sistemáticos de busca para gerar Algoritmos Meméticos Completos e a utilização de Lógica Modal para incorporar conhecimento dependente da instância do problema e assim guiar o processo de busca.

Palavras-Chave: Algoritmos Meméticos, Otimização Combinatória, Pesquisa Operacional.

Abstract

This work studies combinatorial optimization problems and their approximate solution with methods derived from Evolutionary Computation. The objective is to try to identify the problem-dependent features that would help to decide *"a priori"* if evolutionary metaheuristics are recommended or not for the problem under study. The Traveling Salesman Problem and the Number Partitioning problem are studied. We conclude that the classes of computational complexity derived from the concept of approximability of NP Optimization problems are not correlated with the practical difficulty in solving them with these particular type of metaheuristics. As a consequence, new computational complexity classes are proposed to study the hardness of recombining feasible solutions. With this purpose in mind, the usefulness of the Theory of Parameterized Complexity is discussed as well as some optimization problems in graphs as concrete examples of its application. A proposal for future activities in the area of metaheuristics is made, like the hybridization with systematic search methods in order to build Complete Memetic Algorithms. We also comment on the possible utilization of Modal Logic to incorporate instance-dependent and problem-dependent knowledge as well as to guide the search process.

Keywords: Memetic Algorithms, Metaheuristics, Combinatorial Optimization, Operations Research.

Agradecimentos

A meus pais, instituição também conhecida como *"Fundação Moscato para o avanço das Ciências"*. Pela sua dedicação acadêmica que estende-se desde a formação do espírito crítico-científico, à compra de infinitas enciclopédias, até à troca de fraldas e outras atividades extra-curriculares. Não tem país, nem filho, que possa devolver o que vocês me deram, então fico devendo-lhes eternamente, mas deixo-lhes meu compromisso de formar minha própria "Fundação".

A Regina, meu amor, e verdadeiro motor dessa tese.

Tenho também que agradecer a Michael G. Norman, Enrico Santi, José Andrés Tierno, Scott Carder, Dana Hadley, José Fernando Fontanari, e Edoardo Amaldi, meus amigos de Caltech; pois todos eles sempre quiseram ver as três letras "PhD" antes do meu nome. Curiosamente, mas sem dúvida pela verdadeira amizade, sempre me impulsionaram a em obter essas três letras, muito além do meu próprio interesse. Baseado no seu interesse e no seu profissionalismo eu achei motivação para esse trabalho.

A meu irmão César, com meu carinho e minha admiração pelo seu interesse em manter sempre presentes os aspectos sociais e humanos do seu trabalho.

A meus colegas de trabalho, Regina Berretta, Luciana Buriol, Carlos Cotta-Porras, e Alexandre Mendes, pois muito do que finalmente está nessa tese vem de um sempre estimulante trabalho em conjunto. Espero que siga-se repetindo !

A meu orientador no Caltech Concurrent Computation Program, Geoffrey C. Fox, que permiteu que Norman e eu começacemos a experimentar nossos primeiros algoritmos meméticos em Novembro de 1988. Também agradeço a dois extraordinários professores que tive naquela universidade, Yaser Abu-Mostafa e Stephen Judd.

A Christos H. Papadimitriou, por ter lido a proposta inicial dessa tese e dar-me seu apoio. Espero que o resultado não o defraude, mas como vaticinou, o projeto serviu para ter-me ocupado por um bom tempo. O exemplar da Ref. [225] que ele me deu foi bem útil para cobrir vários aspectos da tese. Também a Michael R. Fellows, pelo seu apoio nas etapas finais da tese, seus convites, e pelo exemplar de [107].

Meu agradecimento a Michael J. Dinneen, David Eppstein, Fred Glover, Gregory Gutin, David S. Johnson, Viggo Kann, e Luca Trevisan, pelos conselhos recebidos e por estarem sempre dispostos em questão de minutos a responder alguma questão que me atormenta. Muito obrigado pela ajuda.

Aos vários pesquisadores que atualmente estão trabalhando em algoritmos meméticos, Edmund K. Burke, Natalio Krasnogor, Peter Merz, e tantos outros por manter-me informado das suas últimas descobertas.

Finalmente, a meu orientador, Paulo Morelato França, que me deu a oportunidade de poder mudar de país e de trabalhar tão à vontade que, apesar do meu "Portunhol", jamais me senti um estrangeiro. Também espero que continuemos nossa colaboração. Acho que o melhor que podemos fazer em pesquisa ainda está por vir !

Á FAPESP, pelo apoio financeiro, e por ser uma instituição que deve ser motivo de orgulho de todo brasileiro, muito mais que pelo futebol.

Índice

1 Introdução **1**
- 1.1 Motivação . . . 1
  - 1.1.1 Metodologia: Três aspectos da questão central . . . 2
  - 1.1.2 A expressão "fácil de resolver" . . . 3
  - 1.1.3 Dificuldade de problemas e reduções . . . 4
  - 1.1.4 Aproximabilidade de problemas de otimização NP . . . 5
- 1.2 A tarefa de fornecer soluções aproximadas . . . 8
  - 1.2.1 Exemplo: Aproximabilidade de Min Vertex Cover . 8
- 1.3 Os diferentes graus de aproximabilidade: A classe APX . . . 10
  - 1.3.1 Problemas que são mais difíceis para aproximar . . . 11
- 1.4 Aproximação usando tipos particulares de heurísticas . . . 12
  - 1.4.1 Aproximação por busca local . . . 12
- 1.5 Recombinação . . . 13
- 1.6 Como ler esta tese . . . 15
- 1.7 Notação básica . . . 16

2 Algoritmos Meméticos **19**
- 2.1 Introdução e história . . . 19
- 2.2 Algoritmos Meméticos . . . 21
  - 2.2.1 Conceitos básicos . . . 21
  - 2.2.2 Paisagem de busca . . . 23
  - 2.2.3 Busca Local vs. Busca Baseada em Populações . . . 26
  - 2.2.4 Recombinação . . . 28
  - 2.2.5 Projetando um algoritmo memético . . . 31
- 2.3 Aplicações de algoritmos meméticos . . . 34
  - 2.3.1 Problemas "tradicionais" da classe NPO . . . 35
  - 2.3.2 Outros Problemas de Otimização Combinatória . . . 35
  - 2.3.3 Aprendizado de máquinas e robótica . . . 36
  - 2.3.4 Eletrônica e engenharia . . . 36
  - 2.3.5 Problemas de otimização molecular . . . 36
  - 2.3.6 Outras aplicações . . . 37

3 Aproximabilidade **39**
- 3.1 Definições das classes NP, NPO e PO . . . 39
  - 3.1.1 Auto-redutibilidade . . . 41
  - 3.1.2 Subotimalidade . . . 41

3.2 Medidas de aproximabilidade 43
3.2.1 Erro relativo 43
3.2.2 Erro relativo normalizado 44
3.2.3 Erro absoluto 44
3.2.4 Medida respeitosa do custo 45
3.2.5 Razão de desempenho 45
3.2.6 Aproximação dentro de uma constante 45
3.2.7 Exemplo: O algoritmo de Christofides 45
3.2.8 Aproximação dentro de uma função 47
3.3 Esquemas de aproximação 48
3.3.1 Esquemas de aproximação em tempo polinomial 48
3.3.2 O papel da planaridade e o TSP 48
3.3.3 Esquema de aproximação em tempo completamente polinomial 49
3.3.4 Algoritmos pseudo-polinomiais 49
3.4 NPO PB, problemas NPO polinomialmente limitados 50
3.5 Resultados de não aproximabilidade 51

4 O Problema do Caixeiro Viajante Assimétrico 53
4.1 Introdução e Motivação 53
4.2 MAs para o ATSP 56
4.3 Características da implementação independentes do problema . 59
4.3.1 Tamanho e estrutura da População 60
4.3.2 Atualizações dos agentes 61
4.4 Métodos dependentes do ATSP 63
4.4.1 População inicial 63
4.4.2 Operadores de recombinação 64
4.4.3 O Strategic Arc Crossover (SAX) 64
4.4.4 Distance Preserving Crossover (DPX) 65
4.4.5 Multiple Fragment-Nearest Neighbor Repair Edge Recombination (MFNN) 66
4.4.6 Uniform Nearest Neighbor Crossover (UNN) 66
4.4.7 Similaridades e diferenças entre os operadores de recombinação 67
4.4.8 Recursive Arc Insertion 67
4.5 Mantendo a diversidade da população 69
4.5.1 Política relativa à seleção e recombinação 70
4.5.2 Seleção do procedimento de mutação 70
4.5.3 Restart da população 71
4.6 Experimentos computacionais e resultados 71
4.7 Comparação extra: Resultados em instâncias simétricas 76
4.7.1 Sem estruturas de dados *"ad-hoc"* baseadas nas características geométricas das instâncias 76
4.7.2 Explorando a natureza geométrica das instâncias 79
4.8 Conclusões 81

5 O problema MIN NUMBER PARTITIONING **85**
5.1 Introdução 85
5.1.1 O Problema 86
5.1.2 Problemas NP-completos no sentido fraco 86
5.2 Motivação 87
5.2.1 Relação com outros problemas NP 88
5.3 A Heurística de Karmarkar-Karp 91
5.4 Resultados anteriores 92
5.4.1 Fórmulas de custo esperadas assintoticamente 92
5.4.2 A diferença entre a otimidade local e global 92
5.4.3 A falha de *"Simulated Annealing"* 93
5.5 Uma mudança de representação 94
5.6 O algoritmo evolutivo proposto 96
5.6.1 Representação e espaço de busca 96
5.6.2 Estrutura e tamanho da população 97
5.6.3 O comando **initializePopulation** 98
5.6.4 Os procedimentos de recombinação 98
5.6.5 Os procedimentos de mutação 100
5.6.6 Procedimentos de Busca Local: definindo o comando Local-Search-Engine() 101
5.6.7 RestartPop() 102
5.7 Experimentos Computacionais 102
5.7.1 Instâncias 102
5.7.2 Resultados Finais 103
5.7.3 Resultados com o algoritmo evolutivo proposto 104
5.8 Conclusões 110

6 Complexidade Parametrizada **113**
6.1 Motivação 113
6.2 Definições básicas 113
6.3 Por que a classe FPT é relevante ? 115
6.3.1 Argumento da abundância de problemas naturalmente parametrizados no "mundo real" 115
6.3.2 Um argumento da perspectiva da "falta de progresso" . 117
6.3.3 Um argumento do progresso 118
6.3.4 Um argumento da novidade 119
6.3.5 Intratabilidade como a Hidra de Lerna 119
6.4 Diferentes aproximabilidades de problemas NP-completos . . . 121
6.5 Aproximabilidade e a classe FPT 122
6.5.1 Visão invariante 123
6.5.2 Um *fptas* implica FPT 123
6.5.3 Um *ptas* que também implica FPT 124
6.6 Transformações Parametrizadas e a W-hierarquia 124
6.7 Regras de redução 125

7 Algoritmos Polinomiais de Fusão **129**
7.1 Introdução e Motivação 129
7.2 Algoritmos Polinomias de Fusão e as classes uPMA e PMA . . 131
7.2.1 Recombinação, cegueira e miopia 134
7.2.2 Generalidade da recombinação em MAs 134
7.3 Problemas em PMA 135
7.3.1 Emparelhamento de máxima cardinalidade 135
7.3.2 Tratabilidade a parâmetro fixo e PMA 137
7.3.3 Problemas $W[1]$-completos e a classe PMA 139
7.3.4 Problemas $W[2]$-completos 140
7.3.5 Terceiro circuito hamiltoniano, melhor que o pior 140
7.3.6 O caso do Problema do Caixeiro Viajante 141
7.4 Problemas NP não Aproximáveis e PMA 143
7.5 Na direção de uma definição para a redução em uPMA 145
7.6 Discussões finais e conclusões 152

8 Análise Evolutiva de Algoritmos ? **155**
8.1 Introdução 155
8.2 Estudo de Caso: Algoritmos de Ordenação 156
8.2.1 A base de experimentação escolhida 157
8.2.2 BubbleSort 158
8.2.3 MergeSort 159
8.2.4 ShellSort 160
8.2.5 QuickSort 164
8.3 Conclusões 168

9 Conclusões e Direções Futuras **171**
9.1 Direções Futuras 171
9.1.1 Lições aprendidas 171
9.1.2 Complexidade computacional dos problemas de recombinação 172
9.1.3 Resultados que exploram algoritmos FPT 173
9.1.4 Adição de conhecimento negativo 174
9.1.5 Algoritmos Meméticos Guiados 175
9.1.6 Lógical Modal para MAs e Busca com Crenças 176
9.1.7 Uso de regras de inferência 177
9.1.8 O *MemePool Project*: nosso lugar de encontro para reuso de algoritmos e de conhecimento 181

Bibliografia **183**

Índice Remissivo **215**

Lista de Figuras

1.1 Esta figura, extraída da tese de doutorado de V. Kann [188], mostra os diferentes graus de aproximabilidade de problemas de otimização NP. Essa concepção de "dificuldade" em relação a aproximabilidade e mantida até hoje. 7
1.2 Um grafo $G(V, E)$. Vamos considerar que existe um vértice em todo lugar em que duas arestas são adjacentes num ponto sobre a grade, num total de 80. É possível escolher um subconjunto $V' \subseteq V$ de cardinalidade k, tal que todas as arestas do grafo tenham pelo menos um vértice no extremo que pertence a este subconjunto ? 9
1.3 Um algoritmo 2-approximado para Min Vertex Cover . . . 10
1.4 Uma solução do problema de Min Vertex Cover no mesmo grafo da Fig.1.2 13
1.5 Outra solução factível do problema de Min Vertex Cover para o grafo da Fig.1.2 14
1.6 A solução "descendente" obtida pela recombinação tendo como dado de entrada as duas soluções para Min Vertex Cover mostradas nas Figuras 1.4 e 1.5. 15

2.1 Um algoritmo de busca local 27
2.2 Modelo de um algoritmo de busca baseado em população . . . 31
2.3 Colocando soluções de alta qualidade na população. 32
2.4 O procedimento de criação da nova população. 32
2.5 Pseudocódigo do processo de reinício da população. 34

3.1 Três copias do "gadget" de "OU-exclusivo". O grafo da esquerda só pode ser percorrido nas duas maneiras que são mostradas (centro e direita). 42
3.2 Um procedimento recursivo para resolver o problema de Exact Number Partitioning 50

4.1 O Algoritmo Memético baseado em Busca Local. 57
4.2 Modelo para o nosso MA baseado em Busca Local para o Min ATSP. Nesse nível de generalidade, os detalhes de implementação que levam em conta a estrutura de população, as atualizações, estão ainda ocultos. 58

4.3 Modelo para o algoritmo memético de Merz e Freisleben's para o MIN ATSP. 59
4.4 Modelo para o algoritmo memético de Gorges-Schleuter para o MIN ATSP. 60
4.5 Modelo do algoritmo memético de Walters para o MIN ATSP. 61
4.6 Modelo para o algoritmo memético de Nagata e Kobayashi para o MIN ATSP. 62
4.7 Uma estrutura de população como árvore ternária completa. Os agentes estão organizados hierarquicamente e cada um tem duas soluções factíveis para o problema. 62
4.8 Movimento básico da busca local RAI. 68

5.1 Pseudocódigo para decodificar na representação de Ruml de *"restrições parameterizadas"*. Podemos interpretar a permutação como um *genótipo*, e então *função de crescimento* seria esse pseudocódigo. 95
5.2 Pseudocódigo da implementação de Ruml da meta-heurística de *"Simulated Annealing"*. 95
5.3 Estrutura da população (uma árvore ternária completa) usada em todos os algoritmos "GAs" e MAs deste estudo para o problema de MIN NUMBER PARTITIONING. 97
5.4 O desempenho de *"Simulated Annealing"* (SA), da heurística de Karmarkar-Karp (KK), e do método exato proposto por R. Korf (CKK). Com "nodes" referimos ao número de nós explorados pelo algoritmo de CKK. 104
5.5 O desempenho de quatro tipos de diferentes algoritmos genéticos em comparação com a heurística de Karmarkar-Karp. 105
5.6 O desempenho de GAs e MAs e como ele se comporta em função da representação escolhida. 106
5.7 O desempenho de MAs e GAs: os melhores resultados são obtidos por um algoritmo memético que usa passos de Busca Tabu para diversificar (ME-Dm/100). Um limite de tempo de 120 segundos foi utilizado para poder comparar os algoritmos. . . . 107
5.8 Resultados da heurística de Karmarkar-Karp (KK), de um MA com Busca Tabu para diversificação (MEDm/100), da meta-heurística de *"Simulated Annealing"* de acordo a implementação de Johnson, de uma Busca Tabu com só um agente (TB), e o método "Random Generate-and-Test" de geração aleatória de soluções e avaliação da função objetivo (RGT-TD). 108

6.1 Um grafo e um conjunto dominante com 14 vértices. Por motivos de clareza, só eles são marcados; os outros vértices do grafo correspondem a pontos na grade (42 no total). 126
6.2 O mesmo grafo da Fig. 6.1 e um conjunto dominante que só precisa usar 11 vértices. 127

6.3 A solução descendente. Se preservamos os vértices presentes em ambas as soluções, o problema se reduz a achar um conjunto dominante no grafo composto pelas arestas realçadas e os vértices a elas conectados. A aplicação reiterada da regra de redução discutida acima dá um descendente que só precisa de 9 vértices. 128

7.1 Uma instância do problema ONE-OUT k-FEATURE SET na forma canônica. 147
7.2 Codificando a solução factível $S_2 = \{2, 3, 4\}$ da instância do problema OO-k-FSP da Fig. 7.1 como um conjunto independente dominante num grafo. 148
7.3 Codificando a solução factível $S_1 = \{1, 3, 5\}$ da instância do problema OO-k-FSP da Fig. 7.1 como um conjunto independente dominante num grafo. 150
7.4 No grafo é mostrado um conjunto dominante de vértices que representa a solução factível $S_3 = \{2, 5\}$ da instância do problema de OO-k-FSP da Fig. 7.1. 151

8.1 Pseudocódigo do algoritmo de *BubbleSort* usado nas experiências computacionais. 159
8.2 Variação da função de aptidão em relação aos formae para o problema de *BubbleSort*-ing. 160
8.3 Resultados obtidos ao aplicar o EAA a *BubbleSort*. 161
8.4 Pseudocódigo do algoritmo de *MergeSort* usado nas experiências computacionais. 162
8.5 Variação da função de aptidão em relação aos formae para o problema de *MergeSort*-ing. 163
8.6 Resultados obtidos por EAA aplicado a *MergeSort*. 163
8.7 Pseudocódigo do algoritmo de *ShellSort* usado nas experiências computacionais. 164
8.8 Variação da função de aptidão em relação aos formae para o problema de *ShellSort*-ing. 165
8.9 Resultados obtidos por EAA aplicado a *ShellSort*. Os quadrados representam os dados experimentais, a linha tracejada é um ajuste a uma função do tipo $an^2 + bn$, e a linha sólida e um ajuste do tipo an^b. 166
8.10 Análise da tendência para os dados obtidos por EAA no caso de *ShellSort*. Uma interpolação linear do quociente entre os valores obtidos no experimento e algumas funções de teste. 166
8.11 Pseudocódigo do algoritmo de *QuickSort* usado nas experiências computacionais. 167
8.12 Variação da função de aptidão em relação aos formae para o problema de *QuickSort*-ing. 168
8.13 Resultados obtidos por EAA aplicado a *QuickSort*. Os quadrados representam os dados experimentais, a linha tracejada é um ajuste a $an^2 + bn$, e a linha sólida e um ajuste a $an^2 + bn$, porém restrito a $n \leq 50$. 169

8.14 Evolução do coeficiente quadrático na análise dos dados experimentais obtidos pelo EAA para *QuickSort*. 169

Lista de Tabelas

4.1 Comparação de quatro tipos diferentes de operadores de recombinação para o MIN ATSP. 71
4.2 Comparação do desempenho de nosso método com os de Gorges-Schleuter, Merz e Freisleben, e de Nagata e Kobayashi, usando o mesmo conjunto de instâncias assimétricas. 73
4.3 Comparação do desempenho de nosso método com os de Walters, Stützle e Dorigo, e Amin usando o mesmo conjunto de instâncias assimétricas. 74
4.4 O desempenho de nosso método em todas as instâncias assimétricas disponiveis na TSPLIB. 76
4.5 O desempenho de nosso método em um conjunto de instâncias assimétricas, com solução ótima conhecida, geradas de instâncias simétricas já resolvidas a otimalidade e disponiveis na TSPLIB, para $K_{max} = 5\%$. 77
4.6 O desempenho de nosso método em um conjunto de instâncias assimétricas, com solução ótima conhecida, geradas a partir de instâncias simétricas já resolvidas a otimalidade e disponíveis na TSPLIB, para $K_{max} = 50\%$. 78
4.7 Comparação com Busca Predatória em instâncias simétricas da TSPLIB. 79
4.8 Comparação com Busca Tabu Incremental em instâncias simétricas da TSPLIB. 80
4.9 Comparação com otimização microcanônica em instâncias simétricas da TSPLIB. 80
4.10 Comparação com Merz e Freisleben em instâncias simétricas euclidianas no plano da TSPLIB. 81
4.11 O desempenho de nosso método em um conjunto de instâncias euclidianas disponíveis na TSPLIB. 82

5.1 Uma execução-exemplo da heurística de Karmarkar-Karp. . . . 91
5.2 Comportamentos diferentes utilizados na recombinação entre líder e subordinados. 99
5.3 A notação usada nos resultados experimentais mostrados na Figura 5.5. 105
5.4 O desempenho da meta-heurística de *"Simulated Annealing"* em contraste com a heurística construtiva de Karmarkar-Karp. . . 109

5.5 O desempenho de todos os métodos (para o mesmo conjunto de instâncias). O algoritmo memético que usa Busca Tabu para diversificação domina todos eles, mas não alcança a solução ótima em todos os casos. 109

Capítulo 1

Introdução

1.1 Motivação

Há uma questão central que motiva este trabalho e sua solução é um dos objetivos principais desta tese:

> *Existe alguma maneira de encontrar algoritmos eficientes baseados em princípios de busca evolutiva que sempre fornecem boas aproximações a problemas de otimização para os quais é difícil de encontrar a solução ótima ?*

Desde o começo sabíamos, de uma certa perspectiva, que provavelmente a resposta a esta pergunta é "Não", desde que, sob a suposição comumente acreditada de que P $\neq$ NP, sabemos que há alguns problemas para os quais não pode obter-se soluções aproximadas com algoritmos eficientes [225]. Por outro lado, esta questão tem alguma relatividade pois pode aceitar por resposta: *"depende do problema"*. Sabemos que para alguns, apesar de serem difíceis de se resolver otimamente, pode-se muito bem obter soluções com alguma garantia de desempenho usando algoritmos eficientes. Isto deixa algum espaço para a possibilidade de que alguns problemas podem ser abordados de forma eficaz baseando-se em algoritmos que usam técnicas evolutivas, enquanto para outros não é possível. O propósito da tese, nesta perspectiva, é entender por que aparentemente alguns problemas naturalmente se ajustam ao método heurístico de resolução proposto pela Computação Evolutiva, enquanto outros são mais difíceis de serem adaptados a esta metodologia.

A literatura da Ciência da Computação está atualmente ocupada com muitas referências a *"Algoritmos Genéticos"*, *"Algoritmos Meméticos"*, *"Algoritmos Genéticos Paralelos"*, *"Algoritmos Genéticos Híbridos"*, *"Busca Evolutiva Lamarkiana"*, *"Programação Genética"*, *"Programação Evolutiva"*, *"Busca Evolutiva"*, *"Estratégias Evolutivas"*, *"Computação Evolutiva"*, etc., todos os termos que em geral referem-se a heurísticas para problemas de Otimização Combinatória que são NP-completos ou NP-difíceis. Lentamente, iniciando-se em torno de 1985, depois do Primeiro Congresso Internacional em Algoritmos

Genéticos, uma onda ascendente de interesse tem ganho espaço entre investigadores que trabalham em aplicações e que tem que fornecer uma meta-heurística rápida para os seus problemas de interesse. Este aumento de interesse é evidentemente claro no número de publicações científicas que têm "algoritmos genéticos", ou qualquer uma das frases-chave listadas acima, em seus resumos. Em relação ao uso atual de meta-heurísticas, muitos cientistas vêem um pouco de semelhanças com a situação antes de 1970 quando o algoritmo de Simplex para Programação Linear e técnicas de *"Branch-and-Bound"* eram amplamente utilizadas na prática, mas a teoria de NP-completude ainda não tinha sido desenvolvida.

1.1.1 Metodologia: Três aspectos da questão central

Para estudar a questão apresentada acima, identificamos três direções de pesquisa complementares:

- Identificar problemas de otimização NP para os quais o paradigma da Busca Evolutiva tem provado não ser competitivo contra as melhores heurísticas ou algoritmos de aproximação conhecidos para estes problemas.
- Identificar quais problemas de otimização podem ser abordados usando uma Busca Evolutiva e identificar suas razões.
- Para os problemas do grupo mencionado acima, será importante encontrar ligações com a teoria da complexidade computacional e as classes de complexidade (referente à aproximabilidade) a que estes problemas pertencem.

Concentraremos a discussão agora em dois problemas que são candidatos naturais para o estudo desde a partir dessa perspectiva: o Problema do Caixeiro Viajante conhecido na lingua inglesa como TRAVELING SALESMAN PROBLEM, ou simplesmente pela sigla TSP, e sua versão de otimização, o MIN TSP (um problema clássico de Pesquisa Operacional, Teoria dos Grafos e Ciência da Computação); também discutiremos um problema conhecido como MIN NUMBER PARTITIONING. Este último é um problema com fortes conexões com *Scheduling*, *Bin-Packing*, criptografia, etc. Uma revisão ampla dos resultados recentes neste problema foi primeiro publicada em [275]. No Capítulo 5, esta revisão foi estendida para também incorporar os últimos desenvolvimentos e resultados teóricos.

Para o MIN TSP, provavelmente a melhor heurística conhecida de "agente único" é *"Iterated Lin-Kerninghan"*, uma heurística que incorpora aleatoriedade e uma poderosa busca local. Para MIN NUMBER PARTITIONING, a melhor heurística construtiva conhecida é o *"Método das Diferenças"* de Karmarkar e Karp. Ambos os problemas têm sido objeto de análise de meta-heurísticas baseadas em metodologias populacionais, mas o MIN TSP tem sido também estudado em conexão com *Algoritmos Meméticos*, uma heurística populacional que mistura um (ou vários) métodos de otimização rápidos de um agente único com o uso de operadores de *recombinação de soluções*. No início do trabalho

desta tese, ou melhor, quando o projeto proposto foi formulado, a metodologia memética provou ser ordens de magnitude melhor que algoritmo genético para instâncias planares de MIN EUCLIDEAN TSP [323], mas estava um pouco longe do desempenho de *"Iterated Lin-Kerninghan"* quando o tamanho das instâncias crescia. A situação mudou consideravelmente. No momento de término desta tese existem pelo menos três grupos de pesquisa no mundo que mostram, com resultados computacionais, a vantagem dos algoritmos meméticos em relação à busca de agente único proposto por *Iterated Lin-Kernighan*. Em contraste, também foram desenvolvidas estratégias de busca evolutivas para MIN NUMBER PARTITIONING mas os resultados destes métodos foram significativamente piores que outros métodos, pelo menos para algumas das instâncias que geralmente são usadas na literatura para avaliar empiricamente as heurísticas.

Em relação à terceira direção de pesquisa que apontamos, podemos mostrar uma diferença clara entre estes dois problemas. Alguns problemas são NP-completos somente se valores numéricos exponencialmente grandes são permitidos na entrada e se tornam polinomialmente solúveis sempre que as variaveis inteiras nos dados de entrada sejam polinomiais no tamanho da entrada. Tais problemas são chamados de solúveis em *tempo pseudo-polinomial*. O MIN NUMBER PARTITIONING pertence a esta classe de problemas. A classe complementar de problemas é feita de problemas cuja NP-completude não depende da presença de valores numéricos exponencialmente grandes. A versão de decisão do TSP é um desses problemas, os quais são chamados de *fortemente* NP-*completos*. Embora algumas implementações de algoritmos genéticos tenham sido inventadas para ambos os problemas, parece que a situação é mais difícil para as meta-heurísticas no caso de MIN NUMBER PARTITIONING. Então parece natural investigar se problemas de otimização NP-difícies que podem ser resolvidos em tempo pseudo-polinomial têm uma classe de instâncias que são particularmente "difíceis" (no sentido prático) para algoritmos evolutivos.

Agora vamos à discutir a segunda direção de pesquisa apontada. Investigadores que ainda desenvolvem heurísticas evolutivas não têm um modo, nem mesmo uma dica heurística para uma teoria sobre como responder isto. Apesar disso, abordagens de sucesso baseadas no uso de operadores de *"reprodução"*, *"recombinação"*, ou *"combinação"* sejam geralmente apoiadas por uma codificação adequada das instâncias e o uso de funções de *aptidão* apropriadas, i.e. um mapeamento da função objetivo (fixada pela entrada, uma intocável "medida de mérito" associada com o problema) para valores reais e inteiros. As boas codificações são geralmente aquelas para as quais os efeitos de "rompimento" (em valores de aptidão) induzidos pela recombinação não são significativos em comparação com os valores de aptidão das configurações dos pais [271] [322] [255] [77] [86]. Às vezes, estas codificações saíram de uma percepção que embute as instâncias em algum espaço métrico.

1.1.2 A expressão "fácil de resolver"

É necessário traduzir a expressão cotidiana "fácil de resolver" para um contexto matemático formal. Por essa razão, o conceito de *algoritmos de tempo polinomial* foi introduzido. A definição é a seguinte:

> *"Um algoritmo é dito executar em tempo polinomial se existe um polinômio p tal que o algoritmo aplicado para uma entrada de tamanho n sempre encontra a solução do problema em tempo $p(n)$, isto é, depois de efetuar $p(n)$ instruções simples."*

Podemos comentar alguns fatos: Primeiro, medimos a *complexidade de pior caso*, isto é, o tempo no qual estamos certos que o algoritmo termina *considerando a entrada de tamanho n que colocamos*. Segundo, a "complexidade subjetiva" que mede o esforço de realmente codificar o algoritmo não é relevante nesta definição. Terceiro, a definição é útil porque o tempo de execução de um algoritmo de tempo polinomial cresce bastante lentamente ao aumentar o tamanho da instância, se o comparamos com um algoritmo de crescimento exponencial. Isso é o que torna este segundo tipo inútil, só de interesse prático para instâncias pequenas de problemas. A ordem do polinômio então classifica os algoritmos dentro de uma classe de tempo polinomial. Quarto, a quantidade de memória necessária no cenário de pior caso também não é preocupação, ou diríamos, uma preocupação *separada*.

Como Papadimitriou reconheceu em seu livro "Computational Complexity", *"o argumento para a escolha dos algoritmos polinomiais como conceito matemático que captura a noção informal de 'computação eficiente prática' é aberto a críticas de todos os lados"* [307]. E no mesmo parágrafo ele observa:

> *"Adotar o desempenho de pior caso polinomial como nosso critério de eficiência tem por resultado uma teoria útil e elegante que diz alguma coisa relevante sobre a computação prática, o que seria impossível sem esta simplificação".*

1.1.3 Dificuldade de problemas e reduções

Uma das maneiras mais aceitas para provar que um problema é difícil é prová-lo ser NP-completo. Se um problema de otimização é NP-completo estamos quase certos que não pode ser resolvido otimamente em tempo polinomial.

Se temos um algoritmo que resolve um problema, podemos freqüentemente usá-lo para resolver outros problemas semelhantes. Esta é uma idéia poderosa, usada extensivamente em Teoria da Computação, contudo raramente é aplicada em trabalhos práticos. Queremos acentuar este ponto; dados dois problemas A e B, podemos especificar antecipadamente como usar qualquer algoritmo para o problema B para resolver A. Tal especificação é chamada de uma *redução* de A a B. Se somos capazes de provar que o processo é correto, é dito que *reduzimos* A para B (ou melhor, uma instância genérica do problema A para uma instância particular do problema B). Se a execução em si é *"fácil"* de executar, i.e. o tempo de execução dos algoritmos implementados para a redução rodam em tempo polinomial, então mostramos que o problema B é *no mínimo tão difícil* de resolver como A, no sentido que se podemos resolver B podemos também resolver A em torno do mesmo tempo. De certo modo, uma redução é uma forma de comparação entre problemas e, como conseqüência, reduções são freqüentemente simbolizadas com uma desigualdade $A \leq B$.

Nos anos 70 e 80 muitos *problemas de decisão* (problemas para os quais somente se aceita a resposta do tipo *"Sim"* ou *"Não"*) foram provados serem redutíveis entre si. Todos estes problemas têm uma propriedade em comum: para cada dado de entrada para um problema com uma solução *"Sim"* há uma prova que o dado de entrada tem solução *"Sim"* e esta prova pode ser *verificada* (i.e. checada que é verdade) em tempo polinomial. Qualquer problema com esta propriedade é chamado de um problema NP. A sigla deriva da expressão em lingua inglesa *"non-deterministic polynomial*, e dizer, se uma solução é dada, pode-se verificar a validade em tempo polinomial no tamanho da instância do problema. Dizemos isto para problemas de decisão, mas é fácil deduzir que todos os problemas que podem ser resolvidos através de algoritmos de tempo polinomial satisfazem esta propriedade. Dizemos que $P \subseteq NP$ onde P é o conjunto de problemas que podem ser *resolvidos* por algoritmos de tempo polinomial e NP é o conjunto de problemas NP.

Um problema que pode ser dito, em um certo sentido, ser "maior" ou "mais difícil" que todos os problemas NP, é um problema para o qual todo problema em NP pode ser reduzido, é chamado NP-difícil. Se um problema é provado NP-difícil e também é provado que é um problema da classe NP então ele é chamado de NP-completo. Assim, todos os problemas NP-completos são igualmente difíceis de resolver, pois são irredutíveis. Se há um algoritmo de tempo polinomial para qualquer problema NP-completo então $P = NP$ e todo problema NP pode ser resolvido em tempo polinomial. Apesar de esforços enormes, a pergunta se $P = NP$ ainda não foi respondida. A convicção geral hoje em dia é que $P \neq NP$ e uma grande parte dos pesquisadores em Teoria da Ciência da Computação, tem $P \neq NP$ como uma suposição fundamental.

1.1.4 Aproximabilidade de problemas de otimização NP

A teoria de NP-completude foi o instrumento final que fez a Teoria da Computação estabelecer um de seus paradigmas de pesquisa científica mais influentes. A teoria se tornou uma espécie de "ponte", uma ligação entre Ciência da Computação e outras áreas científicas. Em um recente relatório [5] de alguns dos mais distintos membros em Teoria em Ciência da Computação podemos ler:

> *"Ainda mais, a (teoria de)* NP*-completude tem se tornado uma espécie de "embaixador cultural" de Ciência da Computação para outras áreas, tanto que o conceito é utilizado como evidência indireta de complexidade em contextos extremamente diversos (uma busca bibliográfica encontra aproximadamente 6.000 artigos em conferências e jornais todo ano, em campos que vão desde ciência política a biologia matemática e engenharia de potência, têm 'NP-completo' entre suas palavras chave)."*

É totalmente reconhecido que provar que um certo problema novo é NP-completo é certamente um importante objetivo e um indicativo de que os esforços em pesquisa deveriam ser redirecionados para outras novas metas como inventar algoritmos aproximados, ou, como fazemos, usar métodos evolutivos

para atacá-los e tentar resolvê-los na prática. Uma prova de NP-dificuldade e/ou NP-completude é geralmente vista por muitos pesquisadores (incluindo o autor desta tese) como um sinal verde; heurísticas são conseqüentemente desenvolvidas sem qualquer tipo de culpa.

A teoria de aproximabilidade tem atualmente alcançado o status de "bem-desenvolvida" com muitos desenvolvimentos recentes em dificuldade de aproximação e muitas aplicações. Para uma ampla visão dos muitos aspectos deste assunto nós sugerimos ao leitor os recentes livros [167] e [15]. Há, entretanto, certas insatisfações com o atual estado da área. Para conhecer o poder, e algumas das fraquezas, da teoria necessitaremos uma revisão da área.

Um problema de maximização, que genericamente denotamos como MAX P, é *aproximável em* $1+\rho$ se existe um algoritmo de tempo polinomial A tal que, para todo dado de entrada x

$$\frac{opt_{MaxP}(x)}{A(x)} \leq 1+\rho. \tag{1.1}$$

Para problemas de minimização consideramos a razão inversa. Por $A(x)$ denotamos o valor da função objetivo dado pelo algoritmo A quando este recebe como dado de entrada a instância x e $opt_{MaxP}(x)$ denota o valor ótimo com o dado de entrada sendo a instância x. A notação $1+\rho$ sugere o fato de que quanto mais próximo estamos de 1, melhor é o algoritmo de aproximação. Esta definição coloca uma nova questão em complexidade computacional: *"Dado um problema de otimização* MAX P*, qual é o menor ρ para o qual* MAX P *é $(1+\rho)$-aproximável em tempo polinomial ?"* Esta questão chamou a atenção de muitos investigadores teóricos. A relevância para o trabalho prático é discutível, mas ajuda a caracterizar tipos diferentes de problemas dentro da mesma classe de complexidade computational. Por exemplo, problemas de otimização NP basicamente apresentam os seguintes tipos de comportamento

- Existem problemas que não são aproximáveis dentro de qualquer ρ a menos que P = NP. Este é o caso do Problema do Caixeiro Viajante (MIN TSP) e Programação Inteira.
- Outros problemas são aproximáveis dentro de *alguma* constante ρ. A classe destes problemas é chamada de APX. Por exemplo, MAX CUT é aproximável em 2.
- Outra classe de problemas são os que são aproximáveis dentro de *todo* valor de ρ. Em outras palavras, isto significa que para cada ρ existe um algoritmo de aproximação A_ρ em $(1+\rho)$. A seqüência infinita $\{A_\rho : \rho > 0\}$ é chamada de um *"polynomial time approximation scheme" (ptas)*, que traduzimos como *esquema de aproximação de tempo polinomial*. A classe de problemas que admite tal esquema é chamada de PTAS. Exemplos de problemas em PTAS são MAX INDEPENDENT SET para grafos planares e encontrar uma programação de tarefas de duração total mínima em máquinas paralelas. Observamos que cada A_ρ deve ser executado em tempo polinomial em relação ao tamanho do dado de entrada, mas poderia depender exponencialmente (ou mesmo ser pior) em $1/\rho$. Essa classe será discutida posteriormente em na Seção 3.3.1.

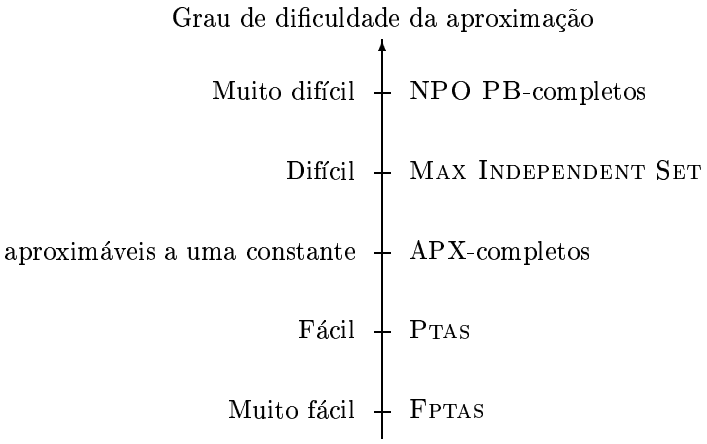

Figura 1.1: *Esta figura, extraída da tese de doutorado de V. Kann [188], mostra os diferentes graus de aproximabilidade de problemas de otimização* NP. *Essa concepção de "dificuldade" em relação a aproximabilidade e mantida até hoje.*

- O último problema mencionado acima é resolvido se podemos encontrar um *"fully polynomial time approximation scheme" (fptas)* que traduzimos como *esquema de aproximação de tempo completamente polinomial*, que é um *ptas* tal que o tempo de execução de cada A_ρ é polinomial em relação ao tamanho de x e $1/\rho$. A classe resultante é denotada como FPTAS. Por exemplo, o problema Max Knapsack está em FPTAS. Essa classe será discutida posteriormente em na Seção 3.3.3

A teoria de algoritmos de aproximação tenta colocar problemas nestas classes provando limitantes superiores, i.e. a existência de um algoritmo de aproximação ou esquema e limitantes inferiores, mostrando que um problema não é aproximável dentro de uma certa precisão.

Um exemplo prático de um problema NP-completo é o *Problema do Caixeiro Viajante* ou Tsp. Uma vendedora[1] tem que visitar n cidades conhecendo antecipadamente as distâncias entre cada par de cidades. Ela quer encontrar um circuito que passe por todas as n cidades, começando e terminando na mesma cidade (um circuito fechado, que também pode ser interpretado como um ciclo hamiltoniano em um grafo completo com pesos nas arestas), minimizando a distância viajada. Nesta formulação, o Tsp ainda não é um problema de *decisão* dado que é um problema de otimização; a resposta é obter uma rota ótima ao invés de *"Sim"* ou *"Não"*. Entretanto, é fácil de formulá-lo como um problema de decisão introduzindo um novo parâmetro B como dado de entrada. A questão agora é: *há uma rota de comprimento no máximo B que passe por todas as cidades ?*

[1] Ou um vendedor, sem perda de generalidade.

O TSP foi dos primeiros problemas a ser provado como sendo NP-completo. Não existe algoritmo de tempo polinomial conhecido que possa resolvê-lo para cada valor B, usualmente referido como, *"budget"* que podemos traduzir como *orçamento*. Infelizmente, até mesmo várias restrições naturais do TSP também dão origem a problemas NP-completos existindo uma prova formal da intratabilidade, por exemplo, quando as distâncias satisfazem à desigualdade triangular.

Qualquer problema de otimização que de modo similar ao TSP possa ser visto como um problema de decisão NP é um problema de otimização NP que denotamos como NPO. Se P $\neq$ NP não há algoritmo que encontre a solução ótima de um problema NPO em tempo polinomial.

1.2 A tarefa de fornecer soluções aproximadas

Existem vários argumentos que podem ser invocados para mostrar que, para algumas aplicações, freqüentemente é suficiente encontrar uma solução *aproximada*, i.e. uma solução *factível* (para o TSP isto significa qualquer tour, não necessariamente o tour ótimo), e existem algumas circunstâncias que permitem fazer isto rápidamente. Além disso, para alguns problemas NPO mencionamos a existência de *algoritmos de aproximação de tempo polinomial* que podem ser utilizados quando a ordem do polinômio os permite ser de utilidade prática. Entretanto, e infelizmente, a aproximabilidade de diferentes problemas NPO difere enormemente.

Por exemplo, o TSP com desiqualdade triangular (i.e. as distâncias entre cada três cidades satisfazem a desigualdade triangular) pode ser resolvido aproximadamente em tempo polinomial dentro de um fator 3/2, i.e. pode-se encontrar uma rota de no máximo 3/2 vezes a rota mais curta possível [72]. Discutiremos em detalhes o algoritmo na Seção 3.2.7. Porém, o TSP geral não pode ser aproximado dentro de qualquer fator se P $\neq$ NP [131]. Outro exemplo é o Problema da Mochila, que é NP-completo mas tem um *ptas*, permitindo assim ser aproximado dentro de cada constante em tempo polinomial [175]. Além disso, o problema MAX KNAPSACK tem de fato um *fptas*.

Em princípio, poderia ser razoável acreditar que, desde que todos os problemas de otimização NP-completos são redutíveis entre si, um algoritmo de aproximação para um problema automaticamente, via redução, daria igualmente um bom algoritmo de aproximação para outros problemas. Usando palavras de Edmonds (em um contexto diferente) podemos dizer: *"é uma legítima possibilidade matemática"*. Infelizmente, este não é o caso. A razão para isto é que as reduções NP não são fortes o suficiente para preservar toda a estrutura subjacente da versão de otimização dos problemas. Muitas tentativas têm sido feitas para encontrar uma teoria que explique porquê um problema aproveita certas propriedades particulares da aproximação (veja [42] para uma revisão e [167] para um tratamento mais recente).

1.2.1 Exemplo: Aproximabilidade de Min Vertex Cover

Discutiremos agora um problema de otimização que será o mais representativo em diversos conceitos e conclusões desta tese. Vamos supor que alguém nos

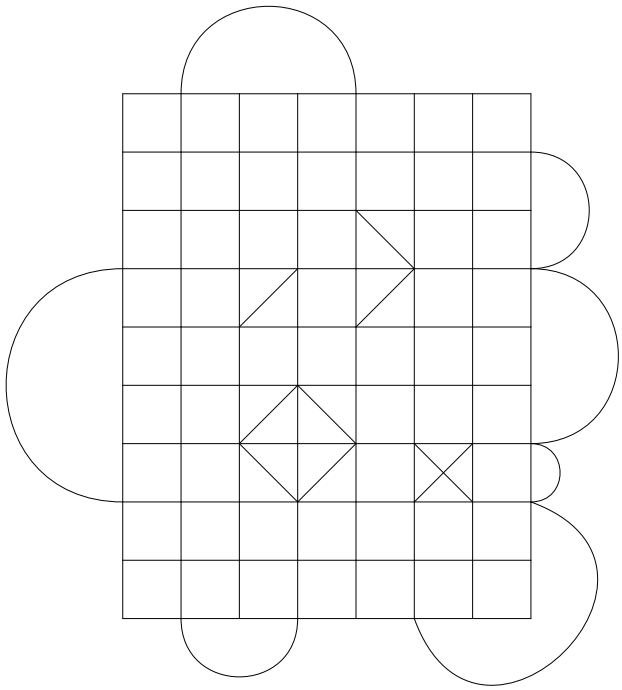

Figura 1.2: *Um grafo $G(V, E)$. Vamos considerar que existe um vértice em todo lugar em que duas arestas são adjacentes num ponto sobre a grade, num total de 80. É possível escolher um subconjunto $V' \subseteq V$ de cardinalidade k, tal que todas as arestas do grafo tenham pelo menos um vértice no extremo que pertence a este subconjunto ?*

fornece o grafo $G(V, E)$ mostrado na Fig. 1.2. Para simplificar a apresentação, desenhamos o grafo no plano tal que só temos vértices em cada ponto da grade e as linhas mostram as arestas entre os vértices. Nenhuma aresta de G está superposta neste esquema, embora, inevitavelmente, haja um par de arestas que se cruzam.

Uma *cobertura de vértices* de um grafo $G(V, E)$ é um subconjunto de vértices de G ($C \subseteq V$) tal que cada aresta em E que conecta os vértices a e b tem $a \in C$ ou $b \in C$, ou ambos. Uma cobertura de vértices C é dita ser *minimal*, se eliminando qualquer vértice de C o resultado é um conjunto que agora não é uma cobertura de vértices. Dado um grafo, podemos construir uma cobertura de vértices minimal em tempo polinomial, mas quão difícil é encontrar a cobertura de cardinalidade mínima ?

Devido a ser possível mostrar que o problema MIN VERTEX COVER é NP-difícil, nossas esperanças de resolver o problema em tempo polinomial são remotas. Este problema tem um grande número de aplicações e é considerado um dos seis problemas NP-completos *essenciais*.

Mostraremos agora um método construtivo que é extremamente simples já que fornece um algoritmo de 2-aproximado para este problema (veja Fig. 1.3). O algoritmo é dito ser *randomizado*, pois ele começa com uma aresta esco-

```
Procedure 2-approx-algo-for-Min-Vertex-Cover (G(V, E))
begin
    C = ∅; (* the cover is initially empty *)
    while (there is an edge left in G) do{
        choose any edge (u, v);
        add u and v to C, and delete them from G;
    }
    return C;
end
```

Figura 1.3: *Um algoritmo 2-approximado para* MIN VERTEX COVER

lhida aleatoriamente. En seguida, o algoritmo insere os vértices dos extremos da aresta em C, que é inicialmente vazio, e elimina os dois vértices extremos e todas as arestas que estão tocando-os (desde que já estão cobertos por eles, não há mais preocupação com essas arestas). Executando este procedimento randomizado básico novamente até que não haja arestas remanescentes, o conjunto final C é uma cobertura e sua cardinalidade $|C|$ é exatamente duas vezes o número de arestas escolhidas. Note, entretanto, que nenhuma das arestas escolhidas tem um vértice em comum (devido à maneira em que foram escolhidas). Isto constitui um *emparelhamento*. Agora observamos que *qualquer* cobertura de vértices, *incluindo a ótima*, deve conter *no mínimo* um vértice de cada uma destas arestas; caso contrário, a aresta não seria coberta. Sendo este o caso, se $|C_{opt}|$ é a cardinalidade da cobertura ótima de vértices $|C| \leq 2|C_{opt}|$. Então

$$\frac{|C|}{|C_{opt}|} \leq 1 + \rho,$$

que significa que temos um algoritmo 2-aproximado, isto é um algoritmo de aproximação com $\rho = 1$.

A simplicidade do argumento contrasta com a falta de melhora para este problema. Esse simples algoritmo de aproximação se manteve por muitos anos como o melhor conhecido, com um valor alto, $\rho = 1$ de acordo com nossa definição prévia. Alguns anos atrás foi provado que *não há algoritmo de aproximação para* MIN VERTEX COVER *para qualquer* $\rho < 1/6$. Retornaremos a discutir este problema em seções futuras da tese, visto que é representativo do sucesso e dos fracassos de diversas abordagens.

1.3 Os diferentes graus de aproximabilidade: A classe APX

Em 1988, Papadimitriou e Yannakakis definiram a classe de complexidade computacional que eles chamaram de MAX SNP mais conhecida hoje por APX, e um novo tipo de redução, a *L-redução*. A principal idéia, que motivou a

introdução desta nova redução, é preservar a aproximabilidade dentro de constantes [309, 310].

O que é provavelmente mais interessante é que a classe MAX SNP contém os problemas que podem ser definidos *sintaticamente* de uma certa maneira. Com isto, Papadimitriou e Yannakakis conseguiram caracterizar muitos problemas NPO polinomialmente aproximáveis *de modo não computacional*, dado que *cada problema* em MAX SNP pode ser aproximado dentro de uma constante em tempo polinomial.

Como é tradicional em complexidade computacional, eles chamaram um problema em MAX SNP, para o qual cada problema em MAX SNP pode ser reduzido, MAX SNP-*completo*. Eles também provaram que diversos problemas NPO são MAX SNP-completos; podemos citar o problema MAX 3-SATISFIABILITY (MAX 3SAT) como um exemplo. Se tivermos uma fórmula booleana na forma normal conjuntiva (uma conjunção de cláusulas onde cada cláusula é uma disjunção de no máximo três literais), MAX 3SAT é o problema de otimização que visa encontrar uma designação de valores verdade para os literais que satisfaz o número máximo de cláusulas.

A *P-redução*, introduzida posteriormente, generalizou a L-redução. Entretanto, em ambos os casos, estas reduções são suficientemente fortes para preservar aproximabilidade dentro de constantes. O significado disto é que se o problema A P-reduz a B, e B tem um *ptas* então A tem também um *ptas*. Em essência, todos os problemas MAX SNP-completos são igualmente difíceis de aproximar: ou todos têm um *ptas* ou nenhum.

Obviamente, havia um forte interesse em saber se problemas MAX SNP-completos poderiam ser aproximados dentro de cada constante. Entretanto, isto foi provado não ser possível (devido a P $\neq$ NP) [12]. Provar resultados de MAX SNP-completude é importante, visto que fornecem uma boa caracterização da aproximabilidade do problema. Isto que dizer que pode ser aproximado dentro de uma constante, mas não para toda constante.

1.3.1 Problemas que são mais difíceis para aproximar

Como já mostramos na Fig. 1.1.4, há problemas mais difíceis que os pertencentes à classe MAX SNP. Um deles é o problema de achar o maior conjunto independente de vértices de um grafo (MAX IND SET), e discutiremos este problema nos últimos capítulos desta tese. A tarefa é então achar o subconjunto de vértices de maior cardinalidade tal que nenhum par de vértices no conjunto é conectado com uma aresta do grafo. Existe uma constante c tal que MAX IND SET não pode ser aproximado dentro de $|V|^c$, onde $|V|$ é o número de vértices no grafo.

A P-redução é uma ferramenta muito importante, pois dado um problema de aproximabilidade desconhecida, se é possível encontrar uma P-redução de MAX IND SET para este problema, é provado que o problema é difícil de resolver.

O problema LONGEST INDUCED PATH IN A GRAPH (LIP) tem uma propriedade que diz que cada problema NPO com um valor ótimo não muito grande (mais formalmente, se o valor do ótimo pode ser limitado por um polinômio em relação ao tamanho do dado de entrada) pode ser P-reduzido a ele. Recorrendo

à aproximabilidade, este fato coloca este problema entre os mais difíceis dos problemas NPO polinomialmente limitados. Podemos dizer que é um problema NPO PB-completo. Assim este problema é até mais difícil de aproximar que o MAX INDEPENDENT SET.

1.4 Aproximação usando tipos particulares de heurísticas

É importante de ressaltar que a definição de aproximabilidade do problema é *indiferente* ao tipo particular de algoritmo que é usado para retornar uma solução factível. Obviamente, uma área importante de pesquisa é aquela de aproximação com outros tipos de algoritmos, além dos algoritmos de tempo polinomial seqüenciais.

Como exemplo, uma alternativa ao processamento seqüencial é o processamento *paralelo*. Sentimos que não há nenhuma necessidade de acentuar a enorme importância que a computação paralela tem para resolver amplos problemas de otimização. Infelizmente, há relativamente poucos resultados em aproximação paralela de problemas NPO[314] [382] [342] [341] [361]. Outras referências podem ser achadas em [167].

1.4.1 Aproximação por busca local

Para o propósito desta tese, uma atenção particular foi dada ao campo de aproximação por *otimização local*.

Resumidamente, um *algoritmo de busca local* para um problema de otimização tenta em cada passo encontrar soluções melhores considerando *"perturbações"* ou *"pequenas modificações"* na atual solução factível. Esta é certamente uma explicação intuitiva, apenas para dar fundamento à discussão, uma definição formal é dada na seção Sec. 2.2.3. Um certo algoritmo é encarregado de gerar estas novas soluções, e se não for possível fornecer uma solução melhor, informa como resultado a solução que havia no início, que é dita ser *localmente ótima*.

É importante observar que algoritmos baseados em busca local, a abordagem *"não sistemática"* de maior sucesso, trabalha com problemas de otimização combinatorial. Com "não sistemática" queremos dizer que, embora tenhamos diversos argumentos heurísticos que guiam o processo de projetá-los, ainda falta uma teoria geral que pode ajudar a predizer o resultado. Eles ainda constituem uma forma de "arte oculta", onde a experiência passada parece ter um papel muito significativo, quase como as habilidades desenvolvidas para a construção de mecanismos para provar reduções e a dificuldade de problemas.

Para os teóricos em ciência da computação são desafios extraordinários. Por exemplo:

- Por que eles funcionam tão bem na prática?
- Quão boa é a solução ótima local em relação à solução ótima ?

- É possível provar que, para um determinado algoritmo de busca local, podemos encontrar ótimos locais em tempo polinomial ?

Em 1988, Johnson, Papadimitriou e Yannakakis definiram uma classe de complexidade chamada de PLS relacionada à busca local, e também definiram problemas completos para esta classe [183]. Durante a última década, diversos problemas têm sido mostrados serem PLS-completos. Inclusive, tem sido conjeturado que talvez a PLS-completude é, em algum senso, o comportamento normal ou "natural" de problemas NP-completos [180] [211] [379] [17] [8] e em particular o excelente "survey" de M. Yannakakis [381] Outras referências podem ser achadas em [167]. "Non-oblivious local-search" e discutida por Khanna e Motwani na Ref. [198].

1.5 Recombinação

Como o paradigma de computação evolutiva trata com problemas de otimização, por exemplo, como o MIN VERTEX COVER mostrado na Seção 1.2.1, se um dos procedimentos básicos que é mais comumente usado é a *recombinação* de soluções factíveis. Por exemplo, vamos supor que através de um algoritmo randomizado construtivo temos duas soluções como as mostradas nas Figs. 1.4 e 1.5.

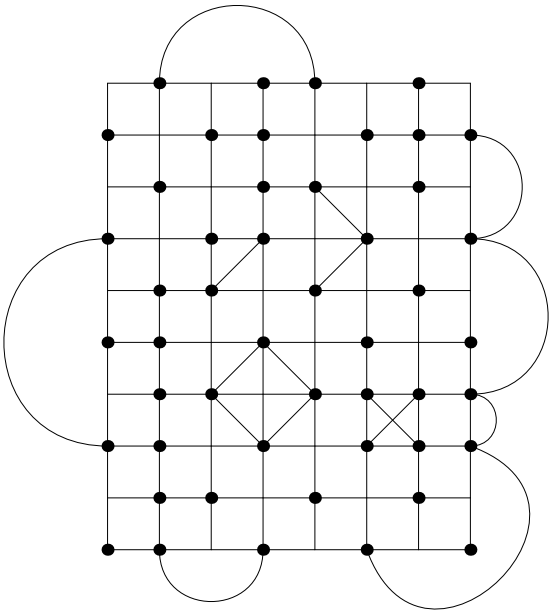

Figura 1.4: *Uma solução do problema de* MIN VERTEX COVER *no mesmo grafo da Fig. 1.2*

Convidamos o leitor a conferir que os vértices marcados de fato são uma cobertura de vértices do grafo (relembramos que há um vértice em cada inter-

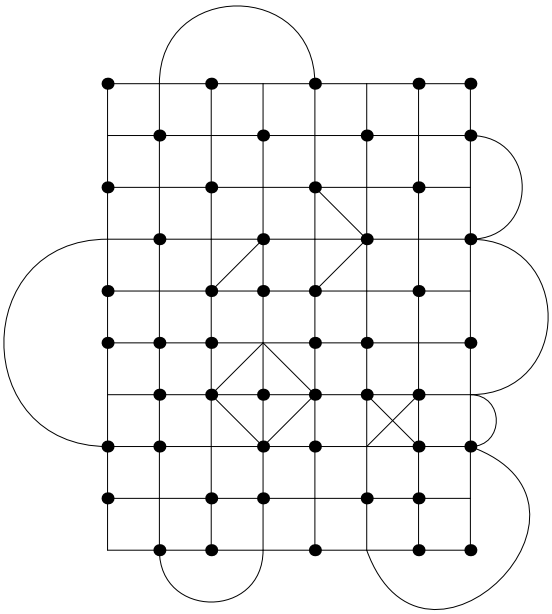

Figura 1.5: *Outra solução factível do problema de* MIN VERTEX COVER *para o grafo da Fig.1.2*

secção da grade). As duas soluções dadas não são coberturas *minimais*. Podemos remover a marca de alguns destes vértices e o conjunto restante ainda é uma cobertura do grafo. Também é interessante pensar que estas soluções podem ter sido geradas através de inspeção "visual" humana, e como conseqüência, alguns "erros" poderiam ter sido introduzidos, como alguns vértices marcados desnecessariamente nas soluções. Eles têm 49 e 50 vértices respectivamente.

Mostraremos agora como uma nova solução pode ser construída através da recombinação destas duas soluções. Existem várias maneiras de como isto pode ser feito, e somente uma delas é aqui usada para ilustrar as idéias gerais. Começamos encontrando os vértices que estão presentes em ambas as coberturas. Na Figura 1.6 selecionamos um tamanho menor para enfatizar os vértices que estão presentes em ambas as soluções factíveis. Este conjunto, denotado como $C_1 \cap C_2$ induz um subgrafo $G'(V', E') \subseteq G(V, E)$ onde $E' \subseteq E$ é o conjunto de arestas cobertas pelos vértices em $C_1 \cap C_2$ e V' é a união de $C_1 \cap C_2$ e o conjunto de vértices que têm todas suas arestas adjacentes já cobertas (isto é, em E'). Este subgrafo é mostrado com arestas finas e os vértices de $C_1 \cap C_2$ são mostrados com menor tamanho na Fig. 1.6.

Com linhas grossas mostramos as arestas de G que não são cobertas pelos vértices em $C_1 \cap C_2$. Note que o grafo não é conectado e tem quatro partes conectadas. Além disso, as arestas de cada um destes quatro fragmentos podem ser cobertas por uma simples *regra de redução* que pode ser descrita como:

> "Se o grafo de entrada tem um vértice u de grau 1, o outro extremo do vértice v *deve estar* na cobertura."

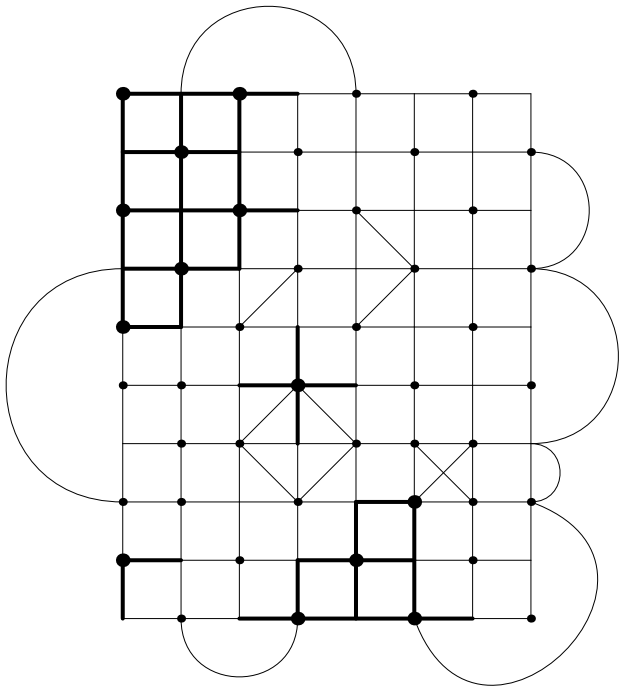

Figura 1.6: *A solução "descendente" obtida pela recombinação tendo como dado de entrada as duas soluções para* Min Vertex Cover *mostradas nas Figuras 1.4 e 1.5.*

Adicionando v à cobertura, garantimos que a aresta (u, v) está coberta, não precisamos incluir o vértice u na cobertura e podemos ter o benefício extra de algumas outras arestas estarem cobertas por v. Convidamos o leitor a checar que de fato a aplicação repetida desta regra de redução conduz a marcar outros 13 vértices como estando na cobertura conduzindo a um total de 44. De maneira interessante neste caso a solução nova obtida por recombinação é diferente e melhor que ambas as soluções iniciais.

1.6 Como ler esta tese

O leitor que tem lido todas as seções deste capítulo consecutivamente achará pelo menos curioso o fato desta seção estar colocada aqui. Outros tipos de leitores que começam através de uma avaliação do índice podem ter começado por esta seção mesmo. Com perdão ao primeiro grupo, algumas palavras introdutórias para o segundo.

Esta tese trata de problemas NPO que são NP-completos em sua versão de decisão. Se o leitor não conhece sobre teoria de NP-completude recomendamos a leitura de capítulos relevantes de [225] primeiro.

Faremos, entretanto, a discussão para problemas que são NP-difíceis e para problemas de complexidade computacional desconhecida. Embora tenha sido feita uma tentativa para que a tese fosse auto-contida, a familiaridade com

noções básicas de Teoria de Computação pode certamente ajudar. Para uma boa avaliação de otimização combinatória o leitor poderia consultar [308].

Este trabalho aborda novos temas e revisões, até certo ponto, artigos na área de computação evolutiva e complexidade computacional. Entretanto, uma ênfase especial foi dada a *algoritmos meméticos*, uma metodologia já destacada pelo autor desta tese em 1989 [270] visto que é uma das meta-heurísticas de maior sucesso atualmente. Mantemos a *"Memetic Algorithms' Home Page"*,[2] um site na web dedicado a informações neste tipo de algoritmos. Muitos desenvolvimentos posteriores ao trabalho desta tese serão encontrados, buscando com a palavra-chave *"memetic algorithms"* na web.

A maioria de nosso próprio material está contido em: Capítulo 2, Capítulo 4, Capítulo 5, Capítulo 7, Capítulo 8 e nas conclusões (Capítulo 9).

No Capítulo 3 apresentamos as definições básicas e a nomenclatura matemática que é usada na tese. Para a notação, seguimos a adotada por Viggo Kann e Pierluigi Crescenzi [89], que começaram a diversos anos atrás um recurso inestimável on-line[3] em aproximabilidade de problemas NPO. Nesse capítulo, a classe de problemas NPO é formalmente definida e diversas medidas de aproximação são discutidas e comparadas. Um especialista em aproximabilidade e teoria da computação pode saltar este capítulo se já é familiar com a notação e o vocabulário.

1.7 Notação básica

Nesta tese tentaremos manter uma notação consistente em todos os capítulos. Assim, pode ser conveniente dedicar uma seção para descrevê-la. Dessa forma, a seguinte notação será utilizada nesta tese. Em relação aos nomes dos problemas, alguns deles estão em lingua inglesa aos efeitos de facilitar a localização de informação usando o "Compendium" de Kann e Crescenzi [89]. Assim, alguns problemas estão identificados usando letras maiúsculas, indicando que tem o mesmo nome nessa referência "on-line" de inestimável valor prático.

Usaremos a seguinte notação:
Conjuntos:

$\mathbb{Z}$	o conjunto dos inteiros
$\mathbb{Z}^+$	o conjunto dos inteiros positivos
$\mathbb{N}$	o conjunto dos números naturais
$\mathbb{Q}$	o conjunto dos números racionais
$\mathbb{R}$	o conjunto dos números reais
$\mathbb{R}^+$	o conjunto dos números reais e positivos
$[a..b]$	o conjunto $\{a, a+1, \ldots, b-1, b\} \subset \mathbb{Z}$
2^A	o conjunto de subconjuntos de A
$\|A\|$	a cardinalidade do conjunto A
$A \subset B$	A é um subconjunto próprio de B

[2] At: `http://www.densis.fee.unicamp.br/~moscato/memetic_home.html`
[3] `http://www.nada.kth.se/~viggo/problemlist/compendium.html`

Grafos:

grafo	um grafo não direcionado
dígrafo	um grafo direcionado
$G = (V, E)$	um grafo com conjunto de vértices V e conjunto de arestas E com arestas (v, v')
$G = (V, A)$	um dígrafo com conjunto de vértices V e conjunto de arcos A com arcos (v, v')
$G = (V, E, W(E))$	um grafo não direcionado com pesos nas arestas com conjunto de vértices V conjunto de arestas E e pesos nas arestas $W(E)$
$G\|_C$	a restrição do grafo G aos vértices o arestas em C

Funções:

log	$\log_2$, i.e. o logaritmo em base 2
ln	$\log_e$, i.e. o logaritmo natural

Capítulo 2

Algoritmos Meméticos

Nesse capítulo, tentaremos oferecer ao leitor uma introdução atualizada e autosuficiente aos Algoritmos Meméticos (MAs), enfocando as características formais mais gerais, mas sem deixar de lado a perspectiva da aplicação prática concreta. Essa introdução à literatura dos MAs nos permitirá depois, no capítulo final da tese, que possamos abordar os assuntos abertos de pesquisa de maior interesse.

2.1 Introdução e história

A denominação genérica *'Algoritmos Meméticos'* é usada para definir uma classe de metaheurísticas (i.e. métodos de propósito geral usados para guiar uma heurística subjacente). O método está baseado em uma população de agentes e provou ser de sucesso prático em uma variedade de problemas. Nós podemos afirmar com segurança que os MAs constituem um dos métodos mais úteis para dar soluções aproximadas para problemas de otimização combinatória e, em particular, para a solução aproximada de problemas de Otimização NP.De fato, durante a última década testemunhamos uma evolução que fez dos MAs uma técnica competitiva (em comparação com outras técnicas de Computação Evolutiva–EC) e uma metodologia sem rival em diversos tipos de problemas de otimização. Ao contrário de métodos de EC tradicionais, os MAs têm intrinsecamente o objetivo de explorar *conhecimento disponível* sobre o problema estudado. A incorporação de conhecimento de domínio do problema não é um mecanismo opcional, mas uma característica fundamental que caracteriza os MAs. Isto filosoficamente é perfeitamente ilustrado pelo termo "memetic". Introduzida pela primeira vez por R. Dawkins [96], a palavra *'meme'* denota um análogo ao gene no contexto da evolução cultural [270]. Em palavras de Dawkins':

> *"Exemplos de memes são melodias, idéias, frases de efeito, modas de roupas, modos de fabricação de panelas ou de arcos de edifício. Da mesma maneira que genes se propagam na piscina de genes saltando de corpo em corpo por espermas ou ovos, assim memes*

> *se propagam na piscina de memes saltando de cérebro em cérebro mediante um processo que, no sentido amplo da palavra, pode ser chamado de imitação."*

Esta caracterização de um meme sugere que, em processos de evolução cultural, a informação não é simplesmente transmitida de maneira inalterada entre indivíduos. Muito pelo contrario, é processada e é aumentada pelas partes que se comunicam. Esta fase é análoga aos processos presentes nos MAs quando incorporam heurísticas, algoritmos de aproximação, técnicas de busca local, operadores de recombinação especializados, métodos exatos truncados, etc. Em essência, a maioria dos MAs pode ser interpretada como estratégias de busca nas quais uma população tenta aperfeiçoar os agentes mediante processos de *cooperação* e *competição* [293]. O notável sucesso dos MAs é provavelmente uma conseqüência direta da *sinergia* dos métodos de busca diferentes que eles incorporam. A característica mais crucial que dos MAs, a inclusão de conhecimento do problema mencionada acima, também é apoiada através de resultados teóricos fortes. Como Hart e Belew [157] inicialmente mostraram e Wolpert e Macready popularizaram depois, no *"No-Free-Lunch Theorem"* [376], a performance de um algoritmo de busca é estritamente dependente da quantidade e qualidade do conhecimento do problema que eles incorporam. Este fato suporta a exploração do conhecimento de problema claramente intrínseco para MAs. Dado que o termo *"hybridization"* [80] é frequentemente usado para denotar o processo de incorporação de conhecimento do problema, não é surpreendente que às vezes os MAs sejam chamados "Algoritmos Evolutivos Híbridos" [95].

Um dos primeiros algoritmos para os quais a denominação MA foi utilizada foi intruzido por Moscato e Norman em 1988 [293], e foi considerado por outros pesquisadores como um híbrido de Algoritmos Genéticos (*traditionais*) (GAs) e o método chamado *Simulated Annealing* (SA). Parte da motivação inicial era descobrir um modo de superar as limitações de ambas as técnicas em um problema de otimização combinatorial bem-estudado, o problema MIN EUCLIDEAN TRAVELING SALESMAN (MIN ETSP). De acordo com os autores, a inspiração original desse algoritmo veio de *torneios computacionais de estratégias de jogos* [171] usados para estudar *"a evolução da cooperação"* [18] (também veja [296] e [286] para outros resultados teóricos mais recentes neste campo). Aquele algoritmo de Moscato e Norman teve várias características que se anteciparam em muitos anos a algoritmos de prática corrente na atualidade. A fase competitiva do algoritmo estava baseada na distribuição de novos pontos de busca no espaço de configurações, mediante um processo que envolve uma *"batalha"* pela sobrevivência seguida pelo processo denominado *"clonagem"*, o qual tem uma semelhança forte com os recentes algoritmos denominados *"go with the winners"* [7] [313]. A fase cooperativa do algoritmo e a busca local que é executada depois em cada agente, poderia ser denomianada *"go-with-the-local-winners"*, pois no algoritmo de Moscato e Norman os *agentes* foram espacialmente organizados com uma topologia de uma grade toroidal em duas dimensões. Depois de algumas experiências iniciais, foi desenvolvida uma certa intuição na particular relevância que a organização espacial, juntamente com um apropriado conjunto de regras, tem no desempenho global de processos de busca da população. Alguns meses depois, Moscato e Norman

descobriram que eles compartilharam visões semelhantes com outros investigadores [279] [140] e outros autores que também propõem *modelos de ilhas* para GAs. A organização no espaço dos agentes está sendo agora reconhecida como um "catalizador", o responsável por uma variedade de efeitos [286] [285]. Este é um assunto de pesquisa importante, atualmente só entendido de forma ainda precária e muito heurística. Porém, alguns resultados formais foram obtidos para problemas relacionados que dão um pouco de esperança a um tratamento mais adequado [147]. Menos de um ano depois, em 1989, Moscato e Norman identificaram vários autores que também estavam a favor da introdução de heurísticas para melhorar as soluções antes de recombiná-las [280] [139] (veja outras referências e a discussão em [270]). Particularmente, no campo dos GAs, outros autores estavam introduzindo *conhecimento do problema* em uma variedade de formas. Em [270] a denominação 'algoritmos meméticos' foi introduzida pela primeira vez. Também foi sugerido que a evolução cultural poderia ser uma metáfora mais adequada para estas metaheurísticas, evitando assim as restrições biológicas que estavam prejudicando o progresso naquele momento. Dez anos depois, embora infelizmente mascaradas na forma de nomes diferentes, os MAs tornaram-se o estado-da-arte como método de otimização aproximada para muitos problemas e ostentam um enorme registro de sucessos em uma variedade de problemas clássicos da classe NP. As últimas seções desse capítulo dão uma lista dessas aplicações.

2.2 Algoritmos Meméticos

Antes de proceder à descrição dos MAs, é necessário prover alguns conceitos básicos e definições. Na primeira subseção, são relacionadas várias noções fortemente ligadas ao campo da complexidade computational. Não obstante, elas podem ser apresentadas em um modo e a um ritmo ligeiramente diferente, por causa do desenvolvimento subseqüente. Estes conceitos básicos darão lugar às noções de busca local e busca baseada em populações, nas quais são fundados os MAs. Este último tipo de busca fornece o marco para estudar a *recombinação*, um mecanismo crucial nos MAs que será estudado com alguma profundidade. Finalmente, serão apresentadas algumas diretrizes para o processo de implementação de um MA.

2.2.1 Conceitos básicos

Um *algoritmo* é um procedimento bem detalhado, passo a passo, que tem o intuito de resolver um *problema computacional*, é dizer, uma seqüência de passos computacionais que transforma um conjunto de dados de entrada em um outro conjunto de dados de salida. Um problema computacional P denota uma classe de tarefas algorítmicas e um conjunto de possíveis dados de entrada, as chamadas *instâncias* I_P. Para cada instância $x \in I_P$, há um conjunto associado $sol_P(x)$ o qual denota as soluções *factíveis* para o problema P, dada a intância x. O conjunto $sol_P(x)$ é também conhecido como o conjunto de soluções *aceitáveis* ou *válidas*.

Quando é esperado que entreguemos um algoritmo que resolva o problema P, isto significa que nosso algoritmo, dada a instância $x \in I_P$, deve retornar pelo menos um elemento y do conjunto de *respostas* $ans_P(x)$ (também chamado *soluções dadas*) que satisfaz às requisições do problema. Este é o primeiro problema de projeto a enfrentar. Em muitas situações, é conveniente definir uma função de factibilidade Booleana $feasible_P(x, y)$ para identificar se uma determinada solução $y \in ans_P(x)$ é aceitável para uma instância $x \in I_P$ de um problema computational P (i.e. conferindo se também $y \in sol_P(x)$).

Tendo definido um problema computational P, a meta óbvia é *resolvê-lo*. Independentemente do tipo de estratégia que usaremos, um problema original pode ser a fonte de outros que, em troca, podem ser classificados considerando seus diferentes tipos de objetivos, por exemplo:

- *encontrar todas* as soluções em $sol_P(x)$, i.e., problemas de *enumeração*.
- *contar quantas* soluções existem em $sol_P(x)$, i.e. problemas de *contagem*.
- determinar se o conjunto $sol_P(x)$ *é vazio ou não*, i.e., problemas de *decisão*.
- encontrar uma solução em $sol_P(x)$ que maximize ou minimize um função dada, i.e., problemas de *otimização*.

Neste capítulo, enfocaremos a última possibilidade, quer dizer, um problema será considerado *resolvido* quando encontrarmos uma certa solução factível entre aquelas que são *ótimas* em $y \in sol_P(x)$, ou quando damos uma indicação de que nenhuma solução possível existe. Um algoritmo é dito *resolve* o problema P se pode cumprir esta condição para qualquer instância determinada $x \in I_P$. Esta definição é certamente muito ampla, uma caracterização mais restritiva para nossos problemas de interesse é necessária. Esta caracterização é provista restringindo-se aos denominados problemas de *otimização combinatória*. Estes constituem uma subdivisão de classe especial de problemas computationais nos quais, para cada instância $x \in I_P$:

- a cardinalidade do conjunto $sol_P(x)$ é finita.
- cada solução $y \in sol_P(x)$ tem um *valor inteiro de mérito* $m_P(y, x)$, obtida através de uma *função objetivo* m_P.
- uma ordem parcial $\prec_P$ é definida para o conjunto de valores de mérito devolvido pela função objetivo e permite determinar qual das soluções é preferível.

Uma instância $x \in I_P$ de um problema de otimização combinatória P é resolvida ao se encontrar a melhor solução $y^* \in sol_P(x)$, i.e., encontrando uma solução y^* tal que nenhuma outra solução $y \prec_P y^*$ existe se $sol_P(x)$ é não vazio. É muito comum ter $\prec_P$ definindo uma ordem total. Neste caso, a melhor solução é aquela que maximiza (ou minimiza) a função objetivo. Como um exemplo de um problema de otimização combinatória considere o

0-1 MULTIPLE KNAPSACK PROBLEM (0-1 MKP). Cada instância x desse problema é definida por um vetor de benefícios $V = \{v_0, \cdots, v_{n-1}\}$, um vetor de capacidades $C = \{c_0, \cdots, c_{m-1}\}$, e uma matriz de restrições de capacidades $M = \{m_{ij} \ : 0 \leq i < m, \ 0 \leq j < n\}$ para as mochilas. Intuitivamente, o problema consiste em selecionar um conjunto de objetos para maximizar o benefício, sem violar as restrições de capacidade. Se os objetos são indexados com os elementos do conjunto $\mathbb{N}_n = \{0, 1, \cdots, n-1\}$, o conjunto resposta $ans_P(x)$ para uma instância x é simplesmente o conjunto das partes de $\mathbb{N}_n$, isto é, cada subconjunto de $\mathbb{N}_n$ é uma possível resposta. Além disso, o conjunto de possíveis respostas $sol_P(x)$ está composto desses subconjuntos cujo vetor de incidência B verifica $M \cdot B \leq C^T$. Finalmente, a função objetivo é definida como $m_P(y, x) = \sum_{i \in y} v_i$, i.e., a soma de benefícios para todos os objetos selecionados e a meta é maximizar este valor. Note que, associado com um problema de otimização combinatória, sempre podemos definir sua versão de *decisão*. Para formular o problema de decisão, um valor inteiro K é considerado, e em vez de tentar achar a melhor solução para a instância x, nós perguntamos se x tem uma solução cuja função objetivo é igual ou melhor que K. No exemplo anterior, poderíamos nos perguntar se é possível achar uma solução y tal que seu lucro associado é igual ou melhor que K.

2.2.2 Paisagem de busca

Como foi mencionado acima, depois de ter definido mais formalmente o conceito de problema de otimização combinatória, um dos problemas é achar pelo menos uma das soluções ótimas para uma determinada instância recebida como dado de entrada. Para este propósito, deve ser usado um algoritmo de busca. Antes de discutir algoritmos de busca, devem ser discutidas três entidades. Elas são: o *espaço de busca*, a *relação de vizinhança*, e a *função guia*. É importante considerar que, para qualquer problema computational determinado, estas três entidades podem ser instanciadas de vários modos e podem dar lugar a tarefas de otimização diferentes. Começaremos definindo o conceito de espaço de busca para um problema de otimização P. Consideramos um conjunto $\mathcal{S}_P(x)$, de elementos que têm as seguintes propriedades:

- Cada um dos elementos $s \in \mathcal{S}_P(x)$ representa pelo menos uma resposta em $ans_P(x)$.
- Para problemas de decisão: pelo menos um elemento de $sol_P(x)$ que representa uma resposta 'Sim', se existir, deve ser representada através de um elemento em $\mathcal{S}_P(x)$.
- Para problemas de otimização: pelo menos um elemento *ótimo* y^* de $sol_P(x)$ é representado por um elemento em $\mathcal{S}_P(x)$.

Cada elemento de $\mathcal{S}_P(x)$ será definido como uma *configuração*, sendo relacionado a uma resposta em $ans_P(x)$ por uma *função de crescimento* $g : \mathcal{S}_P(x) \rightarrow ans_P(x)$. Note que o primeiro requisito refere-se a $ans_P(x)$ e não a $sol_P(x)$, i.e., algumas configurações no espaço de busca podem corresponder a soluções infactíveis. Assim, o algoritmo de busca precisa estar preparado para lidar

com este fato. Se estas exigências foram alcançadas, dizemos que nós temos uma *representação válida* ou *formulação válida* do problema. Para simplificar, nós apenas escreveremos $\mathcal{S}$ para referir a $\mathcal{S}_P(x)$ quando x e P são óbvios a partir do contexto. As pessoas que usam metáforas inspiradas na Biologia gostam de chamar ao $\mathcal{S}_P(x)$ o *espaço dos genótipos* enquanto $ans_P(x)$ denota o *espaço dos fenótipos*, assim referimo-nos adequadamente a g como uma *função de crescimento*, ou mapeamento o genótipo ao fenótipo.

Para ilustrar esta noção de espaço de busca, considere novamente o caso do 0-1 MKP. Desde que soluções em $ans_P(x)$ são subconjuntos de $\mathbb{N}_n$, nós podemos definir o espaço de busca como o conjunto de vetores binários de dimensão n Cada vetor representará o vetor de incidência de um certo subconjunto, i.e., a função de crescimento g é definida como $g(s) = g(b_0 b_1 \cdots b_{n-1}) = \{i \mid b_i = 1\}$. Como foi mencionado acima, muitos vetores binários podem corresponder a conjuntos de objetos que correspondem a soluções infactíveis do problema. Uma outra possibilidade é definir o espaço de busca como o conjunto de permutações de elementos em $\mathbb{N}_n$ [143]. Neste caso, uma função de crescimento pode consistir em aplicar um algoritmo guloso de construção [85], considerando introduzir objetos na ordem provida pela permutação. Ao contrário do espaço de busca binário mencionado previamente, todas as configurações representam possíveis soluções neste caso.

O papel do espaço de busca é prover uma "base" sobre a qual o algoritmo de busca agirá indiretamente movendo-se no conjunto imagem $ans_P(x)$. Propriedades importantes do espaço de busca que afetam a dinâmica do algoritmo de busca são aquelas que têm a ver com as relações de acessibilidade entre as configurações. Estas relações são dependentes de uma *relação de vizinhança* $\mathcal{N} : \mathcal{S} \to 2^{\mathcal{S}}$. Esta função nomeia, para cada elemento $s \in \mathcal{S}$ um conjunto $\mathcal{N}(s) \subseteq S$ de configurações vizinhas de s. O conjunto $\mathcal{N}(s)$ é chamado *vizinhança* de s e cada membro $s' \in \mathcal{N}(s)$ é chamado um *vizinho* de s.

Deve ser notado que a vizinhança depende da instância. Assim, a notação $\mathcal{N}(s)$ é uma forma simplificada de $\mathcal{N}_P(s, x)$ a partir do contexto. Os elementos de $\mathcal{N}(s)$ não necessitam ser listados explicitamente. De fato, é muito habitual defini-los *implicitamente* referindo-se a um conjunto possível de *movimentos*, os quais definem *transições* entre configurações. Os movimentos são usualmente definidos como modificações *"locais"* de alguma parte de s, onde "localidade" se refere ao fato de que o movimento atuando em uma única solução obtém outra única solução. Esta "localidade" é um dos ingredientes chaves da *Busca Local*, e de fato também estendeu o nome a todo o paradigma de busca desse tipo.

Como exemplos de definições de vizinhança concretas, considere as duas representações de soluções para o 0-1 MKP apresentado acima. No primeiro caso (representação binária), podem ser definidas transições como mudando os valores de vários *"bits"*. Se um *"bit"* é modificado de cada vez, a estrutura de vizinhança resultante é o hipercubo binário n-dimensional. No segundo caso (representação de permutação), podem ser definidas transições como aquela do intercâmbio de duas posições na permutação. Assim, duas configurações são vizinhas se, e só se, elas diferem em exatamente duas posições.

A definição de localidade apresentada acima não é necessariamente vinculada a algum tipo de relação de distância entre configurações (exclua-se a situação tautológica na qual a distância entre duas configurações s e s' é definida como o número mínimo de transições necessárias para alcançar s' a partir de s). De fato, é possível dar exemplos comuns de definições de vizinhança muito complexas sem conexão com medidas de distância mais intuitivas.

Uma característica importante, que deve ser considerada quando seleciona-se a classe de transições a ser usada no algoritmo de busca é a *"ergodicidade"* do conjunto de transições. A ergodicidade se refere à habilidade, dada qualquer $s \in \mathcal{S}$, de achar uma sucessão de transições que podem alcançar *qualquer outra*, configuração $s' \in \mathcal{S}$. Em muitas situações, esta propriedade é bastante óbvia e nenhuma demonstração explícita é requerida. Essa propiedade é muito importante desde que tendo-se uma representação válida (recorde a definição acima), é necessário garantir isso *"a priori"*. Assim, pelo menos uma solução ótima é alcançável desde qualquer solução inicial determinada. Novamente, considere a representação binária de soluções para uma instância de 0-1 MKP. Se movimentos são definidos como uma única mudança de valor de um bit, é óbvio que qualquer configuração s' pode ser alcançada desde outra configuração s em exatamente h movimentos, onde h é a distância de Hamming entre essas duas configurações. Entretanto, este não é sempre o caso.

A última entidade que deve ser definida é a *função guia*. Para isso, requeremos um conjunto $\mathcal{F}$, onde os elementos são definidos como sendo os valores de *"fitness"* (tipicamente $\mathcal{F} \equiv \mathbb{R}$), e uma ordem parcial $\prec_{\mathcal{F}}$ em $\mathcal{F}$ (tipicamente, mas não sempre, $\prec_{\mathcal{F}} \equiv <$). A função guia é definida como uma função $F_g : \mathcal{S} \to \mathcal{F}$ que associa a cada configuração $s \in \mathcal{S}$ um valor $F_g(s)$ que avalia a qualidade da solução. O comportamento do algoritmo de busca será "guiado" ou "controlado" por estes valores de aptidão. Note que para problemas de otimização há uma conexão direta óbvia entre a função guia F_g e a função objetivo m_P. De fato, é muito comum forçar esta relação até o ponto que ambas as condições serem consideradas equivalentes. Porém, esta equivalência não é necessária e, em muitas situações, nem mesmo desejável. Para problemas de decisão, desde que uma solução é uma resposta do tipo 'Sim' ou 'Não', as funções guia normalmente associadas são do tipo de *distância da factibilidade*.

Um exemplo típico é o BOOLEAN SATISFIABILITY PROBLEM, i.e., determinar se existe uma designação de valores-verdade aos literais de uma expressão Booleana (em forma normal conjuntiva) que satisfaza todas as cláusulas. Neste caso, as soluções são os valores-verdade das variáveis booleanas. Uma função guia adequada poderia ser o número de cláusulas satisfeitas na configuração atual e a meta seria maximizar este número. Por exemplo, uma função guia para o problema de decisão pode ser a é a função objetivo do problema associado de otimização NP chamado MAX SAT . Porém, pode ser mais conveniente usar outras funções-guia. Por exemplo, uma delas poderia ser $F_g(s) = \sum_i f_i(s)$, a soma de índices de cláusulas i, com $f_i(s) = 0$ para toda cláusula insatisfeita i, e $f_i(s) = 1$ se a cláusula i é satisfeita.

A diferenciação acima entre função-objetivo e a função-guia também é muito importante no contexto de problemas de otimização com restrições, i.e., problemas para os quais, em geral, $sol_P(x)$ é escolhido ser um subconjunto própio

de $ans_P(x)$. Desde que a função de crescimento estabelece um mapeamento de $\mathcal{S}$ para $ans_P(x)$, o algoritmo de busca poderia precisar processar ambas as possíveis soluções (quais valores de função-objetivo são bem-definidos) e soluções infactíveis (quais valores da função-guia são *"ad hoc"* e mal definidos em geral). Em muitas implementações de MAs para estes problemas, uma função-guia é definida como uma soma ponderada do valor da função-objetivo e a distância à factibilidade (o que leva em conta as restrições). Tipicamente, um peso mais alto é dado às restrições, para dar preferência à viabilidade sobre a otimalidade. Vários outros remédios para este problema abundam e incluem o uso de técnicas multi-objetivo de otimização.

A combinação de uma certa instância do problema e as três entidades definidas acima induz ao que em língua inglesa é denominado *"fitness landscape"* e que poderíamos traduzir como *paisagem de aptidão* [270] [186] [257] [246]. Essencialmente, uma paisagem de aptidão pode ser definida como um grafo dirigido com pesos no qual os vértices são configurações do espaço de busca $\mathcal{S}$ e os arcos conectam configurações vizinhas. Os pesos são a diferença da função-guia entre as configurações aos extremos de um arco. A busca pode ser vista assim como o processo de "navegação" na paisagem de aptidão usando a informação provida pela função-guia. Esta é uma metáfora muito poderosa, pois permite interpretar em termos de objetos topográficos muito intuitivos como *picos*, *vales*, *planaltos*, etc, de grande utilidade para visualizar o progresso da busca, e descobrir fatores que afetam o desempenho do processo [322] [254] [257]. Em particular, a noção importante de ótimo *local* é associada a esta definição de paisagem de aptidão. Para sermos precisos, podemos dizer que um ótimo local é um vértice da paisagem de aptidão cujo valor da função guia é melhor que os valores de todos seus vizinhos. Note que diferentes transições definem vizinhanças diferentes e conseqüentemente diferentes paisagems de aptidão, até mesmo quando a mesma instância de problema é considerada. Por isto, a noção de ótimo local não é intrínseca a uma instância de problema como é, às vezes, erroneamente considerada.

2.2.3 Busca Local vs. Busca Baseada em Populações

As definições apresentadas na subseção prévia conduzem à noção de *algoritmo de busca local*. Um algoritmo de busca local, começa a partir de uma configuração $s_0 \in \mathcal{S}$, gerada ao acaso ou construída por algum outro algoritmo. Subseqüentemente, um processo é iterado usando a cada passo uma transição baseada na vizinhança da configuração atual. Transições que conduzem a configurações preferíveis (de acordo com a ordem parcial $\prec_{\mathcal{F}}$) são aceitas, i.e., a configuração recentemente gerada passa a ser a configuração atual no próximo passo. Caso contrário, a configuração atual é mantida. Este processo é repetido até um certo critério de parada ser alcançado. Critérios típicos de parada são a realização de um número pré-especificado de repetições, parar depois de não ter achado melhoria nas últimas m iterações, ou até mesmo mecanismos mais complexos baseados em calcular a probabilidade de se ter atingido um ótimo local [83].

Devido a estas características, a metodologia é metaforicamente chamada

"hill climbing". O processo inteiro é esboçado na Fig. 2.1.

```
Procedure Local-Search-Engine (current)
begin
   repeat
      new ← GenerateNeighbor(current);
      if (F_g(new) ≺_F F_g(current)) then
         current ← new;
      endif
   until TerminationCriterion();
   return current;
end
```

Figura 2.1: *Um algoritmo de busca local*

A seleção do tipo particular de movimentos a usar (o processo também conhecido como *mutação* no contexto de GAs) certamente depende das características específicas do problema e da representação escolhidas. Não há nenhum conselho geral para isto, visto que é uma questão que depende do tempo de computador disponível para o processo inteiro, como também de outras decisões algorítmicas, que incluem facilidade de codificação, etc. Em certos casos, alguns movimentos são distintos, por exemplo, pode ser a mudança do valor de uma única variável ou a troca dos valores de duas variáveis diferentes. Também, às vezes, podem ser compostos, como os de uma cadeia de transições. Por exemplo, em relação com os MAs, Radcliffe e Surry introduziram o conceito de *"Binomial Minimal Mutation"*, onde o número de mutações a executar é selecionado de acordo com certa distribuição binomial [323]. No contexto de paisagens de aptidão, isto é equivalente a uma redefinição da relação de vizinhança e considera duas configurações como vizinhas quando existe uma cadeia de transições que as conectam.

Algoritmos de Busca Local são caracterizados mantendo uma única configuração de cada vez. A generalização imediata deste comportamento é a manutenção simultânea de $k \geq 2$ configurações. A denominação de "algoritmos baseados em populações" foi cunhada para denotar técnicas de busca que se comportam deste modo.

A disponibilidade de várias configurações permite o uso de novos mecanismos poderosos, que atravessam a paisagem de aptidão muito além do operador de mutação padrão. Na próxima seção, será estudado com maior profundidade o mais popular destes mecanismos, o operador de recombinação. Em todo caso, é bom advertir que o funcionamento geral de técnicas de busca baseadas em população é bem parecido com o pseudocódigo descrito na Fig. 2.1. De fato, um algoritmo populacional pode ser entendido como um procedimento no qual consecutivamente visitamos os vértices de um hipergrafo. Cada vértice do hipergrafo representa um conjunto de configurações em $\mathcal{S}_P(x)$, i.e., a população. O próximo vértice a ser visitado, i.e. a nova população, pode ser estabelecido de acordo com a composição das vizinhanças dos mecanismos de transição di-

ferentes usadas no algoritmo de população. Apesar da analogia com a Busca Local, é amplamente aceito na literatura científica aplicar a denominação de 'local' só aqueles procedimentos que visitam uma configuração a cada instante. Por isto, o termo 'local' será usado com esta interpretação no remanescente da tese.

2.2.4 Recombinação

Como foi mencionado na seção anterior, a busca local está baseada na aplicação de um operador de mutação a uma única configuração. Apesar da simplicidade aparente deste mecanismo, a busca local baseada em mutação se revelou um mecanismo muito poderoso por obter soluções de boa qualidade para problemas NP-difíceis (ex., veja [111] [336]). Alguns investigadores tentaram prover uma base teoricamente sólida para esta classe de busca. Nesta linha, é importante mencionar a definição da classe *"Polynomial Local Search"* (PLS) introduzida por Johnson, Papadimitriou e Yannakakis [183]. Basicamente, esta classe de complexidade inclui problemas que compreendem um problema básico de otimização e uma paisagem de busca associada tal que podemos decidir em tempo polinomial se é possíivel achar uma solução melhor na vizinhança. Infelizmente, é muito provável que nenhum problema NP-difícil esteja contido na classe PLS, visto que implicaria que NP=co-NP[381], uma conjetura normalmente considerada falsa. Este fato motiva a pesquisa de mecanismos de busca adicionais serem usado como operadores individuais ou como complementar para a mutação padrão.

Nesta linha, a Busca Baseada em População permitiu a definição de operadores de transições generalizadas, os operadores de *recombinação*. Em essência, a recombinação pode ser definida como um processo no qual um conjunto S_{par} de n configurações (informalmente chamado de "pais") é manipulado para criar um conjunto $S_{desc} \subseteq sol_P(x)$ de m configurações novas (informalmente chamadas de "descendentes"). A criação destes descendentes precisa da identificação e combinação de características dos pais.

Neste momento, é possível considerar propriedades de interesse que podem ser exibidas por operadores de recombinação [323]. A primeira propriedade, *respeito*, representa o carácter explorador ou intensificador da recombinação. É dito que um operador de recombinação é *respeitoso*, relativo a um tipo particular de características das configurações se, e só se, gera descendentes que levam todas as características básicas comums a todos os pais. Note que, se todas as configurações dos pais são idênticas, um operador de recombinação respeitoso é obrigado a devolver a mesma configuração como descendente. Esta propriedade é denominada *pureza*, e pode ser alcançada até mesmo quando o operador de recombinação não é geralmente respeitoso.

Por outro lado, *sortimento* (tradução da palavra em lingua inglesa *"assortment"*) representa o lado diversificador da recombinação. É dito que um operador de recombinação é *"properly assorting"* (que poderíamos traduzir como "adequadamente sortido") se, e só se, pode gerar descendentes que levam qualquer combinação de características compatíveis presentes nos pais. É dito que o sortimento é *fraco* se é necessário executar várias recombinações dentro

da descendência para alcançar este efeito.

Finalmente, *transmissão* é uma propriedade muito importante, que intuitivamente serve para capturar o papel da recombinação. É dito que um operador está *"transmitindo"* se toda característica exibida pela descendência está presente em pelo menos um dos pais. Assim, um operador de recombinação que *"transmite"* combina o presente da informação nos pais mas não introduz informação nova. Esta tarefa posterior permanece delegada ao operador de mutação. Por isto, é dito que um operador de recombinação que não *"transmite"* introduz *mutação implícita*.

As três propriedades são suficientes para qualitativamente descrever o comportamento abstrato de entrada/salida de um operador de recombinação que considera algum conjunto de características particulares. Isto se deve à possibilidade de prover uma caracterização dos possíveis descendentes que podem ser produzidos pelo operador. Não obstante, existem outros aspectos do funcionamento da recombinação que devem ser estudados. Em particular, é interessante considerar como a construção de $\mathcal{S}_{desc}$ é abordada.

Em primeiro lugar, é dito que um operador de recombinação é *cego* se não tem nenhuma outra informação recebida como dado de entrada além do conjunto $\mathcal{S}_{par}$, i.e., não usa nemhuma informação da instância do problema. Esta definição é certamente muito restritiva, e conseqüentemente, às vezes, é relaxada para permitir ao operador de recombinação usar informação relativa às restrições do problema (para construir possíveis descendentes), e possivelmente a função de aptidão das configurações $y \in \mathcal{S}_{par}$ (serve para influenciar a geração de descendentes com características dos melhores pais). Um exemplo típico de um operador de recombinação cego é o clássico *"Uniform Crossover"* [355]. Este operador é definido em espaços de busca $\mathcal{S} \equiv \Sigma^n$, i.e., *"strings"* de n símbolos de um alfabeto Σ. A construção do descendente é completada selecionando aleatoriamente, a cada posição, um dos símbolos que aparecem naquela posição em quaisquer dos pais. Esta seleção aleatória pode ser totalmente uniforme ou pode ser influenciada de acordo com os valores da função de aptidão dos pais, como mencionou-se antes. Além disso, a seleção pode ser feita para obrigar factibilidade (por exemplo, considere a representação binária de soluções no 0-1 MKP). Note que, neste caso, o operador resultante não é nem respeitoso nem transmite, em geral.

O uso de operadores de recombinação cegos esteve normalmente justificado na tentativa de não introduzir excessiva pré-seleção nos processos do algoritmo de busca e assim prevenir a convergência extremamente rápida para soluções subótimas. Isto é, entretanto, questionável. Primeiro, pode-se advertir que o comportamento do algoritmo é de fato parcialmente ligado à escolha da representação e à mecânica dos operadores particulares escolhidos. Segundo, existem mecanismos amplamente conhecidos (por exemplo, o isolamento no espaço) usados para superar estes problemas. Finalmente, pode ser melhor obter uma solução subótima rapidamente e logo reiniciar o algoritmo, em lugar de usar os operadores cegos por muito tempo em perseguição a um comportamento global assintoticamente ótimo (que nem mesmo é garantido na maioria dos casos).

Os operadores de recombinação que usam conhecimento do problema são

comumente chamados *heurísticos* ou *híbridos*. Nestes operadores, é utilizada informação do problema para guiar o processo de construção dos descendentes. Isto pode ser feito em uma variedade de modos para cada problema. Assim, é difícil prover uma taxonomia de operadores de recombinação heurísticos. Não obstante, existem dois aspectos principais nos quais o conhecimento do problema pode ser introduzido: a seleção das características dos pais que serão transmitidas ao descendente, e a seleção de características não-parentais que serão somadas a isto. Um operador de recombinação heurístico pode concentrar-se num destes aspectos, ou em ambos, simultaneamente.

Como um exemplo de operador de recombinação heurístico enfocado no primeiro aspecto, a *Recombinação Dinástica Ótima* (DOR) [82] deve ser mencionada. Este operador explora o conjunto dinástico potencial (i.e. possíveis descendentes) das configurações que são recombinadas para achar a melhor solução deste conjunto (adverte-se que as configurações no potencial dinástico estão completamente compostas de características levadas de quaisquer dos pais: este é um operador que transmite). Esta exploração é obtida usando um algoritmo A^* subordinado que minimiza o custo de atravessar o potencial dinástico. A meta deste operador é achar a melhor combinação de características parentais que dão lugar a uma possível solução viável. Conseqüentemente, este operador é monotônico, devido ao fato de que qualquer solução descendente gerada é pelo menos tão boa quanto o melhor pai.

Como exemplos de operadores de recombinação heurísticos que se concentram na seleção de características não-parentais, podemos citar os operadores *"patching-by-forma-completion"* propostos por Radcliffe e Surry [322]. Estes operadores estão baseados em gerar uma solução incompleta usando um procedimento não-heurístico (por exemplo, o operador RAR_ω [321]), e completar a solução usando um método de busca local restrito ás características não especificadas (*"locally optimal forma completion"*) ou um procedimento de busca global que acha a melhor solução que leva às características especificadas (*"globally optimal forma completion"*). Note a semelhança desta segunda metodologia com DOR.

Finalmente, existem alguns operadores que tentam explorar conhecimento em ambos os aspectos discutidos acima. Um exemplo distinto é o *"Edge Assembly Crossover"* (EAX) [284]. EAX é um operador especializado para o TSP (para instâncias simétricas e assimétricas) no qual a construção do novo tour inclui duas fases: a primeira envolve a geração de uma solução incompleta pelos denominados E-sets (subtours compostos de arestas/arcos que alternativamente pertencem a um dos dois pais); subseqüentemente, estes subtours são fundidos em um único possível subtour, usando um algoritmo de factibilização guloso. Os autores deste operador relataram resultados impressionantes em termos de precisão e velocidade. O operador tem algumas semelhanças com o operador de recombinação proposto em [272].

Um comentário final deve ser feito em relação à complexidade computational da recombinação. Está claro que combinar as características de várias soluções é em geral computationalmente mais caro que modificar uma única solução (i.e. uma mutação). Além disso, a operação de recombinação normalmente será invocada um número grande de vezes. Por isto, é conveniente (e em mui-

tas situações obrigatório) requerer um baixo custo computational. Uma diretriz razoável é considerar um limitante superior $O(N \log N)$ para sua complexidade, onde N é o tamanho da entrada (o conjunto S_{par} é a instância do problema x). Tal limite é facilmente alcançável por operadores de recombinação cegos, sendo o termo *"crossover"* um nome razoável que carrega implicitamente a baixa complexidade (contudo nem sempre usado neste contexto) destes. Porém, este limite pode ser relativamente adstringente no caso de recombinação heurística, principalmente quando a chamada *epistasis* (a influência não aditiva entre características sobre o valor de função de aptidão) é envolvida. Isto conduz a várias soluções que dependem da heurística particular usada. Por exemplo, DOR tem um pior caso de comportamento do tipo exponencial, mas pode ser feito mais tratável escolhendo partes de maior tamanho de cada pai (quanto maior o tamanho destes partes de informação, mais baixo o número deles necessarios para completar a nova solução) [86]. Em todo caso, os operadores de recombinação heurísticos provêem soluções melhores que os operadores de recombinação cegos e, conseqüentemente, eles não precisam ser invocados o mesmo número de vezes.

2.2.5 Projetando um algoritmo memético

Levando em conta as considerações acima, é possível prover um modelo geral para um algoritmo memético. Primeiro devemos prover um modelo para um algoritmo de busca baseado em populações. Como mencionado na Subseção 2.2.3, este modelo é bem parecido a um procedimento de busca local mas que age em um conjunto de $|pop| \geq 2$ configurações. Isto é mostrado na Fig. 2.2:

```
Procedure Population-Based-Search-Engine
begin
    Initialize pop using GenerateInitialPopulation();
    repeat
        newpop ← GenerateNewPopulation(pop);
        pop ← UpdatePopulation (pop, newpop)
        if pop has converged then
            pop ← RestartPopulation(pop);
        endif
    until TerminationCriterion()
end
```

Figura 2.2: *Modelo de um algoritmo de busca baseado em população*

Este modelo requer alguma explicação. Em primeiro lugar, o procedimento GenerateInitialPopulation é responsável por criar o conjunto inicial de $|pop|$ configurações. Isto pode ser feito simplesmente gerando $|pop|$ configurações fortuitas ou usando um mecanismo mais sofisticado (por exemplo, alguma heurística construtiva) por meio do qual são injetadas configurações de alta qualidade na população inicial [354] [236]. Numa outra possibilidade, a máquina de

busca local apresentada na subseção. 2.2.3 poderia ser usada, como mostrado na Fig. 2.3.

```
Procedure GenerateInitialPopulation
begin
   Initialize pop using EmptyPopulation();
   parfor j ← 1 to popsize do
      i ← GenerateRandomConfiguration();
      i ← Local-Search-Engine (i);
      InsertInPopulation individual i to pop;
   endparfor
   return pop
end
```

Figura 2.3: *Colocando soluções de alta qualidade na população.*

A função TerminationCriterion, pode ser definida de maneira muito semelhante ao caso da Busca Local, i.e. fixando um limite para o número total de repetições, alcançando um número máximo de repetições sem melhoria, ou tendo executado um certo número de reinícios da população, etc.

O procedimento GenerateNewPopulation é o núcleo do algoritmo memético. Essencialmente, este procedimento pode ser visto como um processo que inclui n_{op} estágios. Cada um desse estágios consiste de $arity_{in}^{j}$ configurações do estágio anterior, gerando $arity_{out}^{j}$ novas configurações aplicando o operador op^j. Este processo é obrigado a ter $arity_{in}^{1} = popsize$. O processo todo é esquematizado na Fig. 2.4:

```
Procedure GenerateNewPopulation (pop)
begin
   buffer^0 ← pop;
   parfor j ← 1 to n_op do
      Initialize buffer^j using EmptyPopulation();
   endparfor
   parfor j ← 1 to n_op do
      S^j_par ← ExtractFromBuffer (buffer^{j-1}, arity^j_in);
      S^j_desc ← ApplyOperator (op^j, S^j_par);
      for z ← 1 to arity^j_out do
         InsertInPopulation individual S^j_desc[z] to buffer^j;
      endfor
   endparfor;
   return buffer^{n_op}
end
```

Figura 2.4: *O procedimento de criação da nova população.*

Este modelo para o procedimento de criação da nova população normalmente é implementado em GAs deixando $n_{op} = 3$, usando uma seleção, uma recombinação, e um operador de mutação. Tradicionalmente, a mutação é aplicada depois da recombinação, i.e., em cada nova solução gerada pelo operador de recombinação. Porém, se um operador heurístico de recombinação está sendo usado, pode ser mais conveniente aplicar mutação *antes* da recombinação. Como o propósito da mutação é simplesmente introduzir características novas na configuração, usá-la com antecedência também é possível. Além disso, a combinação do aspecto *dedicado* ou *"ad hoc"* executado pelo operador heuristíco não seria perturbado deste modo.

Esta situação é ligeiramente diferente nos MAs. Neste caso, é muito comum deixar $n_{op} = 5$, inserindo uma fase de Busca Local depois de aplicar op^2 e op^4 (respectivamente, recombinação e mutação). Devido ao fato de usar a Busca Local, a ordem de aplicação entre mutação e recombinação não é tão problemático quanto em GAs.

O procedimento UpdatePopulation é usado para reconstruir a população atual que usa a população velha *pop* e a população recentemente gerada *newpop*. Utilizando a terminologia das estratégias evolutivas [328, 339], existem duas possibilidades principais para continuar esta reconstrução: a estratégia *"plus"* e a estratégia *"comma"*. Na primeira, a população corrente é construída com as melhores $popsize$ configurações de $pop \cup newpop$. No último, as melhores $popsize$ configurações são tomadas de *newpop*. Nesse caso, é preciso ter $|newpop| > popsize$, para incluir um pouco de pressão seletiva no processo (quanto maior a razão $|newpop|/popsize$, maior é a pressão). Caso contrário, a busca se reduziria a uma busca aleatória em $\mathcal{S}$.

Há vários estudos que consideram escolhas apropriadas para o procedimento de UpdatePopulation (por exemplo, [21] [144]). Como uma diretriz geral, a estratégia de comma é usualmente considerada menos propensa à estagnação, sendo a relação,

$$|newpop|/posize \simeq 6,$$

uma escolha comum [20]. Não obstante, esta opção pode ser um pouco cara computacionalmente se a função-guia é complexa e demorada de computar. Uma alternativa comum é usar uma estratégia com um baixo valor de $|newpop|$, análoga á denominada *"steady-state"* ou estratégia de substituição em GAs [373]. Esta opção normalmente provê uma convergência mais rápida para soluções de alta qualidade. Porém, têm que ser tomados cuidados com a convergência prematura para regiões subótimas do espaço de busca, i.e., todas as configurações na população que são bem parecidas com outra, dificultando a exploração de outras regiões de $\mathcal{S}$.

A consideração anterior sobre convergência prematura conduz ao último componente do modelo mostrado na Fig. 2.2, o procedimento que reinicia a busca. Em primeiro lugar, deve ser decidido se a população degradou ou não. Para fazer isso, é possível usar alguma medida de diversidade de informação na população como a entropia de Shannon [93]. Se esta medida cai abaixo de um limiar predefinido, a população é considerada em um estado degenerado. Este limiar depende da representação (número de valores por variável,

constrangimento, etc.) e conseqüentemente deve ser determinado de maneira *"ad-hoc"*. Uma possibilidade diferente é o uso de uma metodologia do tipo probabilistíca para determinar com uma confiança desejada que a população convergiu. Por exemplo, em [174] um método do tipo Bayesiano é apresentado para este propósito. Uma vez considerado que a população está em um estado degenerado, o procedimento Restart é invocado. Novamente, isto pode ser implementado de vários modos. Uma estratégia muito típica é manter uma fração da população atual (esta fração pode ser tão pequena quanto uma solução, a melhor atual), e ir substituindo as configurações remanescentes como mostrado na Fig. 2.5.

```
Procedure RestartPopulation (pop)
begin
   Initialize newpop using EmptyPopulation();
   #preserved ← popsize · %preserved;
   for j ← 1 to #preserved do
      i ← ExtractBestFromPopulation(pop);
      InsertInPopulation individual i to newpop;
   endfor
   parfor j ← #preserved + 1 to popsize do
      i ← GenerateRandomConfiguration();
      i ← Local-Search-Engine (i);
      InsertInPopulation individual i to newpop;
   endparfor;
   return newpop
end
```

Figura 2.5: *Pseudocódigo do processo de reinício da população.*

O procedimento mostrado na Fig. 2.5 é também conhecido como a estratégia *"random-immigrant"* [75]. Uma possibilidade diferente é ativar um operador de mutação *forte* ou *pesado* para dirigir a população para longe de sua localização atual no espaço de busca. Ambas as opções têm suas vantagens e desvantagens. Por exemplo, quando a estratégia de *"random-inmigrant"* é usada, tem-se que tomar alguma precaução para evitar que as configurações preservadas "tomem conta" da população (isto pode ser alcançado impondo uma baixa pressão seletiva, pelo menos nas primeiras repetições depois do reinício). Sobre a estratégia de mutação pesada, deve se alcançar uma relação de compromisso entre uma mutação excessivamente forte, que destruiria qualquer informação contida na população atual, e uma mutação não tão forte que faria com que a população convergisse novamente em algumas iterações.

2.3 Aplicações de algoritmos meméticos

Esta seção traz uma avaliação das numerosas aplicações de MAs. Esta avaliação é longe de ser exaustiva visto que estão sendo desenvolvidas novas aplicações

continuamente. Porém, é pretendido que o aqui exposto seja ilustrativo do grande impacto prático destas técnicas de otimização.

2.3.1 Problemas "tradicionais" da classe NPO

Problemas tradicionais de otimização NP constituem um dos campos de batalha mais típicos de MAs. Uma história notável de sucessos pode ser considerada com respeito à aplicação de MAs para problemas NP-hard como o seguinte: Graph Partitioning [43] [44] [254] [257] [258], Min Number Partitioning [34], Max Independent Set [3] [162] [337], Bin-Packing [329], Min Graph Coloring [79] [120] [104] [77], Set Covering [28], Single Machine Scheduling with setup-times and due-dates [223] [124] [125] [262] [123] [247] [246], Parallel Machine Scheduling [69] [71] [264] [248], Min Generalised Assignment [73], Multidimensional Knapsack [85] [29] [143], Nonlinear Integer Programming [356] Quadratic Assignment [41] [63] [251] [257] [256], Set Partitioning [224], e particularmente no Min Traveling Salesman Problem [139] [141] [276] [323] [128] [127] [142] [252] [196] [172] [47] [257] [45] [46].

Relativo à teoria de completude de problemas NP a maioria deles pode ser citada como *"clássicos"* ou *"tradicionais"*, pois eles apareceram no notório trabalho de Karp [192] na redutibilidade de problemas combinatoriais. Notavelmente, na maioria deles a reivindicação dos autores é que eles desenvolveram a melhor heurística para o problema. Isto é importante, visto que estes problemas foram atacados com vários métodos diferentes da caixa de ferramentas da otimização combinatória e quase todas as técnicas algorítmicas de propósito geral foram testadas neles.

2.3.2 Outros Problemas de Otimização Combinatória

O sucesso do paradigma dos MAs não é limitado aos acima mencionados "problemas clássicos". Existem outros problemas adicionais "não clássicos"; eles são problemas de otimização combinatória de complexidade semelhante ou mais alta para os quais a resolução por MAs revelou-se como uma técnica excelente. Como um exemplo destes problemas, podemos citar *emparelhamento parcial de formas* [302], *paisagems de aptidão do modelo NK de Kauffman* [255], *projeto de trajetorias espaciais ótimas* [88], *alocação de frequencias* [195], *"multiperiod network design"* [130], *projeto de árvores geradoras mínimas com restrições* [324], *"uncapacitated hub location"* [1], *telecomunicações* [205] [207], *"adaptive distributed management"* [206], *"placement problems"* [338] [173] [214], *roteamento de veículos* [32] [177], *problemas de transporte* [133] [295], *problemas de designação de tarefas* [154], *escalonamento de tarefas de mantenimento* [54] [55] [56], *"open shop scheduling"* [112] [70] [228], *"flowshop scheduling"* [281] [282] [64] [121] [122], *escalonamento de projetos* [292] [325] [303], *"warehouse scheduling"* [366], *planejamento da produção"* [101] [265], *confeção de horários* [232] [48] [305] [326] [268] [269] [49] [50] [53] [52] [306] [358] *"rostering"* [267] [97], *escalonamento de jogos deportivos* [78], e *otimização combinatória multiobjetivo* [202] [203] [201] [204]. Obviamente, esta lista não está completa e seu

propósito é simplesmente documentar a aplicabilidade para um amplo espectro de problemas de otimização combinatória.

2.3.3 Aprendizado de máquinas e robótica

Aprendizado de máquinas e robótica são dois campos próximos e relacionados, visto que as tarefas diferentes envolvidas no controle de robôs são implementados empregando comumente redes neuronais e sistemas classificadores artificiais. MAs, geralmente citados como "algoritmos genéticos híbridos", foram usados em ambos os campos, i.e., em geral problemas de otimização relacionados a aprendizado de máquinas (por exemplo, o treinamento de redes neuronais artificiais), e em aplicações de robótica. Com respeito ao anterior, MAs foram aplicados em *treinamento de redes neuronais* [272] [383] [360] [176], *"pattern recognition"* [4], *"pattern classification"* [212] [260], e *análise de séries temporais* [299]. Sobre a aplicação de MAs para robótica, trabalho tem sido feito em *"reactive rulebase learning in mobile agents"* [87], *"path planning"* [298] [377] [316], *"manipulator motion planning"* [332], *"time optimal control"* [66], etc.

2.3.4 Eletrônica e engenharia

Eletrônica e engenharia também são dois campos nos quais estes métodos foram ativamente usados. Por exemplo, com respeito a problemas de engenharia, trabalho foi feito nas áreas seguintes: *otimização de estruturas* [384], *modelagem de sistemas* [365], *projetos na aeronáutica* [319] [40], *"trim loss minimization"* [300], *controle de tráfego* [349], e *"power planning"* [362]. Sobre aplicações práticas no campo de eletrônica, a lista seguinte pode ilustrar as numerosas áreas nas quais estas técnicas foram utilizadas: *manufatura de semicondutores* [199], *"projeto de circuitos* [156] [150] [370], *projeto assistido por computador* [30], *"multilayered periodic strip grating"* [19], *síntese de redes analógicas* [148], e *restauração de serviços* [14].

2.3.5 Problemas de otimização molecular

Selecionamos esta classe particular de problemas computacionais que envolvem otimização não-linear, para ajudar o leitor a identificar uma tendência comum na literatura. Infelizmente, os autores continuam se referindo à técnica como 'genética', embora ela seja mais próxima em espírito aos MAs. É válido recordar que o relatório de Caltech que deu seu nome ao, naquele momento incipiente, campo dos MAs [270] discute uma metaheurística que pode ser vista como um híbrido de GAs e SA, desenvolvido em colaboração com M.G. Norman em 1988. Em anos recentes, vários híbridos de GAs com SA ou outros métodos foram aplicados numa variedade de problemas de otimização molecular [304] [386] [345] [344] [235] [26] [98] [372] [189] [227] [149] [219] [185] [388] [91] [263] [103] [249] [239] [357] [129]. Muitos desses métodos de população quase não poderiam ser catalogados como sendo 'genéticos', mas esta denominação apareceu em um trabalho prévio de Deaven e Ho [99] e depois foi citado por J. Maddox em *Nature* [240]. Outros campos de aplicação incluem *"cluster physics"* [291]. Um esforço adicional tem sido feito em [317] [166] [318] [372] [164]

[100]. Outros métodos evolutivos para uma variedade de problemas moleculares podem ser achados em: [363] [327] [103] [259] [245] [163] [158]. O uso deles para problemas de projeto de moléculas está atraindo particular atenção [189] [375] [74]. Eles também foram aplicados na tarefa de projetar proteínas [102] [222], e *"molecular geometry optimization"* [168], provavelmente devido a estrutura da paisagem de energia associada a estes problemas (veja a discussão relativa a proteínas em [272]).

2.3.6 Outras aplicações

Além das áreas de aplicação descritas acima, foram utilizados também MAs em outros campos como, por exemplo, *medicina* [367] [153] [152], *economia* [226] [301], *oceanografia* [283], *otimização de funções* [368], *simulação de equações diferenciais ordinarias imprecisas* [330], *solução global de funções não-diferenciaveis não-lineares* [369], *"imaging science and speech processing"* [244] [58] [213] [333] [385], etc. A lista completa de aplicações de MAs requereria um livro inteiro, como o leitor pode conferir facilmente examinando qualquer banco de dados bibliográfico para as palavras chaves *'memetic-algorithm'* e *'hybrid-genetic-algorithm'*. Tentamos ser ilustrativos em lugar de exaustivos, mostrando algumas referências selecionadas para áreas famosas de aplicação. Isto significa que, com grande probabilidade, podem ter sido inadvertidamente omitidas muitas contribuições importantes.

Capítulo 3

Aproximabilidade

O objetivo deste capítulo é revisar as notações fundamentais sobre aproximabilidade em problemas de otimização. Os resultados e exemplos discutidos neste capítulo podem ser encontrados em vários livros-texto e teses, assim como em artigos de pesquisa. Foi feita uma tentativa de integrá-los em uma discussão auto-suficiente que ajudaria o leitor não especialista a entender rapidamente os conceitos principais. É difícil pensar em um "único" tipo de leitor a quem este capítulo deveria se ajustar. Assim, tivemos que adotar um compromisso e confiamos em uma variedade de fontes que podem ser usadas para complementar a leitura: o Capítulo 3 de [221], assim como os livros [225], [307], [15]. P. Crescenzi e V. Kann iniciaram alguns anos atrás a mais compreensiva coleção de resultados em aproximabilidade[1] disponível na web. Também em todo esse trabalho tentamos concordar com suas definições e as da tese de doutorado de V. Kann.

3.1 Definições das classes NP, NPO e PO

Um artigo fundador na área de aproximabilidade de problemas de otimização NP foi escrito por D.S. Johnson e publicado em 1974 [178]. As definições que seguimos aqui são de trabalhos posteriores como o de Crescenzi e Panconesi [90], assim como também da tese de doutorado de Viggo Kann e do livro [15]. Primeiramente definiremos os problemas de otimização da classe NPO, para os quais a função objetivo é uma função computável em tempo polinomial e a factibilidade de uma dada solução pode ser checada em tempo polinomial.

Definição: Um problema NPO (em um alfabeto Σ) é uma quádrupla $F = (I_F, sol_F, m_F, goal_F)$ onde

- $I_F \subseteq \Sigma^*$ é o espaço de *instâncias de entrada* do problema F. É reconhecível em tempo polinomial.
- $sol_F(x) \subseteq \Sigma^*$ é o espaço de *soluções factíveis* na entrada $x \in I_F$. A única necessidade em sol_F é que exista um polinômio q e um predicado

[1] `http://www.nada.kth.se/~viggo/wwwcompendium/`

computável em tempo polinomial π tal que para todo $x \in I_F$, sol_F possa ser expresso como $sol_F(x) = \{y : |y| \leq q(|x|) \wedge \pi(x, y)\}$ onde q e π dependem somente de F.

- $m_F : I_F \times \Sigma^* \to \mathbb{N}$, é chamada de *função objetivo*, e é uma função computável em tempo polinomial . O valor $m_F(x, y)$ é definido somente quando $y \in sol_F(x)$.

- $goal_F \in \{\max, \min\}$ nos diz se F é um problema de *maximização* ou de *minimização*.

Definição: Resolver um problema de otimização F dada a entrada $x \in I_F$ significa encontrar um $y \in sol_F(x)$ tal que $m_F(x, y)$ é ótimo, e dizer, que seja tão grande quanto possível se $goal_F = \max$ e tão pequeno quanto possível se $goal_F = \min$.

Definição: Dada a entrada $x \in I_F$, o valor $opt_F(x)$ é o valor ótimo de m_F. Isto significa que é o valor do custo da função objetivo do mínimo ou máximo atingível para uma solução factível para $goal_F = \max$ ou $goal_F = \min$, respectivamente.

Uma solução *máxima* de um problema de otimização F é uma solução ótima para $goal_F = \max$ (analogamente será *mínima* para $goal_F = \min$). Assim uma solução *"localmente ótima"* é uma solução *maximal*, e pode ser vista como uma "aproximação" para a solução ótima. Para muitos problemas de otimização NP é muito facil encontrar uma solução maximal mas é difícil encontrar uma solução máxima. Analogamente, para um problema de minimização, uma solução localmente ótima é chamada uma solução *minimal*.
Por exemplo, podemos ver que ambas as Figuras 1.4 e 1.5 são soluções factíveis para o problema MIN VERTEX COVER mas não são *1-minimais* pois há no mínimo um caso em que, removendo um vértice do conjunto de vértices do cobrimento, ainda temos uma solução factível para o problema.

Exemplo: O problema de MIN VERTEX COVER é então um exemplo de um problema em NPO. Seguindo a definição proposta, o problema pode ser representado pela tupla $(I, sol, m, goal)$ onde o espaço de instâncias de entrada I contém todos os grafos não dirigidos $G(V, E)$. O conjunto de soluções factíveis e todo subconjunto de vértices $V' \subseteq V$ para os quais toda aresta $(u, v) \in E$, u ou v (ou ambos) pertencem a V'. A função objetivo e a cardinalidade de V', e o $goal = \min$.

Exemplo: O problema MAX THREE-DIMENSIONAL MATCHING é outro exemplo de um problema NPO. A definição de emparelhamento em um grafo é dada em 3.2.7. Porém, nesse caso, o espaço de instâncias de entrada I contém todos os conjuntos de triplas $T \subseteq X \times Y \times Z$ de conjuntos disjuntos a pares X, Y e Z. O conjunto de soluções factíveis $sol_{\text{MAX3DM}}(T)$ é o conjunto de emparelhamentos de T, onde um emparelhamento $M \in sol_{\text{MAX3DM}}(T)$ é um subconjunto de T onde não há duas triplas que concordem em nemhuma coordenada. A função objetivo $m(T, M)$ é, neste caso, a cardinalidade $|M|$ da solução M e $goal = \max$.

Relacionado com o problema de otimização, existe a *versão de decisão* 3DM que é um conhecido problema NP-completo. Por esse motivo, é amplamente

conjeturado que o problema de otimização MAX 3DM não pode ser resolvido exatamente em tempo polinomial. A relação entre problemas de otimização e as versões de decisão são discutidas mais à frente, em 3.1.1.

Definição: PO é a classe de problemas NPO que podem ser resolvidos em tempo polinomial.

Exemplo: MAX MATCHING IN A BIPARTITE GRAPH (MAX 2DM) é um exemplo de um problema na classe PO.

3.1.1 Auto-redutibilidade

Há uma importante propriedade de SATISFIABILITY e outros problemas NP-completos que é chamada de *auto-redutibilidade*. Ilustraremos como podemos usar um algoritmo para a versão de decisão de um problema para resolver a versão de otimização. Para problemas de otimização, outra técnica será introduzida, a chamada *busca binária*. É possível ilustrar seu funcionamento usando o MIN TSP, outro bem conhecido problema NPO, como nosso exemplo.

Se nosso objetivo original é resolver o MIN TSP, i.e. nossa tarefa é fornecer o tour ótimo para uma dada instância, podemos resolver este problema usando um hipotético algoritmo para a versão de decisão, o TSP (D). Começamos encontrando o custo da solução ótima por um método de busca binária que funciona da seguinte maneira: primeiro, notamos que o custo ótimo é sempre um inteiro entre 0 e 2^n, onde n é o comprimento da codificação da instância. Como uma conseqüência, podemos encontrar o custo perguntando primeiro se, com as cidades dadas e as distâncias dadas, o *"budget"* 2^{n-1} é atingível. Dependendo da resposta, se o *"budget"* 2^{n-2} ou $2^{n-2}+2^{n-3}$ é atingível, e assim por diante, reduzindo o número de valores para o custo ótimo pela metade em cada passo.

Finalmente, o valor do custo ótimo é encontrado, e o chamamos de *opt* como uma abreviação de $opt_{\text{MIN TSP}}(x)$; agora devemos resolver vários problemas de decisão TSP (D) nos quais o budget sempre será *opt*, mas as distâncias entre as cidades serão alteradas, assim estamos trabalhando com outra instância x'. Em particular, tomamos todas as distâncias entre as cidades uma por uma, e perguntamos se cada uma destas instâncias modificadas, somente diferindo em uma particular distância fixa para $opt+1$, tem um tour de custo *opt* ou menos. Se sim, então obviamente o tour ótimo não usa esta distância, assim podemos congelar o custo para $opt + 1$ sem maiores preocupações. Se, entretanto, a resposta for "Não", então é seguro concluir que tal distância é "indispensável" para o tour ótimo, então a armazenamos para ter seu valor original. É fácil ver que, depois que n^2 distâncias tenham sido processadas, os únicos valores da matriz de distâncias que são menores que $opt + 1$ serão n arestas que são utilizadas no tour ótimo. Todo o processo apenas requer um número polinomial de chamadas para o hipotético algoritmo para o TSP (D).

3.1.2 Subotimalidade

Já observamos que para muitos problemas NPO é fácil encontrar uma solução maximal mas é difícil encontrar uma máxima. Mas o que podemos afirmar

sobre o problema de *reconhecer a otimalidade de uma solução quando é dada como entrada ?* Infelizmente, em geral estas questões também correspondem a problemas que podem ser NP-completos, como mostraremos. Recorremos novamente ao MIN TSP como exemplo:

SUBOTIMALIDADE DO TSP
Instância: Uma instância (n, C) do MIN TSP e um tour π de n cidades.
Questão: π é *subótima*, i.e., há outro tour π' de comprimento menor?

Teorema: (Papadimitriou e Steiglitz, 1977) *Dado um grafo $G = (V, E)$ com um circuito Hamiltoniano C, é um problema* NP*-completo determinar se G tem um segundo ciclo Hamiltoniano.*

Prova: O resultado segue a partir da transformação do problema HAMILTONIAN CYCLE o qual é um problema NP-completo. Suponha que $G(V, E)$ é um grafo. Um circuito hamiltoniano artificial pode ser adicionado a G, tal que as arestas deste ciclo não podem se 'misturar' com outras arestas de G para formar um novo ciclo Hamiltoniano. Isto pode ser feito substituindo cada vértice $v \in V$ de G pelo subgrafo S descrito na Fig. 3.1. Note que o subgrafo S pode somente ser atravessado por um circuito hamiltoniano de somente dois modos, mostrados (veja 3.1, centro e direita).

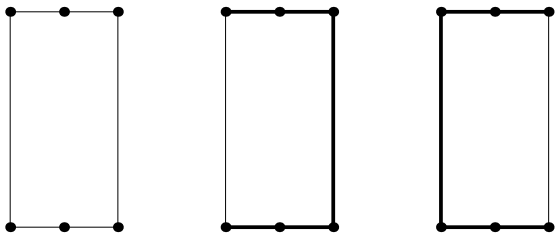

Figura 3.1: *Três copias do "gadget" de "OU-exclusivo". O grafo da esquerda só pode ser percorrido nas duas maneiras que são mostradas (centro e direita).*

Dado que este grafo substituirá cada vértice v de G, chamaremos $A(v)$ ($B(v)$) o vértice do canto superior esquerdo (respectivamente, inferior esquerdo), e $C(v)$ ($D(v)$) o vértice do canto superior direito (respectivamente, inferior direito). Isto significa que há um caminho começando de $A(v)$ que atravessa o grafo, encontrando $C(v)$, $D(v)$, e finalmente $B(v)$ nessa ordem relativa. A única outra maneira de atravessar todos os vértices é começando em $C(v)$ e visitando $A(v)$, $B(v)$, e finalmente $D(v)$ nesta ordem.

Então procedemos adicionando as arestas $\{A(u), B(v)\}$ e $\{A(v), B(u)\}$ para cada arco $\{u, v\} \in E$. Obtemos nosso circuito hamiltoniano artificial ordenando os vértices de G (arbitrariamente) como $v_1, v_2, \ldots, v_n$ e adicionando as arestas $\{C(v_i), D(v_{i+1})\}$, $1 \leq i < n$ e $\{C(v_n), D(v_1)\}$. Pelo procedimento de construção, todas as cópias de S devem ser atravessadas do mesmo modo. Se são atravessadas encontrando $C(v)$, $A(v)$, $B(v)$, e $D(v)$ nessa ordem relativa (Fig. 3.1 direita) então temos nosso circuito hamiltoniano artificial. Se eles são

atravessados seguindo a seqüência $A(v)$, $C(v)$, $D(v)$, $B(v)$ (Fig. 3.1 centro), então obtemos um circuito hamiltoniano se e somente se G é hamiltoniano. Isto completa a prova.

Existem outras novas más notícias em relação à subotimalidade. O seguinte teorema pode ser visto como um corolário do primeiro.

Teorema: SUBOTIMALIDADE do TSP *é* NP*-completo mesmo no caso de matrizes de distância simétricas que obedecem a desigualdade triangular.*

Prova: (Pela inclusão de um caso especial) Faça $G' = (V', E')$ ser o grafo produzido pela construção usada no teorema anterior. Associaremos agora os vértices em V' com as cidades do tipo particular de instância do MIN TSP considerada. Faça as distâncias entre as cidades $d(u, v)$ serem definidas como a seguir.

$$d(u,v) = \begin{cases} 4, & \text{se}\{\text{u, v}\} \notin \text{E}' \\ 3, & \text{se}\{\text{u, v}\} = \{\text{C}(\text{v}_\text{n}), \text{D}(\text{v}_1)\}; \\ 2, & \text{se}\{\text{u, v}\} \in \text{E}' - \{\text{C}(\text{v}_\text{n}), \text{D}(\text{v}_1)\} \end{cases}$$

A matriz de distância correspondente é simétrica e satisfaz a desigualdade triangular. O comprimento do tour possível mais curto é obviamente $2|V'|$, e, além disso, tal tour não pode usar qualquer aresta que não está em G' ou uma aresta 'artificial' (desde que se usada uma aresta artificial, teria que usar $\{C(v_n), D(v_1)\}$). Então um tour de comprimento $2|V'|$ existe (e nosso tour artificial, tendo comprimento $2|V'| + 1$, seria subótimo se e somente se o grafo original tivesse um circuito hamiltoniano). Isto completa a prova.

3.2 Medidas de aproximabilidade

Vimos que a aproximação de um problema de otimização F dado com entrada $x \in I_F$ significa encontrar qualquer $y' \in sol_F(x)$. Quão boa é a aproximação depende do "gap" entre $m_F(x, y')$ e $opt_F(x)$. Entretanto, há muitos critérios e maneiras de se medir como "julgar" quão boa é uma aproximação. Problemas diferentes têm diferentes aproximabilidades, as quais devem ser medidas de maneiras diferentes. As mais comumente utilizadas são *erro relativo*, *erro absoluto*, *respeitosa do custo*, *razão de desempenho* e *melhor razão de desempenho assintótica*. Nesta seção explicaremos, por motivos de completude, essa terminologia que é um padrão de fato.

3.2.1 Erro relativo

A medida de erro relativo é mais comumente utilizada para problemas que não são muito difíceis de aproximar. Podemos formalmente definir:

Definição O *erro relativo* de uma solução factível com respeito ao ótimo de um problema NPO F é definido como

$$\mathcal{E}_F^r(x, y) = \frac{|opt_F(x) - m_F(x, y)|}{opt_F(x)}$$

onde $y \in sol_F(x)$.

Se F é um problema de maximização temos $opt_F(x) \geq m_F(x,y) \geq 0$ e

$$\mathcal{E}_F^r(x,y) = \frac{opt_F(x) - m_F(x,y)}{opt_F(x)} = 1 - \frac{m_F(x,y)}{opt_F(x)}$$

Assim $0 \leq \mathcal{E}_F^r(x,y) \leq 1$.

Se F é um problemas de minimização, temos $m_F(x,y) \geq opt_F(x) \geq 0$ e

$$\mathcal{E}_F^r(x,y) = \frac{m_F(x,y) - opt_F(x)}{opt_F(x)} = \frac{m_F(x,y)}{opt_F(x)} - 1 \geq 0$$

3.2.2 Erro relativo normalizado

É claro que a medida anterior não é "simétrica", então para classificar esta situação Ausiello, D'Atri e Protasi definiram o *erro relativo normalizado* para os problemas de maximização e minimização do seguinte modo:.

Definição: [16] O *erro relativo normalizado* de uma solução factível de um problema NPO F é definido como

$$\mathcal{E}_F^n(x,y) = \frac{|opt_F(x) - m_F(x,y)|}{|opt_F(x) - worst_F(x)|}$$

onde $y \in sol_F(x)$ e

$$worst_F(x) = \begin{cases} \min\{m_F(x,y) : y \in sol_F(x)\} & \text{se } goal_F = \max, \\ \max\{m_F(x,y) : y \in sol_F(x)\} & \text{se } goal_F = \min. \end{cases}$$

A partir da definição, pode ser facilmente observado que $0 \leq \mathcal{E}_F^n(x,y) \leq 1$. Para problemas de maximização $worst_F(x)$ é usualmente zero, e neste caso o erro relativo normalizado coincidirá com o erro relativo. O erro relativo normalizado é dificilmente utilizado na literatura, provavelmente porque não é intuitivo. Veja a Seção 6.4 para um exemplo de uso do erro relativo normalizado.

3.2.3 Erro absoluto

Definição: O *erro absoluto* de uma solução factível com respeito ao ótimo de um problema NPO F é definido como

$$\mathcal{E}_F^a(x,y) = |opt_F(x) - m_F(x,y)|$$

onde $y \in sol_F(x)$.

Para muitos problemas de otimização com variantes de decisão NP-completos, não há esperança de encontrar uma solução com um erro absoluto constante. Conseqüentemente, esta medida é raramente utilizada. Ainda há alguns poucos problemas NPO que têm algoritmos de aproximação com erro absoluto constante garantido, veja [167].

3.2.4 Medida respeitosa do custo

Estas medidas foram generalizadas por Orponen e Mannila [297]. Eles deram uma medida mais geral da qualidade da aproximação.

Definição: [297] Uma *medida de qualidade de aproximação* geral de uma solução factível de um problema NPO F é uma função $\mathcal{E}_F$ que, para cada $x \in I_F$, satisfaz $\mathcal{E}_F(x,y) \geq 0$ se e somente se $y \in sol_F(x)$ e $\mathcal{E}_F(x,y) = 0$ se e somente se $m_F(x,y) = goal_F(x)$.

Definição: Dizemos que a medida é *respeitosa do custo* se

$$\begin{array}{ll} m_F(x,y_1) \leq m_F(x,y_2) \Rightarrow \mathcal{E}_F(x,y_1) \geq \mathcal{E}_F(x,y_2) & \text{se } goal_F = \max, \\ m_F(x,y_1) \leq m_F(x,y_2) \Rightarrow \mathcal{E}_F(x,y_1) \leq \mathcal{E}_F(x,y_2) & \text{se } goal_F = \min. \end{array}$$

Todas as medias definidas acima são respeitosas do custo.

3.2.5 Razão de desempenho

Para analisar um algoritmo de aproximação, na prática a maneira mais comum de expressar a aproximabilidade é dar uma razão de desempenho.

Definição: A *razão de desempenho* de uma solução factível com respeito ao ótimo de um problema NPO F é definido como

$$R_F(x,y) = \begin{cases} opt_F(x)/m_F(x,y) & \text{se } goal_F = \max, \\ m_F(x,y)/opt_F(x) & \text{se } goal_F = \min. \end{cases}$$

onde $x \in I_F$ e $y \in sol_F(x)$.

Existe uma clara relação entre a razão de desempenho e o erro relativo:

$$R_F(x,y) = \begin{cases} \dfrac{1}{1-\mathcal{E}_F^r(x,y)} = \displaystyle\sum_{i=0}^{\infty} (\mathcal{E}_F^r(x,y))^i & \text{se } goal_F = \max, \\ 1 + \mathcal{E}_F^r(x,y) & \text{se } goal_F = \min. \end{cases}$$

3.2.6 Aproximação dentro de uma constante

Agora, colocaremos um exemplo de um algoritmo que fornece uma constante de aproximação para o MIN TSP. Primeiro definiremos:

Definição: Dizemos que um problema de otimização F *pode ser aproximado dentro de* p para uma constante p se existe um algoritmo de tempo polinomial A tal que, para todas as instâncias $x \in I_F$, $A(x) \in sol_F(x)$ e $R_F(x, A(x)) \leq p$.

3.2.7 Exemplo: O algoritmo de Christofides

Quase um quarto de século atrás, Nicos Christofides apresentou um algoritmo que encontra em tempo polinomial um tour de comprimento no máximo de $3/2$ vezes o tour mais curto possível para uma instância simétrica do TSP se as distâncias entre cidades obedecem a desigualdade triangular (este caso é conhecido como o TSP *métrico*) [72]. Para o caso Euclideano, o qual é um caso especial do TSP métrico, este resultado permaneceu como desafio até 1996

quando Arora e Mitchell independentemente desenvolveram um *ptas* diferente para este caso. Infelizmente, para o caso geral é ainda o melhor resultado conhecido.

Definição: Um *grafo euleriano* é um grafo conectado no qual cada vértice tem um grau par.

Não é uma tarefa difícil mostrar que grafos eulerianos são precisamente os grafos que contém um tour euleriano, definidos como a seguir:

Definição: Um *tour euleriano* de um grafo $G(V, E)$ é um ciclo em G que passa por cada aresta exatamente uma vez.

Dado um grafo euleriano $G(V, E)$, um tour Euleriano pode ser computado em tempo $O(|V|)$. Outro conceito de Teoria de Grafos tem um papel central no algoritmo de Christofides. Definimos:

Definição: Dado um grafo $G(V, E)$ um *emparelhamento* M de G é uma coleção de arestas tais que cada ponto final é um vértice de V e é exatamente um elemento de M.

Definição: Dado um grafo com pesos nas arestas $G(V, E, W)$, um *emparelhamento de menor peso* de um conjunto de vértices $V' \subseteq V$ é um emparelhamento M para o qual a soma total de pesos é mínima.

Algoritmo de Christofides [72])
INPUT: Uma instância métrica do MIN TSP, (MIN METRIC TSP), i.e. um grafo completo $G = (V, E, W(E))$ com pesos não negativos nas arestas $w : E \to \mathbb{N}$ que satisfazem a desigualdade triangular.
OUTPUT: Um tour de G de comprimento no máximo $3/2$ vezes o comprimento do mínimo tour.

ALGORITMO DE CHRISTOFIDES

1. Ache a *árvore geradora mínima* ("minimum spanning tree") de G.
2. Identifique V', o conjunto de vértices da *árvore geradora mínima* que tem grau ímpar.
3. Ache um *emparelhamento de menor peso* do grafo induzido por V' e adicione as arestas do emparelhamento à árvore geradora mínima. Agora todo vértice nesse grafo aumentado tem grau par. Então o grafo é euleriano.
4. Ache o *tour euleriano* nesse grafo aumentado.
5. Converta o tour euleriano num tour do TSP tomando "atalhos" nos vértices que já tenham sido visitados.

Analisaremos agora o algoritmo. Uma *árvore geradora mínima* de G pode ser construída em tempo $|V|^2$. Em relação a V', note que a cardinalidade dele é sempre par. A soma dos graus de todos os vértices é um número par (devido a que cada aresta aumenta em dois a soma, para cada um dos vértices dos extremos). Então a cardinalidade de V' (os vértices de grau impar) tem que ser um número par. Isso mostra que o grafo é euleriano.

Consideramos o grafo induzido em G por V'. Temos que analisar qual é o limitante da soma de suas arestas (pois pode ser possível que G já seja um

circuito euleriano). Seja *opt* o comprimento do tour ótimo do problema de MIN TSP. Se consideramos os vértices em V' na ordem em que aparecem nesse tour, podemos determinar dois emparelhamentos M e M' de V' e cada aresta num desses emparelhamentos não pode ter maior comprimento que o segmento correspondente em C_{opt} (pela desigualdade triangular). Então se L indica o comprimento de um emparelhamento M, temos $L(M) + L(M') \leq opt(G)$, então ou M ou M' deve ter comprimento menor ou igual a $opt(G)/2$. Então, como M e M' são dois emparelhamentos dos vértices em V', o comprimento do emparelhamento de peso mínimo deve ser não maior que $opt(G)/2$. Como é conhecido que o cumprimento da árvore geradora mínima é menor que *opt*, temos que o comprimento de G é no máximo $1,5 \cdot opt$.

Cada passo do algoritmo pode ser feito em tempo polinomial, é a complexidade e dominada pelo problema de emparelhamento com uma complexidade de $O(|V|^3)$.

3.2.7.1 Aproximação assintótica dentro de uma constante

Na literatura, é possível encontrar problemas nos quais o caso interessante a considerar é a razão de desempenho apenas para instâncias com valor ótimo alto. Esta exigência, que é de medir o comportamento assintótico, é coberta pela seguinte definição

Definição: Um problema de otimização F *pode ser aproximado assintoticamente em* p para uma constante p se existe um algoritmo de tempo polinomial A e uma constante positiva N tal que para todas as instâncias $x \in I_F$ com $opt_F(x) \geq N$, $A(x) \in sol_F(x)$ e $R_F(x, A(x)) \leq p$.

3.2.7.2 Melhor razão de desempenho

Definição: A *melhor razão de desempenho* $\mathcal{R}[F]$ para um problema de otimização F é definida como

$$\mathcal{R}[F] = \inf\{p : F \text{ pode ser aproximado em } p\}.$$

3.2.7.3 Melhor razão de desempenho assintótico

Definição: A *melhor razão de desempenho assintótico* $\mathcal{R}^{\infty}[F]$ para um problema de otimização F é definida como

$$\mathcal{R}^{\infty}[F] = \inf\{p : F \text{ pode ser aproximada assintoticamente em } p\}.$$

3.2.7.4 Aproximação limitada

Definição Um problema de otimização F tem uma *aproximação limitada* se existe uma constante positiva c tal que F possa ser aproximado em c.

3.2.8 Aproximação dentro de uma função

Podemos estender a definição 3.2.6 para problemas que não têm aproximação limitada do seguinte modo.

Definição Um problema de otimização F *pode ser aproximado em* $p(n)$ para uma função $p : \mathbb{Z}^+ \to \mathbb{R}^+$ se existe um algoritmo de tempo polinomial A tal que para cada $n \in \mathbb{Z}^+$ e para todas as instâncias $x \in I_F$ com $|x| = n$ temos que $A(x) \in sol_F(x)$ e $R_F(x, A(x)) \leq p(n)$.

3.3 Esquemas de aproximação

3.3.1 Esquemas de aproximação em tempo polinomial

Por definição, um algoritmo de aproximação de tempo polinomial que é capaz de aproximar um problema NPO F dentro de $1+\epsilon$ com $\epsilon = 0$ encontra a solução ótima para o problema em tempo polinomial. Para um problema NP-completo, a menos que P = NP, não há esperança de encontrar tal algoritmo. Mas, para muitos problemas NP-completos existem algoritmos que aproximam dentro de $1 + \varepsilon$ para cada $\varepsilon > 0$. Tal algoritmo é chamado de *esquema de aproximação em tempo polinomial*, ou *ptas*, pelo nome em lingua inglesa da classe PTAS de problemas que admitem um *"polynomial-time approximation scheme"*. Essa classe representa os problemas considerados "fáceis" na Fig. 1.1.4 e a classe é definida como segue:

Definição: Um problema de otimização F tem um *esquema de aproximação de tempo polinomial* ou (*ptas*) se existe um algoritmo de aproximação de tempo polinomial (no comprimento da entrada da instância) que tem como entrada uma instância $x \in I_F$ e uma constante $\varepsilon > 0$, e então resulta uma solução que aproxima F em $1 + \varepsilon$.

Observamos que uma outra definição equivalente é que F tenha um *ptas* se há um algoritmo de aproximação em tempo polinomial que tem como entrada a instância $x \in I_F$ e uma constante $\varepsilon > 0$, e fornece como resultado uma solução que aproxima F com um erro relativo menor que ε.

3.3.2 O papel da planaridade e o TSP

É freqüente o caso que um problema NPO, restrito a instâncias que podem ser ditas "planas", pode ser significantemente fácil de resolver. O problema, restrito a todas as instâncias que são deste tipo, pode constituir um caso especial do problema que pode ser colocado em uma diferente classe de complexidade.

Por exemplo, encontrar um conjunto máximo de vértices independentes em um grafo planar é um exemplo de um problema que tem um *ptas* [23]. Mas a complexidade deste *ptas* é infelizmente $O(n \cdot 8^{1/\varepsilon}/\varepsilon)$ onde n é o número de vértices no grafo, e assim o tempo aumenta muito rapidamente com o decrescimento de ε.

Um caso especial do MIN METRIC TSP é onde as distâncias entre cidades são dadas pela métrica do caminho mínimo de um grafo planar sem peso nas arestas (i.e. todas os "custos" das arestas valem um). Em 1995, Grigni, Koutsoupias e Papadimitriou apresentaram um *ptas* para este problema [146]. O esquema alcança uma razão de aproximação de $(1 + \epsilon)$ no tempo de $O(n^{1/\epsilon})$, onde $\epsilon > 0$ é qualquer instância e n é o número de cidades.

3.3.3 Esquema de aproximação em tempo completamente polinomial

Um problema representativo que pertence a esta classe é o MAX KNAPSACK. Também na Fig. 1.1.4 observamos uma classe como sendo "muito fácil". Estes são os problemas em PTAS para os quais a complexidade tem uma dependência polinomial em $1/\varepsilon$. Um *ptas* com tal complexidade é chamado de *esquema de aproximação em tempo completamente polinomial*. Um *fptas* para o MIN KNAPSACK foi introduzido em [175]. A definição formal é:

Definição: Um problema de otimização F tem um *esquema de aproximação completamente polinomial* (*fptas*) se tem um algoritmo de aproximação que tem como entrada uma instância $x \in I_F$ e uma constante $\varepsilon > 0$, e então, em tempo polinomial em $1/\varepsilon$ e o comprimento de x, resulta uma solução que aproxima F em $1 + \varepsilon$.

3.3.4 Algoritmos pseudo-polinomiais

Este é o lugar certo para discutir um assunto relacionado, o que trata de *algoritmos pseudo-polinomiais* Seguiremos a discussão do livro de Lewis e Papadimitriou; primeiro introduziremos o seguinte problema, chamado INTEGER PARTITION em [225]:

EXACT NUMBER PARTITION

Entrada: Um conjunto A de n números inteiros positivos $\{a_1, \ldots, a_n\}$.

Questão: Há uma partição de A, i.e. dois conjuntos disjuntos A_1 e A_2 com $A = A_1 \bigcup A_2$, tal que

$$\sum_{a_i \in A_1} a_i = \sum_{a_j \in A_2} a_j \quad ? \tag{3.1}$$

Seja $H = (\sum_{i=1}^{n} a_i)/2$, se H não é um inteiro, isto significa que os números em S tem como soma um número ímpar e o conjunto não pode ser particionado em somas iguais, então neste caso a resposta é 'Não'. Para cada índice $i = 0, \ldots, n$, definimos:

$$B(i) = \{b \leq H : b \text{ é a soma de alguns subconjuntos de números } \{a_1, \ldots, a_n\}\}$$

Se soubéssemos $B(n)$, poderíamos resolver o problema facilmente apenas testando se $H \in B(n)$. Assim, se há uma soma parcial que termina somando H a resposta é "Sim", caso contrário é "Não". Agora mostraremos que $B(n)$ pode ser computado pelo algoritmo mostrado em 3.2.

Repetiremos aqui a instância particular apresentada em [225] (p. 285) que ilustra como o algoritmo trabalha. Dado o conjunto $\{38, 17, 52, 61, 21, 88, 25\}$, temos $H = 151$ e o $B(i)$ como computado pelo algoritmo mostrado na Fig. 3.2

$$B(0) = \{0\},$$
$$B(1) = \{0, 38\},$$
$$B(2) = \{0, 17, 38, 55\},$$

```
Procedure Recursive_computation_of_B(n)
begin
B(0) = 0
for i = 1, 2, ..., n do
    B(i) = B(i − 1);
    for j = a_i, a_i + 1, a_i + 2, ..., H do
        if j − a_i ∈ B(i − 1) then add j to B(i)
    endfor
endfor
end
```

Figura 3.2: *Um procedimento recursivo para resolver o problema de* EXACT NUMBER PARTITIONING

$$B(3) = \{0, 17, 38, 52, 55, 69, 90, 107\},$$
$$B(4) = \{0, 17, 38, 52, 55, 61, 69, 78, 90, 107, 113, 116, 130, 151\},$$

então, finalmente obtemos

$$B(7) = \{0, 17, 21, 25, 38, 42, 46, 52, 55, 59, 61, 63, 69,$$
$$73, 76, 77, 78, 80, 82, 84, 86, 88, 90, 94, 98, 99, 101, 103,$$
$$105, 107, 109, 111, 113, 115, 116, 124, 126, 128, 130,$$
$$132, 134, 136, 137, 138, 140, 141, 143, 147, 149, 151\}.$$

A resposta é "Sim" para esta instância, porque a metade da soma é $H = 151$ e está contida em $B(7)$ (porém, note que sua primeira ocorrência é em $B(4)$, então teríamos que parar o algoritmo lá). Por indução em i, podemos provar que o algoritmo computa corretamente $B(i)$, para $i = 0, \ldots, n$ e que requer tempo $O(nH)$.

Agora, obteremos a razão do porquê que estes algoritmos são ditos *pseudo-polinomiais*. Note que embora $O(nH)$ seja linear em n *não é polinomial em relação ao comprimento da entrada.* Assumimos aqui que os inteiros são dados como binários e assim sua soma será em geral um valor exponencialmente alto quando comparado com o comprimento da entrada. Por exemplo, se todos os n inteiros são em torno de 2^n, então $H \approx (n2^n)/2$, enquanto o comprimento da entrada é somente $O(n^2)$.

3.4 NPO PB, problemas NPO polinomialmente limitados

Uma importante classe de problemas de otimização NP é a dos problemas *polinomialmente limitados*. Mencionamos na Seção 1.3.1 que o LONGEST INDUCED PATH IN A GRAPH é um deles. A classe representa problemas NPO para os quais o tamanho da solução ótima é limitada por um polinômio no comprimento da instância correspondente [33, 42, 210].

Definição: Um problema NPO F é *limitado polinomialmente* se existe um polinômio p tal que

$$opt_F(x) \leq p(|x|) \text{ for all } x \in I_F,$$

onde $|x|$ é o tamanho da instância x. A classe de todos os problemas NPO limitados polinomialmente é chamada de NPO PB. Esta classe foi chamada de OPTP[$\log n$] por Krentel em [210].

Outros problemas NPO PB que podem ser citados nessa classe são o MAX THREE-DIMENSIONAL MATCHING e MAX INDEPENDENT SET. O último será objeto de análise posterior. Ao contrário, o MIN TSP e MIN $0-1$ PROGRAMMING são exemplos de problemas que não estão em NPO PB.

Observe que se existe um problema NPO PB NP-completo que tem um *fptas*, então podemos usar o *fptas* para encontrar a solução ótima em tempo polinomial. Assim, contanto que P $\neq$ NP, nenhum problema NPO PB NP-completo pode ter um *fptas*.

3.5 Resultados de não aproximabilidade

Alguns dos mais famosos problemas NP-completos tem versões que *não são aproximáveis*. Como este fato pode ser provado ? As demonstrações em geral costumam não ser fáceis. Algumas das mais desafiantes questões em aberto exigem o desenvolvimento de novos métodos matemáticos. Novamente, recorremos ao MIN TSP que pode ser usado para ilustrar a dificuldade de aproximação, embora a demonstração seja fácil nesse caso.

Suponha que seja dado um número alto ϵ, devemos provar que, a menos que P = NP, não há algoritmo de aproximação $(1+\epsilon)$ para o MIN TSP. Para provar a dificuldade, recorremos mais uma vez ao problema de decisão HAMILTONIAN CYCLE que assumimos já estar provado ser NP-completo. Para provar a dificuldade de aproximação, mostraremos que, se há um algoritmo de aproximação $(1+\epsilon)$ para o MIN TSP então há um algoritmo de tempo polinomial para o problema HAMILTONIAN CYCLE. Começamos com uma instância $G(V,E)$ do problema HAMILTONIAN CYCLE . Aplicamos a este uma simples redução de HAMILTONIAN CYCLE para o MIN TSP com as distâncias $d_{u,v}$ sendo agora computadas da seguinte maneira:

$$d(u,v) = \begin{cases} 0, & \text{se u = v} \\ 1, & \text{se (u, v)} \in \text{E;} \\ 2+|V|\epsilon, & \text{nos outros casos} \end{cases}$$

É fácil ver que a instância construída tem a seguinte interessante propriedade; se G tem um circuito hamiltoniano, então o tour ótimo da instância MIN TSP tem custo $|V|$; se, porém, não há um ciclo hamiltoniano, então o comprimento do tour ótimo é estritamente maior que $(1+\epsilon)|V|$. Isto segue do fato de que no mínimo uma aresta de comprimento $2+|V|\epsilon$ deve ser parte do tour. O mesmo vale para no mínimo as outras $|V|-1$ arestas de comprimento no mínimo 1.

O que acontece então se encontramos um algoritmo de aproximação $(1+\epsilon)$ de tempo polinomial A para o MIN TSP. Então seríamos capazes de dizer se G tem um ciclo hamiltoniano como segue: Primeiro, executamos o algoritmo A na instância dada de MIN TSP. Então temos duas classes a considerar:

- Se o tour resultante tem comprimento $\geq |V|(1+\epsilon)+1$ então sabemos que o ótimo não pode ser $|V|$, porque neste caso o erro relativo teria sido

$$\frac{||V|(1+\epsilon)+1-|V||}{|V|} > \epsilon$$

  que contradiz a hipótese de que A é um algoritmo de aproximação $1+\epsilon$. Dado que a solução ótima é maior que $|V|$, concluímos que G não tem um ciclo hamiltoniano.

- Se, porém, o tour tem um comprimento $\leq |V|(1+\epsilon)$, então sabemos que a solução ótima deve ser $|V|$. Isto segue do fato de que, por construção, a instância de MIN TSP não pode ter um tour de custo entre $|V|+1$ e $|V|(1+\epsilon)$. Então, neste caso G deve ter um ciclo hamiltoniano.

Então, podemos construir, em tempo polinomial de qualquer instância genérica G de HAMILTONIAN CYCLE, uma instância particular do MIN TSP tal que, se um algoritmo de aproximação $(1+\epsilon)$ de tempo polinomial existe para o MIN TSP então podemos decidir em tempo polinomial se G tem um ciclo hamiltoniano ou não. Dado que este argumento pode ser feito para qualquer $\epsilon > 0$, então provamos que *em geral* MIN TSP não é aproximável a menos que P = NP. Isto completa a prova.

De acordo com Arora e Lund [11], os recentes resultados de não aproximabilidade podem ser divididos em quatro classes, que são baseadas na razão de aproximabilidade nas que podem ser provados serem difíceis de alcançar. Denotando por n o tamanho da instância, as razões de aproximação que definem as classes são respectivamente $1+\epsilon$ para algum $\epsilon > 0$ fixo, $\Omega(logn)$, $2^{log^{1-\gamma}n}$ para todo $\gamma > 0$ fixo, e n^{δ} para algum $\delta > 0$ fixo. Porém, provar estes resultados pode ser muito difícil [13]. Na referência [11], eles discutem seis problemas NPO "canônicos". Eles são úteis, no sentido de que é possível derivar resultados de não aproximabilidade de maneira simples, apesar de não serem necessariamente os melhores possíveis. Para os quatro problemas anteriores, os problemas representativos são MAX 3-SAT, MIN SET COVER, MAX LABEL COVER (e sua versão de minimização também), e MAX CLIQUE respectivamente. Porém, devemos levar em conta o que os autores deste artigo dizem de acordo com o esquema proposto: *"Deve ser notado que nossa classificação reflete somente (o limite de) nosso atual conhecimento da área; trabalho futuro pode revelar outras classes naturais, ou colapsar duas classes, ou mover problemas de uma classe à outra."* Nos referimos a [160], [13], e [92] para outros resultados nessa área.

Capítulo 4

O Problema do Caixeiro Viajante Assimétrico

4.1 Introdução e Motivação

Este capítulo introduz um novo algoritmo memético (MA) que foi particularmente projetado para as instâncias assimétricas do problema do caixeiro viajante MIN TSP. Este caso especial é chamado o MIN ATSP. O método também inova no uso de uma variante do esquema de movimento de busca local *3-change* que é usado como o principal motor de busca local do MA. Porém, o algoritmo também incorpora muitas outras características que contribuem para sua eficácia, como: a) a população de agentes é topologicamente organizada como uma árvore ternária completa com treze nós; b) a organização hierárquica da população em *"clusters"* com sobreposição conduz a escolher mecanismos de seleção e estratégias de reprodução especiais; c) a estrutura de dados e os algoritmos escolhidos conduzem a um método altamente eficiente.

Experimentos computacionais foram realizados considerando todas as instâncias assimétricas disponíveis na `TSPLIB`, e em um conjunto de instâncias assimétricas geradas a partir de instâncias simétricas com as soluções ótimas conhecidas por construção. Os experimentos mostram que o método teve um bom desempenho, em comparação aos resultados obtidos por várias outras metaheurísticas propostas recentemente para o MIN ATSP.

O MIN TSP é o problema de encontrar o tour de comprimento mínimo entre um conjunto de cidades, tendo como dado de entrada a matriz de distâncias completa entre todas as cidades. Seja c_{ij} um inteiro não negativo que representa uma medida do custo de viajar da cidade i diretamente à cidade j. Uma instância simétrica é qualquer instância em que $c_{ij} = c_{ji}$ para todos os pares i, j. Uma instância do MIN ATSP é qualquer instânica do MIN TSP que tem no mínimo um par de cidades tal que $c_{ij} \neq c_{ji}$. O MIN ATSP pode formalmente ser descrito como: dado como entrada um grafo direcionado completo com peso nas arestas $G = (V, A, W)$, onde $V = \{1, \ldots, n\}$ e $A = \{(i, j) : i, j \in V, i \neq j\}$ são os conjuntos de vértices e arcos de G, respectivamente. Uma solução factível para o MIN ATSP é um circuito hamiltoniano em G (um *"tour"*). O objetivo

é encontrar um tour de comprimento total mínimo, onde o comprimento é a soma dos pesos de cada arco no tour. Dito isto, o MIN ATSP é apenas o problema MIN TSP restrito a instâncias assimétricas. Assim, constitue um caso especial do MIN TSP.

Notamos que as instâncias euclidianas do MIN TSP têm atraído uma grande atenção entre os pesquisadores em décadas recentes. De fato, instâncias euclidianas têm sido usadas como um dos mais importantes *"testbeds"* para novos métodos de otimização combinatória. O crescimento em popularidade é também devido às diversas aplicações importantes do mundo real, principalmente em controle de chão de fábrica ("scheduling"), distribuição de bens e serviços (roteamento de veículos), desenho de produtos ("VLSI layout"), etc. Algoritmos exatos têm sido propostos para casos simétricos e assimétricos. Desde que o MIN TSP tem provado pertencer à classe de problemas NP-difíceis [131], que, como vimos no capítulo anterior, não é aproximável, heurísticas e meta-heurísticas ocupam um lugar importante entre os métodos desenvolvidos para fornecer soluções práticas para instâncias grandes. Para mais informação sobre pesquisas em outros métodos de solução para TSP, o leitor pode referenciar [24], [220], [187]. Se apenas focamos em métodos exatos para o MIN ATSP, o algoritmo *Branch-and-Bound* proposto por Miller e Pekny [261] usa a bem conhecida *relaxação usando o Problema de Designação* (AP) do MIN ATSP. Eles relatam soluções ótimas encontradas para instâncias com até 500.000 cidades em tempo de execução razoável. Deve ser observado, porém, que as instâncias foram geradas aleatoriamente a partir de uma distribuição uniforme em um dado intervalo. Neste caso, as instâncias parecem ser "fáceis" para algoritmos baseados em AP [118]. Usando uma metodologia similar, Carpaneto, Dell'Amico, e Toth [62], resolveram instâncias com até 2.000 cidades em menos de 1 minuto. Porém, há instâncias em que algoritmos baseados em AP podem enfrentar problemas, tais como os custos serem "quase" simétricos (i.e. $c_{ij} \approx c_{ji}$ para todo i, j). Outra classe de instâncias "difíceis de resolver" vem de situações do mundo real relacionadas com o problema de roteamento de veículos que surgem em serviços de entrega de produtos farmacêuticos dentro da cidade de Bologna [119]. Estas instâncias de MIN ATSP e seus tours ótimos estão disponíveis na `TSPLIB` [331], e serão utilizadas neste capítulo (as instâncias `ftv*` com 90 a 160 cidades são obtidas a partir da instância `ftv170`). A metodologia poliédrica usada por Fischetti e Toth [118] desempenha bem para estas duas classes de instâncias difíceis quando comparadas com os métodos baseados em AP. Em nossos experimentos computacionais, também usamos as instâncias destas duas classes para avaliar o método proposto.

Do ponto de vista das heurísticas, um amplo espectro de técnicas meta-heurísticas tem sido proposto, tais como *Busca Tabu* [116], *Redes Neurais* [315], *Colônia de formigas* [352], *"Simulated Jumping"* [9], e *Algoritmos Genéticos* (GAs). De particular interesse para a finalidade desta Tese são os GAs, devido à efetividade alcançada por esta classe de técnicas que têm encontrado boas soluções em tempos computacionais baixos. Porém, os melhores resultados foram obtidos pela combinação de algoritmos evolutivos com métodos de busca local, os *Algoritmos Meméticos* (MAs) [270], [293] [257] [273].

Muitos MAs têm sido propostos para resolver o MIN ATSP. Baseados em

resultados computacionais reportados para estes métodos, podemos dizer que uma boa implementação de um MA deve combinar algumas características essenciais:

- operadores de recombinação e mutação satisfatórios;
- efetivos e rápidos algoritmos de busca local;
- uma população estruturada hierarquicamente;
- estrutura de dados avançada e mecanismos de codificação adequados.

O primeiro requisito também é obviamente inerente a qualquer algoritmo populacional do tipo genético, enquanto o segundo é crucial em MAs porque cerca de 85% a 95% do tempo de CPU utilizado é geralmente gasto nos procedimentos de busca local (e também será mostrado válido em nosso caso). Muitas experiências de trabalhos anteriores mostraram que a adoção de estruturas nas quais os agentes são restritos à recombinação dentro de subpopulações hierarquicamente organizadas provaram ser mais efetivas quando comparadas com implementações não estruturadas [278], [272], [142], [125], [248], [125], [34]. Para algum tipo de problemas, ou para algum tipo de instâncias, tambm é essencial obter uma boa população inicial.

O método GLS (por *"Genetic Local Search"* ou *Busca Local Genética* (GLS) proposto por Freisleben e Merz [127] é um MA de sucesso [257] que introduz um novo operador de recombinação, chamado DPX (*Distance Preserving Crossover*). Eles usam a heurística conhecida como Lin-Kernighan (LK) como busca local para instâncias euclidianas (simétricas) e o procedimento *fast-3-Opt* para as assimétricas. Em um artigo recente, eles melhoraram os resultados adotando uma série de sofisticados mecanismos de implementação que aumentaram o desempenho do método [252] [257]. Para o MIN ATSP, eles adicionaram uma variante de *4-Opt moves* para a busca. Ao invés de investigar métodos complexos de busca local, Gorges-Schleuter [142] escolheu investir em populações estruturadas espacialmente usando uma heurística simples de busca local. Neste método, chamado *Asparagos96*, a população é organizada em *demes* (i.e, um tipo de subpopulação local) espacialmente organizada como um anel e a topologia particular recebeu o nome de *"ladder-population"* [139]. Os algoritmos de busca local empregados são os *"2-repair-mechanism"* e o *"3-repair-mechanism"* para as instâncias simétricas e assimétricas, respectivamente. O operador de recombinação utilizado é o *MPX2*, que difere um pouco de seu anterior *MPX* (*Maximal Preservative Crossover*). Testes computacionais com estas duas metodologias limitados a um pequeno grupo de instâncias da `TSPLIB` mostram que para instâncias simétricas maiores, *Asparagos96* é superior enquanto que GLS é melhor em instâncias com até 783 cidades. Para instâncias assimétricas, *Asparagos96* superou GLS em todas as cinco instâncias usadas na comparação.

Um novo operador de recombinação que pode ser usado para instâncias simétricas ou assimétricas foi recentemente proposto por Nagata e Kobayashi [284]. O *EAX* (*Edge Assembly Crossover*) é capaz de gerar uma ampla variedade de descendentes a partir de um par de pais. É adicionada informação heurística durante o processo de recombinação de tal modo que a fase de busca

local pode ser desconsiderada. Porém, poder-se-ia também considerar este algoritmo como pertencendo à classe dos MAs, dado que agrega conhecimento adicional durante o processo de evolução.

Um mecanismo de recombinação que emprega o mesmo conceito de "grande família" foi desenvolvido por Walters [364] e é baseado em *"soft brood selection"* uma estratégia usada em Programação Genética. Em *"brood selection"*, dois pais podem gerar muitos descendentes e somente o melhor,com respeito ao seu valor de aptidão, é selecionado para o resto do procedimento. De fato, pode-se escolher mais de um descendente por iteração. O operador de recombinação de dois pontos e o método de reparar o cromossomo chamado *DER* (*"Directed Edge Repair"*) foi combinado com um *fast 3-opt* e aplicado a um subconjunto muito reduzido de instâncias simétricas e assimétricas da `TSPLIB` [364].

Neste capítulo, propomos um novo MA para resolver o Min ATSP que incorpora todos os quatro aspectos responsáveis por obter um método efetivo. Também propomos um novo operador de recombinação chamado *SAX* (*"Strategic Arc Crossover"*) capaz de tratar com instâncias do Min ATSP e adaptado a partir de um operador de recombinação similar, chamado *SEX* (*"Strategic Edge Crossover"*) que provou ser efetivo para instâncias simétricas [276] [169] [272] [170]. A população é hierarquicamente organizada como uma árvore ternária completa onde cada nó que não é folha corresponde a uma subpopulação de quatro indivíduos. Dentro de uma subpopulação, um *líder* e outros três indivíduos subordinados podem se recombinar, produzindo descendentes. A estrutura de dados e algoritmos usados permite que soluções muito próximas da ótima sejam encontradas rapidamente.

Este capítulo é organizado como segue. A Seção 4.2 fornece uma introdução que ajuda a comparar esta metodologia com outros MAs desenvolvidos para este problema. Na Seção 4.3, são descritas as principais características (do método proposto) que são independentes do problema, como estrutura e tamanho da população. Na Seção 4.4, os procedimentos dependentes do problema de Min ATSP, como o novo algoritmo de busca local e os operadores de recombinação são apresentados . A Seção 4.5 introduz os procedimentos que evitam uma convergência rápida da população. Nas seções seguintes, nosso método é avaliado em uma série de testes computacionais em conjuntos de dados da literatura. Comparações empíricas com seis outras metaheurísticas também são fornecidas. Os resultados e as conclusões obtidas constituem a Seção 4.8.

4.2 MAs para o ATSP

Uma estrutura genérica para um MA baseado em busca local foi proposta por Moscato em [273] e é reproduzida na Fig. 4.1. Este peseudo-código pode ser considerado como um modelo genérico que caracteriza várias implementações diferentes que podem ser encontradas na literatura. Embora menos genérico que o introduzido no capítulo anterior, já é genérico o bastante para incorporar algum outro pseudo-código baseado em Busca Local para MAs para o Min ATSP, dado que os resultados obtidos por estes métodos alternativos serão mostrados neste capítulo.

Local-Search-based Memetic Algorithm

```
begin
   initializePopulation Pop using FirstPop();
   foreach i ∈ Pop do i := Local-Search-Engine(i);
   foreach i ∈ Pop do Evaluate(i);
   repeat /* generations loop */
      parfor j := 1 to #recombinations do
         selectToMerge a set S_par ⊆ Pop;
         offspring := Recombine(S_par, x);
         offspring := Local-Search-Engine(offspring);
         Evaluate(offspring);
         addInPopulation offspring to Pop;
      endparfor;
      parfor j := 1 to #mutations do
         selectToMutate i ∈ Pop;
         i_m := Mutate(i);
         i_m := Local-Search-Engine(i_m);
         Evaluate(i_m);
         addInPopulation i_m to Pop;
      endparfor;
      Pop := SelectPop(Pop);
      if Pop meetsPopConvCriteria then Pop := RestartPop(Pop);
   until termination-condition=True;
end;
```

Figura 4.1: *O Algoritmo Memético baseado em Busca Local.*

Usando este modelo genérico, o algoritmo proposto apenas precisa ser definido com um pouco de informação extra, como o tipo de operador de busca local (RAI em nosso caso) e o tipo de recombinação a serem usados. Em nosso caso, usamos o SAX, embora também tenhamos testado alguns outros operadores de recombinação, um único de cada vez. O procedimento **selectToMerge** é determinístico e altamente relacionado à estrutura de população hierárquica que usamos. A população inicial usa uma variante *"ad-hoc"* da heurística do vizinho mais próximo. São avaliados os indivíduos de acordo com os valores da função objetivo, de modo que a instância x é necessária para o procedimento Evaluate().

Não esperamos que neste momento o leitor entenda completamente todos os detalhes, apenas que tenha uma visão geral de como a metodologia proposta ajusta-se na descrição geral de um MA baseado em busca local. Porém, alguns detalhes já podem ser entendidos neste nível de generalidade. Por exemplo, cada solução possível tem uma probabilidade *"a priori"* de 7/10 de *não* ser mutada, assim **selectToMutate** é um procedimento randomizado imparcial. A mutação é executada por um procedimento randomizado chamado *"Mutation-Insertion"* a ser descrito mais à frente. O critério para convergência é devido

MA baseado em Busca Local para o MIN ATSP

begin
  Ordered_Neighborhood_List(5); (* list to be used by RAI *)
  $|Pop| = 13$; (* complete ternary tree with 13 agents *)
  $|S_{par}| = 2$; (* two-parent recombination *)
  initializePopulation *Pop* using FirstPop(Variant_of_NN_Heuristic(x));
  repeat /* generations loop */
    parfor $j := 1$ **to** $|Pop|$ **do**
      selectToMerge a set $S_{par} \subseteq Pop$;
      $offspring$:= Recombine(S_{par}, x, SAX);
      $offspring$:= Local-Search-Engine($offspring, x$, RAI);
      Evaluate($offspring, x$);
      addInPopulation $offspring$ to *Pop*;
    endparfor;
    parfor $j := 1$ **to** *#mutations* **do**
      selectToMutate an individual $i \in Pop$;
      i_m := Mutate(i, Mutation-Insertion);
      i_m := Local-Search-Engine(i_m, x, RAI);
      Evaluate(i_m, x);
      addInPopulation i_m to *Pop*;
    endparfor;
    Pop := SelectPop(*Pop*);
    if *Pop* **has converged then** *Pop* := RestartPop(*Pop*);
  until termination-condition=True;
end;

Figura 4.2: *Modelo para o nosso MA baseado em Busca Local para o* MIN ATSP. *Nesse nível de generalidade, os detalhes de implementação que levam em conta a estrutura de população, as atualizações, estão ainda ocultos.*

a uma perda de diversidade e também será explicado depois. A condição de término em nosso caso é um número máximo e fixo de gerações.

O modelo para o nosso MA baseado em Busca Local é mostrado na Figura 4.2. O procedimento RAI(), usado para busca local, precisa de uma lista ordenada dos 5 vizinhos mais próximos de cada cidade. A população é composta por 13 agentes, cada um tendo duas soluções factíveis do problema (a ser descrita em 4.3.1). Em cada geração, a população é reestruturada e se a falta de diversidade é detectada, o procedimento RestartPop() é executado.

No algoritmo de Merz e Freisleben, da Fig. 4.3, o procedimento de busca local executa os movimentos *3-opt* e *4-opt*. A mutação inverte um subcaminho escolhido aleatoriamente de comprimento 6 (executando um *7-change* aleatório no tour corrente). O critério de parada é encontrar a solução ótima e/ou algum outro critério não especificado. O #recombinations e #mutations é de 20. O leitor pode comparar as diferências e similaridades dos pseudocódigos dos algoritmos de Gorges-Schleuter [142], descrito na Figura 4.4, Merz e Freisle-

Algoritmo de Merz e Freisleben para o MIN ATSP

begin
Ordered_Neighborhood_List(m); (*m = 100 for TSP, unspecified for ATSP*)
$|Pop| = 100$; (* unstructured population *)
$|S_{par}| = 2$; (* two-parent recombination *)
initializePopulation Pop using FirstPop(NN_Heuristic(x));
foreach $i \in Pop$ **do** i := Local-Search-Engine(i, x, fast 3-opt, 4-opt);
foreach $i \in Pop$ **do** Evaluate(i, x);
repeat /* generations loop */
parfor $j := 1$ **to** $|Pop|$ **do**
selectToMerge a set $S_{par} \subseteq Pop$;
$offspring$:= Recombine(S_{par}, x, DPX);
$offspring$:= Local-Search-Engine($offspring, x$, fast 3-opt, 4-opt);
Evaluate($offspring, x$);
addInPopulation $offspring$ to Pop;
endparfor;
parfor $j := 1$ **to** $\#mutations$ **do**
selectToMutate an individual $i \in Pop$;
i_m := Mutate(i, random-7-change);
i_m := Local-Search-Engine(i_m, x, fast 3-opt, 4-opt);
Evaluate(i_m, x);
addInPopulation i_m to Pop;
endparfor;
Pop := SelectPop(Pop);
if Pop **has converged then** Pop := RestartPop(Pop);
until termination-condition=True;
end;

Figura 4.3: *Modelo para o algoritmo memético de Merz e Freisleben's para o* MIN ATSP.

ben [252], já discutido e apresentado na Figura 4.3, Nagata e Kobayashi [284], apresentado na Figura 4.6, e Walters [364], apresentado na Figura 4.5.

4.3 Características da implementação independentes do problema

Descreveremos primeiramente algumas características do método proposto que são independentes do problema. Nos referimos a eles separadamente, visto que contêm várias partes de código altamente reutilizáveis em domínios de problemas diferentes.

Algoritmo de Gorges-Schleuter para o Min ATSP

begin
  Ordered_Neighborhood_List(20);
  $|Pop| = 20$; (* stuctured population *)
  $|S_{par}| = 2$; (* two-parent recombination *)
  initializePopulation Pop using FirstPop(Modified_NN_Heuristic(x));
  foreach $i \in Pop$ **do** i := Local-Search-Engine(i, x, 3-repair);
  foreach $i \in Pop$ **do** Evaluate(i, x);
  repeat /* generations loop */
    parfor $j := 1$ **to** $|Pop|$ **do**
      selectToMerge a set $S_{par} \subseteq Pop$;
      $offspring$:= Recombine(S_{par}, x, MPX2);
      i_m := Mutate($offspring$, non-sequential-4-opt-change);
      i_{ml} := Local-Search-Engine(i_m, x, 3-repair);
      Evaluate(i_{ml}, x);
      addInPopulation i_{ml} to Pop;
    endparfor;
    Pop := SelectPop(Pop);
    if Pop **has converged then** Pop := RestartPop(Pop);
  until termination-condition=True;
end;

Figura 4.4: *Modelo para o algoritmo memético de Gorges-Schleuter para o* Min ATSP.

4.3.1 Tamanho e estrutura da População

No método implementado, a população é composta de 13 agentes organizados como uma árvore ternária completa de três níveis. A estrutura e o tamanho da população são fixos. Para o leitor já familiarizado com Algoritmos Evolutivos, uma estrutura de população hierárquica poderia ser entendida como uma variante dos *modelos de ilhas*. Porém, sabendo que a metáfora de MAs está relacionada com evolução cultural, podemos considerar essa estrutura melhor como uma organização hierárquica de "colegas" que revisam e recombinam soluções de um problema, veja Fig. 4.7. Ao invés de estabelecer algum mecanismo de *migração* de agentes e soluções entre subpopulações, em nosso caso cada subpopulação tem quatro agentes, enquanto os indivíduos em um nível intermediário (aqueles que não são a raiz e nem as folhas da árvore) são compartilhados pelas duas subpopulações. Cada subpopulação é composta de um agente *líder* e três agentes *subordinados*, que estão em um nível abaixo na hierarquia. O agente 1, localizado na raiz da árvore, é o líder da subpopulação do topo e tem seus agentes subordinados indexados com os números 2, 3, e 4. O agente 2 tem como subordinados os agentes 5, 6, e 7, e asim por diante. Note que os agentes 2, 3, e 4, interagem como líderes e subordinados.

Novamente, observamos que usamos a denominação de agente, ao invés de

Algoritmo de Walters para o MIN ATSP

begin
  Ordered_Neighborhood_List($N-1$);
  $|Pop| = 20$; (* unstuctured *)
  $|S_{par}| = 2$; (* two-parent recombination *)
  initializePopulation *Pop* using FirstPop(NN_Heuristic(x));
  foreach $i \in Pop$ **do** i := Local-Search-Engine(i, x, fast-3-opt);
  foreach $i \in Pop$ **do** Evaluate(i, x);
  repeat /* generations loop */
    parfor $j := 1$ **to** $|Pop|$ **do**
      selectToMerge a set $S_{par} \subseteq Pop$;
      $offspring$:= Recombine(S_{par}, x, 2-point-modified);
      i_m := Mutate($offspring$, mutate-4-to-7-genes);
      i_{mr} := DER_Repair(i_m);
      i_{ml} := Local-Search-Engine(i_{mr}, x, fast-3-opt);
      Evaluate(i_{mrl}, x);
      addInPopulation i_{mrl} to *Pop*;
    endparfor;
    Pop := SelectPop(*Pop*);
    if *Pop* **has converged then** *Pop* := RestartPop(*Pop*);
  until termination-condition=True;
end;

Figura 4.5: *Modelo do algoritmo memético de Walters para o* MIN ATSP.

indivíduo, como outro modo de diferenciar parte do espírito de MAs. Recordando um conceito de engenharia de software, os agentes também podem codificar métodos de solução para o problema particular. Neste caso, será uma heurística de busca local. Porém, outras formas de conhecimento de domínio são possíveis. Nesta implementação particular para o MIN ATSP, cada agente estará manipulando duas soluções factíveis. Nos referiremos a uma delas como a solução *Pocket* e à outra como a solução *Current* (a metodologia de *Pocket-Current* foi usada em outros trabalhos, por exemplo, [278], [305], [34]). Em cada geração, são criados 13 indivíduos novos na fase de recombinação e eles são postos nas 13 soluções *Current*. Após as fases de recombinação e de mutação, estes indivíduos novos são otimizados pelo procedimento de busca local. As soluções *Pocket* foram previamente encontradas por algum agente da população. Além disso, cada agente traz a informação relativa ao valor da função objetivo associado com as soluções *Pocket* e *Current*: *PocCost* e *CurCost*, respectivamente.

4.3.2 Atualizações dos agentes

Em todo momento, todos os agentes possuem uma solução *Pocket* que é de melhor qualidade que sua *Current*. Quando uma solução *Current* (que é a usada para recombinação, mutação e busca local) é otimizada ao ponto de ser

Algoritmo de Nagata e Kobayashi para o Min ATSP

begin
  $|Pop| = f(x)$; (* and *ad hoc* value for each instance *)
  $|S_{par}| = 2$; (* two-parent recombination *)
  initializePopulation *Pop* using FirstPop(NN_Heuristic(x));
  repeat /* generations loop */
    parfor $j := 1$ **to** $|Pop|$ **do**
      selectToMerge a set $S_{par} \subseteq Pop$;
      $offspring :=$ Recombine(S_{par}, x, EAX);
      Evaluate($offspring, x$);
      addInPopulation $offspring$ to *Pop*;
    endparfor;
    Pop := SelectPop(*Pop*);
    if *Pop* **has converged then** *Pop* := RestartPop(*Pop*);
  until termination-condition=True;
end;

Figura 4.6: *Modelo para o algoritmo memético de Nagata e Kobayashi para o* Min ATSP.

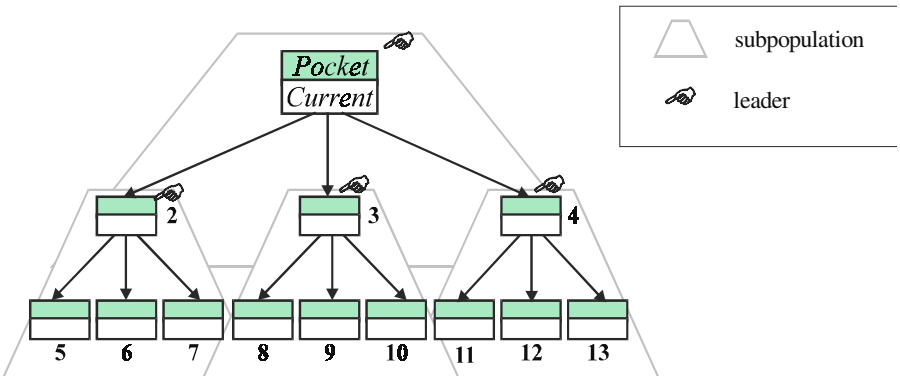

Figura 4.7: *Uma estrutura de população como árvore ternária completa. Os agentes estão organizados hierarquicamente e cada um tem duas soluções factíveis para o problema.*

melhor que seu correspondente *Pocket* (i.e. $CurCost < PocCost$ se o objetivo é minimizar uma função-guia), elas são trocadas, assim podemos pensar que o *Pocket* trabalha tendo o papel de uma memória de boas soluções visitadas no passado. Este procedimento é chamado de *UpdatePocket*. Se 1/4 das soluções *Pocket* tem a mesma solução que *CurCost*, o *UpdatePocket* não é executado. Isto é necessário porque algumas instâncias testadas são altamente degeneradas e têm muitos tours com o mesmo custo. Permitir que a população tenha somente um *Pocket* com o mesmo valor de custo é muito restritivo e não imple-

mentar qualquer proibição causa uma perda de diversidade muito rápida para diversas instâncias.

Depois que todos os agentes tiveram suas soluções *Current* otimizadas e trocadas com seus *Pockets* sempre que apropriado, um procedimento de nome *PocketPropagation* é aplicado. Este procedimento troca as soluções *Pocket* de um líder com seu subordinado, se este tem uma solução *Pocket* que é melhor que seu líder. Note que estes dois mecanismos são suficientes para garantir o fluxo de soluções melhores em direção ao topo da hierarquia (nó raiz). Ambos os mecanismos são inerentemente assíncronos, o que conta para a conveniência dos métodos para uma implementação distribuída, mesmo em um sistema heterogêneo.

4.4 Métodos dependentes do ATSP

Nesta seção, descrevemos os métodos desenvolvidos tendo em mente o problema específico a ser resolvido, isto é, o MIN ATSP. Porém, isto não restringe o possível uso deles em outros domínios. Por exemplo, os operadores de recombinação ou as buscas locais podem ser aplicados prontamente a outros problemas, por exemplo, para representações baseadas em permutação, como os problemas de *"scheduling"* [248].

4.4.1 População inicial

A estrutura de dados adotada para representar uma solução factível (i.e. um ciclo hamiltoniano dirigido) consiste de um vetor bidimensional de comprimento $N \times 2$, onde N é o número de cidades da instância. Cada índice i do vetor corresponde a uma cidade i e o vetor preserva os índices associados com a cidade que precede (*Prev*) e sucede (*Next*) a cidade i.

Há duas razões principais para escolher esta estrutura de dados. Primeiro, devido à inegável simplicidade em comparação com outras estruturas de dados propostas para o MIN ATSP, e segundo, a numerosa quantidade de *trocas de três arcos* recursivamente feitas pelo Local_Search_RAI() que podem ser executadas em tempo fixo, e dizer, independente do tamanho da instância. No caso de instâncias simétricas, é interessante ter uma estrutura de dados mais complexa que permite a inversão de subtours em tempo constante. Em nosso caso, dado que nossa busca local não requer tal atualização, recorremos a esta estrutura de dados por sua simplicidade para as operações necessárias.

Há várias possíveis escolhas para a população inicial. Uma delas é criá-la completamente ao acaso (i.e. escolhendo uniformemente ao acaso entre todas as possíveis permutações), enquanto outra possibilidade é aplicar alguma heurística construtiva gulosa. Escolhemos usar a heurística construtiva do vizinho mais próximo para um procedimento randomizado. As soluções *Pocket* e *Current* do Agente 1 (o agente no topo da hierarquia na Fig. 4.7) foram geradas usando a heurística do vizinho mais próximo que usa uma cidade inicial selecionada ao acaso com distribuição uniforme. Note que, na heurística do vizinho mais próximo, as últimas cidades inseridas são unidas geralmente a vizinhos distantes, assim é interessante manter registro destas e usá-las como

cidades iniciais para criar recursivamente novas soluções. O objetivo é gerar uma população inicial diversificada. Assim, para cada subpopulação, as últimas três cidades inseridas na solução de vizinho mais próximo do *Pocket* e *Current* do líder são consideradas cidades iniciais na construção das soluções *Pocket* e *Current* dos subordinados correspondentes.

4.4.2 Operadores de recombinação

Dado que uma nova busca local está sendo proposta para o MIN ATSP, achamos conveniente testar sua qualidade *"vis-a-vis"* para diferentes operadores de recombinação. É bem conhecido que o desempenho de um MA é dependente de uma boa sinergia entre o algoritmo de busca local e o operador de recombinação utilizado. Da vasta literatura de operadores de recombinação propostos, selecionamos quatro operadores de recombinação com base em resultados no MIN TSP: i) o operador SAX, nossa generalização do SEX (*"Strategic Edge Crossover"*) que foi originalmente proposto para instâncias euclidianas e simétricas de MIN TSP [293]; ii) o DPX (*"Distance Preserving Crossover"*) introduzido por Freisleben e Merz em [127]; iii) o MFNN introduzido por Holstein e Moscato em [172]; e iv) o operador *Uniform Nearest Neighbor Crossover* (UNN). Na seqüência, mostraremos como derivamos a adaptação que conduz à recombinação SAX, como também a idéia básica de cada um dos outros operadores.

4.4.3 O Strategic Arc Crossover (SAX)

Historicamente, SEX pode ser considerado como descendente do procedimento *"Enhanced Edge Recombination"* sugerido em [374], combinado com a heurística introduzida por Karp [193] e [194]. Embora o SEX não tenha sido estudado completamente na literatura, com poucas exceções [276] [169] [272] [170], muitos operadores de recombinação subseqüentes têm características similares. O processo que cria um novo descendente usando o operador SAX pode ser melhor entendido descrevendo-o em duas fases: *geração de strings* (uma string é um caminho dirigido com h cidades, $1 \leq h \leq N$) e a aplicação do procedimento *PatchStrings*, isto é as strings são reconectadas para formar um ciclo hamiltoniano dirigido.

A razão que motivou este operador de recombinação foi *maximizar* heuristicamente o número de arestas nos descendentes que pertencem à união das arestas recebidas como entrada. Ao contrário de outros operadores como o DPX, não há nenhuma aresta que seja proibida, e o processo de conserto *PatchStrings* pode adicionar arestas que não estão presentes nos pais, introduzindo mutações implícitas. Com respeito a este aspecto, a recombinação SEX pode ser considerada como um operador *misturar-e-intensificar*, enquanto outros operadores de recombinação (como *Distance Preserving Crossover* de Merz e Freisleben) tem uma natureza de maior diversificação. Originalmente, a heurística de Karp para conserto foi selecionada para o SEX, dado que tem sido mostrado que, para instâncias tendo uma distribuição de probabilidade aleatória uniforme de cidades, tours com erros percentuais de aproximadamente zero poderiam ser encontradas particionando em subconjuntos e então concatenando os tours

ótimos de cada subconjunto [193].

Dado dois *pais*, tours A e B, um dígrafo G pode ser construído tendo o mesmo conjunto de vértices e o seu conjunto de arcos e constituído pela união de arcos nos dois tours. Assim, todo vértice no dígrafo tem grau de saída, ou entrada, 1 ou 2. As *"strings"* são geradas como indicado:

- Passo 1: Escolha aleatoriamente um vértice não utilizado v em G como o vértice inicial para uma nova *"string"* atual, marque-o como usado, e declare-o ser o final t da string e o início h.
- Passo 2: Enquanto o início h da string atual tem no mínimo um arco-saída para um vértice não usado, faça o seguinte: aleatoriamente escolha um (1 ou 2) dos vértices não utilizados para que h tenha um arco-saída, adicione-o à lista (conectando-o a h), marque-o como usado, e declare-o como sendo o novo h da string.
- Passo 3: Declare o h da string atual como sendo o extremo anterior da string.
- Passo 4: Enquanto o vértice t da string atual tem no mínimo um arco-entrada para um vértice não utilizado, faça o seguinte: aleatoriamente escolha um (1 ou 2) vértice não utilizado para o qual t tenha um arco-entrada, adicione-o à lista (conectando-o a t), marque-o como usado, e declare-o ser o novo t da string.
- Passo 5: Declare o t da string atual como sendo o extremo posterior da string. Se ainda houver vértices não utilizados, retorne ao Passo 1.

Como conseqüência da aplicação dos passos 1-5, terminamos tendo um conjunto de strings, ou seja, um conjunto de caminhos dirigidos em G. Porém, eles podem não ser conectados. Então, para recuperar factibilidade, precisamos conectá-los de forma a termos um ciclo hamiltoniano dirigido. Genericamente chamamos este procedimento de *PatchStrings* e nesse caso novamente recorremos ao algoritmo do vizinho mais próximo que começa com o extremo anterior de uma string e o conecta ao extremo posterior mais próximo de alguma outra string.

Note que este operador de recombinação não garante que um arco comum entre ambos os pais seja preservado no novo tour gerado. Por exemplo, se ambos os pais têm em comum os arcos $a-b$ e $b-c$, e se o vértice início v da primeira string é c, e podemos terminar construindo a string $c-x_1-x_2-\cdots-x_k-a-b$, onde os xs permanecem para qualquer cidade diferente de a, b, c. Então, pode ocorrer o caso de que não podermos colocar o arco $b-c$ nesta string sem violar a condição de exclusão do subtour. Além disso, o procedimento *PatchStrings* pode não ajudar, visto que não força qualquer arco perdido da intersecção ser adicionado de volta.

4.4.4 Distance Preserving Crossover (DPX)

A idéia principal por trás do DPX [127] origina-se a partir da observação de que a distância média entre tours localmente ótimos é proximamente relacionada

com a distância média entre um tour localmente ótimo e o ótimo global. Assim, o DPX é também baseado nas idéias de *correlação de ótimos locais* já discutidos em [270], [272], [257] em conexão com MAs. Neste caso, a distância entre dois tours é calculada contando o número de arestas que pertencem a um tour mas não ao outro. O objetivo do operador DPX é construir um descendente que tenha a mesma distância de cada um de seus pais como os pais tem entre si. Isto pode ser realizado copiando as arestas comuns a ambos os pais para o descendente. Os fragmentos do tour são, então, conectados usando o algoritmo do vizinho mais próximo, mas sem usar quaisquer das arestas que estão presentes em apenas um dos pais. Mais formalmente, seja $A \cap B$ o conjunto de arestas que os tours A e B têm em comum, e seja $A - B$ o conjunto de arestas que pertencem a A mas não a B. Então, o conjunto de arestas usadas para gerar o descendente é $(A \cap B) + E - ((A - B) \cup (B - A))$, onde E representa o conjunto completo de arestas da instância do Min TSP. Preferimos usar o termo aresta aqui, pois esta recombinação foi originalmente proposta para o caso simétrico. A versão assimétrica, com arcos ao invés de arestas, é diretamente implementável sem maiores dificuldades.

4.4.5 Multiple Fragment-Nearest Neighbor Repair Edge Recombination (MFNN)

Este operador de recombinação, introduzido por Holstein e Moscato em [172], emprega uma estratégia diferente na construção do descendente. É fácil adaptá-lo ao caso assimétrico. Aqui o conjunto de arcos usados para construir o descendente são todos os arcos em $A \cap B$ (como em DPX). Como em DPX, em um único passo, eles são todos adicionados à nova solução. Depois, estas arestas que pertencem a $(A - B) \cup (B - A)$ são adicionadas (sempre que possível) em um ordem não decrescente de comprimento. No processo, os empates são decididos arbitrariamente, levando em conta que a não violação das restrições de eliminação do subtour e de designação. O algoritmo do vizinho mais próximo é então aplicado para reconectar os fragmentos e construir um tour válido.

4.4.6 Uniform Nearest Neighbor Crossover (UNN)

No operador de recombinação Uniforme [137], um descendente é gerado através de escolhas aleatórias (com igual probabilidade) dos valores dos genes dos dois pais. Cada valor de gene depende do resultado de uma variável booleana: por exemplo, se verdade, o gene correspondente ao pai A é copiado para o descendente, caso contrário a cópia vem do pai B. Em nossa implementação UNN desta idéia geral, inicialmente todos os arcos comuns aos dois pais são copiados para o descendente. Após a cópia de arcos iguais, para cada cidade i, se o resultado de uma escolha aleatória de igual probabilidade correspondente à cidade i é verdade, então o arco que une i à próxima cidade no pai A é copiada para o descendente, desde que não viole qualquer restrição do TSP. Se é falso, então o arco copiado pertence ao pai B. Em ambos os casos, se uma violação ocorre, então o arco do outro pai é testado. Os fragmentos de tour resultantes são consertados usando o procedimento do vizinho mais próximo.

4.4.7 Similaridades e diferenças entre os operadores de recombinação

Dos operadores de recombinação apresentados aqui, certamente UNN e SAX são os mais semelhantes. Ambos os operadores maximizam o número de arcos no descendente herdados dos pais e eles não usam qualquer ordem de prioridade como o MFNN faz. Mas há duas diferenças importantes entre SAX e UNN. Primeiro, o SAX não garante que o descendente herde todos os arcos comuns dos pais como o UNN faz. Segundo, SAX gera uma string de cada vez, a partir de uma cidade escolhida aleatoriamente entre as não visitadas, e tenta aumentar o comprimento das strings tanto quanto possível com arcos vindos dos pais. O UNN gera todas as strings ao mesmo tempo: arcos dos pais são colocados no descendente em seqüência a partir de um índice de uma cidade inicial escolhida aleatoriamente, e apenas ao fim as strings são identificadas. Os resultados empíricos mostraram que o operador SAX obteve melhores resultados em média. Considerando que os demais operadores são os mesmos nos experimentos computacionais realizados (veja a Tabela 4.1), podemos concluir que as diferenças entre os operadores destacam as características encontradas no operador SAX.

4.4.8 Recursive Arc Insertion

Há várias definições de vizinhança utilizadas em metodologias baseadas em busca local para o MIN TSP. A literatura geralmente se refere à vizinhança de Lin-Kernighan como o mais eficaz otimizador de "agente-único" de busca local. Vários esquemas baseados em variantes desta estratégia mostraram ser úteis, particularmente nas maiores instâncias *geométricas* do MIN TSP. Johnson e McGeoch apresentaram uma variante *3-opt*, baseada em uma redução de vizinhança, com bons resultados para o caso simétrico [182]. Para instâncias assimétricas do MIN TSP, a heurística de Kanellakis-Papadimitriou (e algumas variantes dela) tem-se mostrado ser útil. No entanto, a maioria dos pesquisadores tem preferido usar vizinhanças simples, como a *3-opt*, devido a ser de complexidade computacional inferior.

Neste capítulo, para o Local-Search-Engine() do MA, usaremos um procedimento que nomeamos de *"Recursive Arc Insertion"*. É baseado na eliminação de um arco do tour e uma reinserção entre duas cidades (selecionadas de acordo com o critério descrito a seguir). O procedimento elimina três arcos e adiciona três novos arcos. Dessa forma, a transição básica está sempre contida na vizinhança *3-opt*.

A razão subjacente na nova heurística de busca local é melhorar o tour começando a partir de uma cidade marcada com *"don't look bits"* [31] durante as fase de recombinação e mutação. Isto é análogo ao espírito do mecanismo de *2-repair* proposto por Gorges-Schleuter [139]. Após o passo de recombinação, todas as cidades que resultaram ser extremos posteriores e anteriores de strings, têm seus *"don't look bits"* marcados como falso. Em um passo de mutação, todas as cidades nos extremos dos arcos envolvidos no movimento de inserção terão seus *"don't look bits"* fixados como falso. As cidades fixadas em falso nos *"don't look bit"* são consideradas como cidades "críticas" e são os pontos de

partida para ativar a busca. Antes do operador de recombinação e depois da busca local, todas as cidades têm seus *"don't look bit"* fixos em verdadeiro. Este não é o caso da primeira geração, porque em princípio todas as cidades têm seus *"don't look bits"* fixos em falso, permitindo que a busca local começe em qualquer lugar. Em um determinado tour, fazemos uma única passagem por cada ponto de partida (entretanto, devido à natureza recursiva do algoritmo, pode ser ativado mais de uma vez para qualquer cidade específica).

Dada a cidade i, definimos *s-neighbors(i)* como o conjunto de cidades destino dos $|s|$ arcos mais curtos que saem da cidade i. Analogamente, o *p-neighbors(i)* é o conjunto de cidades $|p|$ de arcos mais curtos chegando à cidade i. Seja i uma cidade crítica inicial selecionada em um tour dado.

- Passo1: (Figura 4.8A) Escolha $j \in$ *p-neighbor(i)*. Se $i = Next(j)$, fixe $Count = 1$ e vá ao Passo 5. Caso contrário, fixe $Count = 0$ e vá ao Passo 2.
- Passo 2: (Figura 4.8B) Considere um movimento de tentativa de adicionar um arco (i, j) e eliminar os arcos $arcs(i, a)$ e (b, j), onde $a = Next(i)$ e $b = Prev(j)$. Calcule $\Delta_1 = c_{ia} + c_{bj} - c_{ij}$. Faça $Count = Count + 1$ e vá ao Passo 3.

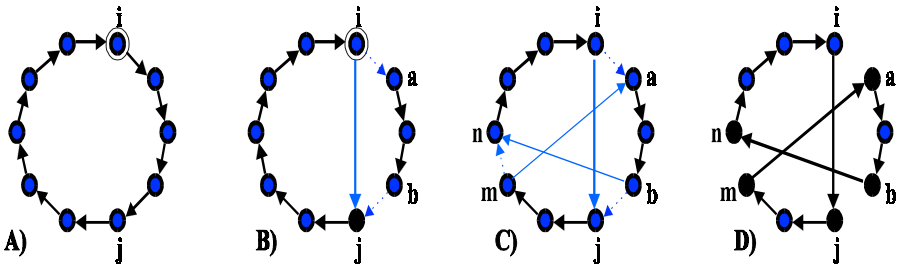

Figura 4.8: *Movimento básico da busca local RAI.*

- Passo 3: (Figura 4.8C) Comece do arco $(j, Next(j))$, selecione o primeiro arco $(m, n) \neq (i, j)$ no subtour $\{j, \ldots, i, j\}$ tal que $\Delta_2 < Delta_1$, $\Delta_2 = c_{ma} + c_{bn} - c_{mn}$. Se encontrar um arco tal (m, n), vá ao Passo 4. Se $\Delta_2 \geq \Delta_1$ para todos os arcos no subtour $\{j, \ldots, i, j\}$, vá ao Passo 5.
- Passo 4: (Figura 4.8D) Execute o movimento, ex., insira os arcos (i, j), (m, a) e (b, n) e remova os arcos (i, a), (b, j) e (m, n). Chame o procedimento recursivamente usando como cidade crítica à inicial (cidade i) e as cidades a, b, n, m, j e i, nesta ordem. Vá ao Passo 5.
- Passo 5: Se $Count = 2$, pare. Caso contrário, selecione $j \in$ *s-neighbor(i)*. Se $j = Next(i)$, pare. Caso contrário, faça $Count = 2$ e troque os índices i e j (fazendo $aux = i$, $i = j$ e $j = aux$), e vá ao Passo 2.

Note que o procedimento poderia usar uma pilha ao invés de recursão. Mas, a recursão foi mais rápida que a pilha em nosso caso. Acreditamos que isto acontece porque a chamada recursiva é muito simples e são realocadas apenas três variáveis em cada chamada. Uma lista ao invés uma pilha foi testada, mas os resultados que usam as duas aproximações alternativas são semelhantes em média. Porém, eles diferem para algumas instâncias. O uso da fila é melhor para algumas instâncias e para outras não. Devido ao fato da recursão, em média, ter mostrado um comportameno melhor, a selecionamos para implementação neste algoritmo.

Os vizinhos *s-nearest* e os vizinhos *p-nearest* de cada cidade i são armazenados em uma matriz com duas dimensões em ordem não decrescente de suas distâncias como um meio de facilitar a busca da cidade j nos Passos 1 e 5. Em nossos experimentos, $|s|$ e $|p|$ são ambos fixados em 5 para todas as instâncias testadas. A escolha da cidade j é influenciada do seguinte modo. Há 40% de probabilidade de que a cidade j seja selecionada como o vizinho mais próximo, 30 % como o segundo mais próximo, 15 % como o terceiro, 10 % como o quarto e 5 % como o quinto. Note que, durante a execução do Passo 3, a inserção do arco (i, j) e a eliminação dos arcos (i, a) e (b, j) criaram um subtour e um string de arcos. Porém, note que pode acontecer que a string pode ter vários, um, ou mesmo nenhum arco (no caso de $a = b$). Em todos os casos, precisamos recuperar a factibilidade. Encontramos o primeiro arco (a partir da cidade m para a cidade n no subtour). Obviamente, $(m, n) \neq (i, j)$ desde que testamos a inserção do novo arco (i, j). A estratégia alternativa, que olha para um arco (m, n) com a maior melhoria, foi testada e os resultados mostraram uma falta de diversidade na população. Testes considerando um número fixo de arcos também provaram não ter sucesso. Acreditamos que o que mais caracteriza o poder da busca é uma apropriada combinação de uma escolha gulosa dos arcos envolvidos no cálculo de Δ_1 com uma busca extensiva pelos arcos relativos ao cálculo de Δ_2.

4.5 Mantendo a diversidade da população

A robustez é um elemento chave no desenvolvimento de qualquer meta-heurística. Conseqüentemente, um método que mostra ser efetivo na prática, isto é, atinge soluções próximas da ótima em um baixo tempo de CPU, independentemente do tamanho da instância e do tipo particular de instância que é resolvida, é altamente desejável. De mesma relevância é a habilidade do método em resolver um conjunto de instâncias com os mesmos parâmetros previamente usados na solução de outras instâncias. Provavelmente, um grande passo para esta meta é a adoção sistemática de uma população estruturada como uma árvore ternária. Além disso, em todas as experiências realizadas com nosso MA, o tamanho de população foi fixado em 13 agentes (26 indivíduos). Comparado a outros MAs já propostos para o TSP, nossa população é notavelmente menor e independente do tamanho da instância. A vantagem é que um número muito maior de gerações pode ser executado no mesmo espaço de tempo. Por outro lado, é provável que a população perca diversidade rapidamente. Para evitar isto, nossa implementação usa vários procedimentos descritos nas subseções

seguintes.

4.5.1 Política relativa à seleção e recombinação

Discutiremos agora as políticas adotadas para selecionar as soluções a serem usadas como entrada para o operador de recombinação. Uma escolha satisfatória de agentes é requerida para a preservação da diversidade da população. Considere a notação adicional usada para cada subpopulação da árvore ternária.

- *líder* - o agente líder da subpopulação;
- *subordinado1, subordinado2* e *subordinado3* - os 3 subordinados da subpopulação;
- *Pocket(i)* - a solução *Pocket* do agente i;
- *Current(i)* - a solução *Current* do agente i.

Relembrando que o resultado da recombinação substitui a solução *Current*, a *Current* do agente nó raiz é gerada pela recombinação da solução *Pocket* do primeiro subordinado com a solução *Current* do segundo subordinado. Os operadores de recombinação para as outras 4 subpopulações são:

- *Current* do *líder* $\leftarrow$ Recombine(*Pocket(subordinado2), Pocket (subordinado3)*);
- *Current* do *subordinado1* $\leftarrow$ Recombine(*Pocket(líder), Current (subordinado2)*);
- *Current* do *subordinado2* $\leftarrow$ Recombine(*Pocket(subordinado1), Current (subordinado3)*);
- *Current* do *subordinado3* $\leftarrow$ Recombine(*Pocket(subordinado2), Current (subordinado1)*).

Para cada subpopulação, a escolha de qual dos três subordinados de um líder será o *subordinado1*, *subordinado2* e *subordinado3* nas expressões acima é feita aleatoriamente.

4.5.2 Seleção do procedimento de mutação

No contexto de MAs, em geral, a mutação é utilizada somente para manter a diversidade da população. Em cada geração, uma solução *Current* tem 30 % de probabilidade de ser escolhida. Se é selecionada, um movimento *one-city-insertion* [276] [272] é aplicado no tour escolhendo as cidades aleatoriamente. As cidades envolvidas no movimento têm seus *"don't look bits"* fixos em *"falso"* para permitir o uso de busca local como pontos críticos para reotimizar o tour.

4.5.3 Restart da população

Depois de uma série de gerações, a população naturalmente perde diversidade. O critério usado para detectar a convergência prematura de uma subpopulação é o número de gerações que a solução incumbente desta subpopulação não tenha se alterado. Quando este número de gerações sem melhora usando esse critério é igual a 20, os cinco procedimentos de mutação são aplicados na subpopulação correspondente, mas, neste caso, os cinco procedimentos de mutação são aplicados em todos os *Pockets* (ao invés dos *Currents*) desta subpopulação (com exceção da raiz da árvore). O procedimento de busca local não é aplicado à solução *Pocket* depois da mutação.

4.6 Experimentos computacionais e resultados

Como previamente mencionado, o propósito dos primeiros experimentos computacionais foi testar o desempenho do MA quando a busca local RAI age em conjunto com 4 operadores de recombinação diferentes. O melhor conjunto de instâncias disponível é constituído por 27 instâncias do MIN ATSP acessadas na `TSPLIB`, onde as soluções ótimas são conhecidas. Em todos os casos, reportamos os valores médios obtidos para uma instância depois de 30 execuções do MA. O código foi escrito em linguagem de programação Java, usando o *Java Development Kit 1.2*, e o computador foi um PC Pentium II, 400 MHz com 128 MB de RAM.

Dois critérios de parada foram utilizados: alcance de um número máximo de $13 * log(13) * log(N^2)$ gerações (*SC1* - para *"stopping criterion one"*) e encontrar o tour ótimo (*SC2*). Os resultados atingidos para cada operador de recombinação são mostrados na Tabela 4.1, onde $Otim./Tent.$ é a média de vezes que o método encontrou a solução ótima em 30 tentativas, *Gap* significa o desvio percentual médio das soluções ótimas $(100 * (Heuristic_solution - Optimal_Solution)/Optimal_Solution)$, *Gen*2 é o número médio de gerações realizadas usando o *SC1* e *SC2* (aquele que ocorrer primeiro), *CPU Tempo1* denota o tempo médio (em segundos) usando somente o *SC1* e *CPU Tempo2* denota o tempo médio (em segundos) usando os critérios *SC1* e *SC2* ao mesmo tempo.

Recombinação	$Otim./Tent.$	*Gap*	*Tempo1*	*Gen*2	*Tempo2*
SAX	22.8/30	0.07 %	58.33	113.20	6.07
DPX	22.6/30	0.06 %	228.87	121.35	28.85
MFNN	11.1/30	0.50 %	8.98	212.56	5.20
UNN	20.7/30	0.13 %	28.02	131.03	4.14

Tabela 4.1: *Comparação de quatro tipos diferentes de operadores de recombinação para o* MIN ATSP.

O melhor desempenho foi atingido pelo operador de recombinação SAX devido à relação *Gap* vs. $Tempo$ obtida.

Uma observação interessante pode ser tirada desta experiência: RAI demonstrou um desempenho muito bom, indiferentemente do operador de recombinação usado com a implementação do MA, mostrando que pode ser considerada uma busca local robusta para MAs projetados para resolver o MIN ATSP. A Tabela 4.1 particulariza o desempenho do MA com RAI e os 4 operadores de recombinação nas duas maiores instâncias assimétricas da `TSPLIB`. O método proposto teve sucesso resolvendo todas as 30 execuções na otimalidade em menos de 10 segundos por instância em média, não importando o operador de recombinação selecionado. Note que os tempos de execução variam em um intervalo estreito entre 4 e 10 segundos.

Nosso segundo experimento computacional foi uma comparação entre nosso RAI/SAX MA e seis outras meta-heurísticas recentemente propostas para resolver o MIN ATSP. Selecionamos da literatura cinco MAs de alta qualidade que empregam várias combinações de recombinação e algoritmos de busca local e eles têm os melhores resultados relatados na literatura. Nesta classe, escolhemos os MAs de Gorges-Schleuter [142], descrito na Figura 4.4 Merz e Freisleben [252], já discutido e apresentado na Figura 4.3, Nagata e Kobayashi [284], apresentado na Figura 4.6, e Walters [364], apresentado na Figura 4.5. As comparações também incluem o algoritmo *"non-standard ant colony"* proposto por Stützle e Dorigo [352] e o algoritmo *"Simulated Jumping"* de Amin [9]. *Simulated Jumping* (SJ) é uma metaheurística introduzida em 1997 por S. Amin e J.L. Fernández-Villacañas, e recentemente aplicada para o MIN ATSP e o MIN QUADRATIC ASSIGNMENT PROBLEM. Recorrendo a uma comparação com o bem conhecido algoritmo de *"Simulated Annealing"* (SA) que pode ser considerado como um processo de resfriamento lento, o algoritmo SJ pode ser entendido como uma série de rápidos processos de aquecimento e resfriamento. Há algumas outras diferenças com o SA. Por exemplo, não se permite ao sistema alcançar o equilíbrio em qualquer temperatura e o processo inicia-se com uma temperatura baixa. O leitor pode consultar [9] para mais detalhes do método proposto, assim como [270] onde é proposto o aquecimento, como estratégia de diversificação nos algoritmos meméticos baseados em SA.

A Tabela 4.2 e a Tabela 4.3 resumem os resultados publicados para estes métodos executados em instâncias da `TSPLIB`. A primeira coluna mostra o nome da instância `TSPLIB`. *PopSize* denota o tamanho da população utilizada pelos MAs, *Gap* é o desvio percentual médio relativo ao valor da solução ótima obtida em um certo número de execuções (tentativas), e *CPU time* é o tempo de execução médio (em segundos) usando os computadores e linguagens indicados. Dependendo dos experimentos conduzidos, os autores empregaram diferentes números de execuções para cada instância do `TSPLIB`. Por exemplo, Gorges-Schleuter, Merz e Freisleben, e Walters fizeram 20 execuções, enquanto Nagata e Kobayashi fizeram 30 execuções, e Stützle e Dorigo fizeram 25 execuções. Amin reporta somente *o melhor resultado de cinco tentativas* para cada instância.

Usamos 30 execuções em nossos experimentos. A linha "Promédios" das Tabelas 4.2 e 4.3 mostram a qualidade média e o tempo de CPU atingido por estes métodos. O desempenho médio de nosso método é calculado com a execução do algoritmo 30 vezes apenas para o conjunto de instâncias utilizado pelos diferentes métodos.

Comparação computacional com métodos anteriores (1997)							
Autor	Gorges-Schleuter		Merz e Freisleben		Nagata e Kobayashi		
Método	MA		MA		MA		
Hardware	SUN UltraSparc		DEC Alpha		Pentium		
Velocidade	170MHz		233MHz		200 MHz		
Linguagem	C		C++		C		
PopSize	Fixa (20)		Fixa (40)		Variável		
Execuções	20		20		30		
Instância	Gap %	Tempo	Gap %	Tempo	PSize	Gap %	Tempo
p43	0.00	2.1	0.00	11			
ry48p	0.00	3.5	0.20	72	100	0.00	3
ft53					100	0.56	5
ft70	0.00	4.4	0.00	111	150	0.00	10
kro124p	0.00	6.0	0.00	35	200	0.02	15
ftv170	0.00	9.5	0.26	100	350	0.00	60
rbg323					300	0.00	129
rbg358					350	0.00	172
rbg403					400	0.00	256
rbg443					450	0.00	268
Promédios	0.00	5.1	0.09	65.8		0.06	102
Nosso MA	0.00	4.09	0.00	4.09		0.00	9.46

Tabela 4.2: *Comparação do desempenho de nosso método com os de Gorges-Schleuter, Merz e Freisleben, e de Nagata e Kobayashi, usando o mesmo conjunto de instâncias assimétricas.*

A Tabela 4.4 reporta o desempenho detalhado de nosso método no conjunto completo de instâncias disponíveis no `TSPLIB`. Isto inclui 30 execuções para cada uma das 27 instâncias com tamanhos variando de 17 a 443 cidades. $Gen2$ é o número médio de gerações realizadas usando os critérios $SC1$ e $SC2$ e as colunas restantes são como na Tabela 4.1. O tamanho da população é fixo em 13 em todas as execuções.

Para os testes executados acima, o desvio percentual médio da solução ótima atingida pela solução do vizinho mais próximo foi de 33 % e o procedimento de busca local gastou 84 % do tempo total. Antes de analisar os resultados, deve ser observado que as comparações incluem computadores diferentes e códigos em linguagens/compiladores diferentes. O desempenho dos computadores é difícil de comparar, enquanto o compilador de Java usado em nossos experimentos deve ser otimisticamente calculado pelo menos em 25% mais lento que a maioria dos compiladores GNU C/C++. Os resultados médios mostrados na Tabela 4.2 indicam uma superioridade de nosso MA, mesmo quando as comparações foram feitas em um número limitado de instâncias. Pode ser dito que os métodos de Gorges-Schleuter e Walters comparam em termos de quali-

Comparação computacional com métodos anteriores (1998-99)							
Autor	Walters			Stützle e Dorigo		Amin	
Método	MA			Ant Colonies		Simulated Jumping	
Hardware	PC Pentium II			UltraSparc II		Dell PC	
Velocidade	300 MHz			167 MHz		300 MHz	
Linguagem	C			C		C	
PopSize	Fixa (20) 50 só para `ftv170`			Fixa (?)		1	
Execuções	20			25		melhor de 5	
Instância	PSize	Gap %	Tempo	Gap %	Tempo	Gap %	Tempo
`br17`						0.00	1
`ftv33`						0.00	71
`ftv35`						0.00	107
`ftv38`						0.00	65
`p43`						0.00	10
`ftv44`						0.00	940
`ftp47`						0.00	1450
`ry48p`	20	0.00	0.4	0.00	2.3	0.00	78
`ft53`						0.00	185
`ftv55`						0.00	1067
`ftv64`						0.00	70
`ft70`	20	0.00	2.0	0.00	37.2	0.00	430
`ftv70`						0.00	190
`kro124p`	20	0.00	1.6	0.00	7.3	0.00	1223
`ftv170`	50	0.00	7.3	0.00	56.2	1.49	482
`rbg323`						0.00	11140
`rbg358`						0.09	4308
`rbg403`						0.00	544
`rbg443`						0.00	11280
Promédios		0.00	2.8	0.00	25.7	0.08	1771
Nosso MA		0.00	5.0	0.00	5.0	0.02	4.76

Tabela 4.3: *Comparação do desempenho de nosso método com os de Walters, Stützle e Dorigo, e Amin usando o mesmo conjunto de instâncias assimétricas.*

dade de soluções e tempo de execução. Porém, as instâncias `TSPLIB` usadas em suas experiências podem ser consideradas fáceis. Note que as instâncias `ftv38`, `ftv44`, `ftv90`, `ftv100`, `ftv110`, `ftv120`, `ftv130`, `ftv140`, `ftv150`, e `ftv160` parecem ser mais difícies de resolver e a maioria delas não foi resolvida por qualquer método anterior. Outra observação concerne ao uso de parâmetros que dependem da instância sendo resolvida. Por exemplo, nos testes realizados por Nagata e Kobayashi, como também por Walters, o tamanho da população

é adaptado em função do tamanho da instância. Em nossos testes, mostrados na Tabela 4.4, *usamos os mesmos parâmetros para todas as 27 instâncias*. O algoritmo memético SAX/RAI foi capaz de encontrar no mínimo cinco soluções ótimas em todas as instâncias. Considerando o total de 810 execuções usadas nos experimentos computacionais, o método encontrou a solução ótima em 76 % das vezes com um desvio médio do ótimo de 0.07%. Excluindo as dez instâncias difíceis `ftv*` mencionadas acima, o *Gap* médio melhora para 0.0007%, e a porcentagem de soluções ótimas encontradas aumentou para 96.10 %, e o tempo computacional médio caiu para 5.17 segundos.

As Tabelas 4.5 e 4.6 reportam o desempenho de nosso método em algumas instâncias para o MIN ATSP geradas a partir de instâncias simétricas (com as soluções ótimas conhecidas) disponíveis em `TSPLIB`. Seguimos uma metodologia similar à sugerida por Fischetti e Toth [118] para gerar instâncias assimétricas a partir de simétricas. Suponha que fornecemos como entrada a matriz de distância diagonal superior de uma instância de TSP simétrica. Para criar uma instância assimétrica, precisamos dar valores à matriz de distância diagonal inferior. Seja c_{ij} a distância da cidade i à cidade j, $j > i$ (matriz diagonal superior) e s o valor da distância média computada para todo par i, j. Então, para todo par de cidades (j, i) tal que $j > i$, se o arco (j, i) não é parte do tour ótimo da instância simétrica, fixamos o custo entre as cidades $c_{ji} = c_{ij} + ks$, onde k é um fator uniformemente gerado de forma aleatória no intervalo $[0, K_{max}]$. Assim, dada uma matriz de distâncias simétrica correspondente a uma instância para o TSP com a solução ótima conhecida, somente alteramos incrementalmente os valores c_{ji} correspondentes aos arcos que não são parte do tour ótimo. Dito isto, asseguramos que o tour ótimo da instância assimétrica gerada ainda tem o mesmo custo da instância simétrica. Neste novo experimento testamos dois valores para K_{max}, 5% e 50%, para todas as instâncias que o tour ótimo está disponível em `TSPLIB`.

Os resultados da Tabela 4.6 mostram que o algoritmo tem um desempenho similar quando comparado aos resultados da Tabela 4.5. Para algumas instâncias a qualidade decresce e poucas soluções ótimas foram encontradas em 30 tentativas. A versão assimétrica da instância `brg180` pode ser classificada como difícil para nosso MA, possivelmente por causa da pobre população inicial encontrada pela nossa heurística do vizinho mais próximo (1007%). Note que a performance global do método não é muito sensível ao parâmetro K_{max}. O gerador de instâncias MIN ATSP usando as instâncias simétricas com a solução ótima conhecida foi implementada em linguagem de programação Java. Um bom aspecto de um gerador Java que implementamos é que os dados das instâncias geradas serão sempre os mesmos, indiferentemente do computador ou compilador utilizado. O gerador, como também os dados do problema, já estão em domínio público. A instância `pr2392`, neste caso, foi rodada apenas 5 vezes em lugar das 30, por isso a média não foi apresentada.

Instância	*Hits*	*Gap*	*Gen*1	Tempo1	*Gen*2	Tempo2	OPT time
br17	30	0.00	188	0.61	1.50	0.00	0.00
ftv33	30	0.00	235	1.25	11.33	0.09	0.09
ftv35	28	0.00	238	1.48	36.07	0.23	0.14
ftv38	5	0.11	244	1.52	215.33	1.33	0.47
p43	30	0.00	250	2.49	38.07	0.37	0.37
ftv44	18	0.27	253	2.05	141.07	1.12	0.57
ftv47	30	0.00	258	2.48	17.13	0.20	0.20
ry48p	27	0.00	258	2.68	133.73	1.34	1.15
ft53	27	0.00	264	2.96	74.93	0.89	0.69
ftv55	30	0.00	268	2.81	20.80	0.26	0.26
ftv64	30	0.00	278	3.71	32.27	0.49	0.49
ft70	22	0.01	283	6.22	155.27	3.27	2.61
ftv70	30	0.00	284	4.62	66.07	1.17	1.17
ftv90	13	0.19	300	6.11	189.33	3.85	1.11
kro124p	30	0.00	307	7.11	45.80	1.64	1.64
ftv100	20	0.10	307	8.05	146.73	3.83	1.92
ftv110	9	0.19	314	9.63	234.70	7.24	1.91
ftv120	10	0.16	319	12.07	242.50	9.16	3.84
ftv130	16	0.11	325	13.23	194.87	7.99	3.78
ftv140	5	0.26	330	15.98	297.20	14.47	9.84
ftv150	20	0.17	334	17.08	190.87	10.19	6.85
ftv160	11	0.42	338	20.78	270.40	16.83	10.38
ftv170	26	0.02	342	23.55	221.97	13.85	10.42
rbg323	30	0.00	385	220.51	27.63	16.83	16.83
rbg358	30	0.00	392	274.02	43.90	32.14	32.14
rbg403	30	0.00	400	384.03	4.27	7.55	7.55
rbg443	30	0.00	406	527.82	2.73	7.59	7.59
Promédios	22.8	0.07	300	58.33	113.20	6.07	4.59

Tabela 4.4: *O desempenho de nosso método em todas as instâncias assimétricas disponiveis na* TSPLIB.

4.7 Comparação extra: Resultados em instâncias simétricas

4.7.1 Sem estruturas de dados *"ad-hoc"* baseadas nas características geométricas das instâncias

É instrutivo mostrar o desempenho do método também para instâncias planares euclidianas do Min TSP. Esta "comparação" é de particular relevância, já que o algoritmo apresentado neste capítulo foi desenvolvido para a versão assimétrica, e não usa muitas das vantagens que as versões geométricas e simétricas do Min TSP têm para oferecer.

Por exemplo, Linhares em [233] descreve resultados da nova meta-heurística

Resultados de nosso SAX/RAI MA nas instâncias assimétricas construídas usando instâncias simétricas da TSPLIB ($K_{max} = 5\%$)							
	SC1				SC2		
Instância	*Hits*	Gap	Gen1	Tempo1	Gen2	Tempo2	OPT time
ulysses16	30	0.00	184	0.54	3.20	0.01	0.01
ulysses22	30	0.00	206	0.74	4.17	0.02	0.02
gr24	30	0.00	211	0.81	9.30	0.04	0.04
fri26	28	0.08	217	0.85	54.23	0.22	0.17
bayg29	30	0.00	224	1.04	4.90	0.03	0.03
bays29	30	0.00	224	1.05	6.93	0.04	0.04
att48	30	0.00	258	2.25	15.70	0.21	0.21
gr48	30	0.00	258	2.09	12.30	0.15	0.15
eil51	23	0.38	262	2.47	106.87	1.15	0.66
berlin52	30	0.00	263	2.23	5.10	0.09	0.09
st70	30	0.00	283	2.80	6.97	0.17	0.17
eil76	8	1.46	288	4.64	239.17	3.94	1.92
pr76	30	0.00	288	3.50	12.00	0.31	0.31
gr96	28	0.10	304	5.51	73.20	1.84	1.52
kroa100	29	0.14	307	5.09	37.63	0.90	0.72
kroc100	30	0.00	307	5.20	14.20	0.41	0.41
krod100	27	0.57	307	6.50	108.37	2.90	2.38
rd100	29	0.22	307	5.44	32.10	0.96	0.72
eil101	28	0.11	307	8.05	79.97	2.78	2.34
lin105	29	0.33	310	5.67	22.20	0.76	0.53
gr120	30	0.00	319	8.11	31.97	1.75	1.75
ch130	29	0.04	324	9.53	70.53	3.33	3.10
ch150	30	0.00	334	10.38	24.23	1.44	1.44
brg180	0	10.86	346	48.20	346.00	48.20	n.f.
gr202	29	0.05	354	23.15	74.63	9.02	8.45
tsp225	2	1.04	361	31.34	345.17	30.39	18.53
a280	19	0.41	375	39.50	224.20	27.54	19.47
pcb442	30	0.00	406	122.66	148.97	61.83	61.83
pa561	0	2.70	422	490.18	422.00	490.18	n.f.
gr666	3	1.03	433	721.49	417.33	713.81	480.92
pr1002	0	4.31	460	2132.28	460.00	2132.28	n.f.
pr2392	28	0.00	518	3833.22	239.77	2717.75	1372.11
Promédios	23.72	0.74	311.47	235.52	114.16	195.45	-

Tabela 4.5: *O desempenho de nosso método em um conjunto de instâncias assimétricas, com solução ótima conhecida, geradas de instâncias simétricas já resolvidas a otimalidade e disponiveis na TSPLIB, para* $K_{max} = 5\%$.

nomeada como *Busca Predatória*. Os resultados da comparação estão na Tabela 4.7.

O algoritmo de Busca Predatória foi executado 50 vezes com um código

Resultados de nosso SAX/RAI MA nas instâncias assimétricas construídas usando instâncias simétricas da TSPLIB ($K_{max} = 50\%$)							
	SC1				SC2		
Instância	*Hits*	Gap	Gen1	Tempo1	Gen2	Tempo2	OPT time
ulysses16	30	0.00	184	0.51	2.60	0.02	0.02
ulysses22	30	0.00	206	0.68	2.37	0.01	0.01
gr24	30	0.00	211	0.79	3.23	0.02	0.02
fri26	30	0.08	217	0.79	6.00	0.03	0.03
bayg29	30	0.00	224	0.93	2.00	0.01	0.01
bays29	30	0.00	224	0.94	2.10	0.01	0.01
att48	30	0.00	258	1.86	3.77	0.06	0.06
gr48	30	0.00	258	1.67	3.00	0.04	0.04
eil51	30	0.00	262	1.87	11.33	0.14	0.14
berlin52	30	0.00	263	2.08	2.27	0.04	0.04
st70	30	0.00	283	2.86	8.37	0.20	0.20
eil76	30	0.00	288	3.36	11.00	0.27	0.27
pr76	30	0.00	288	3.38	5.03	0.16	0.16
gr96	30	0.00	304	4.97	30.40	0.81	0.81
kroa100	30	0.00	307	4.79	8.10	0.31	0.31
kroc100	30	0.00	307	4.97	7.17	0.26	0.26
krod100	30	0.00	307	4.93	6.23	0.28	0.28
rd100	30	0.00	307	4.58	5.30	0.21	0.21
eil101	30	0.00	307	5.21	10.07	0.58	0.58
lin105	30	0.00	310	6.08	2.57	0.14	0.14
gr120	30	0.00	319	6.96	11.73	0.59	0.59
ch130	30	0.00	324	7.60	22.70	1.27	1.27
ch150	30	0.00	334	9.44	10.63	0.77	0.77
brg180	0	30.77	346	62.86	346.00	62.86	n.f.
gr202	25	0.36	354	26.27	155.50	15.49	12.15
tsp225	1	3.66	361	29.36	353.13	29.86	11.29
a280	0	2.57	375	47.71	375.00	47.71	n.f.
pcb442	30	0.00	406	102.74	56.60	30.05	30.05
pa561	28	0.02	422	194.62	190.10	128.35	125.29
gr666	0	5.75	433	1166.21	433.33	1166.21	n.f.
pr1002	0	13.86	460	3876.20	460.00	3876.20	n.f.
pr2392	0	10.80	518	39642.35	518.00	39642.35	n.f.
Promédios	23.45	2.12	311.47	1413.42	96.07	1406.39	-

Tabela 4.6: *O desempenho de nosso método em um conjunto de instâncias assimétricas, com solução ótima conhecida, geradas a partir de instâncias simétricas já resolvidas a otimalidade e disponíveis na TSPLIB, para* $K_{max} = 50\%$.

rodando em um PC usando um processador AMD 586 com 133 Mhz. A razão entre parênteses mostra quantas vezes a solução ótima foi encontrada em um total de 50 execuções. Os autores comentam que o tempo de execução cresce

Instância	Cidades	Ótimo	Busca Predatória	MA
`berlin52`	52	7542	8.42%-2.27% (26/50)	0.00 % (30/30)
			1 seg. - 9 seg.	0.15 seg.
`pr124`	124	59030	6.95%-1.58% (6/50)	0.00 % (30/30)
			29 seg. - 102 seg.	1.14 seg.
`rat195`	195	2323	3.62%-1.40% (4/50)	0.7% (0/30)
			443 seg. - 1043 seg.	30.35 seg.
`m400`	400	4000	0.20%-0.07% (33/50)	n.a.
			3510 seg. - 6294 seg.	

Tabela 4.7: *Comparação com Busca Predatória em instâncias simétricas da `TSPLIB`.*

como $O(N^2)$ com N sendo o número de cidades na instância. Um processador 133 MHz foi também utilizado neste estudo. A comparação com *Busca Tabu Incremental* (proposta por Linhares e Gomes) na Tabela 4.8 pode também ajudar a avaliar o método.

Em [234], os autores apresentaram uma meta-heurística nomeada como *otimização microcanônica* (μO) e foi testada a performance do método utilizando um conjunto de seis instâncias do TSP euclidiano 2-dimensional, variando de 76 a 439 cidades. A partir de experimentos computacionais conduzidos por eles, os autores apontam a superioridade de μO em relação a metodologias como Simulated Annealing, Annealing Microcanônico, Tsallis Annealing, e Busca Tabu. Na Tabela 4.9 apresentamos os resultados obtidos com μO nas instâncias que pertencem ao `TSPLIB` e para as quais foram dados os melhores e piores resultados em 50 execuções.

4.7.2 Explorando a natureza geométrica das instâncias

O algoritmo memético mais rápido para instâncias euclidianas planares do MIN TSP é provavelmente o desenvolvido por Merz e Freisleben [257]. Acreditamos que é provável ser a heurística mais eficiente para este problema que conhecemos. Reproduzimos na Tabela 4.10 os resultados apresentados em sua última publicação [257]. A coluna DPX indica o tempo médio requerido pelo seu MA baseado em Busca Local usando somente *Distance Preserving Crossover* (DPX). Apresenta o tempo médio em segundos necessários para alcançar o ótimo (que foi encontrado em um conjunto de 30 execuções independentes). O MA baseado em mutação apenas mostra resultados piores, mostrando o benefício de empregar alguma forma de recombinação. O algoritmo de busca local utilizado é baseado na heurística de Lin-Kernighan enquanto que a mutação é realizada com um operador não-seqüencial do tipo *"four-change"* [231].

Finalmente, no mesmo formato mostrado nas Tabelas 4.6 e 4.5 apresentamos na Tabela 4.11 os resultados para instâncias euclidianas no plano. Isto corresponderia a um valor de $K_{max} = 0$, i.e. sem perturbação das instâncias e ajudaria a avaliar os efeitos de aumentar a assimetria em conexão com nosso método. Também neste caso o algoritmo só executa cinco vezes usando a

Instância	Cidades	Opt.	Incremental TS	MA
berlin52	52	7542	0.00%	0.00 %
			19 seg.	0.15 seg.
pr76	76	108159	0.85%	0.00 %
			36 seg.	0.52 seg.
rat99	99	1211	0.99%	0.23 %
			60 seg.	2.65 seg.
grid100	100	3700	0.00%	
			61 seg.	
kroa100	100	21282	0.41%	0.05 %
			62 seg.	1.6 seg.
krob100	100	22141	2.49%	0.19 %
			66 seg.	5.48 seg.
pr124	124	59030	0.89%	0.00 %
			96 seg.	1.14 seg.
rat195	195	2323	1.51%	0.7 %
			270 seg.	30.35 seg.
kroa200	200	29368	1.86%	0.07 %
			287 seg.	17.65 seg.
grid400	400	9000	1.07%	
			2123 seg.	
pr439	439	107217	3.20%	0.48 %
			1572 seg.	234 seg.
pcb442	442	50778	3.48%	1.50 %
			1594 seg.	406 seg.

Tabela 4.8: *Comparação com Busca Tabu Incremental em instâncias simétricas da TSPLIB.*

Instância	Cidades	Opt.	μO	MA
pr76	76	108159	108159 - 109085	108159
			52 seg. - 154 seg.	0.52 seg.
kroa200	200	29368	29696 - 30941	29389
			52 seg. - 155 seg.	17.65 seg.
pr439	439	107217	>108000 - >124000	107732
			>300 seg. - >1260 seg.	234 seg.

Tabela 4.9: *Comparação com otimização microcanônica em instâncias simétricas da TSPLIB.*

instância pr2392 em lugar das 30, por isso promédio não foi apresentado.

Instância	Cidades	DPX	Mutação	MA SAX/RAI
`lin318`	318	25 seg.	38 seg.	
`pcb442`	442	35 seg.	63 seg.	1.5 % - 406 seg.
`att532`	532	131 seg.	230 seg.	1.13 % - 735 seg.
`rat783`	783	61	88 seg.	

Tabela 4.10: *Comparação com Merz e Freisleben em instâncias simétricas euclidianas no plano da TSPLIB.*

4.8 Conclusões

Propusemos um novo algoritmo memético com um novo tipo de busca local (o *Recursive Arc Insertion* especificamente projetado para instâncias do MIN ATSP. Nossa implementação de um MA introduz diversos procedimentos para melhorar o desempenho, como uma população hierarquicamente estruturada organizada como uma árvore ternária completa de 13 agentes agrupados em 4 subpopulações. Uma subpopulação é composta de um líder e 3 subordinados. Cada agente manipula duas soluções factíveis: uma *Current* usada para operações realizadas pelo método como a recombinação, mutação e busca local, e uma solução *Pocket* que é atualizada com a solução *Current* sempre que a última tenha uma solução de melhor qualidade que uma armazenada na *Pocket*. A árvore ternária é atualizada de tal modo que o nó raiz sempre contenha a melhor solução encontrada até o momento. O mecanismo de atualização consiste em trocar a solução *Pocket* do líder com a solução *Pocket* de um de seus subordinados sempre que a última tenha uma solução de melhor qualidade que a armazenada na primeira. Testamos quatro operadores de recombinação agindo em conjunto com a busca local RAI. Resultados computacionais visando selecionar o operador mostraram que o melhor desempenho foi obtido pelo *Strategic Arc Crossover*.

Experimentos computacionais adicionais foram conduzidos em 27 instâncias para ATSP do `TSPLIB` (30 execuções para cada instância). Nosso método foi capaz de encontrar 76% de soluções ótimas em um tempo médio de CPU de 6.07 segundos com uma qualidade média de soluções de 0.07%. Testes com outra classe de instâncias assimétricas geradas a partir de tours ótimos conhecidos de instâncias simétricas do `TSPLIB` mostraram comportamento similar do método. Comparações usando um pequeno subconjunto de instâncias com 6 outras meta-heurísticas demonstraram que nosso método superou as metodologias anteriores. Acreditamos que o desempenho de nosso método pudesse ser melhorado se os parâmetros fossem ajustados ao conjunto específico de instâncias usado para comparações. Ao invés disso, como robustez é uma característica chave para a aplicabilidade do método, nossos parâmetros foram colocados para o conjunto inteiro de instâncias e foram usados da mesma maneira em todos os testes. A pesquisa futura inclui a utilização de heurísticas construtivas mais eficazes para encontrar a população inicial, visto que nossa população inicial é muito pequena. Sabe-se que as metodologias populacionais convergem rapi-

Resultados de nosso SAX/RAI MA em instâncias euclidianas no plano							
	SC1				SC2		
Instância	*Hits*	Gap	Gen1	Tempo1	Gen2	Tempo2	OPT time
ulysses16	30	0.00	184	0.63	2.40	0.02	0.02
ulysses22	30	0.00	206	0.84	2.27	0.01	0.01
gr24	30	0.00	211	0.90	3.10	0.02	0.02
fri26	30	0.00	217	0.98	1.03	0.00	0.00
bayg29	30	0.00	224	1.13	4.23	0.03	0.03
bays29	30	0.00	224	1.17	5.70	0.04	0.04
att48	30	0.00	258	2.83	42.37	0.54	0.54
gr48	29	0.00	258	2.48	33.03	0.36	0.30
eil51	20	0.08	262	2.65	103.13	1.10	0.32
berlin52	30	0.00	263	2.49	7.70	0.15	0.15
st70	30	0.00	283	4.71	42.23	0.89	0.89
eil76	30	0.00	288	4.92	45.17	1.04	1.04
pr76	30	0.00	288	4.47	18.83	0.52	0.52
gr96	13	0.10	304	6.89	189.67	4.56	1.41
kroa100	29	0.05	307	7.14	52.43	1.60	1.36
kroc100	30	0.00	307	6.67	23.83	0.85	0.85
krod100	16	0.34	307	7.70	183.33	4.95	2.37
rd100	30	0.00	307	7.13	92.80	2.70	2.70
eil101	24	0.15	307	9.43	150.83	5.23	4.12
lin105	30	0.00	310	6.41	23.70	0.91	0.91
gr120	09	0.37	319	13.46	277.57	12.17	8.74
ch130	23	0.18	324	13.05	180.97	8.37	6.89
ch150	12	0.40	334	16.92	244.37	13.27	7.23
brg180	04	2.46	346	36.49	300.00	31.66	0.02
gr202	04	0.40	354	41.59	342.30	40.71	36.89
tsp225	00	1.16	361	53.05	361.00	53.05	n.f.
a280	06	0.53	375	75.53	241.80	53.05	48.35
pcb442	00	1.50	406	405.94	406.00	405.94	n.f.
pa561	00	4.07	422	687.98	422.00	687.98	n.f.
gr666	00	2.51	433	1383.79	433.00	1383.79	n.f.
pr1002	00	5.12	460	3630.90	460.00	3630.90	n.f.
pr2392	00	7.26	518	36908.75	518.00	36908.75	n.f.
Promédios	19.03	0.84	311.47	1352.26	166.09	1354.66	-

Tabela 4.11: *O desempenho de nosso método em um conjunto de instâncias euclidianas disponíveis na TSPLIB.*

damente para soluções próximas da ótima quando são adicionados indivíduos de boa qualidade à população inicial. Por exemplo, os métodos construtivos introduzidos por Glover, Gutin, Yeo e Zverovich (em [135]) possivelmente melhorariam o desempenho global de nosso método. Tambem o uso de instâncias "fractais" para testar a performance, como proposto em [242] [243] [294] [277]

para o caso euclídeano e interessante de pode ser feito modificando-as para torna-las assimétricas com o gerador apresentado nesse capítulo.

Capítulo 5

O problema Min Number Partitioning

> *"Como pagamento de pequeno lote de carneiros (três criadores) receberam aqui em Bagdá uma partida de vinho, muito fino, composta de 21 vasos iguais, sendo: 7 cheios, 7 meio-cheios, 7 vazios. Querem, agora, dividir os 21 vasos de modo que cada um deles receba o mesmo número de vasos e a mesma porção de vinho. Repartir os vasos é facil. Cada um dos sócios deve ficar com sete vasos. A dificuldade, a meu ver, está em repartir o vinho sem abrir os vasos, isto é, conservando-os exatamente como estão. Será possível, ó Calculista, obter uma solucão para este problema ?"*

De *"O Homem que Calculava"* de Malba Tahan, Capítulo VIII.

5.1 Introdução

Houve muitas tentativas frustradas para classificar o que faz um problema ser *"difícil"* para os Algoritmos Genéticos e isto também estende-se ao campo da Computação Evolutiva. Em relação ao campo das metaheurísticas, sob um ponto de vista científico, é frustante ver que a maioria dos resultados informam aplicações de "sucesso" de uma certa técnica, enquanto muitos resultados *"negativos"* e fracassos raramente chegam a ser publicados. Uma exceção notável para a regra é o estudo de Johnson, Aragon, McGeoch, e Schevon na aplicação de *"Simulated Annealing"* [275] para o problema por eles chamados de Min Number Partitioning [181].

O Number Partitioning (D) pertence à classe de problemas NP-completos e representa certamente um desafio em aberto para metaheurísticas amplamente usadas como os *Algoritmos Genéticos* (GAs) [184] [334] e o *"Simulated Annealing"* [181] [348] [335], *"Problem Space Local Search"* [351], GRASP [10], e Busca Tabu [136].

O propósito deste capítulo é ajudar a entender as razões para este aparente fracasso, identificar suas causas e sua extensão, e prover novas perspectivas que

ajudem a entender os principais mecanismos responsáveis do sucesso de uma implementação, assim como entender a razão dos fracassos, dos métodos de Computação Evolutiva para otimização. Em particular, trataremos o problema das limitações de um método quando uma representação inadequada para as soluções do problema foi escolhida.

5.1.1 O Problema

O problema NUMBER PARTITIONING (D) é amplamente citado como sendo um membro distinto na classe dos NP-completos. Seguindo o estilo de apresentação de Garey e Johnson [131], enunciaremos o problema como o seguinte par Dados-Questão:

NUMBER PARTITIONING (D)

Input: Um conjunto A de n números inteiros positivos $\{a_1, \ldots, a_n\}$.

Questão: Existe uma partição de A, i.e. dois conjuntos disjuntos A_1 e A_2 com $A = A_1 \bigcup A_2$, tal que

$$\left| \sum_{a_i \in A_1} a_i - \sum_{a_j \in A_2} a_j \right| \leq 1 \ ? \tag{5.1}$$

Como declarado, NUMBER PARTITIONING (D) é um problema de *decisão*. Isto significa que para cada *instância* (cada possível conjunto A de n números inteiros constitui uma instância $x \in I_P$ em nossa notação), os únicos tipos de resposta são 'Sim' ou um 'Não'. Associado a este problema, há um problema de *otimização* combinatória ou uma *versão de otimização* (denotada MIN NUMBER PARTITIONING (O)). Este problema pode ser visto como a tarefa de encontrar um conjunto $y = \{v_1, \ldots, v_n\}$, onde v_i pode ser ou 1 ou -1, tal que y minimiza a seguinte função objetivo (neste caso podemos chamá-la de *custo*)

$$m_P(y, x) = \left| \sum_{i=1}^{n} a_i v_i \right| . \tag{5.2}$$

MIN NUMBER PARTITIONING (O) é fácil de descrever, mas difícil de resolver. Tem sido mostrado ser particularmente difícil para as metaheurísticas como *"Simulated Annealing"* [181] [348] e *Algoritmos Genéticos* [184] [334].

5.1.2 Problemas NP-completos no sentido fraco

É importante remarcar uma diferença entre MIN NUMBER PARTITIONING (O) e outros problemas de otimização da classe NP. Nesse caso, o problema pode ser resolvido com um *algoritmo pseudo-polinomial*. Isto significa que, se A é um conjunto de n inteiros tal que a soma deles é B, então existe um algoritmo que pode encontrar a partição de custo mínimo em tempo nB, i.e. polinomial em n. Dizemos "tempo pseudo-polinomial" porque nB não é limitado por

nenhum polinômio em $nlog_2(B)$, que é uma medida do espaço necessário para armazenar a instância A.

O algoritmo pseudo-polinomial é derivado do fato de que, para qualquer subconjunto $A' \subseteq A$, a soma $\sum_{x_i \in A'} x_i$ deve ser um inteiro entre 1 e B. Por programação dinâmica, é então perguntado se existe $A' \subseteq A$ tal que $\sum_{x_i \in S'} x_i = 1, 2, \ldots$ e assim por diante até $\lfloor B/2 \rfloor$. Um valor de u da função objetivo é atingível se e somente se $(B-u)/2$ é uma soma parcial possível; então, a soma parcial mais próxima de $B/2$ dá o mínimo u. O algoritmo pseudo-polinomial baseado em programação dinâmica requer tempo e espaço linear pelo menos em B. Infelizmente a utilidade do algoritmo pode ser limitada na prática quando B é grande. Lembramos ao leitor que já discutimos algoritmos pseudo-polinomiais na seção 3.3.4.

Alguns outros problemas de decisão NP-completos, tais como SATISFIABILITY, HAMILTONIAN PATH, BIN PACKING, TRAVELING SALESMAN PROBLEM, TRIPARTITE MATCHING, etc., como também outros problemas estudados nessa tese, permanecem NP-completos, mesmo quando se restringe uma instância de tamanho n a conter inteiros de tamanho no máximo $p(n)$. Estes problemas são chamados de fortemente NP-completos. Sob a conjetura que P $\neq$ NP, problemas fortemente NP-completos não têm algoritmos pseudo-polinomiais que os resolvem [131].

5.2 Motivação

Seguindo a discussão anterior, o problema que estudaremos é um membro da classe para qual existe um algoritmo pseudo-polinomial. Porém, foi mostrado ser muito difícil para metaheurísticas baseadas em Busca Local (como SA) e Buscas Baseadas em População (como GAs). Também foi um desafio difícil para uma metaheurística conhecida como GRASP [10]. Pensamos que um melhor entendimento das deficiências destes métodos resultariam em metaheurísticas melhores, e provavelmente mais robustas. Também apontaria dificuldades inerentes a problemas NP-completos *fracos*. Finalmente, poderia nos ajudar no futuro para uma melhor caracterização (i.e. em condições próprias de complexidade computational) do que pode ser visto como *"hardness"* para métodos baseados em computação evolutiva.

Em [181], Johnson, Aragon, McGeoch e Schevon estudaram o desempenho de MIN NUMBER PARTITIONING (O) e também de dois outros problemas NPO: MIN GRAPH COLORING e MIN GRAPH PARTITIONING. Em contraste, os autores concluem:

> *"Os resultados para Number Partitioning foram, como esperado, decididamente negativos; o simulated annealing foi superado pelo substancialmente mais rápido algoritmo de Karmarkar-Karp e até mesmo batido (em base de tempo) pelo multiple-start local otimisation (MSLO)."*

O algoritmo de Karmarkar-Karp é um algoritmo construtivo rápido que, para a família de instâncias que eles estudaram, parece que superou o desempenho

de SA até mesmo quando o segundo é permitido ter execuções longas. Eles concluem o seguinte:

> *Assim, para* NUMBER PARTITIONING, *pelo menos* ***para os tipos de instâncias aleatórias*** *que consideramos, ilustra as limitações de* "Simulated Annealing" *como uma técnica geral.* ***Quando o espaço de soluções é suficientemente acidentado****, a vantagem de* "Simulated Annealing" *sobre* "multiple start local otimisation" *pode ser inteiramente perdida. Além disso,* ***outras metodologias, não vinculadas ao conceito de navegar em um espaço de solução****, podem ser capaz de superá-la substancialmente.*

Nós acrescentamos o uso de ênfase para destacar a motivação para esta experimentação. De um ponto de vista prático, é também interessante ver se o que podemos aprender quando tratamos este problema pode ser também estendido para outros problemas de otimização da classe NP. Sabe-se que MIN NUMBER PARTITIONING (O) está relacionado a outros problemas de decisão como BIN PACKING, PRODUCTION PLANNING, KNAPSACK, OPEN-SHOP SCHEDULING, SCHEDULING TO MINIMIZE WEIGHTED COMPLETION TIME, e outros problemas listados no Apêndice A de [131].

Jones e Beltramo, em [184] usaram um GA para uma generalização de NUMBER PARTITIONING (O), o qual tem sido denominado como *"problema das pilhas iguais"*, no qual o objetivo é dividir um conjunto de números em $k \geq 2$ subconjuntos de exatamente a mesma soma. Obviamente NUMBER PARTITIONING (O) é um caso especial desse problema quando $k = 2$.

5.2.1 Relação com outros problemas NP

5.2.1.1 O Problema da Mochila 0/1 (D)

O problema da MOCHILA 0/1 (D), conhecido na língua inglesa como EXACT KNAPSACK 0/1, é o seguinte:

EXACT KNAPSACK 0/1

Input: Um conjunto I de m inteiros $\{a_1, \ldots, a_m\}$ e um inteiro C (a capacidade da mochila), todos representados em binário.

Questão: Há um subconjunto de inteiros tais que a soma seja exatamente C ?

Este problema pode ser reduzido a NUMBER PARTITIONING (D). Seja $z = \sum_{i=1}^{m} a_i$. Assumimos que $C > z/2$. Seja $d = 2C - z$. Procedemos somando o inteiro d ao conjunto I. Se este conjunto de $n = m + 1$ inteiros pode ser perfeitamente particionado (i.e. se temos uma resposta 'Sim'), então o subconjunto de partição perfeita que não contém o inteiro d (vamos chamá-lo de S_1) é um subconjunto dos números inteiros originais e sua soma é exatamente C. Agora suponha que o valor da soma de elementos de S_1 é z_1, então temos uma partição perfeita $z_1 = 2C - z + (z - z_1)$ é $C = z_1$. Se não temos uma

partição perfeita, então não há solução para o problema da KNAPSACK 0/1 (D). Assumimos que $C > z/2$, se $C \leq z/2$ o mesmo argumento se aplica mas com $z - C$ ao invés de C.

Além destes fatos, o problema de decisão tem também aplicações práticas que surgem em criptografia. As instâncias onde NUMBER PARTITIONING (D) tem uma resposta 'Sim' podem ser usadas para aplicações de criptografia de acordo com [343].

5.2.1.2 O problema de *"scheduling"* em duas máquinas

Em [225] podemos encontrar outra relação deste problema com o campo de "scheduling". O problema TWO-MACHINE SCHEDULING (D) é o seguinte:

TWO-MACHINE SCHEDULING (D)

Input: Um conjunto de m tarefas que podem ser executadas em duas máquinas. As tarefas têm tempos de execução idênticos $\{a_1, \ldots, a_m\}$ em qualquer máquina. Um prazo final D, todos representados em binário.

Questão: É possível completar todas as tarefas antes do prazo final ?

Suponha agora temos uma instância de TWO-MACHINE SCHEDULING (D) com tempos de execução $\{a_1, \ldots, a_m\}$, e um prazo final D. Faça $Id = 2D - \sum_i^m a_i$. Representa o *tempo inativo* em qualquer solução factível (i.e. um schedule que satisfaz o prazo final). Uma interpretação alternativa para Id é como a quantia de *inatividade* ou *"slack"* que temos resolvendo o problema de scheduling. A seguir, somaremos vários números novos para o conjunto de durações de tarefas, tal que: 1) a soma destes números novos é Id; e 2) além disso, nós podemos compor qualquer soma entre 0 e Id, somando um subconjunto destes números novos. Não é difícil ver que, se pudéssemos fazer isto, a instância de NUMBER PARTITIONING assim criada seria equivalente à instância original de TWO-MACHINE SCHEDULING (D). Então, podemos transformar qualquer schedule factível das tarefas originais em uma partição eqüitativa do conjunto novo de números somando, a cada um dos dois conjuntos, um subconjunto dos números recentemente introduzidos que trarão a soma de ambos para D.

Dito isso, tudo que temos que fazer é fornecer um conjunto de inteiros e adicionar Id, de forma que qualquer número entre zero e Id é a soma desses números. À primeira vista, isso parece ser uma tarefa trivial para fazer: *"apenas adicionando Id cópias de inteiros 1"*. Isto seria a tarefa errada a fazer se pensássemos em ter uma redução em *tempo polinomial*, pela mesma razão que o algoritmo Da Figura 3.2 é pseudo-polinomial. A razão é que o inteiro Id não é limitado por um polinomio no tamanho do input. Recorde que o input consiste das durações de tarefa e o prazo final, tudo codificado em binário, o número Id poderia ser exponencialmente grande no tamanho do input.

Isto requer pensar um pouco mais. Para sair da situação, uma alternativa é somar números que são todos potências de 2 e menores que $Id/2$, adicionar outro inteiro de tal modo que a soma alcance Id. Então se $Id = 243$, adicionamos os números $\{1, 2, 4, 8, 16, 32, 64, 116\}$. Com isto, a descrição da redução a partir de TWO-MACHINE SCHEDULING (D) a NUMBER PARTITIONING está

completa. A redução é polinomial desde que os números inteiros que introduzimos sejam limitados pelo *logaritmo de Id* e *Id* é menor que o tamanho do input.

5.2.1.3 Aproximabilidade

Para TWO-MACHINE SCHEDULING (D) não há nenhum limite de quão próximo do ótimo podemos estar; para qualquer $\epsilon > 0$, há um algoritmo ϵ-aproximável para este problema.

Esta família de algoritmos está baseada em uma idéia que já vimos. Na Seção 3.2, mostramos que o problema EXACT NUMBER PARTITIONING pode ser resolvido em tempo $O(nS)$ (onde n é um número de inteiros, e S é sua soma). É muito fácil ver que este algoritmo é trivialmente adaptável para resolver o problema TWO-MACHINE SCHEDULING (D) (i.e. encontrar o menor D): Os conjuntos $B(i)$ são estendidos para incluir somas até S (não só até $H = S/2$). A menor soma em $B(n)$, que é $\geq S/2$, é o mínimo desejado D.

Uma outra idéia é necessária para chegar ao nosso algoritmo de aproximação. Novamente recorremos a um exemplo numérico que pode ser encontrado em [225]. Considere uma instância de TWO-MACHINE SCHEDULING (D) com estas durações de tarefas:

45362, 134537, 85879, 56390, 145627, 197342, 83625, 126789, 38562, 75402,

com $n = 10$ e $S \approx 10^6$. Resolvendo com nosso algoritmo exato $O(nS)$ ele necessitaria 10^7 passos, um alto custo computacional. Mas suponha que arredondamos para cima as durações de tarefa à próxima centena . Nós obtemos a instância:

45400, 134600, 85900, 56400, 145700, 197400, 83700, 126800, 38600, 75500,

que é *a mesma instância*:

454, 1346, 859, 564, 1457, 1974, 837, 1268, 386, 755,

normalizada em 100. Assim, podemos resolver esta instância em aproximadamente 10^5 passos. Sacrificando um pouco em precisão (o ótimo do novo problema é claramente não muito longe do original), diminuímos as exigências de tempo em duas ordens de magnitude.

Se arredondamos para cima para a próxima k-ésima potência de dez, a diferença entre as duas soluções ótimas não pode ser maior que $n10^k$. Para calcular o erro relativo, esta quantidade deve ser dividida pelo ótimo, que, obviamente, não pode ser menor que $S/2$. Temos assim um algoritmo de aproximação $2n10^k/S$, que requer tempo de execução $O(nS/10^k)$. Fixando $2n10^k/S$ igual a qualquer valor $\epsilon > 0$ desejável, chegamos a um algoritmo que tem um tempo de execução $O(n^2/\epsilon)$ que é certamente polinomial em n.

5.3 A Heurística de Karmarkar-Karp

A heurística de Karmarkar-Karp (KK) [190] trabalha substituindo os dois números maiores no conjunto pela diferença entre eles. Isto é uma conseqüência de decidir que cada número vá para diferentes conjuntos e a diferença representa o valor da partição. Por exemplo, considere o conjunto de inteiros $\{15, 12, 10, 9, 4\}$. Se pusermos o número 15 na partição A_1 e o número 12 na partição A_2, então substituímos estes por 3 (15-12). Este valor 3 é equivalente a colocar a diferença no conjunto A_1. O KK trabalha repetindo o processo até que só reste um número que é o valor da partição. Se continuamos o exemplo, teremos

números	partições	diferenças
15,12,10,9,4	-	-
3,10,9,4	{15} {12}	3
3,1,4	{15}{12} e {10}{9}	3,1
1,1	{15}{12,4} e {10}{9}	1,1
0	{15,10}{12,4,9}	0

Tabela 5.1: *Uma execução-exemplo da heurística de Karmarkar-Karp.*

Dado um conjunto $A = \{a_1, a_2, \ldots, a_n\}$ de n inteiros, a heurística KK gera uma árvore que tem inteiros do conjunto A como folhas ou nós. Quando uma árvore que tem n nós é finalmente completada, uma partição dos nós é feita criando uma coloração válida da árvore com duas cores. Cada cor representa uma partição de A; a partição em dois conjuntos é única. Esta parte do algoritmo é executada em tempo $O(n)$.

A árvore é gerada do seguinte modo: começamos com n nós isolados (nó i representará o inteiro a_i), um para cada elemento de A. Inicialmente, um peso de $w(i) = a_i$ é nomeado a cada nó i. Declaramos todos como *'vivos'*. Então os seguintes dois passos, são repetidos até que só um nó esteja "vivo".

- Encontre os dois nós vivos u e v com o maior peso (empates são escolhidos arbitrariamente). Assumimos que $w(u) \geq w(v)$.
- (Criação de uma nova aresta) Ligamos u e v com uma aresta e declaramos v como *'morto'*. Ao peso do nó u é dado um novo valor subtraindo o valor de $w(v)$

É fácil de ver que o segundo passo é uma decisão *relativa* de colocar os nós u e v em diferentes partições. A decisão *"absoluta"* corresponde ao passo final que fornece uma coloração válida da árvore com duas cores. Seguindo estes dois passos o valor final produzido é o peso do último nó vivo.

Um importante algoritmo ótimo que usa KK é o *"Complete Karmarkar-Karp"* (CKK) [208] [209]. CKK é um algoritmo exato que tira vantagens da performance de KK.

5.4 Resultados anteriores

5.4.1 Fórmulas de custo esperadas assintoticamente

Para alguns tipos de instâncias, fórmulas de custo assintóticas podem ser derivadas. Em relação aos resultados de Karmarkar *et al.* [191] os autores de [181] estabeleceram várias experiências que usam instâncias que chamaremos UI_n. Cada uma é um conjunto de n números reais gerados aleatoriamente com probabilidade uniforme no intervalo $[0, 1]$. Esta arena experimental foi escolhida devido ao resultado teórico previamente conhecido que mostra que se $OPT(UI_n)$ é o custo do mínimo global de uma instância UI_n, então o valor médio em cima dessas instâncias é $\langle OPT(UI_n)\rangle = O(2^{-n}\sqrt{n})$ [191]. O problema de avaliar a fórmula exata adquiriu recente atenção de físicos que trabalham em mecânica estatística de sistemas desordenados. Ferreira e Fontanari [115] usaram a *"annealed approximation"* para derivar um rigoroso limitante inferior

$$\langle OPT(UI_n)\rangle \geq 2^{-n}\sqrt{\frac{\pi n}{6e^2}} \approx 0.266\ 2^{-n}\sqrt{n}. \tag{5.3}$$

Muito recentemente [250], S. Mertens forneceu uma expressão exata para esta quantidade:

$$\langle OPT(UI_n)\rangle = \sqrt{\frac{2\pi}{3}}2^{-n}\sqrt{n}. \tag{5.4}$$

Veja também [238] para resultados neste tópico e o problema relacionado de "Subset-Sum".

5.4.2 A diferença entre a otimidade local e global

Para o mesmo tipo de instâncias, o valor esperado do algoritmo de Karmarkar e Karp é $O(1/n^{\alpha log(n)})$ para um $\alpha > 0$. Fischetti e Martello fizeram uma análise de pior caso em [117].

Deve ser mencionado que Johnson *et al.* comentaram que este resultado foi demonstrado para uma variante do algoritmo KK descrito acima [181]. Esta variante simplificou o tratamento analítico e eles notaram que não há nenhuma razão para acreditar que é um pouco melhor que o algoritmo original. Este assunto foi recentemente tratado por B. Yakir [378] (para pesquisas anteriores desse assunto veja [237], [76], e [76]). É fácil observar que existe uma grande diferença entre o custo das soluções ótimas e as fornecidas por KK. A implementação de *"Simulated Annealing"* de Johnson e seus colegas não foram capazes de superar esta diferença.

Johnson *et al.* definem uma série de grafos de vizinhança para definir a transição entre configurações (nesse caso são partições). Por exemplo, quando movemos um inteiro a_j de A_1 para A_2 isso corresponde a uma mudança de sinal da variável associada v_j (quando usamos a representação direta). O grafo de vizinhança associado é nomeado como SW_1. Em geral, eles definem o grafo de vizinhança SW_k que é gerado conectando configurações que são o resultado de intercambiar k ou menos elementos.

5.4.3 A falha de *"Simulated Annealing"*

Podemos então identificar um dos fatores que fazem a implementação direta de SA usando o grafo de vizinhança SW_1 ser um fracasso. Seguindo a discussão do capítulo de introdução aos algoritmos meméticos, notamos que o fracasso é em uma *paisagem de aptidão* que combina o grafo de vizinhança e a função-guia. No caso geral, o valor esperado da menor diferença de custo entre duas configurações vizinhas em SW_k é $1/n^k$. Isto está novamente em contraste com o fato de que o valor esperado do valor ótimo para UI_n, dado por $\langle OPT(UI_n)\rangle$, é aproximadamente $1.447\sqrt{n}2^{-n}$. Dito isto, é claro que esperamos que os melhores mínimos locais sejam localizados em *"vales muito profundos"* no espaço de configurações, visto que eles são rodeados por soluções de custo mais alto. Esta situação se torna cada vez pior com o aumento do tamanho do problema.

Para SA, isto não é definitivamente uma boa paisagem de aptidão. Isto significa que para se mover em SW_k precisamos de temperaturas que são ordens de magnitude maiores (fica necessário fazer saltos de ordem $1/n^k$ na função de custo) que aqueles necessários para eficientemente amostrar (usando o algoritmo de Metropolis) aquelas regiões do espaço de configurações que têm mínimos locais de boa qualidade (que são da ordem $O(\sqrt{n}/2^n)$). Se o valor esperado do resíduo do algoritmo de KK é $O(1/n^{\alpha log(n)})$, assintoticamente menor que o mínimo valor de custo em SW_k, então KK (embora um algoritmo construtivo) é uma estratégia notavelmente competitiva quando confrontada com uma implementação de SA baseada em SW_k.

Os resultados computacionais em [181] confirmam essas hipóteses. Para uma instância com 100 elementos (veja Fig. 9 em Ref. [181]) embora a melhor solução encontrada por SA seja melhor que KK, o valor para qual o qual SA convergiu (usando SW_2) foi radicalmente pior que KK. Deve-se enfatizar que o valor final de SA só era ligeiramente pior que a média nos últimos 25% do tempo de CPU utilizado por SA. Um modo de interpretar este fato é que SA ficou preso em uma região não promissora do espaço de configurações depois que a temperatura atingiu valores menores que o valor mais baixo esperado de intercâmbio de custo.

Em experiências com 200 e 500 elementos, eles encontraram que, em contraste com MIN GRAPH PARTITIONING e MIN GRAPH COLOURING, a solução final dada por SA não melhora quando deixamos o algoritmo executando por muito tempo. Pelo contrário, permanece na vizinhança do valor esperado do mais baixo custo de intercâmbio, que é bem abaixo do custo dado por KK. A melhor solução tende ao custo de KK quando é usado um tempo de CPU 10.000 vezes maior que o usado por KK (1.1 segundos!!). Este tempo computacional seria o bastante para tentar 100.000 passos de busca locais (MSLO)! Para instâncias com 200 elementos, SA usando SW_1 e MSLO deram aproximadamente os mesmos resultados, enquanto MSLO usando SW_2 é claramente superior a SA e é competitivo com KK. Para instâncias com 500 elementos, KK claramente vence visto que MSLO usando SW_2 não é competitivo devido ao fato de que sua complexidade é quadrática no tempo. Também um MSLO usando SW_1 é claramente superior a SA.

5.4.3.1 O número médio de ótimo locais

Ferreira e Fontanari calcularam o número de ótimos locais para a vizinhança SW_1. Seus resultados, a partir de 2^n configurações, foi o número

$$\sqrt{\frac{24}{\pi}}\ \frac{2^n}{n^{3/2}} \approx 2.764\ \frac{2^n}{n^{3/2}}. \tag{5.5}$$

de mínimos locais [115]. Note que a razão de mínimos locais para o número total de configurações (informalmente, a densidade de mínimos locais) decresce como $O(n^{-1.5})$. Este fato está em contraste com outras paisagems de aptidão em outros problemas de otimização combinatória, para os quais o decréscimo é exponencial.

5.5 Uma mudança de representação

Motivado pelos resultados desfavoráveis de SA de [181] e os de Jones e Beltramo usando GAs [184], Wheeler Ruml [334] [335] conduziu uma série de experimentos que tinham como objetivo encontrar uma melhor representação para este problema (a representação usada em [181] foi chamada de *'direta'*) que faria o SA competitivo com KK. Os resultados com muitas implementações diferentes de GAs foram desapontadores e mostraram os problemas que SA e GA tiveram até mesmo usando representações diferentes [334].

Entretanto, a tese de Ruml deixa em aberto a possibilidade de que uma representação mais apropriada seria melhor para SA e GAs. Em [335], os autores apresentaram uma maneira interessante de misturar KK com uma representação que cria uma *prepartição* de seus itens.

Uma *solução preparticionada* é uma seqüência $P = (p_1, p_2, \ldots, p_n)$ de inteiros $p_i \in \{1, 2, \ldots, n\}$. Como em outras representação que diferenciam genótipo de fenótipo, o processo de decodificação tem dois passos. Inicialmente, elementos de S são somados entre eles de acordo a seqüência P. Gera-se então um novo conjunto $S' = (x'_1, x'_2, \ldots, x'_n)$ enquanto que o segundo passo é a aplicação de KK ao conjunto S'. Podemos ver o processo de preparticionamento como um passo que é um tipo de preprocessamento antes de aplicar KK para S', onde cada x'_i ficam acumulando enquanto que a seqüência P especifica qual subconjunto de S é obrigado a estar na mesma partição.

Ruml *et al.* em [335] dão uma seqüência-exemplo que reproduziremos aqui. Suponha que temos $S = \{6, 5, 4, 4, 3\}$ e $P = (1, 1, 3, 4, 5)$ então $S' = \{11, 0, 4, 4, 3\}$ dado que $11 = 6 + 5$ (obrigamos 6 e 5 a estarem na mesma partição). A aplicação de KK para A' gera uma partição onde os elementos 6 e 5 estão em uma partição enquanto que todos os outros estão na outra partição, de custo zero. O pseudo-código da Fig. 5.1 explica o procedimento. Este modo de representar o problema tem uma clara vantagem. É evidente que a permutação $P = (1, 2, \ldots, n)$ retorna o mesmo valor que KK, então é no mínimo tão bom quanto KK. De outro lado, qualquer seqüência de números positivos ou negativos das variáveis y_i pode ser representada (por exemplo $p_i \leftarrow y_i + 2\forall i$).

Ruml *et al.* o desenharam como uma *"representação completa"*. Eles viram este modo de codificar configurações como um princípio mais geral de desenho,

$S' \leftarrow (0, 0, \ldots, 0)$
for $i = 1$ **to** n
    $x'_{p_i} \leftarrow x'_{p_i} + x_i$
endfor
for $i = 1$ **to** $n - 1$
    $j \leftarrow$ the index of the largest element of S'
    $k \leftarrow$ the index of the 2nd largest element of S'
    $x'_j \leftarrow x'_j - x'_k$
    $x'_k = 0$
endfor

Figura 5.1: *Pseudocódigo para decodificar na representação de Ruml de* "restrições parameterizadas". *Podemos interpretar a permutação como um* genótipo, *e então* função de crescimento *seria esse pseudocódigo.*

denominando como *"restrições parametrizadas"*.

$\xi \leftarrow$ *Randomize()*
for *iter* $= 1$ **to** *MaxIter*
    $\xi' \leftarrow$ *Perturb*(ξ)
    $\Delta u \leftarrow u(\xi') - u(\xi)$
    if $\Delta u \leq 0$
        then $\xi \leftarrow \xi'$
        else $\xi \leftarrow \xi'$ with probability
$exp(-\Delta u / T(iter))$
endfor
return ξ

Figura 5.2: *Pseudocódigo da implementação de Ruml da meta-heurística de* "Simulated Annealing".

Ruml *et al.* [335] definem a meta-heurística SA como uma *máquina de busca* nesta codificação. Como sempre é o caso, precisamos definir um movimento no espaço de configuração associado. O procedimento *Randomize* gera cada elemento de P usando uma distribuição aleatória uniforme. O procedimento *Perturb* faz $p_i \leftarrow j$ onde $i = rand(1, n), j = rand(1, n)$ e $j \neq p_i$. O processo escolhido para o decréscimo da temperatura foi conforme a fórmula $T(iter) = 0.01(0.8)\lfloor iter/300 \rfloor$.

Em seus experimentos computacionais, Ruml *et al.* [335] definem um *"batch"* como 100 execuções dos algoritmos acima para $MaxIter = 30000$ iterações. Definindo $u^* \equiv u/u_{KK}$ onde u_{KK} é o custo da partição encontrada por KK ($u_{KK} = 1.084 \times 10^{-7}$ para a instância de 100 elementos $I \in I_{100}$) temos que na média SA encontrou melhores soluções com $u^* \approx -1.5$ enquanto que o melhor resultado tem um valor de $u^* < -2$. Este sucesso foi devido

a uma melhor codificação, isto é, usando outra *"máquina de busca"*, o RGT (para *"Random Generate and Test"*) que, sob as mesmas circunstâncias, fornece melhores resultados que três outras codificações também estudadas em [335].

Os autores observam que a seleção dos parâmetros de SA pode até melhorar estes valores, porém eles conjeturam que um característica da paisagem de aptidão, que eles nomeiam como *gravitação*, é a razão principal atrás de um possível fracasso [335].

5.6 O algoritmo evolutivo proposto

Nesta seção explicaremos os algoritmos que implementamos. Toda a notação será consistente com a que usamos no capítulo de introdução aos algoritmos meméticos (MAs) desta tese. Comentamos que nossa implementação de um "GA" é diferente das usadas por Ruml em [334]. Uma razão para esta mudança é que ele já tinha estabelecido bem o pobre desempenho desses GAs " tradicionais", que usam a representação direta. Em nosso caso, nossos "GAs" são a mesma estratégia do MA básico sem qualquer uso de busca local, ou seja, sem período de melhoria após as recombinações ou mutações. Estes "GAs" foram introduzidos por motivos de completude metodológica e para propósitos de comparação.

5.6.1 Representação e espaço de busca

Um dos objetivos desta pesquisa era avaliar se a *representação direta* [335] é definitivamente inapropriada para algoritmos de busca estocásticas, como foram as conclusões dos resultados de [335]. A representação direta (ρ_D) é *completa* visto que pode gerar todas as possíveis partições (o espaço de busca associado tem 2^n configurações). Notamos, no entanto, que qualquer partição dada (qualquer solução $y \in sol_P(x)$) é representada por duas configurações ($s, s' \in S$). Por exemplo, ambas as representações $s = \{1, -1, -1, -1, -1\}$ e $s' = \{-1, 1, 1, 1, 1\}$ têm a mesma partição (a mesma solução factível $y \in sol_P(x)$) de cinco inteiros.

Outra representação que usamos pode ser chamada de *ternária direta*(ρ_{TD}), dado que permite que os alelos tenham um dos três diferentes valores, '0', '1', or '-1'. Obviamente, o espaço de busca associado é também completo, pois contém o espaço de busca de ρ_D.

Para ambas as representações usamos a mesma função-guia, que é de fato a função objetivo $m_P(y, x)$. Entretanto, quando usamos ρ_{TD}, qualquer valor de alelo '0' deve estar *"decidido"*, i.e. dando ou um '-1' ou um '1' para computar $m_P(y, x)$. Escolhemos então usar um procedimento *guloso*. A cada passo toma-se o maior inteiro com um '0' e fornece-se um valor ('1' ou '-1') que minimiza o desequilíbrio atual de todos os inteiros já decididos.

A intuição aqui é que o ρ_{TD} corresponde a outro "estado de partição" que pode ser nomeado a um inteiro. Pesquisadores no campo de GAs podem preferir pensar no espaço de busca de ρ_{TD} como um espaço de *genótipos* enquanto que o procedimento de decodificação gulosa é visto como o desenvolvimento

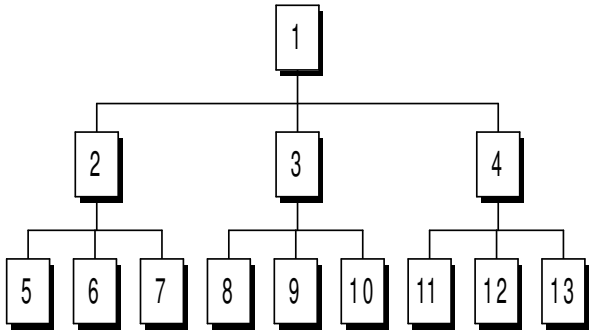

Figura 5.3: *Estrutura da população (uma árvore ternária completa) usada em todos os algoritmos "GAs" e MAs deste estudo para o problema de* MIN NUMBER PARTITIONING.

de um *fenótipo*. Procedimentos gulosos como este aqui são os chamados de *função de crescimento* na literatura de GAs [321]. Notamos que uma função de crescimento que usa o algoritmo de KK pode ser uma escolha natural. Porém, decidimos não incluir os resultados computacionais fazendo uso dela pois isso poderia ser uma comparação injusta.

5.6.2 Estrutura e tamanho da população

A população (usada em ambas as implementações, os "GAs" e MAs) tem um tamanho fixo de 13 agentes. Uma relação de vizinhança espacial entre os agentes está baseada em uma árvore ternária completa de três níveis (veja Fig. 5.3) e é também fixa.

A estrutura da população pode ser interpretada como uma variante dos *modelos de ilhas* de GAs, em que cada subpopulação tem quatro indivíduos enquanto que indivíduos em um nível intermediário (os quais não são nem a raiz nem as folhas da árvore) são *compartilhados* pelas duas subpopulações. A estrutura é a mesma que foi utilizada no capítulo anterior, para o problema do caixeiro viajante assimétrico. Cada subpopulação é composta de um nó *"líder"* onde três *"subordinados"* estão em um nível abaixo na hierarquia. O agente 1, a raiz da árvore, é o líder da subpopulação do topo e tem como subordinados os agentes 2, 3, e 4. O agente 2 tem como subordinados os agentes 5, 6, e 7, e assim por diante. Note que os agentes 2, 3, e 4 são ao mesmo tempo líderes e subordinados.

Cada agente da população contém dois individuos, quer dizer, duas configurações no espaço de busca associado. Um é chamado de *"Pocket"* e o outro é o *"Current"*. Sempre que um indivíduo *Current* tem menor custo que o *Pocket*, eles são trocados, funcionando assim como uma "memória" de soluções boas passadas. Analogamente, outro procedimento chamado *PocketPropagation*

troca duas soluções *Pocket* se o líder tem uma solução *Pocket* que é pior que um de seus seguidores. Estes dois mecanismos são suficientes para garantir o fluxo de soluções melhores para o indivíduo no topo da hierarquia.

Em nossa implementação de MA, os agentes otimizam suas soluções de tipo *Current* com períodos de busca local que alternam-se com recombinação. Como mencionamos antes, chamaremos um "GA" uma implementação que só difere do MA pela ausência do uso de busca local, enquanto todas as outras regras do algoritmo permanecem as mesmas. Para esta particular implementação de um MA, escolhemos chamar como uma *geração* o processo no qual os agentes evoluíram suas soluções *Current* para um mínimo local e depois eles se ocupam da recombinação.

5.6.3 O comando initializePopulation

Para ρ_D o comando **initializePopulation** cria 13 agentes, cada um com um cromossomo *Pocket* e um *Current*. Cada cromossomo representa uma partição factível $y \in sol_P(x)$. O espaço de busca $S(\rho_D)$ é o conjunto de 2^n cromossomos e valores a serem escolhidos, '1' ou '-1', com igual probabilidade. Para ρ_{TD} o procedimento de inicialização é o mesmo, mas o espaço de busca de genótipos (denotado por $S(\rho_{TD})$) tem 3^n configurações e o valor de allelo '0' é selecionado aleatoriamente com uma probabilidade de 1/2 enquanto os dois outros valores ('1' e '-1') têm igual probabilidade de 1/4.

5.6.4 Os procedimentos de recombinação

A cada geração, é substituído o cromossomo *Current* de cada agente por um novo gerado pelo procedimento Recombine() que age no cromossomo *Pocket* do agente e seu líder. Por exemplo, o *Current* do indivíduo 2, na Fig. 5.3, será substituído pelo resultado de Recombine(), que tem como entrada o *Pocket* do agente 1 e o *Pocket* do agente 2; o *Current* do agente 9, será substituído pelo resultado de Recombine() usando o *Pocket* do agente 3 e o *Pocket* do agente 9; etc.

Então, em uma iteração do laço de gerações, todos os indivíduos exceto o agente 1, terão seu cromossomo *Current* substituído. Em relação ao pseudo-código do capítulo de introdução aos algoritmos meméticos, o número de recombinações *#recombinations* executado a cada passo de geração é 12. Além disso, o comando **selectToMerge** pode ser considerado como *totalmente restrito*, desde que cada agente sempre se ocupa da recombinação com os mesmos agentes.

A topologia de vizinhança da árvore ternária para os indivíduos sempre obriga os subordinados a recombinar com seus líderes. A estrutura de população e esta regra sugerem que o modelo de ilhas de GAs (levado como uma metáfora para evolução com colonização e difusão) não é o mais representativo para este MA. Ao invés disso, o método poderia se assemelhar a uma organização hierárquica na qual "idéias" somente são discutidas com o líder imediato do grupo.

5.6.4.1 Comportamentos

Antes que um líder recombine sua solução com qualquer de seus três subordinados, um *comportamento* diferente é designado a cada um. Um comportamento pode ser entendido como um parâmetro de controle extra no controle fixado de um determinado operador de recombinação [323]. Cada subordinado terá um comportamento diferente, que será decidido aleatoriamente com igual probabilidade. Os três tipos de comportamento usados serão descritos abaixo. Para simplificar a apresentação, vamos supor que a representação direta está sendo usada e o pai P_1 é o líder e o pai P_2 é um dos subordinados.

comportamento	primeira coisa a copiar para o *descendente*
rebelde	alelos de P_2 que são diferentes de P_1
conciliador	alelos em comum entre P_1 e P_2
bajulador	alelos de P_1 que são diferentes de P_2

Tabela 5.2: *Comportamentos diferentes utilizados na recombinação entre líder e subordinados.*

Por exemplo, considere P_1 = {-1,-1,1,-1,1} e P_2 = {1,-1,1,1,-1}, a recombinação que usa os mesmos pais como entrada e cada tipo de comportamento é como segue. Aqui o 'x' representa um valor que será decidido por algoritmos que nós explicaremos posteriormente.

rebelde	{1	x	x	1	-1}
conciliador	{x	-1	1	x	x}
bajulador	{-1	x	x	-1	1}

Os três nomes foram escolhidos com referência à semântica para preservar sua informação (valores de alelos). O comportamento *conciliador* é um exemplo de um procedimento de recombinação que *respeita* a representação [322] [321] pois *"cada filho criado contém todos os alelos comuns de seus dois pais"* (i.e. $A \cap B$). Ele compartilha a propriedade de ser uma recombinação *respeitosa*, como também é o caso da *recombinação uniforme*. Neste caso, visto que todos os alelos que não estão em $A \cap B$ tem o valor '-1' ou '1', a recombinação com o comportamento conciliador é dito estar também *transmitindo alelos* [321] desde que *"cada alelo no filho está presente no mínimo em um de seus pais"*.

Os comportamentos *rebelde* e *bajulador* não respeitam (no mesmo sentido apontado por Radcliffe [321]) representações binárias, já que eles podem não incluir alelos em $A \cap B$. Também, dado que é permitido criar alelos que não estão presentes em ambos os pais, os comportamentos *rebelde* e *bajulador* fornecem tipos de recombinações em que não se transmitem alelos.

Para decidir as posições no *descendente* onde nenhum valor de alelo foi escolhido (as marcas 'x') temos que investigar diversas variantes. Para a representação ρ_D, três tipos de algoritmos foram utilizados: *transmissão generalizada* (GT), *GT com reparação gulosa* ($GTgr$), e o *GT semente aleatória - reparação gulosa* ($GTrsgr$).

O *GT* decide aleatoriamente entre '-1' ou '1' com igual probabilidade. O *GTgr* usa um algoritmo guloso, i.e. uma regra determinística que seqüencialmente decide entre '-1' ou '1' para minimizar o desequilíbrio atual. A seqüência é baseada em uma ordem não decrescente da seqüência dos inteiros ainda não decididos. O *GTrsgr* começa selecionando aleatoriamente um inteiro tal que seu valor de alelo tenha que ser decidido primeiro e continua novamente de tal modo que minimize o desequilíbrio, continuando de modo determinístico como o *GTgr*. De acordo com as definições, todos eles são operadores de reprodução, enquanto que o *GT* é o único que é cego, pois não usa a informação da instância do problema.

Por exemplo, considere novamente a instância $A = \{15, 12, 10, 9, 4\}$ e os dois pais P_1 e P_2 com o comportamento conciliador. O *descendente* primeiro recebe os valores de alelos comuns, i.e. $O = \{x, -1, 1, x, x\}$. Com *GT*, cada $'x'$ será decidido aleatoriamente com a mesma probabilidade (neste caso temos o tradicional *operador uniforme*. Com *GTgr*, escolhemos $O[1] = 1, O[4] = -1$ e $O[5] = -1$, nesta ordem. Note que, neste caso, decidimos por '1' ou '-1' de acordo com a diferença absoluta atual da partição parcial.

Para ρ_{TD}, investigamos dois operadores cegos de reprodução. Um é análogo a *GT* mas, desde que temos que tratar com ρ_{TD} podemos nomeá-lo *GT(0,5)*. Ele transmite os valores de alelo relativos ao comportamento exatamente do mesmo modo que o *GT* faz, porém os valores restantes são decididos com probabilidade uniforme onde é nomeada uma probabilidade de 0,5 para estimar '0' e são 0,25 a '-1' e '1'. O outro operador de crossover nomeia um valor de '0' a qualquer alelo não transmitido, assim chamamos de *GT(1,0)* dado que é como se nós estivéssemos selecionando un '0' com uma probabilidade 1.

Por exemplo, considere P_1 = {1,-1,0,0,1} e P_2 = {1,0,0,0,-1}. O filho com comportamento bajulador será $O = \{x, -1, x, x, 1\}$. Com *GT(0,5)* o $O[1], O[3]$ e $O[4]$ será aleatoriamente decididos, mas com probabilidade 0,5 para '0' e 0,25 para '1' e '-1'. Usando *GT(1,0)* o descendente será $O = \{0, -1, 0, 0, 1\}$, i.e., '0' para todo $'x'$.

5.6.5 Os procedimentos de mutação

Para ambas as representações, implementamos dois tipos de mutação, *Simple* e *Minimal*.

Quando usamos ρ_D, a mutação *Simple* recebe como entrada um indivíduo e para todos os valores de alelo decide mudá-lo (de '-1' para '1' ou de '1' para '-1') com uma probabilidade fixa de 1/10. Para ρ_{TD}, sempre que é decidido pela alteração de um valor de alelo (novamente com uma probabilidade fixa de 1/10), se é '-1' ou '1' é alterado para '0', mas se seu valor é '0' é alterado para '-1' ou '1' com probabilidade igual.

A mutação *minimal* foi inspirada na *Mutação Binomial Mínima* como discutida em [323]. Porém, uma diferença notável é que em nosso caso levamos em conta as características da função objetivo (de onde procede o rótulo 'minimal'). Para ρ_D, decidimos se um alelo terá seu sinal não alterado com a mesma probabilidade usada pela mutação *simple* (i.e. 0,1). Vamos supor que o alelo i tem valor $v_i = 1$ e foi selecionado para ser mutado. Vamos supor que repre-

sentamos os cromossomos como um vetor linear tradicional de valores de alelo, tal que o posicionamento corresponde a uma ordem não-crescente dos inteiros da instância A. Isto significa que o valor de alelo de v_1 corresponde à partição nomeada ao maior inteiro em A (inteiro a_1). Identificamos o índice do primeiro alelo na *"direção esquerda"* (referindo-se ao vetor linear) j_l e o primeiro alelo na *"direção direita"* j_r que tem sinais diferentes ($v_{jr} = v_{jl} = -1$). Então v_{jl} é o valor de alelo do menor inteiro que é maior ou igual ao inteiro a_i e é nomeado para uma diferente partição. Analogamente, v_{jr} é o alelo do maior inteiro que é menor ou igual ao inteiro a_i mas em diferente partição. Então selecionamos um dos j_r e j_l que minimiza $|a_i - a_j|$. Trocamos seus valores de alelos com o valor de v_i. Para ρ_{TD}, o procedimento é o mesmo, mas se o alelo i tem valor $v_i = 1$, identificamos o alelo j tal que $v_j = -1$ ou $v_j = 0$ e que minimiza $|a_i - a_j|$, então trocamos seus valores de alelos.

Podemos dizer, analogamente aos procedimentos de recombinação, que a mutação *Simple* é um operador cego enquanto que a mutação *Minimal* não é.

5.6.6 Procedimentos de Busca Local: definindo o comando Local-Search-Engine()

Implementamos dois diferentes tipos de algoritmos de busca local. Um foi chamado de *GreedyImprovement* e o outro foi chamado de *TabuImprovement*. Em ambos, a entrada é um cromossomo, e a saída é um cromossomo com o mesmo ou melhor valor de função objetivo $m_P(y, x)$.

Usando o *GreedyImprovement* com a representação ρ_D, começamos selecionando um alelo i aleatoriamente. Então, como nós fizemos com a mutação mínima, identificamos a posição do primeiro alelo na *direção "direita"* (j_r) e a posição do primeiro alelo na *direção "esquerda"* (j_l), tal que $v_{jr} = v_{jl} \neq v_i$ (pode acontecer de encontrarmos apenas um que satisfaça a condição). Decidimos trocar os valores de v_i com v_{jl} ou v_{jr} se o valor da função objetivo ($m_P(y, x)$) é reduzido. Se em ambos isso acontece, escolhemos entre v_{jl} e v_{jr} que cause a maior redução. Repetimos esse processo até que $(FailuresTries - SuccessTries) > MaxNumberOfTries$. Onde $SuccessTries$ é o número de tentativas que resulta um melhor valor para $m_P(y, x)$ e $FailuresTries$ o oposto. Usamos diversos valores para o parâmetro $MaxNumberOfTries$ que será comentado mais à frente.

Com ρ_{TD} o procedimento é o mesmo, mas selecionamos para trocar apenas entre os alelos de valor '1' e '-1'. Embora, a cada tentativa necessitamos calcular o valor $m_P(y, x)$ com o procedimento guloso, i.e, qualquer alelo com valor '0' "decide-se" ter '-1' or '1' para computar $m_P(y, x)$.

O *TabuImprovement* usa a mesma idéia do *GreedyImprovement*. A diferença é que neste caso usamos uma Busca Tabu básica [136] dentro do *GreedyImprovement*. A cada troca entre i e j, uma matriz tabu ($TABU$) de inteiros armazena, na posição $TABU[i][j]$, o número de tentativas de trocas entre i e j que será 'tabu'. Em *TabuImprovement* podemos permitir trocas que não reduzem o valor de $m_P(y, x)$, mas a saída deste procedimento é a configuração com melhor valor $m_P(y, x)$ encontrado. Repetimos esse processo até que $(FailuresTries - SuccessTries) > MaxNumberOfTries$. A variável

SuccessTries, neste caso, é o número de tentativas que poderiam ser feitas, i.e., o número de vezes que uma tentativa foi não tabu. Além disso, usamos um critério de aspiração simples [136]. Se uma troca causa uma melhora na função objetivo num valor que nunca foi antes atingido, fazemos a troca, mesmo que o movimento esteja declarado como 'tabu'.

5.6.7 RestartPop()

Durante a evolução da população, os agentes recombinam apenas dentro de sua sub-população. Mais especificamente, um agente só recombina com seu indivíduo líder. Porém, eles podem envolver os indivíduos de subpopulações diferentes quando uma *"crise de diversidade"* ocorre. Isto acontece quando todos os três subordinados da mesma subpopulação têm configurações similares.

O critério que usamos para definir uma crise de diversidade é o seguinte. Selecionamos ao acaso 20% de alelos do cromossomo *Current* dos três subordinados. Se os três indivíduos têm os mesmos valores de alelos, então uma *DiversityCrisis* aconteceu.

Se *DiversityCrisis* é detectada, o líder não recombinará com seus três subordinados, mas os três subordinados irão recombinar com subordinados que pertencem a subpopulações diferentes. Por exemplo, em geral o agente 2 recombina com seus três subordinados, i.e., os agentes 5, 6 e 7, mas foi detectado *DiversityCrisis* (i.e. 5, 6, e 7, tem configurações similares). Então a recombinação será feita entre $\{5, 6$ e $7\}$ e $\{8, 9,$ e $10\}$ ou entre $\{5, 6,$ e $7\}$ e $\{11, 12$ e $13\}$. Três recombinações modificarão os cromossomos *Current* dos indivíduos 5, 6, e 7; i.e., se foi escolhido $\{5, 6,$ e $7\}$ e $\{8, 9,$ e $10\}$ (a outra subpopulação foi escolhida aleatoriamente), os pares para recombinação podem ser: 5 com 9, 6 com 10 e 7 com 8. Os pares são escolhidos aleatoriamente.

Com esta *"external mating"* conseguimos uma diversidade extra. Seu benefício potencial é incorporado desde a base da árvore devido a que posteriores recombinações envolvendo os nós líderes ocorrerão nas gerações seguintes.

5.7 Experimentos Computacionais

Montamos um conjunto de experimentos caracterizados por representarem *instâncias difíceis* deste problema para identificar as razões do fracasso da implementação direta de GAs. O objetivo era adquirir alguns *"insights"* acerca dos problemas identificados em [184] e [334]. Quisemos nos convencer de que não há nenhuma esperança grande de obter bons resultados quando usamos a representação direta, até mesmo usando formas mais sofisticadas de processos de seleção e operadores de recombinação. A partir desta perspectiva inicial, os resultados finais foram uma surpresa.

5.7.1 Instâncias

Nossa primeira decisão foi a respeito de quais instâncias deveriam ser usadas. Uma possibilidade foi usar as instâncias de ponto flututante UI_n dos autores

de [181] (ou trabalhar com um pacote de precisão numérica múltipla como os autores de [335]), e confiar nas fórmulas de custo assintótico para comparar os resultados.

Ao invés disso, devido à disponibilidade do código exato de Korf (CKK), fomos capazes de resolver instâncias com 10 dígitos em um razoável tempo em um PC. Estamos nos referindo às instâncias mais difíceis do conjunto.

Criamos um conjunto de 100 instâncias que usamos em todos os experimentos que reportaremos. Este conjunto é composto de 10 subconjuntos com 10 instâncias cada. Cada subconjunto tem o mesmo valor de n, i.e. 10 instâncias com $n = 15$, 10 instâncias com $n = 25$, etc., até $n = 105$. Cada instância é composta de n inteiros positivos com $d = 10$ dígitos significativos. Isto nos permite usar o código exato de Korf para encontrar a solução ótima para todas as instâncias, incluindo instâncias perto da *"transição de fase"*.

Para evitar qualquer possível correlação entre os inteiros (devido à estrutura dos geradores de número aleatórios que usamos) decidimos criar cada dígito individualmente ao acaso, com probabilidade de distribuição uniforme entre 0 e 9.

5.7.2 Resultados Finais

Primeiro, executamos o código exato de Korf (CKK) usando nosso conjunto de instâncias (veja Fig. 5.4). Na figura, cada ponto representa o valor médio sobre o conjunto de dez instâncias que têm o valor particular n e usamos escala logarítmica para o eixo vertical. Os pontos foram ligados com uma linha para ajudar a ilustrar as tendências para valores crescentes de n.

Verificamos a existência da chamada *transição de fase* [208] em torno de $n = 35$, como era esperado a partir das análises teóricas [134]. Para alguns problemas de otimização combinatória, quando usamos um algoritmo ótimo (como CKK), ao aumentar o tamanho da instância o problema se torna mais difícil até que um certo tamanho crítico seja alcançado. Depois disto, o problema é mais fácil. Este fenômeno é chamado de *transição de fase*, e neste caso temos uma transição *fácil-difícil-fácil*.

Neste caso, a *transição de fase* ocorre quando a probabilidade de encontrar um partição perfeita é 1/2. Karmarkar *et al.* [191] mostraram que para um conjunto de tentativas independentes encontrar uma partição em n números entre '0' e '1', a média da diferença mínima é $\sqrt{n}/2^n$. Visto que no nosso caso as instâncias têm 10 dígitos decimais, o valor médio do custo é $c10^{10}\sqrt{n}/2^n$, para alguma constante c. Korf em [209] não conhecia ainda o resultado de Mertens, e resolveu para c na equação $c\sqrt{35}10^{10}/2^{35} = 1$ obtendo $c = 1,72$, e $c = 0,873$ para $n = 36$. Isto significa que as instâncias com $n = 35$ serão as instâncias mais difíceis para CKK em nosso conjunto de testes.

Com $n > 35$, na maioria das vezes as instâncias têm uma partição perfeita (um ótimo de custo 0 ou 1). Para instâncias com $n < 35$ é altamente improvável ter uma partição perfeita. Como resultado, a partição ótima tende a ter custos maiores com a redução de n. Um lado interessante destes efeitos é refletido contando o número de nós usado no procedimento exato (veja Fig. 5.4). Para $n \geq 45$ o problema começa a ser mais fácil para o método exato, aparentemente

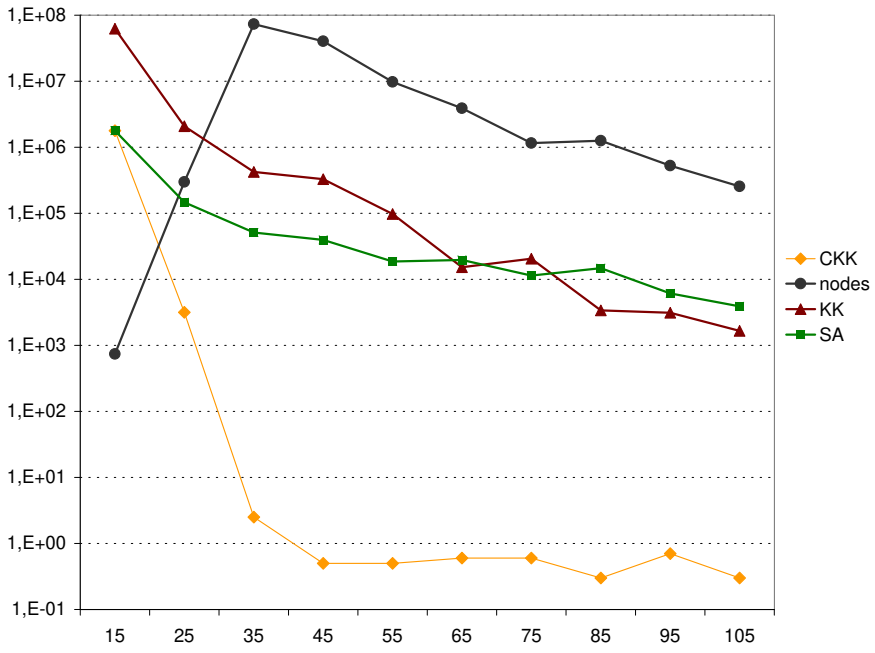

Figura 5.4: *O desempenho de* "Simulated Annealing" *(SA), da heurística de Karmarkar-Karp (KK), e do método exato proposto por R. Korf (CKK). Com "nodes" referimos ao número de nós explorados pelo algoritmo de CKK.*

seguindo uma taxa linear em uma escala logarítmica.

O algoritmo de KK parece estar melhorando com o aumento de n. Isto é possivelmente devido ao fato de que o número de partições perfeitas aumenta com o aumento de n [250], e possivelmente o KK obtém algum benefício também deste fato. Os valores obtidos por KK parecem seguir o que prediz a teoria: $O(1/n^{\alpha}logn)$ [378], com $\alpha \approx 0,702$ obtido a partir de um ajuste de nossos dados experimentais. Nosso algoritmo de SA foi codificado para propósitos de comparação e segue exatamente a implementação proposta em [181]. Parece que, para este conjunto de instâncias, o SA é uma alternativa melhor que KK para $n \leq 55$.

5.7.3 Resultados com o algoritmo evolutivo proposto

Os próximos experimentos mostram a preformance dos "GAs" (neste caso, sendo o nosso MAs sem o Local-Search-Engine) com as representações ρ_D e ρ_{TD}, com 4000 gerações. Reportamos os resultados dos algoritmos que usaram a recombinação *GTrsgr*, para $S(\rho_D)$, e a *GT(1,0)* para $S(\rho_{TD})$, posto que eles obtiveram os melhores resultados. Estes experimentos são mostrados na Figura 5.5 onde usamos a notação da Tabela 5.3.

Os resultados com a representação direta ρ_D usando mutação *simple* (a

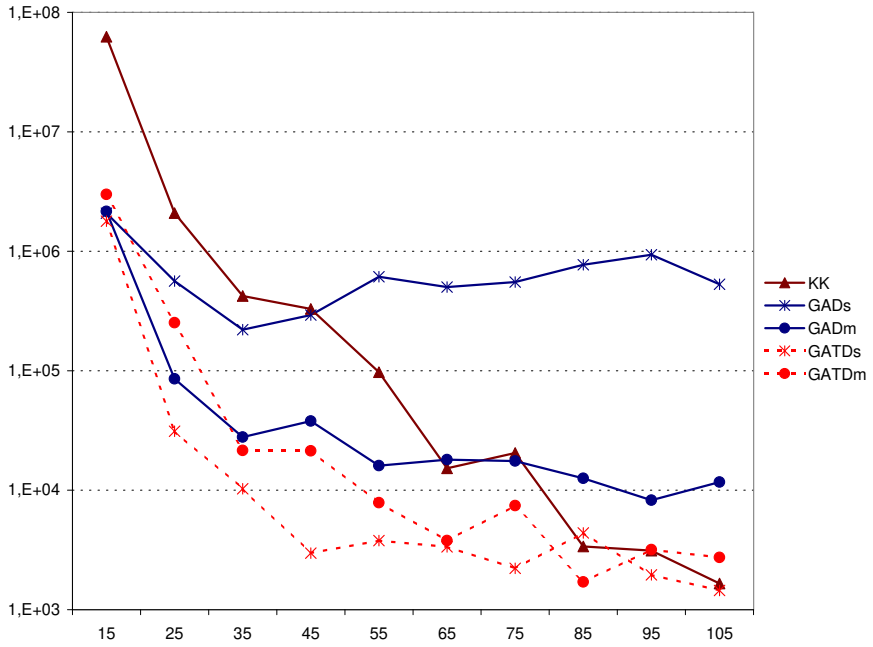

Figura 5.5: *O desempenho de quatro tipos de diferentes algoritmos genéticos em comparação com a heurística de Karmarkar-Karp.*

notação	representação	mutação
GA_{Ds}	ρ_D	simple
GA_{Dm}	ρ_D	minimal
GA_{TDs}	ρ_{TD}	simple
GA_{TDm}	ρ_{TD}	minimal

Tabela 5.3: *A notação usada nos resultados experimentais mostrados na Figura 5.5.*

curva chamada de GA_{Ds}) foi significativamente ruim, como era esperado pelos resultados prévios de [334]. Não é nem mesmo capaz de competir com a heuristística de KK para $n \geq 45$. Pode ser considerado como um *"benchmark"* para outros "GAs". Porém, em comparação com a mutação *Simple*, note que o uso de mutação *minima* (até mesmo quando estamos usando a mesma representação) melhorou significativamente os resultados, começa a ser pior que KK somente para $n > 75$. Ao contrário, o uso de mutação *Simple* melhorou significativamente os resultados quando é usada a representação $S(\rho_{TD})$, sendo melhor que KK para $n < 75$. Os resultados parecem ser melhores até mesmo para o mesmo algoritmo usando a mutação *minimal*.

Depois que estes resultados foram encontrados, incorporamos o procedimento Local-Search-Engine(). Testamos o *GreedyImprovement* e o *TabuIm-*

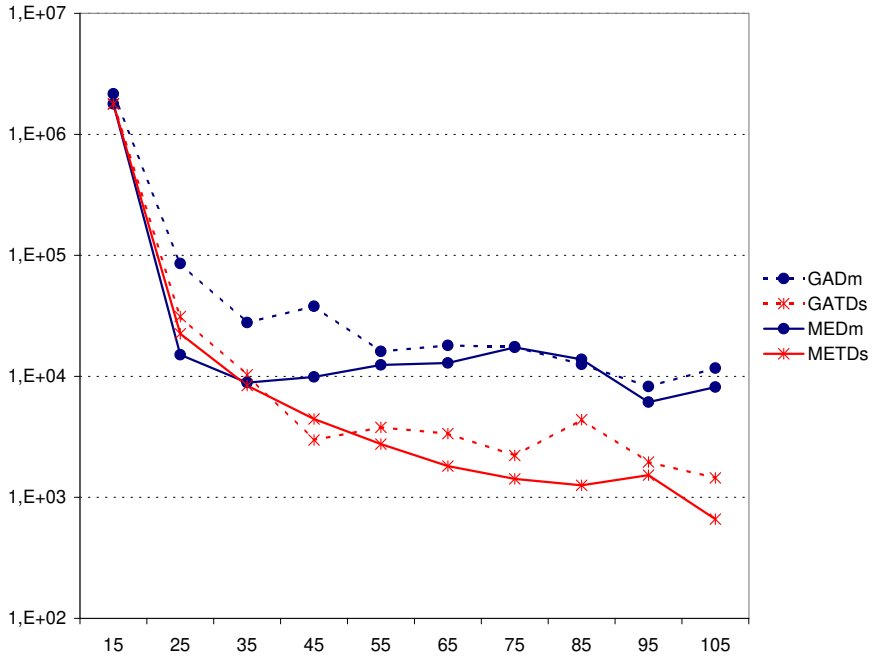

Figura 5.6: *O desempenho de GAs e MAs e como ele se comporta em função da representação escolhida.*

provement com $MaxNumberOfTries = 10$. Também neste caso, limitamos o número de gerações em 4000. Os resultados da introdução do *GreedyImprovement* estão apresentados na Fig. 5.6, onde mostramos com linha tracejada os valores obtidos por GA_{Dm} e GA_{TDs} (também foram mostrados na Fig. 5.5 e estão incluídos na Fig. 5.6 como uma referência). Os resultados dos dois algoritmos, chamados de ME_{Dm} e ME_{TDs} não mostram nenhuma melhora impressionante sobre GA_{Dm} e GA_{TDs}, que não usaram busca local.

Das Figuras 5.5 e 5.6, notamos que a representação ρ_{TD} parece ter melhores resultados quando comparado com a ρ_D nas experiências discutidas acima. Neste momento, estávamos um pouco frustrados com os resultados destes dois algoritmos meméticos, ME_{Dm} e ME_{TDs}, onde não foram mostradas melhoras relevantes. Os resultados usando *TabuImprovement* usando $MaxNumberOfTries = 10$ foram similares e não foram incluídos na figura.

Numa outra experiência, decidimos usar Local-Search-Engine() com um valor maior de $MaxNumberOfTries$. Porém, decidimos não usar Local-Search-Engine() a cada geração, mas só depois de um número fixo de iterações ($GenerationForLocal$). Esse método é análogo ao apresentado em [272] para o problema de aprendizado do perceptron binário. A idéia chave é usar menos freqüentemente a busca local baseada em Busca Tabu. Em compensação, fazê-la mais exaustiva quando aplicada.

Nossos experimentos incluíram três tipos de valores para *GenerationFor-*

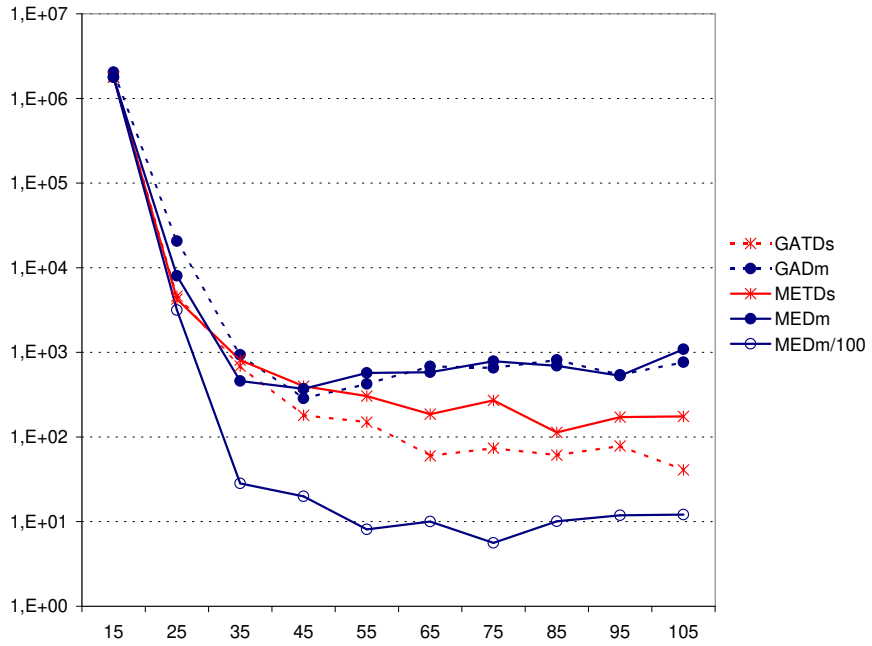

Figura 5.7: *O desempenho de MAs e GAs: os melhores resultados são obtidos por um algoritmo memético que usa passos de Busca Tabu para diversificar (ME-Dm/100). Um limite de tempo de 120 segundos foi utilizado para poder comparar os algoritmos.*

Local (10, 50, e 100). Isto significa que depois de 10, 50, ou 100 gerações o procedimento Local-Search-Engine() é aplicado ao *Pocket* de todos os indivíduos. Os valores de $MaxNumberOfTries$ testados foram de 1000, 10000, 50000, e 100000. Buscando uma comparação mais justa entre os diferentes métodos, decidimos que o melhor era limitar o tempo de execução em 120 segundos. Quando usamos o *GreedyImprovement* os resultados não foram sensíveis para qualquer representação (i.e. ρ_D ou ρ_{TD}) sendo usada. Entretanto, quando usamos o *TabuImprovement* com ρ_D os resultados melhoraram significativamente sobre o uso de ρ_{TD}. A razão é que o ρ_{TD} requer a decodificação gulosa para decidir os valores dos 'zeros' e isto consome um tempo precioso quando usamos um limite de 120 segundos.

Na Figura 5.7, mostramos os resultados dos mesmos algoritmos, GA_{Dm} e GA_{TDs}, e das duas variantes que usaram busca local ME_{Dm} e ME_{TDs}. Estas curvas são mostradas na Figura 5.6 também. Como uma comparação, agora nós adicionamos a curva $ME_{Dm/100}$, um MA que que usa uma Busca Tabu (com $MaxNumberOfTries = 50000$) como um passo de diversificação a cada 100 gerações. Estes resultados são agora para um limite de 120 segundos de execução. As curvas para GA_{Dm} e ME_{Dm} são muito similares. Entrentanto, para ρ_{TD} o GA_{TDs} parece superar o ME_{TDs}.

Relativo a estes resultados, poderíamos concluir que a busca local, pelo

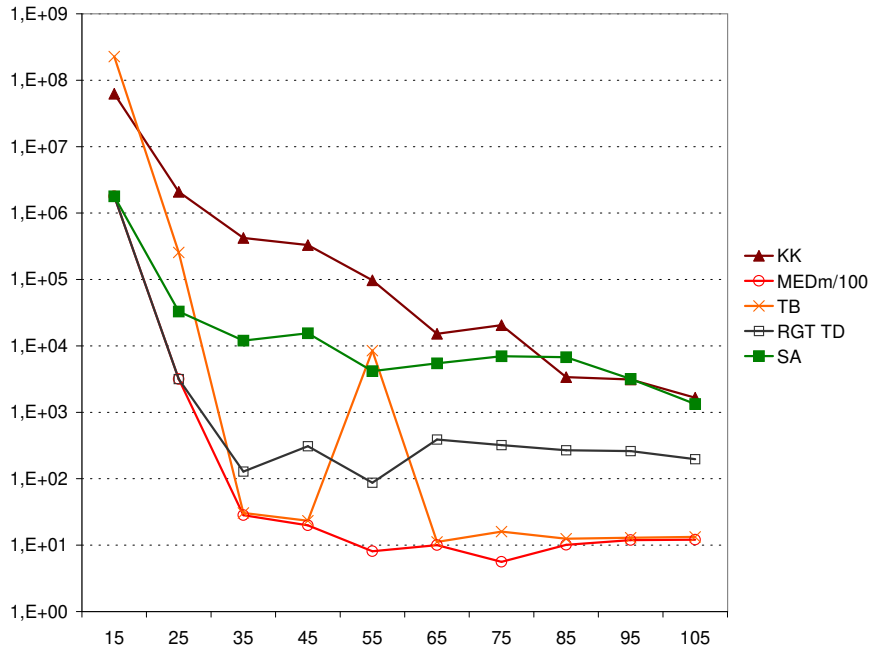

Figura 5.8: *Resultados da heurística de Karmarkar-Karp (KK), de um MA com Busca Tabu para diversificação (MEDm/100), da meta-heurística de* "Simulated Annealing" *de acordo a implementação de Johnson, de uma Busca Tabu com só um agente (TB), e o método "Random Generate-and-Test" de geração aleatória de soluções e avaliação da função objetivo (RGT-TD).*

menos do modo que foi implementada nas representações diretas, é de pequena utilidade. Mas a melhor curva na Figura 5.7 é um algoritmo memético que usa a representação ρ_D, com mutação *minima* e um período de *TabuImprovement* tendo $MaxNumberOfTrials = 50.000$ que é executado a cada 100 gerações. Os resultados são significativamente melhores. A maioria das outras variantes de MAs para diferentes pares de $MaxNumberOfTrials$ e $GenerationForLocal$ produzem resultados similares.

Os resultados do melhor algoritmo memético apresentado na Figura 5.7 nos conduz a realizar uma experiência para testar o desempenho de um "único-agente", o procedimento *TabuImprovement*. Neste caso, nenhum método baseado em população foi utilizado. Os resultados foram muito bons (veja a curva TB na Figura 5.8) mas não tão robusto e sem dominar os resultados do algoritmo memético usando Tabu.

Na Figura 5.8, podemos ver um resumo de alguns dos resultados. A figura exibe os resultados da heurística de Karmarkar-Karp (KK). As outras curvas correspondem a execuções de 120 segundos (todas usando ρ_D) de: a implementação de *"Simulated Annealing"* (SA), a Busca Tabu (TB), e o algoritmos memético que usa a mesma estratégia Tabu da Figura 5.7 ($ME_{Dm/100}$ com

(100,50000)). Além disso, apresentamos os resultados da estratégia random-generate-and-test para ρ_{TD} usando a decodificação gulosa (curva RGT_{TD}).

Ficamos surpresos com o desempenho de $ME_{Dm/100}$ que usa esporadicamente períodos de Busca Tabu. O desempenho do procedimento simples de Busca Tabu, embora menos robusto, também é notável, devido à simplicidade da sua implementação. Porém, notamos que não pôde encontrar as soluções ótimas para as instâncias com $n \leq 25$ como é o caso do $ME_{Dm/100}$. Apresentamos abaixo os valores (cada um é a média de 10 instâncias, arredondada para o inteiro mais próximo) que os diferentes métodos obtiveram. O CKK nos fornece os valores ótimos e é o único método que não foi limitado em 120 segundos.

n	CKK	KK	SA
15	1785469	62529719	1785469
25	3161	2087226	32958
35	3	422345	11996
45	1	329255	15492
55	1	97390	4177
65	1	15250	5478
75	1	20488	6995
85	0	3389	6755
95	1	3122	3180
105	0	1656	1332

Tabela 5.4: *O desempenho da meta-heurística de* "Simulated Annealing" *em contraste com a heurística construtiva de Karmarkar-Karp.*

n	CKK	SA	RGT_{TD}	GA_{TDs}	TB	$ME_{Dm/100}$
15	1785469	1785469	1785469	1785469	227423824	1785469
25	3161	32958	3161	4610	255580	3161
35	3	11996	129	688	31	28
45	1	15492	309	181	23	20
55	1	4177	87	150	8498	8
65	1	5478	388	60	11	10
75	1	6995	321	74	16	6
85	0	6755	269	61	13	10
95	1	3180	260	79	13	12
105	0	1332	198	41	13	12

Tabela 5.5: *O desempenho de todos os métodos (para o mesmo conjunto de instâncias). O algoritmo memético que usa Busca Tabu para diversificação domina todos eles, mas não alcança a solução ótima em todos os casos.*

5.8 Conclusões

A partir das experiências computacionais, podemos concluir que este problema ainda é uma fonte de interessantes perguntas abertas. As instâncias aleatórias com 10 dígitos decimais apresentam muitos desafios para as implementação de GAs e outras meta-heurísticas baseadas na codificação binária direta. Porém, um algoritmo memético que usa várias idéias inéditas dominou todas as outras meta-heurísticas neste estudo. Ao contrário de nossas expectativas prévias, o melhor MA usa a codificação direta (binária) e usa uma mutação *minimal*, assim como uma Busca Tabu muito simples para diversificação (como proposta em [272]). Não obstante, não excluímos a possibilidade de obter resultados até melhores usando outras representações.

Também notamos que em nenhum caso "semeamos" a população ou o SA com uma solução fornecida pela heurística de KK. Observamos que é parte da filosofia dos MAs introduzir todo conhecimento disponível e teria sido uma opção válida. O KK teria sido uma boa heurística para fazer um algoritmo randomizado para a população inicial. Também é um método construtivo bom para usar ao invés do procedimento guloso de recombinação. Testamos estas idéias experimentalmente, mas escolhemos não incluí-las neste estudo para poder fazer uma comparação clara.

A distância da otimalidade local até a otimalidade global é incrível para este problema e representou um desafio árduo para as metaheurísticas. A natureza deste desafio parece estar na forte dependência de várias metaheurísticas com o valor da função-objetivo $m_P(y,x)$ quando é usada como função-guia. Este fato, junto com o número exponencial de mínimos locais encontrado na representação direta, é uma combinação difícil de superar. Nosso algoritmo memético, combinado com o uso de passos de Busca Tabu, parece ser uma alternativa adequada para este problema (até mesmo nesta paisagem da função de aptidão desfavorável), particularmente onde as instâncias difíceis para CKK parecem ser, i.e. para $n \approx 35$ neste caso. Para maiores valores de n, desde que esperamos ter um número exponencial de partições perfeitas [250], o desempenho de CKK dificilmente pode ser superado.

Os resultados da Busca Tabu simples foram surpreendentes para nós. Ficamos intrigados porque este método simples não foi aplicado previamente em outros estudos de comparação. Notamos que o problema de encontrar a configuração de custo mínimo para este problema (quando a função objetivo é $m_P(y,x)$ na introdução) é equivalente ao problema de encontrar o "ground-state" do Hamiltoniano

$$H = (m_P(y,x))^2 = \sum_{i,j=1}^{N} v_i a_i a_j v_j \tag{5.6}$$

que é um vidro de spin de alcance infinito com acoplamento antiferromagnético de Mattis $J_{ij} = -a_i a_j$ [250]. Os bons resultados de Busca Tabu reportados em [216] para o vidro de spin 2-dimensional também estão correlacionados com o que obtivemos. Devido ao desempenho mais robusto de nosso algoritmo memético que usa TS como uma estratégia de diversificação, pensamos que o

método apresentado aqui também pode ter aplicações interessantes na Física de sistemas desordenados.

Capítulo 6

Complexidade Parametrizada

6.1 Motivação

Neste capítulo, introduziremos alguns resultados da teoria de *Complexidade Parametrizada* defendida por R. Downey e M. Fellows [107]. Acreditamos que há esperança de desenvolver uma nova e interessante linha de pesquisa ao unir apropriadamente alguns resultados da teoria de *"fixed-parameter tractability"*, que traduzimos como "tratabilidade a parâmetro fixo", e o desenvolvimento de novos algoritmos de recombinação. Esta é uma linha de pesquisa que segue ambas as direções. Em uma direção, a partir de uma perspectiva pragmática, algoritmos de parâmetro fixo eficientes (i.e. de tempo polinomial) podem ser usados como ferramentas *"fora da caixa"* para criar procedimentos de recombinação de tempo polinomial. Pode ser como os procedimentos de *recombinação dinástica ótima* que garantem, no mínimo, fornecer uma solução factível tão boa quanto a melhor solução pai recebida como entrada [82]. Alternativamente, podemos estar satisfeitos com um algoritmo de recombinação de tempo polinomial que fornece uma *nova* solução factível que é melhor que o pior pai recebido como entrada [273]. Este assunto será tema do Capítulo 7, onde os discutiremos em maior profundidade.

Na direção oposta, sabendo-se que algoritmos meméticos são tipicamente projetados para tratar de instâncias de grande porte e tendem a funcionar bem com o aumento de tamanho, usar ambas as técnicas juntas pode produzir MAs *completos* que permitem estender os benefícios da tratabilidade de parâmetro fixo. Nesse caso, desde a perspectiva da engenharia de software, a combinação é perfeita pensando na reutilização de algoritmos.

6.2 Definições básicas

Um problema parametrizado pode ser geralmente visto como um problema que tem como entrada dois componentes, i.e. um par $\langle x, k \rangle$. O primeiro é geral-

mente uma instância (i.e. $x \in I_P$) de algum outro problema de decisão P e o último é algum aspecto numérico do primeiro (geralmente um número positivo, supondo $k \ll |x|$), que constitue um *parâmetro*. Para um problema de maximização (resp. minimização) P, a linguagem induzida $L_P(param)$ é uma linguagem parametrizada consistindo de todos os pares $\langle x, k\rangle$ onde $x \in I_P$ e $opt_P(x) \geq k$ (resp. $opt_P(x) \leq k$). Se existe um algoritmo que resolve $L_P(param)$ no tempo $O(f(k)|x|^\alpha)$, onde $|x|$ é uma medida do tamanho da instância, $f(k)$ uma função *arbitrária* dependente de k somente, e α uma constante independente de k ou n, o problema parametrizado é dito ser *tratável a parâmetro fixo* e o problema de reconhecimento de linguagem pertence à classe de complexidade FPT .

Ainda que surpreendente, embora o argumento não seja difícil, observamos que a classe FPT é inalterada se a definição acima é modificada substituindo $f(k)|x|^\alpha$ por $f(k) + |x|^\alpha$ [60].

Para ilustrar este tópico, podemos discutir um dos mais emblemáticos problemas FPT:

k-Vertex Cover

Instância: Um grafo $G(V, E)$ e um inteiro positivo k. **Parâmetro:** k.

Questão: Há um conjunto $V' \subseteq V$, tal que para toda aresta $(u, v) \in E$, no mínimo u ou v é um membro de V' e $|V'| \leq k$?

Em geral, os algoritmos FPT eficientes são baseados nas técnicas de *redução a um núcleo de problema* e *árvores de busca limitadas*. O leitor interessado em um rápido entendimento destes métodos pode encontrar no artigo de Chen, Kanj e Yia, um tratamento muito apropriado para esses fins [68]. Nele, os autores mostram como a versão parametrizada de Vertex Cover (quer dizer, o k-Vertex Cover definido acima) pode ser resolvida no tempo $O(1.271^k k^2 + kn)$, onde k é a cardinalidade do subconjunto de vértices procurado e $n = |V|$.

A definição da classe FPT permite à função $f(k)$ crescer arbitrariamente rápido. Para Planar Dominating Set o melhor algoritmo conhecido foi $O(11^k n)$ por alguns anos [107]. Num trabalho muito recente, Alber, Bodlaender, Fernau, e Niedermeier, mostram que o problema pode ser resolvido em $O(c^{\sqrt{k}} n)$, onde $c = 3^{6\sqrt{34}}$ [6], mas para outros problemas, como k-Vertex Cover esta situação não é tão dramática.

Do lado positivo, um método geral novo que melhora algoritmos FPT que executam no tempo $O(q(k)\beta^k + p(n))$ (onde p e q são polinomiais e β é uma constante pequena) foram desenvolvidos por Niedermeier e Rossmanith [290]. O método intercala a redução para um núcleo de problema e métodos de busca limitados. Se é o caso que β^k é o tamanho da árvore de busca limitada e $q(k)$ é o tamanho do problema kernel, a nova técnica é capaz de eliminar o fator $q(k)$ permitindo o algoritmo ser $O(\beta^k + p(n))$. Com este algoritmo, o algoritmo apresentado por Chen, Kanj e Yia em [68] pode ser melhorado, dando lugar a um algoritmo que resolve a versão parametrizada de Vertex Cover no tempo $O(1.271^k + n)$.

Para discutir outro problema que pertence à classe FPT, primeiro apresentaremos seu problema associado da classe NPO:

MIN HITTING SET
Instância: Uma coleção C de subconjuntos de um conjunto finito S.
Solução: Um conjunto, i.e. um subconjunto $S' \subseteq S$, tal que S' contenha no mínimo um elemento de cada subconjunto em C.
Medida: Cardinalidade do conjunto, i.e. $|S'|$.

De acordo com os conceitos de aproximabilidade, o problema foi provado ser equivalente a MIN SET COVER em [16]. Ou seja, algoritmos de aproximação e resultados de não aproximabilidade para MIN SET COVER valem também para MIN HITTING SET. A variação restrita na qual a entrada é também tem definido um subconjunto T de S, e o problema é encontrar o "hitting set" que contém o maior número de elementos de T, é não aproximável em $|S|^{\epsilon}$ para algum $\epsilon > 0$. Entretanto, alguns casos especiais nos quais é dado, por exemplo, subconjuntos compactos de $\mathbb{R}$, e o objetivo é encontrar um conjunto de linhas retas de mínima cardinalidade tal que cada subconjunto dado é "hitted" por pelo menos uma linha, são aproximáveis [159].

Um caso especial deste problema é:

k-HITTING SET FOR SIZE THREE SETS
Instância: Uma coleção C de subconjuntos de tamanho três de um conjunto finito S e um inteiro positivo k. **Parâmetro:** k.
Questão: Existe um subconjunto $S' \subseteq S$, com $|S'| \leq k$ que permite S' conter no mínimo um elemento de cada subconjunto em C ?

Este problema, em troca, pode ser visto como uma generalização do problema VERTEX COVER mas para hipergrafos, onde há um "hiper-arco" entre três vértices (ao invés de somente dois como em VERTEX COVER), e a tarefa da versão de otimização é encontrar um subconjunto mínimo de vértices que cobrem todos os hiper-arcos. Recentemente, Niedermeier e Rossmanith apresentaram um algoritmo que resolve este problema no tempo $O(2.311^k + n)$ [288], usando seu mais recente método de aceleração.

6.3 Por que a classe FPT é relevante ?

6.3.1 Argumento da abundância de problemas naturalmente parametrizados no "mundo real"

É difícil articular uma defesa mais completa das razões que justificam a conveniência da pesquisa em tratabilidade de parâmetro fixo, e complexidade parametrizada em geral, que as disponíveis nos mais recentes artigos do assunto. Veja-se por exemplo [109] [287], e particularmente [108].

Obtendo emprestado um argumento "estatístico", estima-se atualmente que "em torno da metade" dos problemas NP-completos listados no livro de Garey e Johnson [131] tem versões naturalmente parametrizadas que estão em FPT incluindo três dos seis problemas básicos no Capítulo 3 de [131]. Isto

é importante visto que há muitas maneiras nas quais números que são pequenos ou moderadamente grandes (ex., $k \leq 40$) naturalmente aparecem nas especificações dos problemas e podem ser considerados como parâmetros. Por exemplo, *problemas em base de dados e lógica* são freqüentemente definidos como tendo uma entrada consistindo em uma fórmula (que pode ser pequena e relativamente invariante), e algumas outras estruturas (tal como base de dados) que são tipicamente grandes e variáveis. O tamanho da fórmula, ou outros aspectos da estrutura das fórmulas, pode ser um parâmetro relevante [380]. O *número de processadores ou máquinas a serem seqüenciadas* podem ser limitadas por um valor tal como $k \leq 20$. O número de graus de liberdade em um problema de planejamento de movimentos em robótica é comumente $k \leq 10$. Grafos de intervalo de *"pathwidth"* $k \leq 10$ têm aplicações em problemas de reconstrução de seqüências de DNA [35]. Outras aplicações em desenho de VLSI e manufatura são descritas em [108].

Por outro lado, os resultados de FPT geralmente exploram algumas estruturas presentes em casos especiais de um problema, da mesma forma que MAs são geralmente bons quando podem explorar alguma forma de estrutura presente nas instâncias. A idéia de explorar a estrutura dos problemas está longe de ser nova; podemos citar como exemplo um artigo publicado por H. Fujiwara em 1988. Ele mostrou que problemas de *geração automática de padrões de teste* em circuitos digitais (ATPG) (que foi provado NP-completo por Ibarra e Sahni in 1975) podem ser resolvidos eficientemente, isto é em tempo polinomial em relação ao tamanho do circuito para alguns tipos de instâncias. Ele introduziu a definição de *circuitos k-limitados*; resumidamente, um circuito é dito ser k-limitado se seus nós podem ser particionados em blocos disjuntos tal que cada bloco tem no máximo k entradas, e os blocos formam um grafo acíclico dirigido sem caminhos reconvergentes. Isto, basicamente, significa que toda reconvergência dos circuitos tem uma certa natureza *local* e limitada dentro de blocos de k entradas. Então, pode ser provado que existe um algoritmo que resolve ATPG e que é exponencial em k, mas para uma constante k, é polinomial em relação ao tamanho do circuito. Enquanto que para circuitos VLSI gerais a "k-limitação" pode ser muito restritiva, o resultado é ainda importante posto que alguns circuitos como *"ripple-carry adders"*, *"decodes"*, e arranjos celulares em uma e duas dimensões são deste tipo. Por outro lado, desde que técnicas usadas para geração de padrões de testes são atualmente usadas para problemas de verificação [215], o estudo de tratabilidade de parâmetro fixo pode ser útil como uma ferramenta de desenho de circuitos, assim como pode ser de outros tipos de sistemas complexos.

Dito isto, podemos dizer que "parametrização" é o caso mais geral encontrado na prática, e não a exceção. Além disso, Complexidade Parametrizada é dita estar baseada em um *"trato com o Diabo da Intractabilidade"*, que tenta identificar onde a parte principal do problema contribui para a complexidade global de um modo "bom" (quer dizer, polinomialmente), e ao mesmo tempo isola quais partes da entrada (esperançosamente o parâmetro) que contribuem pobremente para a complexidade (quer dizer, exponencialmente). O objetivo é *"confinar a explosão combinatorial"* [107].

6.3.2 Um argumento da perspectiva da "falta de progresso"

Nas últimas duas décadas temos visto um pequeno, ou nenhum, progresso para provar a conjectura que P $\neq$ NP. Isto pode ser visto como uma formalização da intuição de que há problemas que são *inerentemente* difíceis e isto é amplamente considerado como um dos problemas não solucionados mais importantes em Ciência de Computação e dos fundamentos da Matemática. A resolução final dessa conjectura vale muito mais do que o prêmio de 1.000.000 de dólares americanos atualmente oferecido[1].

Uma das mais profundas conseqüências da pesquisa em algoritmos de aproximação da última década é que tem sido encontrados muitos resultados que sugerem que a maioria dos problemas NP-difíceis que podem ser formulados como problemas de otimização associados *também* não têm bons algoritmos de aproximação [160], [11], [13], [92].

Recorremos novamente ao MIN VERTEX COVER como um exemplo ilustrativo, neste caso, de "falta de progresso". Por exemplo, em relação ao algoritmo de aproximação visto no capítulo inicial dessa tese (com $\rho = 1$, foi apresentado na Fig. 1.3), Lewis e Papadimitriou afirmam em [225], página 337, *"Podemos fazer melhor ? Desanimadoramente, este simples algoritmo de aproximação é o melhor conhecido para o problema."* Muito recentemente, Halperin melhorou isto para $2 - \frac{2 \ln \ln |V|}{\ln |V|}(1 - o(1))$ [155]. Também recentemente foi possível provar que, a menos que alguma conjectura forte não seja verdade (P $\neq$ NP) não existe algoritmo de aproximação que garanta fornecer uma solução tendo menos vértices que 7/6 vezes a cardinalidade ótima [161].

Enquanto o algoritmo 2-aproximado da Fig. 1.3 tem o benefício de ser mais simples, é inaceitável na maioria dos problemas práticos. Cientistas que trabalham mais perto da prática conhecem muitos métodos alternativos, particularmente meta-heurísticas como os MAs, que dão resultados bem melhores e também não são difíceis de implementar. Para muitos teóricos, *a facilidade de codificar não é relevante*, e possivelmente tenham razão de uma certa perspectiva, mas a radicalização desses argumentos só aumenta a distância entre teoria e prática. Além disso, métodos que não têm garantias teóricas de desempenho são desconsiderados, apesar de todas as evidências empíricas de sua utilidade em situações práticas. Entretanto, Downey, Fellows e Stege, em [108], alertam sobre a *"desconfortável posição"* na qual a Teoria da Computação se acha atualmente quando afirmam (nossa tradução):

> O problema principal em considerar heurísticas como um programa sistemático para enfrentar a intratabilidade é que não é um programa matemático coerente. O conjunto positivo de ferramentas inclui Vodu e a *"kitchen sink"*. Como um programa, não pede qualquer teorema, somente desempenho empírico. Os sucessos inegáveis de algoritmos heurísticos "sem inteligência", que geralmente não são analisáveis, colocam a Teoria de Ciência de Computação em uma posição incômoda.

[1] `http://www.claymath.org/prize_problems/p_vs_np.htm`

6.3.3 Um argumento do progresso

Em contraste com a falta de melhoria em algoritmos de aproximação, recentemente justificada pela obtenção de provas formais da dificuldade de resultados de aproximação, houve um avanço em algoritmos FPT para o problema. Resumimos aqui as realizações principais para o problema k-VERTEX COVER. Seja então $n = |V|$, o número de vértices do grafo.

- Fellows e Langston, em 1986, observaram primeiro que a teoria de *"graph minors"* desenvolvida por Robertson e Seymour pode ser usada para concluir que, para cada valor fixo de k, o problema k-VERTEX COVER pode ser resolvido em tempo $O(f(k)n^3)$ [114].

- D.S. Johnson observou em 1987 [179] que um algoritmo $O(f(k)n^2)$ para k-VERTEX COVER poderia ser obtido usando uma combinação de *decomposição em árvore* e técnicas de programação dinâmica de estado finito.

- Em 1988, M. Fellows apresentou um algoritmo para k-VERTEX COVER baseado em uma técnica de busca em árvore [113] com um tempo $O(2^k n)$.

- Em 1989, S. Buss desenvolveu um algoritmo que executa em $O(kn + 2^k k^{2k+2})$, veja [57]. O método foi baseado no que agora conhecemos como *redução ao núcleo de problema*. Foi aperfeiçoado como um método geral em [106].

- Em 1992, Balasubramanian, Downey, Fellows, e Raman, combinaram idéias de [113] e as desenvolvidas por Buss. Posteriormente, Balasubramanian, Fellows, e Raman em [25] reportaram estes descobrimentos em um novo algoritmo com tempo de execução de $O(kn + 2^k k^2)$.

- Usando o algoritmo de máximo emparelhamento como uma subrotina, Papadimitriou e Yannakakis apresentaram um algoritmo $O(3^k n)$ em 1993 [311]. Note também o fato deste produzir um algoritmo em tempo polinomial para k-VERTEX COVER nos casos onde k é $O(log(n))$ [312] Veja também o trabalho independente de Bonet, Steel, Warnow, e Yooseph [39].

- Em [25], Balasubramanian, Fellows, e Raman, descreveram um algoritmo que executa em $O(kn + (4/3)^k k^2)$ baseado numa combinação e refinamento de métodos anteriores.

- Em 1999, o algoritmo $O(kn + 1.324718^k k^2)$ de [25] foi melhorado para $O(kn + 1.31951^k k^2)$ in [109].

- Num trabalho mais recente, Niedermeier e Rossmanith melhoraram o resultado para $O(kn + 1.29175^k k^2)$ [289].

- Finalmente, reportamos os dois resultados que já mencionamos. Chen, Kanj e Yia melhoraram o resultado para $O(kn + 1.271^k k^2)$ [68]. Combinado com os métodos gerais introduzidos em [290] isto poderia dar um impressionante $O(1.271^k + n)$, i.e., linear em n para k fixo.

6.3.4 Um argumento da novidade

Desde o início das pesquisas em alguns problemas NP-difíceis, importantes esquemas de aproximação em tempo polinomial foram encontrados para problemas NP-completos, tais como BIN PACKING e KNAPSACK. Desde o início do desenvolvimento da teoria sobre problemas NP-completos até a atualidade, podemos observar que o projeto de encontrar algoritmos de aproximação tem atraído enormes esforços durante as três últimas décadas. Como destacamos com o MIN VERTEX COVER, à exceção de bons resultados em problemas que aceitam um *ptas* ou *fptas*, é agora claro que, com as poderosas técnicas novas de prova [12], resultados deste tipo *não são típicos* para a classe de problemas NP-difíceis e não poderíamos esperar que esquemas de aproximação sejam encontrados (a menos que algumas das suposições normalmente acreditadas, como por exemplo P $\neq$ NP, não aconteçam).

Em comparação com o número de investigadores que trabalham na perspectiva de encontrar esquemas de aproximação eficientes, o número de investigadores que trabalham em complexidade parametrizada é muito pequeno. É atribuído a Benoit Mandelbrot a seguinte citação:

> *"Science would be ruined if (like sports) it were to put competition above everything else, and if it were to clarify the rules of competition by withdrawing entirely into narrowly defined specialties. The rare scholars who are nomads-by-choice are essential to the intellectual welfare of the settled disciplines."*

que tentamos traduzir da seguinte maneira:

> A Ciência ficaria arruinada se (como nos esportes) a competição fosse colocada acima de tudo, e se fossem estabelecidas as regras de competição ao concentrar-se completamente em especialidades estreitamente definidas. Os raros estudiosos que são "nômades por própria escolha" são essenciais ao bem-estar intelectual das disciplinas já estabelecidas.

O campo da Complexidade Parametrizada, desenvolvido por diversos destes "nômades por própria escolha", parece estar tendo avanços significativos e, o que é mais importante, avanços sistemáticos em problemas particulares. Isto acontece às custas de estudar em alto detalhe muitas características particulares dos problemas, o que é um assunto semelhante àquilo que fazemos heuristicamente com algoritmos de recombinação. A novidade da metodologia, e os resultados encontrados nos últimos anos, sugere que é uma estratégia promissora para nos ajudar a superar a distância atual entre a falta de teoria e o sucesso das meta-heurísticas na prática.

6.3.5 Intratabilidade como a Hidra de Lerna

A visão de Downey e Fellows da intratabilidade como um "demônio" e o desenvolvimento do *"FPT Toolkit"* (de algoritmos eficientes para problemas particulares da classe FPT) como um "trato com o Diabo" (veja [107], página 487),

pode ser bem substituída com outra metáfora. Assim, recorremos à mitologia grega; a versão do mito que conhecemos é a seguinte:

> *"A segunda tarefa de Hércules foi matar um horrendo monstro conhecido como a Hidra de Lerna. A Hidra era uma enorme serpente com nove cabeças. Além disso, seu hálito era tão venenoso que quem a sentisse cairia morto. Palas-Atena, a deusa da sabedoria e da guerra prudente, ajudou a localizá-la. Cada vez que Hércules decepava uma das nove cabeças da Hydra com sua espada, duas cresciam em seu lugar. Esta besta não era tão fácil de derrotar, porque uma de suas nove cabeças era imortal e, portanto, indestrutível. Iolaus, que foi com Hércules, sugeriu queimar os tocos antes que eles tivessem a chance de crescer de volta; e com a ajuda de algum objeto quente (uma tocha ou um pau - dependendo da versão) Iolaus queimou as cabeças da serpente, cauterizando cada toco com tochas de ferro quente tão logo Hercules decepava cada cabeça e antes que elas tivessem chance de crescer novamente. Com o jorro de sangue detido, as cabeças da Hidra pararam de se multiplicar. Uma vez que as oito cabeças mortais foram removidas e destruídas, Hercules decepou a nona cabeça imortal. Hércules cortou fora a última cabeça e enterrou-a debaixo de uma rocha que pode ser encontrada nos dias de hoje no caminho entre Lerna e Elaeus. Hércules então molhou suas flechas com o sangue da Hidra, e assim conseguiu fazê-las venenosas e mortais."*

No momento de escrever esta tese, milhares de problemas foram categorizados como sendo NP-difíceis; a intratabilidade é um monstro com ordens de magnitude que correspondem a mais cabeças que a Hidra. Cada semana, mais problemas estão sendo reconhecidos como pertencentes a este conjunto, e assim os problemas continuam se multiplicando, até mesmo sem intervenção de Hércules ! Relativo à teoria sobre problemas NP-completos, podemos pensar em cada cabeça como um problema. Assim podemos ver o desenvolvimento de algoritmos de FPT como uma tentativa para reconhecer uma característica comum de problemas que permitem os milhares de Iolaus "queimar sistematicamente e controlar" alguns deles. Porém, os níveis mais altos da W-hierarquia indicam que há problemas que são "mais imortais" que outros em um certo sentido (discutiremos a W-hierarquia em 6.6). Se precisamos de outro paradigma para "sistematicamente queimá-los e controlá-los" ainda é uma pergunta aberta, mais relacionada ao fato de a hierarquia colapsar ou não.

O mito e a metáfora permitem uma nova interpretação filosófica de como esta batalha científica contra a intratabilidade está sendo administrada. Pensamos que este é um assunto crucial e sua extensão realmente nos preocupa. *Até que ponto o desenvolvimento de algoritmos de aproximação e algoritmos de parâmetro fixo são abordagens "escaláveis" para tratar com intratabilidade ?* No lado positivo, os métodos parecem complementares, começando pelo Min Vertex Cover, um exemplo paradigmático em que é complementada a falta de progresso em uma direção com o trabalho sendo feito na outra. Mas o número de problemas identificados parece ser impossível de contar, e está cres-

cendo mais rapidamente do que o desenvolvimento de algoritmos específicos. Em contraste, metodologias como Programação Inteira, Programação com Restrições, ou Programação Lógica, parecem estar se concentrando em achar um único método, uma única "bala de prata" para lidar com eles. Como tal, estas metodologias propõem uma representação restrita de problemas e instâncias, e por isso, podem ter completamente perdido as vantagens de uma representação mais adequada. A escolha de representações apropriadas, por outro lado, geralmente é a razão atrás do sucesso de métodos baseados em Busca Local e meta-heurísticas.

Não temos uma única resposta para estes assuntos. Apenas nos limitamos a levantar nossas preocupações. Ao mesmo tempo em que os fundos para pesquisa em Ciência de Computação Teórica são escassos, assim como a teoria diverge da prática próspera, vemos que a intratabilidade está presente em toda parte no "mundo-real". Neste momento, nem mesmo uma lista de todos os problemas NP-difíceis é mantida em domínio público ! Como tal, o "monstro" tem muitas cabeças, a maioria delas nem mesmo são conhecidas dos investigadores e dos que trabalham na prática. E o que mais perturba, aparte de nossa *"desiderata"* [126] para o Projeto *MemePool*, é que até o momento não temos escutado outras vozes preocupadas em reunir algoritmos que trabalhem juntos em um único sistema computacional [274]. Como coordenar então o trabalho de milhares de "Iolauses" se nem um repositório de algoritmos e informações foi desenvolvido como fruto da colaboração nos últimos trinta anos ?

6.4 Diferentes aproximabilidades de problemas NP-completos

Agora veremos como às vezes fica difícil reconhecer como podemos tratar, de acordo com as medidas de aproximabilidade, as diferentes "cabeças da Hidra".

Dado um problema NPO $F = (I_F, sol_F, m_F, goal_F)$, o correspondente problema (natural) de decisão $F^d = (I_F, sol_F, m_F, goal_F, K_F)$ é decidir, para uma entrada $x \in I_F$, se há uma solução $y \in sol_F(x)$ tal que $m_F(x, y) \geq K_F$ se $goal_F = \max$ e $m_F(x, y) \leq K_F$ se $goal_F = \min$. Já discutimos na Seção 3.1.1 o conceito de auto-redutibilidade e como a busca binária relaciona versões de decisão e otimização. Como uma conseqüência, todos os problemas NPO podem ser escritos como problemas de decisão e dado que todos os problemas NP-completos são redutíveis entre eles, pode-se conjecturar que todos os problemas NPO deveriam ter a mesma aproximabilidade. Por diversas razões isso não é verdade.

Uma das razões é que uma redução NP entre dois problemas NPO F e G (formalmente, uma redução entre as versões dos problemas de decisão de dois problemas NPO) raramente *preserva* a função objetivo. Se, não obstante, temos uma redução na qual a função objetivo de F se transforma na função objetivo de G, pode não ser o caminho que preserva a aproximabilidade. Considere por exemplo o problema Max Independent Set que abreviaremos como Mis, e o problema Min Vertex Cover que abreviaremos como Mvc, de um grafo não direcionado $G(V, E)$. Um subconjunto de V é independente se e

somente se seu complementar é uma cobertura de vértices. Assim, há uma cobertura de vértices de tamanho σ se e somente se há um conjunto independente de tamanho $|V| - \sigma$. É igualmente difícil encontrar uma solução para MIS com erro absoluto e e encontrar uma solução para MVC com erro absoluto e, mas é muito mais difícil encontrar uma solução para MIS com erro relativo ε do que encontrar uma solução para MVC com o mesmo erro relativo.

Os vértices de um emparelhamento maximal do grafo sempre formam uma cobertura de vértices de no máximo duas vezes o tamanho ótimo, considerando que nenhum algoritmo de aproximação existe, devido a P $\neq$ NP.

Aqui temos dois problemas de otimização que de uma certa maneira são os mesmos, mas têm diferentes aproximabilidades devido à definição de erro relativo. Neste caso específico temos sorte, porque obteremos melhores resultados se usarmos o erro relativo normalizado ao invés do erro relativo ordinário, mas geralmente não podemos esperar por isto.

Vamos considerar os problemas MIS e MVC, $x = G(V,E) \in I_{\text{MIS}} = I_{\text{MVC}}$, $y \in sol_{\text{MIS}}(x)$, $\bar{y} = V - y$, $worst_{\text{MIS}}(G(V,E)) = 0$ e $worst_{\text{MVC}}(G(V,E)) = |V|$. Agora os erros relativos normalizados são:

$$\mathcal{E}^n_{\text{MIS}}(x,y) = \frac{|opt_{\text{MIS}}(x) - m_{\text{MIS}}(x,y)|}{|opt_{\text{MIS}}(x) - worst_{\text{MIS}}(x)|}$$

$$\mathcal{E}^n_{\text{MIS}}(x,y) = \frac{opt_{\text{MIS}}(x) - m_{\text{MIS}}(x,y)}{opt_{\text{MIS}}(x)}$$

e

$$\mathcal{E}^n_{\text{MVC}}(x,\bar{y}) = \frac{|opt_{\text{MVC}}(x) - m_{\text{MVC}}(x,\bar{y})|}{|opt_{\text{MVC}}(x) - worst_{\text{MVC}}(x)|}$$

$$\mathcal{E}^n_{\text{MVC}}(x,\bar{y}) = \frac{m_{\text{MVC}}(x,\bar{y}) - opt_{\text{MVC}}(x)}{worst_{\text{MVC}}(x) - opt_{\text{MVC}}(x)}$$

$$\mathcal{E}^n_{\text{MVC}}(x,\bar{y}) = \frac{(|V| - m_{\text{MIS}}(x,y)) - (|V| - opt_{\text{MIS}}(x))}{|V| - (|V| - opt_{\text{MIS}}(x))}$$

$$\mathcal{E}^n_{\text{MVC}}(x,\bar{y}) = \frac{opt_{\text{MIS}}(x) - m_{\text{MIS}}(x,y)}{opt_{\text{MIS}}(x)}.$$

Assim $\mathcal{E}^n_{\text{MIS}}(x,y) = \mathcal{E}^n_{\text{MVC}}(x,\bar{y})$.

6.5 Aproximabilidade e a classe FPT

Veremos agora como o paradigma de algoritmos de aproximação e o *"FPT Toolkit"*, representado pelos algoritmos eficientes para problemas da classe de

FPT, estão relacionados. Seguimos o Capítulo 4 de [107] que se refere a resultados obtidos por Cai e Chen [59]. Como conseqüência: esta Seção não oferece resultados novos e está incluída para prover uma completa discussão do tópico.

Definição: Seja Q um problema parametrizado, dizemos que Q é *uniformemente tratável a parâmetro fixo* se há um algoritmo Φ, uma constante c, e uma função arbitrária $f : \mathbb{N} \to \mathbb{N}$ tal que

- o tempo de execução de $\Phi(\langle x, k\rangle)$ é no máximo de $f(k)|x|^c$,
- $\langle x, k\rangle \in Q$ se e somente se $\Phi(\langle x, k\rangle) = 1$.

6.5.1 Visão invariante

Lema: (Cai e Chen, 1997 [59]) *Seja Q um problema* NPO *arbitrário. Então os três problemas parametrizados $Q_=, Q_\leq$, e $Q_\geq$ são todos (fortemente) (uniformemente) tratáveis a parâmetro fixo, ou todos (fortemente) (uniformemente) intratáveis a parâmetro fixo.*

Prova: Suponha que um problema parametrizado $Q_\leq$ é tratável a parâmetro fixo (somente argumentaremos aqui para FPT uniformes visto que os argumentos são similares). Se $Q_\leq \in$ FPT, então existe um algoritmo $A_\leq$ que pode resolver $Q_\leq$ em tempo $O(f(k)n^c)$, onde f é uma função e c é uma constante independente do parâmetro k.

Primeiro note que a condição $k = opt_Q(x)$ é equivalente a simultaneamente preencher duas condições: $k \leq opt_Q(x)$ e $(k+1) > opt_Q(x)$. Então, dada uma instância $\langle x, k\rangle$ para o problema parametrizado $Q_=$ podemos executar o algoritmo $A_\leq$ com a entrada $\langle x, k\rangle$ e com a entrada $\langle x, k+1\rangle$ para decidir se $k = opt_Q(x)$. O tempo total de execução desta alternativa é limitado por $O((f(k) + f(k+1))n^c)$, que prova que $Q_= \in$ FPT.

Falta ainda provar na outra direção. Suponha que sabemos que $Q_= \in$ FPT. Então existe um algoritmo $A_\leq$ que resolve o problemas $Q_=$ no tempo $O(f'(k)n^{c'})$, onde f' é uma função recursiva e c' é uma constante independente do parâmetro k. Então, dada uma instância $\langle x, k\rangle$ do problema parametrizado $Q_\leq$, podemos executar o algoritmo $A_=$ com k entradas $\langle x, 0\rangle, \langle x, 1\rangle, \ldots, \langle x, k-1\rangle$. Obviamente, $\langle x, k\rangle$ não é uma instância do problema parametrizado Q_{leq} se e somente se $A_=$ conclui 'Sim' para no mínimo uma destas k entradas. Note que o tempo de execução do processo acima é limitado por $O(kf'(k)n^{c'})$. Assim, $Q_\leq \in$ FPT.

Analogamente, podemos provar que $Q_\geq$ é tratável a parâmetro fixo se e somente se $Q_=$ é tratável a parâmetro fixo.

6.5.2 Um *fptas* implica FPT

Teorema: (Cai, 1992; Cai e Chen, 1997 [59]) *Se um problema* NPO *tem um esquema de aproximação completamente polinomial, então ele é tratável a parâmetro fixo.*

Prova: Vamos supor que um dado problema NPO $Q = (I_Q, sol_Q, m_Q, opt_Q)$ tem um esquema de aproximação completamente polinomial A. Então, o algoritmo A tem como entrada uma instância $x \in I_Q$ e uma exigência de precisão $\epsilon > 0$. No tempo $O(p(1/\epsilon, |x|))$ (onde p é um polinômio de duas variáveis), A fornece uma solução factível tal que o erro relativo $R_A(x)$ é limitado por ϵ.

Vamos supor primeiro que Q é um problema de maximização. Pelo resultado de invariância anterior(da Sec. 6.5.1), tudo que precisamos mostrar é que o problema parametrizado $Q_{\leq}$ é tratável a parâmetro fixo. Seja $\langle x, k\rangle$ uma instância de $Q_{\leq}$, executaremos o algoritmo A com a entrada x e $1/2k$. Então A produz uma solução factível $y \in sol_Q(x)$ no tempo $O(p(2k, |x|))$. O erro relativo de y é:

$$R_A(x) = \frac{opt_Q(x) - m_Q(x,y)}{m_Q(x,y)} \leq \frac{1}{2k} < \frac{1}{k} \tag{6.1}$$

Se $k \leq m_Q(x,y)$, então certamente $k \leq opt_Q(x)$ dado que Q é um problema de maximização. Se $k > m_Q(x,y)$, então $(k-1) \geq m_Q(x,y)$. Combinando isto com $opt_Q(x) - m_Q(x,y)/m_Q(x,y) < 1/k$, temos $k > opt_Q(x)$. Então, $k \leq opt_Q(x)$ se e somente se $k \leq m_Q(x,y)$. Dado que A fornece uma solução y no tempo $O(p(2k, |x|)$, a conclusão é que o problema NPO original Q é tratável a parâmetro fixo.

Quando Q é um problema de minimização, podemos analogamente provar a conclusão mostrando que $Q_{\leq} \in$ FPT.

6.5.3 Um *ptas* que também implica FPT

Corolário:

O teorema da Sec. 6.5.2 também demonstra que qualquer problema de otimização com um *ptas* $A(x, \epsilon)$, com a propriedade que existe uma constante c independente de ϵ tal que $A(x, \epsilon)$ executa em um tempo $O(|x|^c)$ para um ϵ fixo, é tratável a parâmetro fixo.

6.6 Transformações Parametrizadas e a W-hierarquia

Definição: Uma *transformação parametrizada* de uma linguagem parametrizada L para uma linguagem parametrizada L' é um algoritmo que computa a partir de uma entrada que é um par $\langle x, k\rangle$, um par $\langle x', k'\rangle$ tal que:
(1) $\langle x, k\rangle \in L$ se e somente se $\langle x', k'\rangle \in L'$,
(2) $k' = g(k)$ é uma função somente de k, e
(3) a execução é feita no tempo $f(k)n^{\alpha}$, onde $n = |x|$, α é uma constante independente de n e k, e f é uma função arbitrária.

As transformações parametrizadas diferem das reduções de tempo polinomial ordinárias. Por exemplo, se $G = (V, E)$ é um grafo de n vértices, um conjunto de vértices $V' \subseteq V$ é um k-clique em G (i.e. un subgrafo completo de G com k vertices) se e somente se $V - V'$ é uma cobertura de vértices no grafo

complementar G', onde vértices são adjacentes se e somente se eles são adjacentes em G. Isto fornece uma redução de tempo polinomial do problema CLIQUE parametrizado naturalmente para o problema VERTEX COVER parametrizado, transformando a instância (G, k) de CLIQUE para a instância (G', k') de VERTEX COVER. Entretanto, isto não é uma transformação parametrizada, visto que $k' = n - k$ não é puramente uma função de k. A conjetura atual é que não há uma transformação parametrizada nesta direção. Existe uma elaborada transformação parametrizada a partir do problema CLIQUE parametrizado naturalmente para o problema DOMINATING SET parametrizado naturalmente, mapeando (G, k) para (G', k') onde $k' = 2k$ [105, 107]. Novamente, todas as evidências aparentemente indicam que não há transformação parametrizada na outra direção.

A propriedade essencial da transformação parametrizada é que se L transforma para L' e $L' \in$ FPT, então $L \in$ FPT. Isto naturalmente conduz a um programa de completude baseado na hierarquia de classes de problemas parametrizados:

$$FPT \subseteq W[1] \subseteq W[2] \subseteq \cdots \subseteq W[SAT] \subseteq W[P] \subseteq AW[P] \subseteq XP$$

O análogo parametrizado de NP é $W[1]$, e a $W[1]$-dificuldade é a evidência básica de que um problema parametrizado não é provavelmente tratável a parâmetro fixo. O k-STEP HALTING PROBLEM FOR NONDETERMINISTIC TURING MACHINES é $W[1]$-completo (onde a quantidade de não determinismo em cada passo da máquina de Turing é ilimitada). Desde que o $q(n)$-STEP HALTING PROBLEM é essencialmente o problema que define NP, a analogia entre NP e $W[1]$ é forte. A classe de complexidade parametrizada $W[P]$ é um análogo de PSPACE [2]. A classe XP é o conjunto de todas as linguagens parametrizadas L para as quais existem funções f e g e um algoritmo para determinar se $\langle x, k \rangle \in L$ no tempo $f(k)|x|^{g(k)}$.

6.7 Regras de redução

A Complexidade Parametrizada permite um certo tipo de "diálogo" com um problema único. Depois de demonstrar que um problema é $W[1]$-difícil, o próximo passo natural é encontrar se existem versões dele, usando alguma parametrização diferente, que podem estar na classe FPT. Note que é sempre possível parametrizar um problema de várias maneiras, que são tratáveis a parâmetro fixo. Assim, alternativamente, não é surpresa que muitos problemas parametrizados aparentemente *não* pertencem a FPT.

O problema DOMINATING SET parametrizado naturalmente mencionado acima, no qual o parâmetro é a cardinalidade do conjunto dominante de vértices, é um destes. Entretanto, o PLANAR DOMINATING SET está em FPT. Também mencionamos antes que uma das técnicas usadas para desenvolver um algoritmo FPT é baseada nos métodos de *busca em árvores limitadas* e *redução a um núcleo de problema*. A idéia principal do último é reduzir uma instância do problema $x \in I_P$ a uma instância "equivalente" x' onde o tamanho de x' é limitado por uma função do parâmetro k. A instância x' é então exaustivamente analisada, e uma solução para o x' pode ser levada a uma solução para

x, no caso em que solução existe. Para o método de *árvores de busca limitadas*, o uso de *regras de redução* são essencialmente regras heurísticas, mas provadas corretas, para "encolher" as árvores de busca envolvidas. Por exemplo, para o MIN VERTEX COVER já introduzimos uma regra de redução no procedimento de recombinação discutido na Seção 1.5 e mostrado na Figura 1.6. A regra de redução pode ser escrita como:

> "Se o grafo de entrada tem um vértice u de grau 1, o outro vértice final v *deve estar* na cobertura."

Podemos reescrever esta regra nos seguintes termos: *"Se existe um aresta $(u, v) \in E$, onde E é um conjunto de arestas de um grafo não direcionado $G(V, E)$, e v tem grau 1, então o grafo G tem uma cobertura com k vértices se, e somente se, o grafo G', o grafo obtido a partir de G removendo os vértices u e v e todas as arestas a eles conectadas, tem uma cobertura com $k-1$ vértices."* Note que as regras de redução são altamente dependentes do problema, mas algumas delas podem ser usadas para outros problemas. Por exemplo, para o problema DOMINATING SET podemos também provar que:

> "Se o grafo de entrada tem um vértice u de grau 1, o outro vértice v a ele conectado *deve estar* no conjunto dominante."

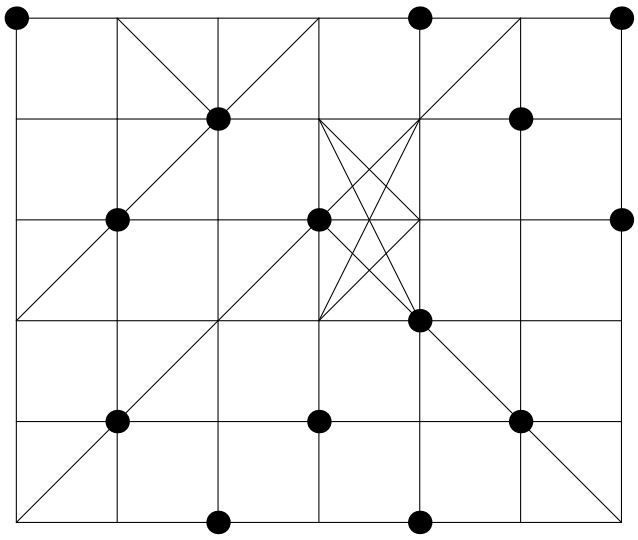

Figura 6.1: *Um grafo e um conjunto dominante com 14 vértices. Por motivos de clareza, só eles são marcados; os outros vértices do grafo correspondem a pontos na grade (42 no total).*

Podemos ver na prática o efeito de aplicar esta regra de redução como parte de um algoritmo de recombinação. Suponha que temos dois conjuntos dominantes de vértices, nas Figuras 6.1 e 6.2. Como fizemos no Capítulo 1 (Fig. 1.6), em nosso desenho o grafo tem vértices somente em cada um dos

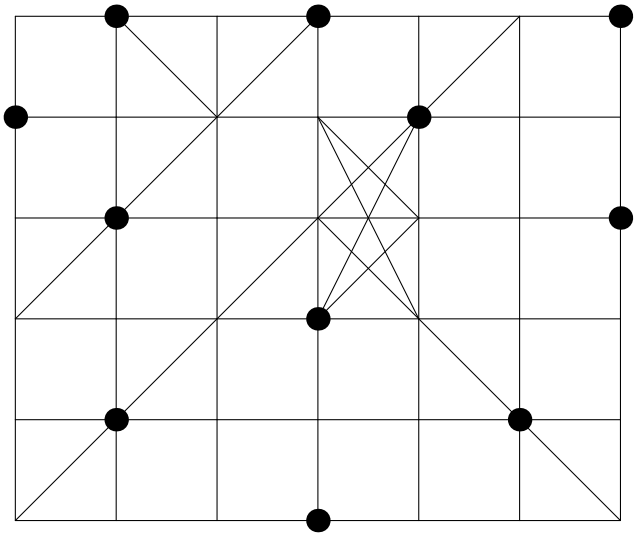

Figura 6.2: *O mesmo grafo da Fig. 6.1 e um conjunto dominante que só precisa usar 11 vértices.*

"pontos da grade". Dito isto, o grafo tem 42 vértices e a Fig. 6.1 mostra um conjunto dominante D_1 que usa 14 vértices. Na Fig. 6.2 mostramos outra solução factível D_2 com 11 vértices.

Agora, a Figura 6.3 mostra o resultado de um procedimento de recombinação que respeita os vértices que são comuns a ambos os conjuntos dominantes. Este procedimento de recombinação trabalha da seguinte maneira:

- **Passo 1** Identificar todos os vértices em $D_1 \cap D_2$. Seja V_D o conjunto dos vértices de G que não estão em $(D_1 \cap D_2)$ mas já estão dominados por um vértice em $(D_1 \cap D_2)$.
- **Passo 2** Remova de G todas as arestas u, v tal que u ou v (ou ambas) estejam em V_D.
- **Passo 3** Remova de G todas as arestas u, v tal que u ou v (ou ambas) estejam em $D_1 \cap D_2$. Remova todos os vértices em $(D_1 \cap D_2)$.
- **Passo 4** Remova todos os vértices, agora isolados, em $V_D \cup (D_1 \cap D_2)$.
- **Passo 5** Encontre a solução ótima do problema Min Dominating Set do grafo restante.

Note que o **Passo 5** do procedimento de recombinação proposto acima pode recorrer a uma regra de redução válida como uma das anteriormente propostas. Por aplicação repetida, podemos encontrar uma solução ótima que somente requer 4 vértices para dominar o grafo restante, assim conseguimos construir uma solução *descendente* usando somente 9 vértices. Por outro lado, pode ser o caso de continuarmos aplicando todas as regras de redução válidas

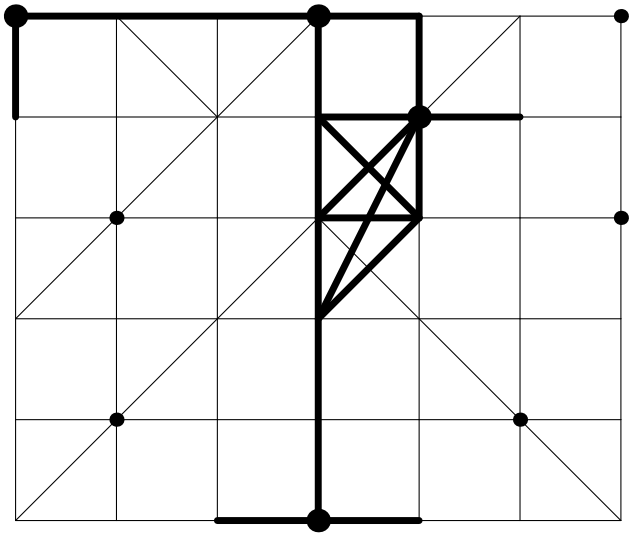

Figura 6.3: *A solução descendente. Se preservamos os vértices presentes em ambas as soluções, o problema se reduz a achar um conjunto dominante no grafo composto pelas arestas realçadas e os vértices a elas conectados. A aplicação reiterada da regra de redução discutida acima dá um descendente que só precisa de 9 vértices.*

até que nenhuma outra regra possa ser usada e, neste caso, podemos recorrer a um método sistemático, como *"Branch-and-Bound"* para resolver a instância restante.

Note também que pode ser o caso em que, enquanto tentamos resolver um problema dado que não está em FPT, podemos realmente, depois da recombinação, estar com um subproblema que é a instância de um problema em FPT. Por exemplo, o grafo restante na Fig. 6.3 pode ser mostrado ser planar. Isto significa que uma direção alternativa para a pesquisa teórica é a de recombinar soluções, tal que os problemas de otimização a serem resolvidos estejam em FPT (ou ainda melhor, se um algoritmo de tempo polinomial existe para tal problema). Trataremos este tema no próximo capítulo.

Do lado prático, isto significa que muito código pode ser apropriadamente utilizado outra vez. Outra possibilidade, que é extremamente útil na prática e não deve subestimada, é recorrer a algumas meta-heurísticas para resolver o subproblema se este não pertence a FPT ou se o tamanho da instância é muito grande.

Em termos da comparação mitológica proposta, podemos dizer que regras de redução, os algoritmos FPT, e os métodos já desenvolvidos anteriormente, podem ser como as flechas de "Hércules ", permitindo-nos usá-las sempre que apropriadas.

Capítulo 7

Algoritmos Polinomiais de Fusão

7.1 Introdução e Motivação

Em 1978, D. West levantou a questão:

> *"Sob quais circunstâncias as arestas de um par de circuitos hamiltonianos podem ser redistribuídas para formar outro par de circuitos hamiltonianos com a mesma união e intersecção?"*

Um dos objetivos deste capítulo é mostrar como perguntas como esta estão inerentemente relacionadas com assuntos de pesquisa atuais em meta-heurísticas paralelas e Computação Evolutiva. A conjectura de West pode ter aplicação direta para desenvolver melhores algoritmos meméticos para o MIN TSP, por exemplo. Neste capítulo apresentamos, pela primeira vez, de acordo com o nosso conhecimento, vários problemas de grafos que surgem da definição da classe denominada em inglês como *"Polynomial Merger Algorithms"* (PMA), que poderíamos traduzir como *"Algoritmos Polinomiais de Fusão"*. Essa é uma classe de complexidade computational recentemente proposta em [273] como uma formalização de vários problemas que aparecem quando projetamos algoritmos meméticos. A versão irrestrita, uma classe mais geral, é chamada uPMA e é análoga à classe chamada PLS devido a seu nome em inglês, i.e. *Polinomial Local Search*. As classes uPMA e PMA tentam dar um marco de trabalho para a análise teórica de operadores de *reprodução* e *recombinação*, o que é relevante para *Algoritmos Genéticos* e meta-heurísticas paralelas. Diversas oportunidades para algoritmos paralelos são destacadas.

Como aconteceu com o paradigma de *Busca Local*, que estava sendo amplamente utilizado antes que qualquer abordagem sistemática para caracterizar suas classes de complexidade associadas tenha sido tentada, pesquisas em MAs seguiram um caminho semelhante. Entretanto, em [273] fizemos uma primeira tentativa de formalização. Visto que a maioria dos MAs que aparece na prática faz uso de *recombinação* de soluções e períodos de otimização individual, decidimos em [273] formalizar uma classe de complexidade computacional que

envolve problemas e procedimentos de *recombinação* ou *fusão*. Neste capítulo fazemos uma breve introdução destas idéias, assim como de vários problemas interessantes em aberto relacionados à Teoria de Grafos.

Há certos paradoxos nesta iniciativa. Muitos teóricos da Ciência da Computação estão atualmente procurando razões matemáticas que justificariam o uso de meta-heurísticas de *"agente único"* (como *Simulated Annealing*). Para quem utiliza MAs, este é um desperdício de esforços, porque estas meta-heurísticas de agente único tipicamente executam relativamente pior em comparações justas com MAs para uma ampla gama de problemas. Um exemplo paradigmático é o MIN TRAVELING SALESMAN PROBLEM (MIN TSP) [257]. Além disso, as abordagens baseadas em Teoria de Probabilidade às vezes estão condenadas desde o início, dado que elas não podem manipular (ou estão perdendo como conseqüência da média estatística) algumas características específicas das instâncias e dos casos de complexidade de pior caso dos subproblemas associados.

Um dos paradoxos que encontramos é que sempre, na literatura da Ciência da Computação e da Matemática, há pioneiros que levantaram perguntas semelhantes no passado. Como um exemplo disso, *trinta e cinco* anos atrás, Lin conjeturou que a união de quaisquer dois circuitos hamiltonianos de arestas disjuntas de qualquer grafo G também é a união de *dois outros* circuitos hamiltonianos de arestas disjuntas de G. Em [371], D. West estendeu a conjectura para incluir todos os pares cuja interseção contém exatamente uma única aresta. Em essência, isto imediatamente sugere uma direção de pesquisa interessante. *É possível usar esta propriedade para construir uma boa heurística paralela para o* MIN TSP *?* Não sabemos se este outro assunto foi parte do programa de trabalho de pesquisa de Lin, mas levou mais de treze anos até que a conjectura tenha sido demonstrada verdadeira por A. Thomason [359]. Naquele momento, o sucesso notório da heurística de busca local de Lin-Kernighan para o MIN TSP estava influenciando extremamente a Otimização Combinatória. A idéia de usar soluções múltiplas teve que esperar até que as deficiências do paradigma da busca local começaram a ser evidentes para então sim permitir aos MAs entrar em cena.

A decepção com algumas meta-heurísticas de agente único tem sido uma das principais razões que aumentaram o interesse em MAs, assim como também a disponibilidade de sistemas paralelos. Quando usadas como uma heurística, os MAs combinam o poder de busca baseada em população com as heurísticas *ad-hoc* para o problema em questão. Os MAs podem ser levados a ser uma busca denominada *completa* com a adição de algum método de busca sistemático, como *"Branch-and-Bound"* ou *"Branch-and-Cut"*. Porém, a maioria das aplicações usa alguma forma de busca local. Dado que há uma classe de complexidade associada que relaciona busca local a problemas de otimização, pensamos que outra classe é necessária para analogamente relacionar a *recombinação de soluções*.

7.2 Algoritmos Polinomias de Fusão e as classes uPMA e PMA

Por analogia com a definição de classe PLS, de *"Polynomial Local Search"* podemos definir outras classes para os Algoritmos Polinomias de Fusão, em língua inglesa *"Polynomial Merger Algorithms"* (PMA) e a classe de fusão *irrestrita* (uPMA) que contém a primeira.

Definiremos as classes PMA e uPMA nos referindo a três algoritmos análogos aos que definem a classe de problemas *Polynomial Local Search* (PLS). Estas definições (especialmente para a classe PMA) são particularmente dependentes do algoritmo chamado *k-merger*, que ajuda a formalizar a noção de *recombinação de um conjunto de k soluções dadas*, como geralmente usadas pela maioria dos MAs (assim como também em outras abordagens populacionais). Notamos que esta abordagem é nova e diferente das provenintes da abordagem anterior denominada *"Formae Theory"* [323]. .

Seja x uma instância de um problema de otimização P (escrevemos $x \in I_P$). Seja $sol_P(x)$ o conjunto de soluções factíveis do problema P dada uma instância x. Se $y \in sol_P(x)$, seja $m_P(y,x)$ a função objetivo. Denotamos como $\mathcal{M}_P(S_{par},x) \subseteq sol_P(x)$ o conjunto de todos os possíveis resultados que o algoritmo *k-merger* pode dar se recebe como entrada o par (S_{par},x) para o problema P. O conjunto $\mathcal{M}_P(S_{par},x)$ generaliza a idéia de *potencial dinástico* para mais de dois pais [82]. A entrada de um algoritmo k-merger é um conjunto S_{par} de $k \geq 2$ soluções *factíveis*, então $S_{par} \subseteq sol_P(x)$. Elas podem ser informalmente chamadas de soluções "pais" e, se tiver sucesso, o k-merger gera *no mínimo uma* solução factível. Para a classe uPMA a geração da nova solução é menos restrita que para PMA. Em geral, os processos de recombinação podem ser muito complexos e com muitas restrições envolvendo a detecção, a preservação ou a proibição, e a combinação factível de *características* já apresentadas nas soluções pais.

Definição (uPMA): Seja x uma instância de um problema de otimização P. Com $\mathcal{M}_P(S_{par},x) \subseteq sol_P(x)$ denotamos o conjunto de todos os possíveis resultados que o algoritmo *k-merger* pode fornecer se ele recebe como entrada o par (S_{par},x) para o problema P. (Note que o conjunto $\mathcal{M}_P(S_{par},x)$ generaliza a idéia de *potencial dinástico* para mais que dois pais [82].)

Um problema de recombinação $P/\mathcal{M}$ pertence a uPMA se existem três algoritmos de tempo polinomial *p-starter*, *p'-evaluator*, e *k-merger* (onde p, p' e k são números inteiros tais que $p' \geq p \geq k \geq 2$) que satisfazem as seguintes propriedades:

- Dada uma entrada x (formalmente um string $\in \{0,1\}^*$), o *p-starter* determina se $x \in I_P$ e, nesse caso, produz um conjunto de p diferentes soluções factíveis $\{y_1, y_2, \ldots, y_p\} \subseteq sol_P(x)$.

- Dada uma instância $x \in I_P$ e uma entrada (formalmente um string $\in \{0,1\}^*$), o *p'-evaluator* determina se esta entrada representa um conjunto de soluções factíveis, i.e. $\{y_1, y_2, \ldots, y_{p'}\} \subset sol_P(x)$ e neste caso computa o valor da função objetivo associada a cada uma delas, i.e. $m_P(y_j,x), \forall j = 1, \ldots, p'$.

- Dada uma instância $x \in I_P$ e um conjunto de k soluções factíveis $S_{par} \subseteq sol_P(x)$, o *k-merger* determina se o conjunto S_{par} é um *k-merger optimum*, e, se não é, retorna no mínimo uma nova solução factível $y' \in \mathcal{M}_P(S_{par}, x)$ com valor estritamente melhor de m_P, (i.e. $m_P(y', x) < \max\{m_P(y_1, x),\ m_P(y_2, x), \ldots, m_P(y_k, x)\}$ para um problema de minimização, e $m_P(y', x) > \min\{m_P(y_1, x),\ m_P(y_2, x),\ \ldots, m_P(y_k, x)\}$ para um problema de maximização).

Analogamente, a classe PMA é mais restrita, pois incorpora um tipo particular de recombinação. Para uPMA o tipo de recombinação é implícito e de acordo com a maneira com que a vizinhança $\mathcal{M}$ é definida. Entretanto, a definição para PMA é geral o bastante para englobar a maioria dos procedimentos de recombinação utilizados na prática pelas metodologias baseadas em população.

Definição (PMA). Um problema de recombinação $P/\mathcal{M}$ pertence a PMA se existem três algoritmos de tempo polinomial *p-starter*, *p'-evaluator*, e *k-merger* (onde p, p' e k são números inteiros tais que $p' \geq p \geq k \geq 2$), tais que o *p-starter* e *p'-evaluator* satisfazem as mesmas propriedades necessárias pela classe uPMA mas o *k-merger* é restrito a ser de um tipo particular, i.e.:

- Dada uma instância $x \in I_P$ e um conjunto de k soluções factíveis $S_{par} \subseteq sol_P(x)$, o *k-merger* determina se o conjunto S_{par} é um *k-merger optimum*, e, se não é, faz o seguinte:
  - Para cada $y \in S_{par}$, resolve n_1 problemas de decisão de tempo polinomial $\{\Pi_1(y), \ldots, \Pi_{n_1}(y)\}$. Seja D uma matriz de $k \times n_1$ coeficientes Booleanos formados pelos resultados de todos os problemas de decisão, i.e. $D_{i,j} = \Pi_j(y_i)$.
  - Cria um conjunto de n_2 *restrições* C, tal que C possa ser particionado em dois subconjuntos, i.e. $C = C_{in} \cup C_{out}$. Cada restrição $c \in C$ é representada por um predicado π_c tal que seu problema de decisão associado $\Pi_c(y)$ possa ser resolvido em tempo polinomial para todo $y \in sol_P(x)$. Todo predicado π_c é uma função computável em tempo polinomial que tem como entrada a matriz Booleana D e a instância x. É necessário que *no mínimo um* predicado π_c^* seja uma função não constante de *no mínimo dois elementos diferentes de* S_{par}.
  - Retorna no mínimo um *descendente*, quer dizer uma nova solução factível $y' \in \mathcal{M}_P(S_{par}, x)$ com um valor estritamente melhor de função objetivo m_P, (i.e. $m_P(y', x) < \max\{m_P(y_1, x),\ m_P(y_2, x),\ \ldots,\ m_P(y_k, x)\}$ para um problema de minimização, e $m_P(y', x) > \min\{m_P(y_1, x),\ m_P(y_2, x),\ \ldots,\ m_P(y_k, x)\}$ para um problema de maximização) sujeito a

$$\max_{y' \in sol_P(x)} [(\sum_{(c \in C_{in}) \wedge \Pi_c(y')} w_c) - (\sum_{(c \in C_{out}) \wedge \Pi_c(y')} w_c)] \qquad (7.1)$$

    onde w_c é um peso inteiro associado à restrição c.

Para continuar com a analogia com a classe PLS, podemos identificar o conjunto de potencial dinástico $\mathcal{M}_P(S_{par}, x)$ como uma *"vizinhança de grupo"*, quer dizer, uma extensão do conceito de uma vizinhança mas não para uma única solução, mas para um conjunto de (*no mínimo duas*) soluções factíveis. Acreditamos que aqui podemos esboçar uma apropriada fronteira com o conceito de *macromutação*, o qual, não importando quão complicado ele possa ser, incorpora a idéia de *solução única*, e conseqüentemente uma vizinhança "local". Algoritmos evolutivos que só melhoram os indivíduos através de macromutação, por mais complicados que sejam, podem ser classificados como apenas outras formas de busca local randômicas. Também observamos que a definição de uPMA é de fato o que é mais próximo à definição de PLS. Ela na verdade permite o conjunto $\mathcal{M}_P(S_{par}, x)$ ser arbitrariamente definido para cada entrada diferente S_{par}.

Nos referimos a um algoritmo *k-merger* (não necessariamente um de tempo polinomial) como um *operador de recombinação*. Não vemos qualquer vantagem em evitar o uso de vocabulário de algoritmos genéticos sempre que nós nos referimos a algoritmos ou objetos semelhantes. O uso da palavra 'recombinação' parece particularmente satisfatório para um algoritmo *k-merger* que satisfaz os requisitos da classe PMA. O conjunto $\mathcal{M}_P(S_{par}, x)$ de "descendentes potenciais", pode ser visto como uma extensão do conceito de vizinhança, e por isso algoritmos que só confiam em recombinação e mutação (como GAs clássicos) foram considerados por alguns autores como membros da família de busca local. Porém, para uma maioria grande de pesquisadores, esta é certamente uma classificação discutível e nossa definição ajudaria a ordenar este assunto.

A definição da classe PMA é necessariamente mais complicada que a uPMA, embora a única diferença resida na definição do algoritmo *k-merger*. O caso PMA tenta formalizar muitos operadores de recombinação usados em Computação Evolutiva. A condição "sujeito a" (veja Eq. 7.1) apresentada pela classe PMA pode ser, na realidade, uma descrição formal de alguns algoritmos muito simples, usados na prática.

Para trabalhar em nosso exemplo preferido, se estamos resolvendo um problema de MIN TSP, então o conjunto S_{par} é um conjunto de k soluções factíveis, tours nesse caso, e $D_{i,j} = \Pi_j(y_i)$ pode ser uma matriz booleana tal que $D_{i,j} = true$ indica que a solução y_i inclui a aresta e_j. Então $1 \leq j \leq n(n-1)/2$, com n indicando o número de cidades de x, a instância TSP. Com x e D computamos um certo conjunto de restrições $C = C_{in} \cup C_{out}$. Uma suposição razoável é que C_{in} deva levar em conta o conjunto de arestas que são comuns a todos os tours em S_{par}. Assim, o C_{out} pode representar o conjunto de arestas do pior tour em S_{par} que não estão em C_{in}. Quer dizer, estamos heuristicamente culpando-os de serem responsáveis pela qualidade ruim daquele tour. Se supomos $w_c = 1$ para todo c, estamos basicamente exigindo encontrar um novo tour, melhor que o pior tour em S_{par}, tal que deva maximizar o número de arestas na intersecção menos o número de arestas em C_{out}. Como nós veremos na última seção, estes tipos de problemas de otimização podem ser realmente muito difíceis (com um difícil grau de aproximabilidade), e uma boa idéia seria achar casos especiais tratáveis para desenvolver algoritmos de recombinação

eficientes. Isto, às vezes é possível. Assim este amplo tipo de complexidades sugere que encontrar resultados de tratabilidade e intractabilidade que surgem da classe PMA provê uma excitante nova área de pesquisa.

7.2.1 Recombinação, cegueira e miopia

O *k-merger* é dito ser *cego* se não usa qualquer informação *da instância de entrada* x do problema P. Sua entrada é somente o conjunto S_{par}, e no máximo, o conjunto de valores da *função-guia* associada a cada elemento do conjunto. A função-guia em MAs pode ser a função objetivo (para problemas de otimização) como também qualquer outra função de mérito escolhida *ad-hoc* para guiar o processo de busca.

Um algoritmo 2-*merger* é um *operador de crossover* se é cego, e se sua complexidade computacional não é "alta", i.e. *no máximo* $O(|N|\log(|N|))$ onde $|N|$ representa uma medida do tamanho da entrada (relembre que um algoritmo *k-merger* tem como entrada o conjunto S_{par} e a instância x). Esta definição que nós adotamos para "crossover" é completamente arbitrária, e aqui somente refletimos o uso geral dos termos da literatura de Computação Evolutiva. Exemplos de operadores de "crossover" são os bem conhecidos *"one-point-crossover"* o "crossover" de um ponto, *"two-point crossover"* o "crossover" de dois pontos, ou o *"generalized N-point crossover"* o "crossover" generalizado de N pontos [323].

Dizemos que a recombinação é *míope* se não usa *toda* a informação de x. Isto parece ser vago, mas o conceito de *miopia* será importante para a exposição posterior de alguns problemas abertos. Por exemplo, dados dois ciclos hamiltonianos de um grafo completo com pesos nas arestas, é fácil de imaginar muitos modos possíveis de se criar outro ciclo por um processo de dois-passos. Por exemplo, primeiro um procedimento guloso que ordena as arestas de ambos os ciclos em ordem não decrescente e os adiciona evitando criar um subtour ou deixar um vértice com grau três. Se o conjunto de arestas em ambos os pais termina antes que um ciclo completo seja formado, os pontos finais dos substrings resultantes são conectados ao acaso. A única informação usada de x é o peso das arestas de ambos os pais e, dado que o problema P supõe que você tem como instância o grafo *completo*, esta recombinação pode ser chamada de 'míope'.

7.2.2 Generalidade da recombinação em MAs

Procedimentos de recombinação em MAs têm várias diferenças com outras metodologias baseadas no uso de múltiplos pais. Tentando formalizar GAs, em [321] um operador de recombinação foi definido como sendo *puro* se dado como entrada um conjunto de $k \geq 2$ soluções *idênticas* ele retorna a mesma solução que foi dada como entrada. A partir das definições acima, no caso de todos os pais serem idênticos, o algoritmo *k-merger* ou declararia o conjunto S_{par} ser um ótimo *k-merger ou daria uma solução estritamente melhor*. Também em MAs *não restringimos* a recombinação a satisfazer $S_{par} \subseteq \mathcal{M}_P(S_{par}, x)$.

Uma diferença importante com a maioria dos algoritmos inspirados na natureza, é que não restringimos o operador de recombinação a ser uma função simétrica de seus argumentos. Dadas duas soluções-pai $y_1, y_2 \in sol_P(x)$, é possível que $\mathcal{M}_P(y_1, y_2, x) \neq \mathcal{M}_P(y_2, y_1, x)$. Um algoritmo *k-merger* será chamado *pai-simétrico* se o conjunto $\mathcal{M}_P(S_{par}, x)$ é o mesmo para todas $k!$ permutações dos k pais de entrada. Quase todas as implementações de GAs usam recombinação pai-simétrica. Para MAs, como também para as classes PMA e uPMA, esta é uma restrição desnecessária.

Para métodos inspirados biologicamente, em geral os *descendentes* de um algoritmo *2-merger* são *mais similares* aos pais (assumindo uma apropriada medida de distância d) que os dois pais. Formalmente,

$$\max\{d(y_1, y'), d(y_2, y')\} \leq d(y_1, y_2) \quad \forall y' \in \mathcal{M}_P(y_1, y_2, x). \tag{7.2}$$

Também neste caso, esta restrição é desnecessária e limita a recombinação, e como uma conseqüência não a necessitamos em MAs. Uma restrição fraca, com *'min'* ao invés de *'max'* também não é necessária.

7.3 Problemas em PMA

É bastante evidente como deduzir que a classe PMA não é vazia. Pela brevidade, deixamos ao leitor a análise do problema de árvore geradora mínima e como uma idéia baseada na definição de PMA e uPMA pode substituir estratégias baseadas em Busca Local. Por razões de clareza, ilustraremos com o seguinte problema de tempo polinomial, mas que ainda é um desafio devido ao grande tamanho de instâncias que ocorrem na prática.

7.3.1 Emparelhamento de máxima cardinalidade

Começaremos revendo algumas definições básicas. Um *emparelhamento* de um grafo não direcionado $G(V, E)$ é um subconjunto de arestas $M \subseteq E$ tais que não existem duas arestas que toquem um vértice em comum. Um vértice $v \in V$ é dito ser *não emparelhado* com respeito ao emparelhamento M se nenhuma aresta de M toca v. Um *emparelhamento perfeito* é um emparelhamento que contém uma aresta incidente em todo vértice de G. Um *emparelhamento maximal* é um emparelhamento que não está contido em nenhum outro emparelhamento. Um *emparelhamento de máxima cardinalidade* é um emparelhamento com um número máximo de arestas. Um *caminho alternado* em G com respeito ao emparelhamento M é um caminho cujas arestas estão alternadamente dentro e fora de M. Um *caminho aumentante* é um grafo alternado que começa e termina em um vértice não emparelhado.

Para mostrar que temos os três algoritmos de tempo polinomial necessários, precisamos provar alguns proposições notavelmente óbvias. Porém, as consideramos ilustrativos do que geralmente é necessário.

Proposição: *Existe um algoritmo de tempo polinomial que, dado um grafo não direcionado $G(V, E)$ ele retorna um conjunto de $p = |E|$ emparelhamentos*

de G que diferem de no mínimo uma aresta e tal que a união dos conjuntos de arestas em todos estes emparelhamentos cobrem E.

Prova: Seja $G_i = G - \{e_i\}$ para $i = 1, \ldots, |E|$. Um emparelhamento máximo para G_i pode ser construído executando um algoritmo $O(n+m)$ (veja por exemplo [346]), no grafo $G_i' = G_i - E(e_i)$ (onde $E(e_i)$ é o conjunto de arestas adjacentes à aresta e_i em G). Chamando M_i' o emparelhamento de G_i'; então o emparelhamento para G_i é somente $M_i' \cup \{e_i\}$. Claramente, pela proposição associada ao conjunto G_i, todas as arestas em E pertencem no mínimo a um emparelhamento.

Embora trivial, somente queremos declarar um corolário para referência adicional.

Corolário: *Existe um algoritmo de tempo polinomial que, dado um grafo não direcionado $G(V, E)$ e um emparelhamento M de G, cria um conjunto de $|M|$ diferentes emparelhamentos de G que diferem de no mínimo uma aresta.*

A proposição e o corolário acima implicitamente sugerem diversos modos que permitem a criação de algoritmos p-starter (com p sendo tão grande quanto $|E|$). O caso do p'-evaluator é bem óbvio. Agora, para a recombinação podemos provar o seguinte.

Proposição: *Existe um algoritmo de tempo polinomial* 2-merger *tal que, dado dois emparelhamentos M_1 e M_2 de G com $M_1 \neq M_2$, e sem perda de generalidade $|M_1| \geq |M_2|$, o algoritmo produz um novo emparelhamento de G (denotado $M_{1,2}$) satisfazendo $|M_{1,2}| > |M_2|$ e $M_{1,2} \neq M_1$, ou pára (em tempo polinomial).*

Prova: Seja U um grafo formado pela união dos dois emparelhamentos de entrada M_1 e M_2. Então, cada aresta de U pertence ou a M_1 ou M_2 ou pertence a ambos. Cada vértice de U tem grau 1 ou 2. Então, cada componente conectado de U é um caminho ou mesmo um ciclo (com arestas alternadas vindas dos dois emparelhamentos). Os caminhos ímpares, começando e terminando com arestas em M_1 (em M_2), são ditos serem relativamente aumentados para M_2 (M_1, respectivamente). Se p é um caminho aumentante relativo a M_2, a substituição das arestas p pertencentes a M_2 com as p *não* pertencentes a M_2 (assim pertencendo a M_1) resulta em um novo emparelhamento denotado por $M_2 \oplus p$, o qual obviamente satisfaz $|M_2 \oplus p| = |M_2| + 1$. Então, segue que podemos construir um novo emparelhamento $|M_{1,2}| > |M_2|$ identificando todos estes caminhos aumentantes relativos a M_2 ou determinar que nenhum emparelhamento existe, e pode ser feito em tempo polinomial.

Corolário: *O algoritmo 2-merger descrito também satisfaz $|M_{1,2}| \geq |M_1|$.*

Argumentos extremamente simples. O desafio pode ser o seguinte:

Problema Aberto: *Encontrar um algoritmo correto e completo para o problema de emparelhamento de máxima cardinalidade em um grafo não direcionado que seja baseado em PMA e que desafia os melhores métodos seqüenciais e paralelos na prática.*

Pensamos que esta é uma interessante direção de pesquisa considerando os recentes resultados reportados por Kececioglu e Pecqueur em grafos esparsos [197]. Também fornece uma nova perspectiva para desenvolver metodologias paralelas, podendo ser distribuídas e tolerante a falhas, baseadas em PMA para este problema.

7.3.2 Tratabilidade a parâmetro fixo e PMA

Discutimos há pouco um problema para o qual já existem algoritmos de tempo polinomial. Existem problemas de otimização NP-difíceis, e triplas apropriadas de algoritmos, tal que os problemas associados também pertencem a PMA ? Acreditamos que os resultados de *Fixed Parameter Tractability* têm um papel central para responder diversas questões relacionadas. Como um exemplo, podemos discutir alguns problemas.

7.3.2.1 Recombinação de coberturas de vértices

Nosso primeiro exemplo é baseado no problema de Vertex Cover.

Proposição: *Dado um grafo $G(V, E)$ e duas diferentes coberturas de vértices minimais C_1 e C_2 de G, tal que $C_1 \not\subseteq C_2$ e $|C_2 \setminus C_1| \leq k$, e sem perda de generalidade $|C_1| \leq |C_2|$, existe um algoritmo de recombinação de tempo polinomial que pára ou fornece outra cobertura de vértices de G, denotado C_3 tal que $(C_1 \cap C_2) \subset C_3$ e $|C_3| < |C_1|$ se tal cobertura existe.*

Tal algoritmo pode ser apropriadamente nomeado como *recombinação respeitosa dinasticamente ótima de coberturas de vértices*. Dada a proposição acima temos o seguinte resultado:

Corolário: *O Problema* Min Vertex Cover *com a recombinação respeitosa dinásticamente ótima de cobertura de vértices proposta na proposição anterior e algoritmos apropriados do tipo* p-starter *e* p'-evaluator *pertencem a classe uPMA.*

O corolário segue a proposição, pois o primeiro é um resultado muito mais forte. Garante que podemos encontrar um cobertura que *preserva os vértices em comum* e retorna uma cobertura melhor que o melhor entre as duas coberturas, passadas como dado de entrada, em tempo polinomial; ou retorna como resposta que nenhuma existe. Tal cobertura obviamente é melhor que o pior. Se tal cobertura não existe, há ainda a possibilidade de melhorar o valor da pior, e o mesmo algoritmo de recombinação pode ser usado como um algoritmo *k-merger* como requerido pela definição de uPMA.

Prova: É bastante óbvio como criar um algoritmo *p-starter*, e desde que MIN VERTEX COVER é NP, o p'*-evaluator* é trivial. O que é interessante para o propósito de discussão, é destacar como um algoritmo eficiente *2-merger* pode ser criado. Dados como entrada duas coberturas de vértices minimais C_1 e C_2, com $C_1, C_2 \subseteq V$, tal que $|C_1| \leq |C_2|$, $|C_2 \setminus C_1| \leq k$ e $C_1 \neq C_2$, seja $S = C_1 \cap C_2$ e $G_R = G \setminus S$. A questão: *"Pode G_R ter uma cobertura de vértices com estritamente menos que $|C_1| - |S|$ vertices?"* pode ser resolvida no tempo de $O(|C_1| - |S|)$ conforme vários algoritmos FPT (onde k é assumido fixo na proposição). Se a resposta é afirmativa, seja C_R uma cobertura de vértices de $G_R(S)$ com menos de $|C_1| - |S|$ vértices, então $C_R \cup S$ tem $|C_R| + |S|$ vértices, e $|C_R| + |S| < |C_2|$. Assim se C_1 e C_2 são duas coberturas, o conjunto $C_R \cup S$ é a nova cobertura de vértices *"descendente"*, alcançável em tempo linear. Assim está completa a prova.

7.3.2.2 O problema "Feedback Vertex Set"

Discutimos agora o seguinte problema:

k-FEEDBACK VERTEX SET
Instância: Um grafo $G(V, E)$ e um inteiro k. **Parâmetro:** k.
Questão: Há um conjunto U de k vértices de G tal que cada ciclo de G passa por algum vértice de U ?

Dado que a decomposição de um grafo em ciclos pode ser feita em tempo polinomial, um algoritmo *2-merger* para o MIN FEEDBACK VERTEX SET pode ser criado como se segue: dados como entrada dois conjuntos de vértices U_1 e U_2, tal que $|U_1| \leq |U_2|$ e $U_1 \neq U_2$, com $U_1, U_2 \subset V$ de $G(V, E)$

- Identifique todos os vértices em $U_1 \cap U_2$ e coloque-os em uma solução descendente criada. Chame este conjunto de S. Identifique todos os ciclos que passam pelos vértices em S.
- Remova de G todas as arestas (v_i, v_j) tais que $v_i \in S$ ou $v_j \in S$ (ou ambos). Remova todos (os agora isolados) vértices em S. Chame o grafo restante $G_R(S)$.
- A questão: *"Pode $G_R(S)$ ter um conjunto de vértices feedback com menos que $|U_2| - |S|$ vértices ?"* pode ser resolvida em tempo $O(n)$ (veja [107], página 214).

Se a resposta é afirmativa, faça U_R ser um conjunto de vértices feedback de $G_R(S)$ com menos que $|U_2| - |S|$, então $U_R \cup S$ tem $|U_R| + |S|$ vértices, e $|U_R| + |S| < |U_2|$. Assim $U_R \cup S$ pode ser a solução *descendente*.

Destacamos novamente a conexão com a classe de problemas FPT . É estimado que em torno de metade dos problemas NP-difíceis na referência clássica de Garey e Johnson têm problemas parametrizados que estão em FPT. Veja [107] para uma lista compreensiva de problemas na classe FPT achados até 1997, como também a metodologia usada para as provas. Se esse é o caso,

os resultados de FPT podem ser facilmente adaptados para desenvolver algoritmos de *k-merger* que podem executar em tempo polinomial para uma ampla variedade de problemas NP-difícieis.

Por outro lado, visto que há muitos trabalhos que indicam bons desempenhos obtidos com busca local baseados em MAs que somente usam um pequeno número de agentes (tipicamente de 13 a 20), outra possibilidade surge para paralelização. Notamos que alguns computadores paralelos MIMD têm arquiteturas com 64 ou 128 processadores. Então o problema de paralelização dos algoritmos de FPT ganham um renovado interesse, dado que podemos pensar em agentes que podem usar mais de um processador e podem ter a oportunidade extra de paralelização do passo de recombinação.

7.3.3 Problemas $W[1]$-completos e a classe PMA

A seguir, focalizamos nossa atenção em outros problemas para os quais os problemas de decisão não são tratáveis a parâmetro fixo mas não obstante alguns problemas relacionados podem ser mostrados que pertenecem a uPMA. Relembrando que o problema parametrizado CLIQUE é $W[1]$-completo [107].

CLIQUE

Instância: Um grafo $G(V, E)$ e um inteiro $k > 0$. **Parâmetro** k.

Questão: Existe um conjunto $Q \subseteq V$, que forma um subgrafo completo de G, tal que é uma coleção de pares de vértices adjacentes de cardinalidade k ?

Tal conjunto Q é chamado de *clique*. Um clique Q é *maximal* se não existe outro clique Q' tal que $Q \neq Q'$ e $Q \subset Q'$. A versão de otimização MAX CLIQUE pode ser vista como a tarefa de encontrar um clique de máxima cardinalidade em um grafo.

Em 1996, Balas e Niehaus desenvolveram uma heurística que trabalha recursivamente encontrando um máximo clique em um subgrafo induzido pela união dos vértices de dois cliques. Eles mostram que esta operação (chamada MERGE) pode ser feita em tempo polinomial. A relevância deste método para metodologias baseadas em população foi logo reconhecida por Aggarwal, Orlin e Tai [3]. Eles confiam neste algoritmo para um mecanismo de recombinação "otimizada" para o problema MAX INDEPENDENT SET. Usando a idéia proposta por Balas e Niehaus como um algoritmo *2-merger*, podemos estabelecer o seguinte resultado:

Teorema: *O problema* MAX CLIQUE *junto com o algoritmo* k-merger *baseado na operação de MERGE de Balas e Niehaus está em uPMA.*

Prova: (esboço) É trivial construir um algoritmo *p-starter*. O *p'-evaluator* somente precisa computar a cardinalidade de cada clique p'. Precisamos mostrar que a operação MERGE é realmente um algoritmo 2-*merger* que satisfaz a definição da classe uPMA. O algoritmo recebe como entrada dois cliques Q_1 e Q_2 (que não são necessariamente disjuntos) e eficientemente, i.e. em tempo polinomial, encontra um máximo clique no grafo H, o grafo induzido

por $Q_1 \bigcup Q_2$. Sem perda de generalidade, vamos supor que $|Q_1| \leq |Q_2|$. Seja $H = G(Q_1 \cup Q_2)$ e $\overline{H}$ seu complemento. MERGE primeiro computa um emparelhamento máximo M no grafo bipartido $\overline{H}$ (tomamos $N_1 = Q_1$ e $N_2 = Q_2 \backslash Q_1$ como uma possível bipartição dos nós). Então a complexidade de encontrar M é $O(|N_1 \cup N_2|^{5/2})$. Se N_1 é perfeitamente emparelhado por M, então Q_2 é o clique de máxima cardinalidade em H. Caso contrário, um conjunto máximo independente em $\overline{H}$ pode ser identificado (usando os métodos de "labeling" padrões começando a partir dos vértices não emparelhados de N_1). O conjunto independente máximo resultante em $\overline{H}$ é então um clique de cardinalidade máxima em H, com maior cardinalidade que Q_2. Isto completa a prova.

Relembramos que é trivial estender ou usar o resultado prévio com o propósito de desenvolver um algoritmo *k-merger* com $k > 2$.

7.3.4 Problemas $W[2]$-completos

O problema MIN DOMINATING SET é outro membro da classe NP-difícil e a versão parametrizada pela cardinalidade do conjunto dominante, k-DOMINATING SET, é $W[2]$-completo [107].

Um *conjunto dominante de vértices* de um grafo $G(V, E)$ é um conjunto $D \subseteq V$ tal que, para todo $u \in V - D$, há um $v \in D$ tal que a aresta $(u, v) \in E$. Como no problema acima, estamos interessados em identificar se o problema de otimização da seguinte variação de MIN DOMINATING SET está em uPMA.

INDEPENDENT DOMINATING SET

Instância: Um grafo $G(V, E)$ e um inteiro $k > 0$.

Questão: Há um conjunto D' de k vértices $D' \subseteq V$ que é um conjunto independente (i.e. tal que nenhum par de vértices em D' estão unidos por uma aresta em E) e um conjunto dominante (i.e. para todo $u \in V - D'$ há um $v \in D'$ tal que $(u, v) \in E$) ?

Sabemos que INDEPENDENT DOMINATING SET também é $W[2]$-completo [107]. Veremos que ambos os problemas representariam uma função interessante por provar ser difícil de resolver ou provar que certos problemas de recombinação estão em uPMA (veja Seção 7.5).

7.3.5 Terceiro circuito hamiltoniano, melhor que o pior

Outro problema interessante surge no contexto de desenvolver Algoritmos Meméticos para o MIN TSP. Primeiro considere a seguinte situação: suponha que estamos interessados em resolver um problema MIN WEIGHTED HAMILTONIAN CYCLE em um grafo não direcionado com pesos $G(V, E, W)$, e fornecemos *dois* diferentes, mas não necessariamente disjuntos, circuitos hamiltonianos em G, C_1 e C_2 de comprimentos L_1 e L_2. Vamos supor, sem perda de generalidade, que $L_1 \leq L_2$. Qual é a complexidade computacional de encontrar um *terceiro* circuito hamiltoniano C_3 de comprimento $L_3 < L_2$ (obviamente satisfazendo, $C_1 \neq C_3$) ?

Notamos que muitos procedimentos de recombinação usados em MAs, em geral algoritmos 2-*merger*, são na realidade heurísticas *randomizadas* desenvolvidas para este problema básico. Os algoritmos geralmente nem podem ser chamados de algoritmos de aproximação, pois comumente nem mesmo têm uma garantia de desempenho. Eles geralmente só são uma heurística *ad-hoc* para o problema há pouco declarado, de fato um subproblema se visto de uma perspectiva mais ampla. Podemos indentificá-lo como o seguinte problema:

Problema Aberto: *Qual é a complexidade computacional de decidir se existe um terceiro, melhor que o pior Ciclo Hamiltoniano?*

Restringir o problema acima ao caso em que o grafo G é formado pela união dos dois ciclos, aparentemente não ajuda muito. Considerando que os dois ciclos não são disjuntos, nem mesmo podemos garantir (usando o teorema de Thomason [359]) a existência de um terceiro. Porém, notamos que dado que estamos interessados em um ciclo diferente de C_1, podemos usar a estratégia de remover da união de grafos $C_1 \cup C_2$, uma aresta $e_i \in (C_1 - C_2)$, e assim, *uma aresta a cada vez*, tentar encontrar um *segundo* ciclo hamiltoniano em $(C_1 \cup C_2) - \{e_i\}$ (nenhuma aresta de C_2 é removida).

Embora o problema possa ser NP-difícil, há boas esperanças de se achar algoritmos de aproximação para casos especiais. Em [27], os autores mostram como construir um esquema de aproximação em tempo linear para encontrar um segundo ciclo hamiltoniano em grafos *cúbicos*. Notamos que, enquanto os matemáticos estiveram aparentemente mais interessados em provar resultados para grafos cúbicos (Bazgan et al. também apresentam um resultado de não aproximabilidade para LONGEST CYCLE IN CUBIC HAMILTONIAN GRAPHS no mesmo artigo), queremos neste momento ressaltar a relevância de estudar grafos não direcionados de máximo grau quatro e grau mínimo dois. Isto, junto à definição da classe PMA, sugere outro problema:

Problema Aberto: *Existe um valor de k tal que, dado um grafo não direcionado com pesos $G_U \subset G$ formado pela união de k diferentes Ciclos Hamiltonianos de G, existe um algoritmo de tempo polinomial que encontra outro melhor que o pior Ciclo Hamiltoniano ?*

7.3.6 O caso do Problema do Caixeiro Viajante

Não esperamos que algumas destas questões sejam fáceis de serem resolvidas, devido a que parecem, pelo menos à primeira vista, relativamente análogas às dos *"existentially polytime theorems"* [61]. Resultados em grafos de grau máximo par parecem ser mais difíceis de conseguir que seus sósias de grau ímpar. Porém, tendo ou não uma prova de intratabilidade ou de não aproximabilidade, a falta de teoria não pára os pesquisadores "práticos" sagazes em dominar estes assuntos usando heurísticas apropriadas. Assim foi possível desenvolver MAs excelentes para o MIN TSP. Por exemplo, em recentes estudos computacionais no MIN TSP, Merz e Freisleben apresentam resultados notáveis com o *"Distance Preserving Crossover"*(DPX). Dadas duas soluções

diferentes, mas não necessariamente disjuntas, i.e. dois tours, T_1 e T_2, o DPX cria um novo tour T tal que todas as arestas em comum (o conjunto $S = T_1 \cap T_2$) está em T, e tenta *evitar todas as arestas que não são comuns* ($(T_1 \cup T_2) - S$). Quando o procedimento tem êxito, o novo tour tem $|T| - |S|$ arestas diferentes ao mesmo tempo de T_1 e T_2 sugerindo o nome. O método provou ser altamente efetivo, particularmente quando um algoritmo baseado na heurística de Lin-Kernighan foi usado para a busca local. [257]

Resolver na otimalidade instâncias do MIN TSP de centenas de cidades em segundos em um PC é obviamente muito importante (particularmente porque os MAs são altamente paralelizáveis, e perfeitamente adequados para um conjunto heterogêneo de computadores). Com a esperança de que esses métodos crescentemente adquirirão (da parte dos teóricos) a atenção que merecem (particularmente em comparação com outros métodos estocásticos), mostramos outro interessante assunto de pesquisa. Ainda não está claro como podemos predizer *"a priori"* a sinergia entre uma busca local e um operador de recombinação. Já em [270] discutimos o uso de medidas de correlação estatística para ajudar a entender e controlar estes aspectos. Por exemplo, o DPX trabalha bem com o algoritmo de Lin-Kernighan, mas outras recombinações dão melhores resultados quando unidas a métodos de Busca Local menos poderosos. Uma regra boa é que a "vizinhança de grupo" $\mathcal{M}$ de um operador de recombinação e a vizinhança $\mathcal{N}$ da busca local associada deveriam ser, *na média*, significativamente *díspares* tal que *a priori* pudéssemos esperar que uma boa sinergia ocorresse. Como um exemplo, notamos que já é conhecido que dois ciclos Hamiltonianos de arestas disjuntas de um grafo G não podem ter vértices vizinhos na casca convexa dos circuitos hamiltonianos de G[371]. Mais pesquisa nesta direção é necessária. Foi demonstrado recentemente que o algoritmo de Thomason para encontrar um segundo circuito hamiltoniano que passe por uma determinada aresta em um grafo cúbico pode levar tempo exponencial. Remarcamos a necessidade em focalizar o estudo em grafos formados pela união de $k \geq 2$ circuitos hamiltonianos, como sugerida pela classes uPMA e PMA. Esquemas de aproximação em tempo linear também serão muito bem-vindos.

Notamos que aqui não comentamos, de propósito, um *único* problema aberto. É de fato uma área de pesquisa inteira que identificamos neste momento. Pensamos que provando certos resultados de intratabilidade, como por exemplo provar que certos pares de problemas/algoritmos são PMA-completos, embora um desafio interessante, poderiam não ser totalmente pertinentes. De uma perspectiva prática, este assunto tem certa analogia com o fato de provar que o algoritmo *k-opt* para o MIN TSP toma tempo exponencial, isto não necessariamente significa que não pode ser a base fundamental de bons MAs. De fato, provar resultados de tratabilidade (provando que um conjunto de problemas/algoritmos estejam em uPMA) deve nos dar melhores ferramentas para atacar problemas NP-difíceis, particularmente se para o mesmo problema também existem algumas provas relacionadas à classe PLS. Algoritmos associados a resultados de aproximabilidade podem ser particularmente relevantes e fornecerem interessantes operadores de recombinação. Resultados mais fortes para problemas de otimização NP *polinomialmente limitados* podem ser de interesse.

7.4 Problemas NP não Aproximáveis e PMA

Temos um *Esquema de Aproximação de Tempo Polinomial Eficiente* (*eptas*) se um algoritmo computa uma solução dentro de um fator $(1+\epsilon)$ do ótimo em tempo $O(f(\epsilon)n^c)$ para alguma constante c. Para um problema de maximização (resp. minimização) P, a linguagem induzida $L_P(param)$ é a linguagem parametrizada que consiste de todos os pares (x,k) onde $x \in I_P$ e $OPT(x) \geq k$ (resp. $OPT(x) \leq k$). Por exemplo, o MIN EUCLIDEAN 2-DIM TSP é um caso distinto de um problema conhecido que está em PTAS, mas não é conhecido se pertence a EPTAS. É atribuído a C. Bazgan (1995) (independentemente provado por Cesati e Trevisan em [65]) este resultado: *"Se $P \in EPTAS$, então $L_P(param) \in FPT$"*. Como uma conseqüência, se uma linguagem parametrizada $L_P(param)$ é difícil para $W[1]$, então o problema P não pode ter um *eptas* a menos que $FPT = W[1]$.

Visto que muitos algoritmos de recombinação tipicamente começam por selecionar algumas *características* para estarem no descendente, é relevante estudar como as *"restrições"* (relembre sua importância em relação à definição de PMA) devem afetar a aproximabilidade. Quão "distante" está essa fronteira ? Para ilustrar a discussão, vamos considerar o seguinte problema:

CONSTRAINED MAX VERTEX COVER
Entrada: Grafo não direcionado $G = (V, E)$, $S \subseteq V$, e uma cobertura de vértices de tamanho k.
Saída: O máximo, sobre todas as coberturas de vértices C de tamanho k, de $|C \cap S|$.

Este, como também outros problemas nesta seção, foram provados serem não aproximáveis por D. Zuckerman em [387]. Então a partir de um problema de otimização facilmente aproximável como MIN VERTEX COVER, com uma linguagem parametrizada em FPT, há um *"pequeno salto"* (usando as próprias palavras de Zuckerman), para uma versão não aproximável.

Notamos, porém, que desde que esta não aproximabilidade contém o pior caso (i.e. dado *alguma* eleição de um conjunto S), o desafio então é encontrar casos especiais tratáveis para embasar algoritmos de recombinação a partir deles. A pergunta subjacente então é: qual tipo de informação pode conduzir a um algoritmo eficiente e poder explorar esta característica ? Por exemplo, podemos desenvolver um algoritmo *2-merger* análogo à MIN FEEDBACK VERTEX SET que discutimos antes. O conjunto S pode ser o conjunto de vértices pertencentes a *ambos* os pais.

Outros dois problemas com restrições vêm de um cenário já discutido

CONSTRAINED MAX DIRECTED (UNDIRECTED) HAMILTONIAN CIRCUIT
Entrada: Grafo direcionado (não direcionado) $G(V, E)$, $S \subseteq E$, e um circuito Hamiltoniano em G.
Saída: O máximo, em circuitos Hamiltonianos $C \subseteq E$, de $|C \cap S|$.

então se certas arestas (arcos, no caso direcionado) são considerados "interessantes", pode ser o caso que o problema associado de encontrar uma solução

"estruturalmente ótima" pode ser muito complexo de fato. Recorde que este é o caso quando a função objetivo tenta maximizar o número de restrições presentes na solução. Dito isto, embora este resultado seja interessante como um pior caso, não sombreia nossas esperanças completamente na classe de PMA. Porém, note que isto diretamente conduz a um resultado de não aproximabilidade para a tarefa de criar um operador de recombinação que maximiza o conjunto de arestas "interessantes" da união das arestas de um conjunto S_{par} de tours disjuntos. Esta tarefa naturalmente surge em muitos MAs para o MIN TSP e é ainda válida quando buscamos um operador de recombinação cego.

O mesmo pode ser dito para outro problema no qual MAs provam ser bons na prática, devido ao impacto de MAs para problemas de tabelas de horário, (veja [52], e também [51] e referências neles contidas). A versão restrita de coloramento de grafos é:

CONSTRAINED MAX CHROMATIC NUMBER

Entrada: Grafo $G = (V, E), v \in V, S \subseteq V$, e um k-coloramento de G.

Saída: O máximo, em todos os k-coloramentos $\mathcal{C}$ de G, de $|C \cap S|$, onde C é o conjunto de vértices na mesma classe de cores como v em $\mathcal{C}$.

Neste ponto, podemos reconhecer que estas linguagens/problemas têm uma estrutura em comum. Fazemos isso recordando a representação básica de uma linguagem L de NP, como $x \in L$ se e somente se, $(\exists y \in \{0,1\}^{p(|x|)})\pi(x, y)$ onde π denota um predicado de tempo polinomial e $p(|x|)$ é um polinômio no tamanho de x. Analogamente para o problema de maximização P temos $(x, k) \in L$ se e somente se, $(\exists y, m_P(y, x) \geq k)\pi(x, y)$ (enquanto para o problema de minimização verificamos $m_P(y, x) \leq k$).

Para a nova versão restrita de maximização discutida aqui, notamos que as restrições podem ser expressas usando a correspondência natural de $\{0,1\}^{p(|x|)}$ e subconjuntos de $\{1, \ldots, p(|x|)\}$. Denotando como uma entrada adicional o conjunto $S \subseteq \{1, \ldots, p(|x|)\}$, nos permite ver y como um subconjunto de $\{1, \ldots, p(|x|)\}$. Os problemas de maximização restritos acima correspondem à maximização de $|S \cap y|$ em y tal que $\pi(x, y)$, dado um tal y. Concluímos a seção com uma lista de outros problemas restritos e desafios abertos:

CONSTRAINED MAX CUT

Entrada: Grafo não direcionado $G = (V, E), v \in V, T \subseteq V$, e um corte de peso no mínimo W com respeito à função peso $w : V \to Z$ (i.e. um conjunto $S \subseteq V$ tal que

$$\sum_{\{u,v\} \in E, u \in S, v \notin S} w(\{u, v\} \geq W) \tag{7.3}$$

Saída: O máximo, em cortes C de peso no mínimo W com $v \in C$, de $|C \cap T|$.

CONSTRAINED MAX CLIQUE COVER

Entrada: Grafo $G = (V, E), v_0 \in V, S \subseteq V$, e um clique cover de G de tamanho no máximo de k, i.e. uma representação de G como a união de no máximo k cliques.

Saída: O máximo, de todos os cliques covers $\mathcal{C}$ de G de tamanho no máximo k, de $|C \cap S|$, onde C é um clique contendo v em $\mathcal{C}$.

Constrained Max Steiner Tree

Entrada: Grafo não direcionado $G = (V, E), S \subseteq E$, e uma árvore de Steiner de peso no máximo de k com respeito a $R \subseteq V$, função de peso $w : E \to Z$, (i.e. uma sub-árvore de peso no máximo k contendo o conjunto de nós em R).

Saída: O máximo, dos árvores de Steiner $T \subseteq E$ de peso no máximo de k com respeito a R e w, de $|T \cap S|$.

Constrained Max Three-Dimensional Matching

Entrada: Hipergrafo $H = (V, F), F \subseteq V \times V \times V, S \subseteq F$, e um emparelhamento 3-dimensional (i.e. $M \subseteq F, |M| = |V|$ sem que dois elementos de M concordem em nenhuma coordenada).

Saída: O máximo, em emparelhamentos 3-dimensionais N, de $|N \cap S|$.

Constrained Max Feedback Node (Arc) Set

Entrada: Dígrafo $G = (V, A)$, $S \subseteq V$ (respectivamente, $S \subseteq A$), e um conjunto de nós (respectivamente, arcos) *feedback* de tamanho k (i.e. um subconjunto $R \subseteq V$ (respectivamente, $R \subseteq A$) de tamanho k que contém um vértice (respectivamente, arco) de todo ciclo direcionado).

Saída: O máximo, sob o conjunto de nós *feedback* (respectivamente, arcos) C de tamanho k, de $|C \cap S|$.

Problemas Abertos: *Para cada um dos problemas restritos, há algum caso especial de tratabilidade de parâmetro fixo tal que podemos projetar algoritmos polinomiais de fusão para os problemas originais de otimização NP?*

O excelente resultado reportado por Merz and Freisleben no problema Min Graph Bi-Parititioning [253], como também a implementação de Ozcan e Mohan de MAs para *emparelhamento parcial de formas* em uma aplicação de visão de computadores [302], sugerem que há uma boa motivação para expandir a lista de problemas discutida aqui.

7.5 Na direção de uma definição para a redução em uPMA

O que faremos nesta seção é apresentar um exemplo que visa motivar a pesquisa nesta direção (conduzindo possivelmente à definição de uma redução uPMA e o estabelecimento de problemas uPMA-completos).

De acordo com Davies e Russell [94], o k-Feature Set é o seguinte problema de decisão:

- **Entrada**: Dado um conjunto X de exemplos (onde cada exemplo é composto de um valor binário especificando o valor da *característica alvo* e

um vetor de n valores binários especificando os valores de outras características) e um inteiro positivo $k > 0$.

- **Questão**: Existe um conjunto S de k características *não-alvo* tal que:
  - $S \subseteq \{1, \cdots, n\}$,
  - $|S| = k$,
  - não existem dois exemplos em X que têm valores idênticos para todas as características em S e tem valores diferentes para a característica alvo ?

Este problema de decisão pode ser convenientemente reformulado como segue:

k-FEATURE SET

Instância: Um conjunto de m exemplos $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$, tal que para todo i, $x^{(i)} = \{x_1^{(i)}, x_2^{(i)}, \ldots, x_n^{(i)}, t^{(i)}\} \in \{0,1\}^{n+1}$, e um inteiro $k > 0$.

Questão: Existe um *conjunto de características* S, $S \subseteq \{1, \cdots, n\}$, com $|S| = k$ tal que, para todos os pares de exemplos $i \neq j$, se $t^{(i)} \neq t^{(j)}$, existe $l \in S$ tal que $x_l^{(i)} \neq x_l^{(j)}$?

O problema ONE-OUT k-FEATURE SET (OO-k-FSP) é um caso especial do problema k-FEATURE SET tal que o vetor coluna Booleano $T = \{t^{(1)}, t^{(2)}, \cdots, t^{(m)}\}$ (i.e., a coluna mais á direita de X) tem todos os valores idênticos em todas as posições com exceção de uma. Assim, o OO-k-FSP modela as situações em que temos $m-1$ exemplos que representam uma classe "boa" (ou "atrativa") e um exemplo a partir de uma classe "ruim" (ou "repelente") ou *vice versa*.

Sem perda de generalidade, assumimos que T é da forma $T = \{1, 1, \cdots, 1, 0\}$. Além disso, usualmente achamos conveniente considerar instâncias OO-k-FSP em sua forma "canônica": seja $\langle X, k \rangle$ uma instância de OO-k-FSP. A forma canônica $\langle Y, k \rangle$ é construída a partir de I fazendo $y_j^{(i)} = x_j^{(i)} \mathrm{XOR} \mathrm{x}_\mathrm{j}^{(\mathrm{m})}$ para $1 \leq i \leq m$, e $1 \leq j \leq n$. Note que esta transformação não modifica o conjunto de soluções da instância (o poder distintivo de uma característica se encontra na mudança de seus valores com o conjunto de exemplos, em lugar de seu valor atual em um exemplo específico), mas tem o efeito de inverter as colunas nas quais um 1 estava na última fila (o repelente), fazendo assim ser todos 0s.

Notamos que o ONE-OUT k-FEATURE SET também é NP-completo como segue da redução a partir de VERTEX COVER usada em [94], dado que para provar que k-FEATURE SET é NP-difícil eles realmente geraram uma instância de OO-k-FSP.

Tendo provado que o OO-k-FSP é NP-completo estamos particularmente interessados em ver se podemos construir um algoritmo 2-*merger* para o problema de otimização, ou se podemos provar ser um membro do uPMA. É claro que o problema aberto que temos em mente é o seguinte:

Problema Aberto: *Dada uma instância do OO-k-FSP e dois conjuntos de características S_1 e S_2 com $S_1 \not\subseteq S_2$ e $|S_1| \leq |S_2|$, é possível encontrar em*

tempo polinomial um outro conjunto de características $S_3 \subset (S_1 \cup S_2)$ *tal que* $S_3 \neq S_1, S_2$ *e* $|S_3| < |S_2|$ *?*

Como foi bem colocado no Capítulo 3 de [221] (p. 49), há basicamente três tipos de redução: a) reduções que provam que um problema é fácil, b) reduções que provam que um problema é difícil, c) *reduções que não provam nada.* O que queremos discutir é uma redução deste último grupo. O que Johnson e Papadimitriou enfatizaram com seu comentário (*"não provam nada"*) é que reduzimos um problema A de complexidade desconhecida a um problema B que é, ou suspeita-se ser, difícil.

Vamos supor que temos uma instância do OO-k-FSP, onde temos que substituir o valor '1' para "atrativo" e '0' para "repelente", e que também fornecemos como entrada dois conjuntos de características $S_1 = \{1, 3, 5\}$ e $S_2 = \{2, 3, 5\}$.

0	0	1	0	1	atrativo
1	0	0	1	1	atrativo
0	1	0	1	1	atrativo
1	1	0	0	0	atrativo
0	1	1	0	0	atrativo
0	1	1	1	0	atrativo
0	0	0	0	0	repelente

Figura 7.1: *Uma instância do problema* One-Out k-Feature Set *na forma canônica.*

Convidamos o leitor a verificar se de fato S_1 e S_2 são duas soluções factíveis do problema (tendo três características cada uma). *É possível usar esta informação para construir um outro conjunto de características com somente duas características ?* Esta é uma instância de um problema de complexidade desconhecida, entretanto uma inspenção visual pode facilmente encontrar a resposta nesse caso.

Agora construiremos um grafo G como o da Figura 7.2. É recomendável entender o processo de construção separando os vértices em grupos diferentes.

- **Centrômero:** Começamos com um único vértice que nomearemos, não sem certa provocação implícita para a imaginação, *centrômero*. Outro nome apropriado seria *vértice polar* dado que está no topo do grafo da Fig. 7.2.

- Q **group:** O centrômero está conectado a cinco vértices que no desenho estão cercados com uma linha pontilhada, eles são o Q group. As linhas não são esboçadas em benefício da clareza. Há um vértice para cada característica e, visto que a matriz booleana de entrada tem cinco colunas (mais uma para o objetivo, é claro), temos cinco vértices. O vértice no topo de Q representa a característica 1 (coluna 1 da matriz booleana de entrada) e então, procedendo à esquerda, o próximo representa característica 2, e assim por diante até a característica 5. Chamamos

este subconjunto de vértices como Q, desde que eles formam um clique (quer dizer, eles estão todos conectados um ao outro). Novamente, não desenhamos nenhuma extremidade entre os pares de vértices em Q, para fazer o desenho mais compreensível.

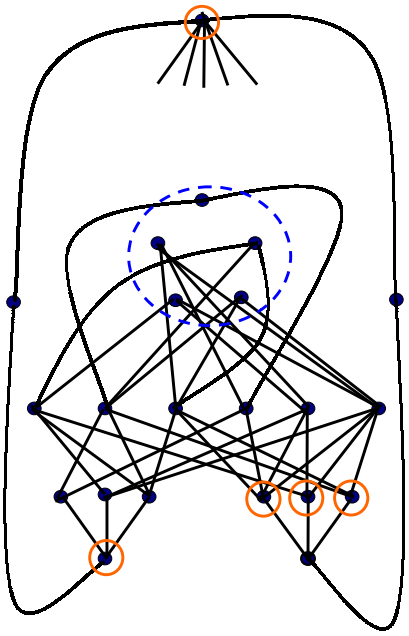

Figura 7.2: *Codificando a solução factível $S_2 = \{2, 3, 4\}$ da instância do problema OO-k-FSP da Fig. 7.1 como um conjunto independente dominante num grafo.*

- **Grupo X:** Os vértices em Q são conectados aos vértices do grupo X. Os vértices em X são um conjunto independente de G (i.e. nenhum par de vértices de X compartilham uma aresta). Cada vértice em X representa um exemplo, então há seis deles na Fig. 7.2 (organizados horizontalmente). No desenho, o vértice mais à direita corresponde ao exemplo 1 (linha 1 da matriz de entrada), o próximo corresponde ao exemplo 2 (linha 2), e o mais à direita corresponde ao exemplo 6.
- **Grupos S_1 e S_2:** Como o nome indica, estes dois grupos, com três vértices cada um, neste caso, representam os conjuntos de características nas duas soluções factíveis. Eles são conectados aos vértices do grupo X . O grupo mais à direita é S_1, e os vértices correspondem às características $\{1, 3, 5\}$ nesta ordem (da esquerda à direita). O grupo mais à direita é S_2, e os vértices correspondem às características $\{2, 3, 4\}$ nessa ordem.
- **Guardas:** Abaixo dos grupos S_1 e S_2 há dois vértices que são chamados

de *guardas*. Então nesse caso temos um guarda S_1 e um guarda S_2. Eles são conectados a todos os vértices em S_1 e S_2, como indicado na Fig. 7.2.

- **Isoladores:** Os guardas S_1 e S_2 não estão conectados diretamente ao centrômero, existem dois nós que chamaremos de *isoladores* que evitam que sejam adjacentes.

Agora precisamos especificar o conjunto de arestas para cada grupo.

- O centrômero está conectado a todos os vértices em Q e também aos dois isoladores.

- Todos os vértices em Q estão conectados entre eles e estão conectados ao centrômero. Para cada vértice $v \in Q$ representando a característica $f_j \in (S_1 \cup S_2)$ há uma aresta no vértice $v' \in X$ representando o exemplo $x^{(i)}$ se, e somente se, $X_{ij} = 1$.

- Para cada vértice $v \in S_1$ representando uma característica $f_j \in S_1$, há uma aresta no vértice $v' \in X$ representando um exemplo $x^{(i)}$ se, e somente se, $X_{ij} = 1$, e fazemos analogamente para S_2.

- Todos os nós nos grupos S_1 e S_2 estão conectados aos seus respectivos guardas.

- Os guardas estão conectados aos seus respectivos isoladores, e os isoladores ao seu respectivo centrômero.

Agora, note o conjunto de cinco vértices cercados na Fig. 7.2. Todos os vértices em S_2 estão marcados como também os guardas para S_1 e o centrômero. Além disso, note que pelo procedimento de construção, este conjunto de cinco vértices é um *conjunto independente dominante* deste grafo. O centrômero, sendo marcado, domina os isoladores e todos os nós no grupo Q. O guarda S_1, sendo marcado, domina seu isolador e todos os vértices em S_1. Os vértices no grupo S_2, sendo todos eles marcados, dominam o guarda S_2, e mais importante, eles *dominam todos os vértices no grupo* X. Isto segue do fato de que se S_2 é uma solução factível da instância OO-k-MFS na forma canônica, então sabemos que para cada exemplo deveria haver uma entrada não-nula pelo menos para uma característica em S_2. Por construção, isto corresponde a uma aresta, então marcando todos os vértices em S_2 garantimos que todos os vértices do grupo X estão dominados.

É claro que este conjunto de vértices é independente desde que: a) os vértices de S_2 são um conjunto independente e são conectados somente aos vértices de X e aos seus guardas, b) o guarda S_1 está somente conectado aos vértices de S_1 e ao isolador, c) o centrômero está somente conectado ao isolador e aos vértices de Q.

A mesma linha de argumento pode ser usada para mostrar que, se marcamos todos os vértices em S_1, o centrômero e o guarda S_2, temos outro conjunto independente dominante de todo o grafo que codifica a solução factível S_1. Isto está mostrado na Fig 7.3

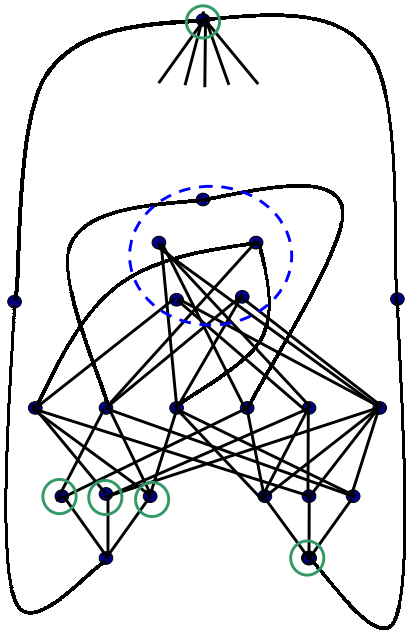

Figura 7.3: *Codificando a solução factível $S_1 = \{1, 3, 5\}$ da instância do problema OO-k-FSP da Fig. 7.1 como um conjunto independente dominante num grafo.*

Agora notaremos o seguinte: para a instância particular do OO-k-FSP que estamos considerando, existe uma resposta 'Sim', i.e. existe outro conjunto de características, $S_3 = \{2, 5\}$, tal que satisfaz as exigências.

Na Fig. 7.4 temos marcados os vértices em Q que correspondem às características 2 e 5. Se também marcamos *ambos* os guardas de S_1 e S_2, temos um conjunto dominante tendo um vértice a menos que os dois conjuntos independentes dominantes anteriores, das Figuras 7.2 e 7.3. Notamos, porém, que este novo conjunto dominante de quatro vértices não é independente, visto que dois vértices marcados em Q são, por construção, conectados.

Em essência, esta construção mostra que o nosso novo problema alvo é o seguinte (é suposto ser difícil):

Problema Aberto: *Dada uma instância do problema* DOMINATING SET *(i.e. um grafo $G(V, E)$), $V' \subset V$, e dois conjuntos independentes dominantes de G, ID_1 e ID_2 com $ID_1 \not\subseteq ID_2$ e $|ID_1| \leq |ID_2|$, é possível encontrar em tempo polinomial um conjunto dominante D_3, não necessariamente independente, tal que $(D_3 \cap V') = \emptyset$ e $|D_3| < |ID_2|$?*

onde neste caso $V' \subset (X \cup S_1 \cup S_2)$. Outro problema interessante pode ser

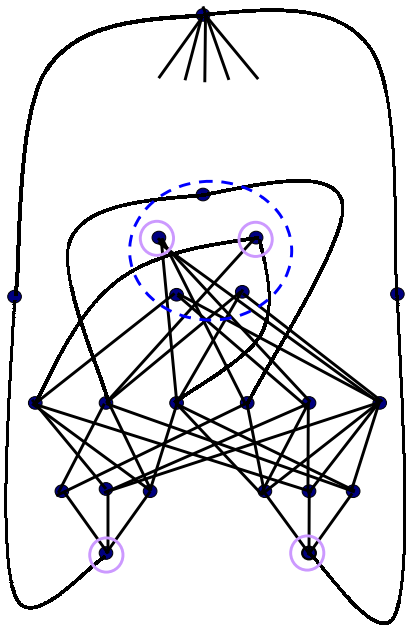

Figura 7.4: *No grafo é mostrado um conjunto dominante de vértices que representa a solução factível* $S_3 = \{2, 5\}$ *da instância do problema de OO-k-FSP da Fig. 7.1.*

estudar a versão não restrita onde $V' = \emptyset$.

Para resumir a discussão, pensamos que uma redução uPMA é realmente muito necessária para formalizar este assunto. Embora seja tentador, no momento nos limitamos em sugerir uma definição formal, embora mostramos que existe uma generalização direta da redução PLS. Além disso, parece interessante estabelecer uma classe para problemas parametrizados na qual o problema de recombinação de "encontrar uma solução melhor que a pior dos pais" está em FPT. Podemos chamar esta classe de p-uPMA, então deveríamos buscar uma apropriada "redução de uPMA parametrizado" (ou p-uPMA-redução) isso satisfaria a proposição abaixo:

Proposição: *Se* Π_1 *e* Π_2 *são problemas de recombinação parametrizados em p-uPMA tal que* Π_1, p-uPMA-reduz *a* Π_2 *e há um algoritmo FPT para encontrar uma outra solução factível melhor que a pior dos pais para* Π_2, *então há também um algoritmo FPT para encontrar uma solução factível melhor que a pior dos pais para* Π_1.

No momento, investigar essa possibilidade está fora de nosso alcance, mas é uma direção de pesquisa interessante, que deve ser analizada.

7.6 Discussões finais e conclusões

Apresentamos neste capítulo vários problemas abertos em Teoria de Grafos que são de natureza fundamental para o campo crescente de Computação Evolutiva. Também descobrimos áreas de pesquisa novas e tentamos estabelecer uma ponte entre o desenvolvimento de algoritmos de recombinação bons e algumas classes de complexidade computational associadas.

Também mostramos que um assunto já levantado por D. West em [371] é muito relacionado aos nossos interesses em desenvolver "potencialmente bons" ou eficientes operadores de recombinação para Algoritmos Meméticos. Em [371] o autor perguntou: *"Sob quais condições as arestas em um par de circuitos hamiltonianos podem ser redistribuídas e formar um outro par de circuitos com a mesma união e intersecção?"*, enquanto para alguns pares que cruzam em duas arestas não podem ser reorganizados daquele modo. Estas perguntas, já discutidas duas décadas atrás, podem ser a "ponta do iceberg" de um campo de pesquisa enorme: o de combinar eficientemente as soluções de problemas que aceitam certificados sucintos.

De acordo com a *Teoria dos Formae* é dito que um operador de recombinação é *respeitoso* se, e somente se, gera novas soluções tendo *todas as características comuns* aos pais. Também, um operador de recombinação é dito ser *"transmitting"* se, e somente se, todas as características dos descendentes gerados estão presentes em pelo menos um dos pais. Vamos considerar as arestas de um ciclo hamiltoniano como nossa característica básica. Neste caso, um algoritmo como o buscado por West, agindo em um par de tours, pode ser considerado como um operador particular de recombinação, que respeita e transmite as arestas. Ter ambas as propriedades não é geralmente fácil, dado que respeito não geralmente implica transmisão ou *vice versa* [82].

Em contraste, uma propriedade não relacionada com a questão de West é *"assortment"*, que podemos traduzir como 'variedade', e um operador de recombinação o tem se, e somente se, pode gerar descendentes tendo uma combinação arbitrária de características *compatíveis* dos pais. Novamente, *assortment* e respeito não são geralmente compatíveis ou podem ser deduzidos a partir do outro. Enquanto respeito representa a *intensificação* ou o comportamento "conciliador" da recombinação, assortment representa a *diversificação* ou o lado *explorativo* . Um bom balanço é, portanto, necessário, e a introdução do conhecimento do problema, como proposto pela abordagem memética é essencial para aumentar a velocidade do processo influenciando a busca. Também apoiamos a introdução de buscas mais sistemáticas em Algoritmos Meméticos tal como *Recombinação Dinástica Ótima* proposta em [82] ou buscas sistemáticas truncadas em $\mathcal{M}_P(S_{par}, x)$ (por exemplo, usando algum tipo de procedimento *Branch-and-Bound* truncado). Também damos boas-vindas ao desenvolvimento de MAs *completos*, que podem retornar uma solução com prova de otimalidade.

Pode ser possível que a análise de complexidade computational destas propriedades deva seguir abordagems diferentes. A tarefa de provar que uma determinada recombinação, relativa a algumas "características definidas", respeita e transmite produziriam alguns problemas de decisão interessantes. Para

assortment, de uma perspectiva baseada em projeto, pode ser o caso de ser mais pertinente adaptar os conceitos de *análise de dominância* [151] ou *análise comparativa* [77] para ajudar a comparar operadores de recombinação. Também parece que os conceitos de *"treewidth"* e *"branchwidth"* ([36] e as referências aí contidas) podem ter um papel central para criar eficientes procedimentos de recombinação em grafos esparsos.

Este capítulo visou fazer uma ponte entre problemas abertos teóricos, novos problemas em grafos que oferecem novos desafios para metaheurísticas paralelas, e uma abordagem altamente próspera na prática. Também sugere que as classes uPMA e PMA podem ter um importante papel como um *framework* sistemático para encontrar tratabilidade úteis de subproblemas surgindo no campo de algoritmos evolutivos. Esperamos que trabalho adicional nos ajudará a entender se a abordagem formal proposta aqui pode nos ajude a projetar algoritmos meméticos mais eficientes, em novas áreas de pesquisa.

Capítulo 8

Análise Evolutiva de Algoritmos ?

8.1 Introdução

A pergunta central que gostaríamos de colocar pode ser enunciada como:

> *"É possível desenvolver sistematicamente, com auxílio de um computador, métodos baseados em adversários com o objetivo de ajudar na análise de algoritmos e nas implementações concretas em código executável ?"*

Por que é esta uma indagação interessante ? Talvez seja possível, em lugar de criar operadores de recombinação e de busca local, delegar essa tarefa para algum sistema algorítmico de tipo evolutivo. Por outro lado, todos os que realizam trabalhos do tipo prático e os investigadores sabem, assim como é ensinado em todos os livros-texto, que a análise assintótica de algoritmos baseada no pior caso é provavelmente o paradigma mais próspero em computação. Porém, o processo de *"encontrar"* a instância de pior caso para o algoritmo particular em escrutínio é um processo bastante complexo, que muito depende de inspiração. Desnecessário dizer, é uma tarefa dura que também envolve habilidades matemáticas e um bom conhecimento das muitas ferramentas teóricas, desenvolvidas durante o último século.

Ilustraremos o uso de algoritmos evolutivos [22] para ajudar a tarefa de encontrar as instâncias do pior caso e apresentamos uma metodologia para este propósito. Nesse capítulo, ao falar de algoritmos evolutivos nos referimos a técnicas heurísticas de busca baseadas nos princípios de evolução natural ou cultural, isto é, pela adaptação e sobrevivência do mais apto. A partir de um conjunto de soluções do problema de interesse, um processo iterativo é executado incluindo seleção (propagação das "melhores soluções" de acordo com uma função de "aptidão" *ad-hoc* projetada pelo usuário), reprodução (geração de soluções novas através da recombinação e mutando soluções propagadas) e substituição (substituição das piores soluções do conjunto pela recentemente

criada). Quando um certo critério de parada é conhecido (normalmente alcançando um número fixo de avaliações da solução), a melhor solução encontrada é devolvida. Como já foi discutido nessa tese, os Algoritmos Meméticos e os Algoritmos Genéticos são exemplos de algoritmos de tipo evolutivo.

Nesse capítulo, propomos usar este tipo de heurística de busca para encontrar instâncias difíceis para algoritmos particulares. As instâncias encontradas deste modo proverão um limitante inferior da complexidade do pior caso. Não é difícil de imaginar situações em que alguém muito envolvido numa aplicação prática, enquanto desenvolve códigos e existem prazos finais apertados para término do trabalho, é obrigado a tomar uma decisão rápida dentre várias alternativas de algoritmos. Uma análise evolutiva preliminar pode ser suficiente para já descartar alguns algoritmos sem atrativo ou invalidar alguma hipótese de trabalho. Além disso, pode ser o caso que não se conhece uma distribuição de probabilidade de instâncias do problema e tudo o que é conhecido é o maior tamanho das instâncias que podem ser encontradas. Uma análise de pior caso restrita a estes cenários de "tamanho-finito" ainda faz sentido. A análise evolutiva proveria um limitante inferior útil na prática.

Para o teórico e o investigador em Ciências da Computação, a análise evolutiva pode ajudar como um tipo de *"musa"* de inspiração. Funcionaria como uma heurística válida para encontrar as instâncias "difíceis de resolver", que poderiam ser logo analisadas e poderiam então ser revisadas para desenvolver uma análise assintótica formal de pior caso, seguindo os métodos rigorosos de prova. Para qualquer problema de tamanho-finito, a análise evolutiva de algoritmos (EAA daqui em diante) proveria um limitante inferior útil da complexidade do pior caso.

Para encontrar uma base comum para uma análise inicial, na qual todos os profissionais de computação podem se relacionar facilmente, selecionamos *'Ordenação'*, um problema estudado a fundo do ponto de vista teórico e experimental. Ainda será mostrado que a análise da maioria dos algoritmos que se destacam para sua solução nos entrega um teste não-trivial que ajuda a estabelecer uma metodologia experimental. Em essência, nesse capítulo, estamos respondendo de um modo sistemático a desafios propostos por Sedgewick em [340] quando, relativo a um algoritmo de $O(n^{3/2})$ para *ShellSort* dizia:

> *"A prova desta propriedade está além do âmbito deste livro, mas o leitor não só pode apreciar sua dificuldade mas também estar convencido que Shellsort executará bem na prática tentando construir um arquivo para o qual Shellsort executa lentamente."*

8.2 Estudo de Caso: Algoritmos de Ordenação

Independentemente do desafio de Sedgewick, acreditamos que a análise de outros algoritmos de ordenação provê um bom *"benchmark"* para uma avaliação inicial da viabilidade da técnica proposta. Estudando a ordenação, beneficiamo-nos de um número grande de algoritmos para os quais uma análise teórica rigorosa está disponível. As conclusões obtidas depois da análise evolutiva podem ser facilmente checadas para otimalidade e ajudariam a entender

e sistematizar a metodologia de trabalho. Além disso, para algoritmos de ordenação, uma análise matemática elaborada é exigida para obter limitantes de complexidade. A comparação com a "quase" automática técnica evolutiva dará alguma luz na utilidade potencial deste método.

8.2.1 A base de experimentação escolhida

Os algoritmos de ordenação considerados neste trabalho são *BubbleSort*, *MergeSort*, *QuickSort*, and *ShellSort*. Em cada caso, um algoritmo genético padrão (o GA) ($PopSize = 100$, $p_c = 0.9$, $p_m = 1/n$, $maxevals = 100n^2$) usando seleção ordenada ($\eta = 2.0$) e o operador de mutação de troca [241] foi utilizado (onde n é o número de items a serem ordenados). Propositalmente, decidimos usar um esquema evolutivo muito simples para testar a viabilidade da idéia original. Nos referimos a [67] para a implementação detalhada de um GA. Nossa função de aptidão (i.e. a função objetivo nesse caso) é o número de operações elementares (comparações ou trocas) que os algoritmos fazem para encontrar a seqüência ordenada dada uma permutação inicial. Buscamos maximizar isto, pois corresponderá a uma instância de pior caso.

Uma instância do problema para um algoritmo de ordenação é uma seqüência de elementos a serem ordenados de acordo com um valor-chave. Sem perda de generalidade, esta seqüência é escolhida como uma permutação do conjunto de números naturais $\{1, 2, \cdots, n\}$. A tarefa do algoritmo genético será encontrar permutações difíceis para o algoritmo de ordenação considerado. Para isso, a análise evolutiva deveria identificar as características de uma seqüência que apresenta a maior "dificuldade relativa" para ordená-los usando o algoritmo investigado. Para tal propósito, usaremos uma técnica chamada *Forma Analysis* que provou ser de uso em Algoritmos Genéticos e Meméticos [81, 170, 322].

Uma *forma* (pl. *formae*) pode ser definida como um conjunto de soluções do problema que compartilham uma característica particular (nos referimos a [320] para uma definição formal) . Alternativamente, um forma pode ser vista como a própria característica. Um GA que tenha sucesso deveria ser projetado para gerar soluções novas manipulando os *"formae"* pertinentes [217]. Mais precisamente, e dado que a recombinação tem um papel chave na fase de reprodução de GAs, Análise de Forma nos dá uma boa dica de qual operador de recombinação deveria ser usado.

No caso de permutações, essencialmente existem quatro características diferentes que podem ser identificadas [84]:

1. relações de *precedência* entre elementos, i.e., quais elementos ocorrem antes/depois de um dado elemento. Por exemplo, as permutações 123456 e 654312 compartilham a característica comum 'o 1 ocorre antes do 2'.

2. relações de *adjacência* entre elementos, i.e., quais elementos ocorrem depois de um dado elemento. Por exemplo, as permutações 123456 e 245361 compartilham a característica comum 'o 5 ocorre depois do 4'.

3. *posições absolutas* dos elementos, i.e., quais elementos ocorrem em uma dada posição. Por exemplo, as permutações 123456 e 643512 compartilham a característica comum '3 ocorre na terceira posição'.

4. *blocos* de elementos, i.e., quais seqüências de elementos começam em uma dada posição. Por examplo, as permutações 123456 e 613452 compartilham a característica comum 'a seqüência 345 começa na terceira posição'.

Dependendo do algoritmo de ordenação considerado, uma destas características será a mais representativa da complexidade do algoritmo responsável pela ordenação de uma seqüência particular. Quer dizer, exibirá a melhor correlação com a dificuldade do problema. Análise de Forma é usada para discernir isto e provê uma indicação de quais dos operadores de recombinação deveriam ser usados em cada caso. Isto é feito como segue: primeiro, uma amostra do espaço de busca é realizada. Esta amostra é feita fixando-se algumas características das soluções, e medindo a variância da função de aptidão de uma amostra aleatória de soluções que exibem estas mesmas características. Este processo é repetido para cada possível representação, variando o número de características fixas de nenhuma (as soluções são então completamente não especificadas) até todas (a solução é completamente especificada). As características para as quais a variância da função de aptidão diminui a uma taxa mais alta com a ordem de forma (a ordem de uma forma é o número de características fixas) será a mais correlacionada com a aptidão. Uma vez que estas características tenham sido identificadas, o operador de recombinação é projetado para manipulá-las adequadamente [322].

8.2.2 BubbleSort

BubbleSort é um dos algoritmos de ordenação mais simples e mais ineficientes. O método é baseado em fazer $O(n)$ passagens pelo conjunto a ordenar, executando comparações par a par (e trocas se apropriado) entre elementos em posições sucessivas. Pode ser facilmente argumentado que este algoritmo tem um pior caso quadrático de comportamento. Além disso, a complexidade exata de pior caso requere $n(n-1)/2$ comparações/trocas. O pseudocódigo na linguagem Java desse algoritmo é mostrado na Figura 8.1.

Devido à sua simplicidade, este algoritmo foi escolhido como a primeira "etapa" para o desenvolvimento da técnica de análise evolutiva. Em primeiro lugar, devem ser identificadas as características salientes de permutações difíceis com respeito ao *BubbleSort*. A Fig. 8.2 mostra uma análise da variância da função de aptidão de diferentes representações em um problema com 50 elementos. Para ser preciso, para cada tamanho de forma, foram gerados 100 formae aleatórios e a variância na aptidão de 100 instâncias aleatórias de cada forma foi medida.

Estes resultados indicam que a representação baseada na posição é a mais adequada para capturar a dificuldade do problema neste caso, enquanto que a adjacência é a pior representação. Este é um resultado esperado, visto que sabendo que o elemento i é localizado depois de j, simplesmente indica que para haver uma troca j deve ser maior que i. Ao contrário, o fato de i aparecer, por exemplo, na posição j provê uma informação muito definida sobre o custo de ordenação na seqüência (i.e., pelo menos $|i-j|$ trocas são requeridas). De fato, adjacência será mostrada ser um indicador pobre da dificuldade do problema para todos os algoritmos de ordenação considerados neste trabalho.

```
void BubbleSort (int list[]) {
     boolean continue = true;
     for (int i = list.length-1; (i>0) && continue; i--) {
         for (int j = 0; j<i; j++) {
             if (list[j] > list[j+1]) {
                int temp = list[j];
                list[j] = list[j+1];
                list[j+1] = temp;
                continue = true;
             }
         }
     }
}
```

Figura 8.1: *Pseudocódigo do algoritmo de* BubbleSort *usado nas experiências computacionais.*

De acordo com esses resultados, o operador de reprodução Uniform Cycle (UCX) [84] foi selecionado. Este operador é projetado para combinar características de posição de duas soluções sem introduzir informações exógenas. Subseqüentemente, o algoritmo evolutivo foi aplicado para tamanhos diferentes de problema, variando de 10 até 70 elementos. Um número total de 20 execuções foi feito em cada tamanho de problema. A Fig. 8.3 mostra os resultados. Como pode ser visto, existe uma correlação perfeita entre os dados experimentais providos pelas instâncias de pior caso geradas pelo GA (para valores diferentes de n, o eixo-X) e a parábola $y = an^2 + bn$ para $a = 0,5$ e $b = -0,5$. Em essência, o EAA emparelhou o comportamento de pior caso assintótico exato para *BubbleSort*.

8.2.3 MergeSort

MergeSort é também um algoritmo recursivo muito simples, geralmente usado para ensinar o princípio da estratégia denominada "Dividir-e-conquistar". O método é baseado em dividir a lista em duas sublistas iguais que são ordenadas recursivamente. As sublistas ordenadas resultantes são fundidas para obter a lista ordenada final. O pseudocódigo Java deste algoritmo é mostrado na Figura 8.4.

Consideramos a versão denominada *"in place"* de *MergeSort* cuja complexidade de pior caso é $O(n^2)$. Esta complexidade poderia ser reduzida a $O(n \cdot \log n)$ usando um buffer auxiliar externo, mas nos restringimos aos algoritmos de ordenação internos.

Como fizemos antes, começamos o EAA executando uma análise da variação da função de aptidão. Os resultados são mostrados na Fig 8.5. Novamente, formae de posição parecem ser bem correlacionados com a dificuldade

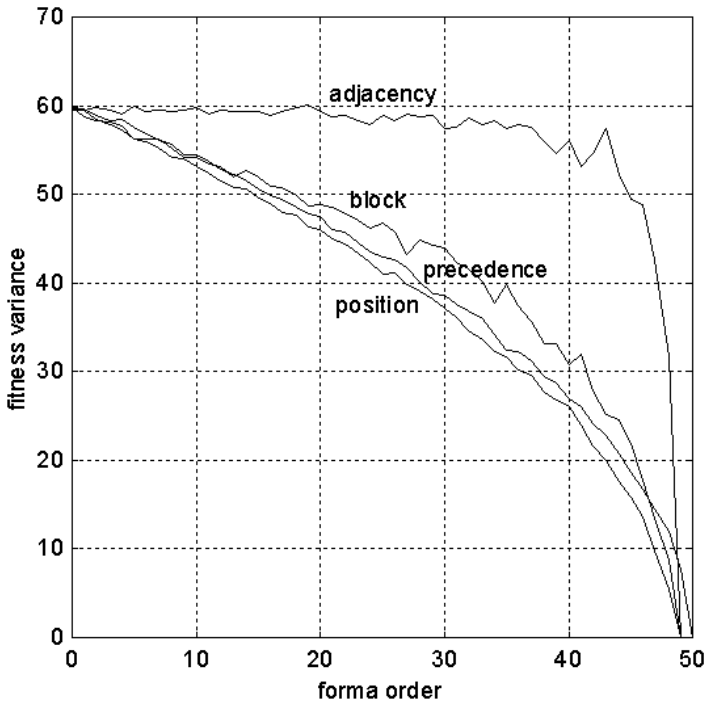

Figura 8.2: *Variação da função de aptidão em relação aos formae para o problema de* BubbleSort-*ing.*

do problema. Assim UCX foi escolhido neste caso. São mostrados os resultados providos pelo EAA na Fig 8.6. Como pode ser visto, um emparelhamento quadrático perfeito é encontrado. Para ser preciso, os dados experimentais ajustaram a parábola $y = 0,5153n^2 + 1,3735n$. O coeficiente do termo quadrático concorda fortemente com o valor teórico de 1/2. Note que a complexidade de pior caso exata inclui um termo $n \cdot \log n$, não considerado na parábola acima. Se um ajuste para uma curva $y = an^2 + bn + cn \log n$ é buscado, os coeficientes $a = 0,4981$, $b = -0,8711$, e $c = 0,8074$ são obtidos. Isto representa um acordo muito melhor com o limitante teórico. De fato, deve-se notar que o GA acha o pior caso exato (uma seqüência ordenada em ordem decrescente) de maneira sistemática.

8.2.4 ShellSort

O algoritmo **Shellsort** [347] é assim chamado em homenagem a D.L. Shell que o propôs quatro décadas atrás. É um dos métodos mais antigos de ordenação. Baseado em ordenação por inserção, este algoritmo procede da esquerda para a direita pela lista em uma seqüência de passos interlaçados. Estes passos são feitos na base de uma certa seqüência de incremento. Esta seqüência é um

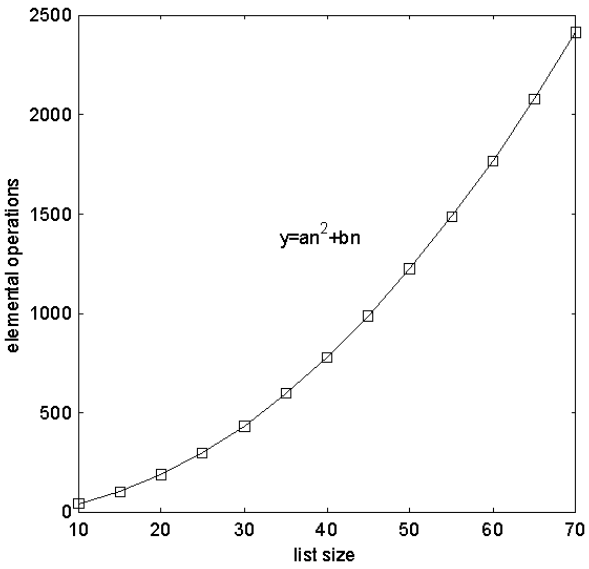

Figura 8.3: *Resultados obtidos ao aplicar o EAA a* BubbleSort.

elemento crucial de *Shellsort* e representa um papel central no comportamento do algoritmo e em sua subseqüente complexidade. O pseudocódigo Java do algoritmo que testamos é mostrado na Figura 8.7.

Apesar de sua simplicidade aparente, este algoritmo mereceu uma quantia enorme de estudos teóricos e empíricos. Nesse sentido, um dos pontos chaves é o estudo de seqüências de incremento diferentes e a determinação dos limitantes de complexidade correspondentes.

A seqüência que consideramos aqui ($h_0 = 1$, $h_{i+1} = 3h_i + 1$) foi proposta por D. Knuth, e é freqüentemente usada por ser fácil de computar, usa relativamente poucos (em torno de $\log_3 n$) incrementos, e se comporta bem em estudos empíricos. Também está no coração do desafio de Sedgewick (veja [340], página 110) que já foi citado antes.

Devido ao interesse provocado pelo desafio de Sedgewick, tivemos uma especial curiosidade em testar a análise evolutiva e como poder avaliar este algoritmo. Seguindo a mesma metodologia proposta, executamos a análise de formae de variação da função de aptidão primeiro. Os resultados são mostrados na Fig. 8.8. Em ambos, a representação baseada em posição e a representação baseada em bloco parecem capturar as características salientes das instâncias difíceis de um modo semelhante, representação baseada em posição é ligeiramente melhor. Assim, UCX foi novamente escolhido como o operador de recombinação para nosso GA. Comentamos isso em essência, usamos o mesmo GA (sem mudança de parâmetros) para os algoritmos estudados. São mostrados os resultados para tamanhos que variam de 10 até 100 na Fig. 8.9 (direita).

```
void MergeSortAux (int list[], int l, int r) {
     if (l<r) {
        int mid = (l+r)/2;
        int i = l;
        MergeSortAux(list, l, mid);
        MergeSortAux(list, mid+1, r);
        while ((i<=mid) && (mid<r)) {
              if (list[i]>list[mid+1]) {
                 int temp = list[mid+1];
                 for (int j = mid+1; j>i; j--)
                     list[j] = list[j-1];
                 list[i] = temp;
                 mid++;
              }
              i++;
        }
     }
}

void MergeSort (int list[]) {
     MergeSortAux(list, 0, list.length-1);
}
```

Figura 8.4: *Pseudocódigo do algoritmo de* MergeSort *usado nas experiências computacionais.*

Como pode ser visto, os dados experimentais podem ser ajustados em ambos por uma parábola ($a = 0,092$, $b = 8,519$) e uma exponencial ($a = 2,792$, $b = 1,397$). Não obstante, note que o coeficiente quadrático é relativamente pequeno e sugere que não é muito significativo. Este fato pode ser explorado mais adiante analisando a tendência das curvas obtidas dos dados empíricos. Para isso, consideramos várias funções candidatas, $f_1(\cdot), \cdots, f_k(\cdot)$. Computamos a relação entre os dados empíricos e os valores providos pelas funções candidatas. Finalmente, calculamos a inclinação dos dados obtidos. A razão atrás disto é que o melhor emparelhamento entre os dados empíricos e a função candidata e obtido quanto maior é a tendência da pendente da relação a ser zero.

A Fig. 8.10 é um modo interessante de exibir esta informação. Mostra os resultados desta análise para as funções candidatas seguintes: $f_1(n) = n^2$, $f_2(n) = n \log n$, $f_3(n) = n^{3/2}$, $f_4(n) = n^{\sqrt{2}}$, e $f_5(x) = n^{\phi}$ onde $\phi = (1 + \sqrt{5})/2 \approx 1,618$. Os dados obtidos são conclusivos, f_1 claramente cresce mais rapidamente que os dados empíricos. O mesmo para f_5. Ao contrário, f_2 cresce menos rapidamente que os dados experimentais. A função exata se encontra entre f_3 e f_4, respectivamente limitante superior e inferior dos dados empíricos. Novamente, isto concorda de maneira excelente com a teoria.

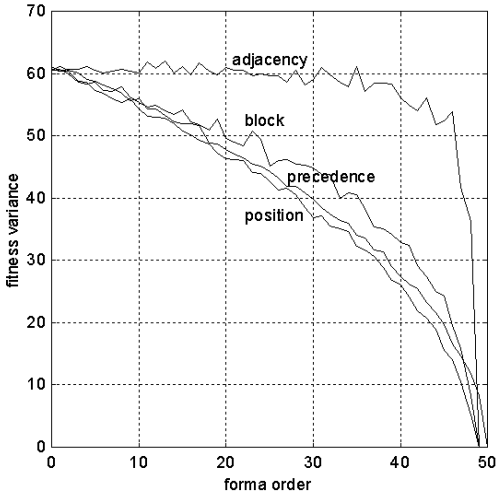

Figura 8.5: *Variação da função de aptidão em relação aos formae para o problema de* MergeSort-*ing.*

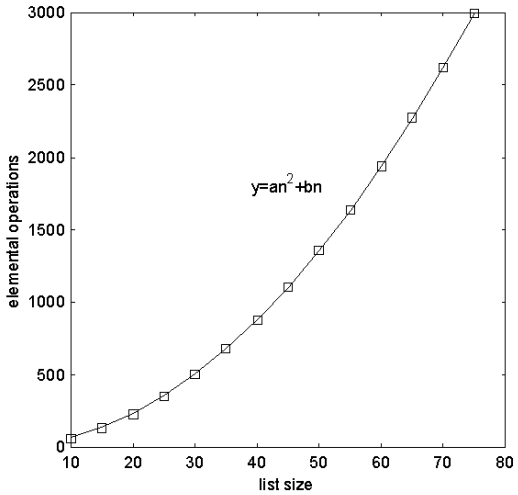

Figura 8.6: *Resultados obtidos por EAA aplicado a* MergeSort.

```
void ShellSort (int list[]) {
     int incs[14] = { 2391484, 797161, 265720, 88573,
                      29524, 9841, 3280, 1093, 364,
                      121, 40, 13, 4, 1 };
     for (int k = 0; k < 14; k++)
         for (int h = incs[k], i=h; i<r; i++) {
             int v=list[i];
             int j=i;
             while ((j>h) && (list[j-h]>v)) {
                   list[j] = list[j-h];
                   j -= h;
             }
         }
}
```

Figura 8.7: *Pseudocódigo do algoritmo de* ShellSort *usado nas experiências computacionais.*

8.2.5 QuickSort

QuickSort é o último algoritmo [165] que estudamos. Este algoritmo é considerado um dos procedimentos de ordenação mais eficientes, e será usado para ilustrar assuntos de escalabilidade. *QuickSort* inicialmente escolhe um elemento de pivô e divide a lista em duas sublistas: uma com elementos abaixo ou igual ao pivô, e outro com elementos maiores que o pivô. Ordenando recursivamente cada uma destas sublistas, a lista inteira é ordenada. O pseudocódigo em Java deste algoritmo é mostrado na Figura 8.11.

O número de comparações entre os elementos deste algoritmo é conhecido ser $O(n^2)$ no pior caso, correspondendo à situação na qual o pivô escolhido ocorre ser o maior ou o menor elemento na sublista atual. O EAA é assim aplicado para buscar este cenário de pior caso.

A Fig. 8.12 mostra o resultado da análise da variação da função de aptidão para este problema. Ao contrário dos algoritmos anteriores, *formae* (pl. de *forma* de bloco são neste caso as características mais adequadas para capturar a dificuldade do problema. Conseqüentemente, a recombinação de Bloco Uniforme (UBX) [84] foi escolhida como operador de recombinação. São mostrados os resultados da aplicação subseqüente do EAA na Fig. 8.13. Note um fato muito interessante: enquanto a aproximação quadrática global dos dados experimentais é bastante pobre (o melhor ajuste é $y = 0,073n^2 + 19,206n$), uma aproximação quase perfeita ($y = 0,4939n^2 - 0,3662n$) é encontrada para tamanhos até 50 items. Além disso, a parábola $y = 0,5n^2 - 0,5n$ é alcançada para tamanhos até 40.

Claramente, a qualidade dos resultados produzidos pelo GA naturalmente degradam quando a dimensionalidade do problema aumenta. Dependendo do algoritmo que é analisado, esta degradação pode ser significativa em um ponto

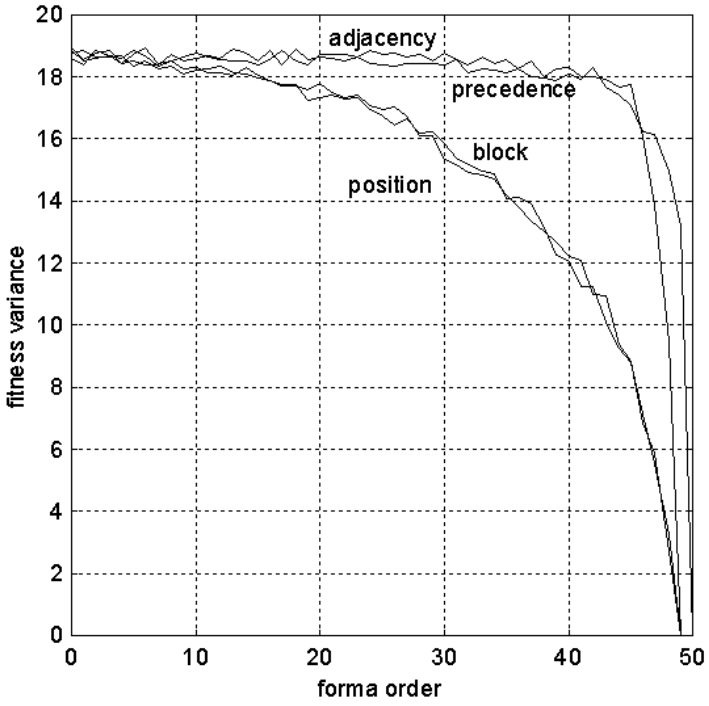

Figura 8.8: *Variação da função de aptidão em relação aos formae para o problema de* ShellSort*-ing.*

diferente. Isto é esperado, de um certo modo a *maldição da dimensionalidade* é inevitável. Porém, nos experimentos que conduzimos até o momento para testar o EAA, sempre o caso das instâncias de baixo tamanho geradas já foram muito úteis para deduzir a tendência assintótica de pior caso. É claro que estamos impossibilitados de garantir que este comportamento também acontecerá com todo tipo de algoritmo. Em particular, para alguns algoritmos exatos para problemas de otimização combinatória este não é o caso. Apesar deste fato, parece razoável esperar que para muitos algoritmos, quanto maior a instância, mais difícil de resolver, então o "comportamento monotônico" pode ser uma base útil para extrapolar a tendência de pior caso.

Nos casos que usamos para exemplificar o EAA, e particularmente na análise de *ShellSort*, e *QuickSort*, estava claro a necessidade de algum tipo de análise estatística *"ad-hoc"* para encontrar o valor de corte do tamanho de instância para o qual os resultados de GA são uma sub-estimação do cenário de pior caso. Para *QuickSort*, se estabelecemos um corte final entre 20 a 50, e tentando ajustar uma curva quadrática, notamos que para todas as curvas providas, o coeficiente quadrático é ao redor de 0,5, mas uma pequena queda ocorre além deste ponto.

O gráfico da Figura 8.14 indica que os resultados obtidos pelo GA para

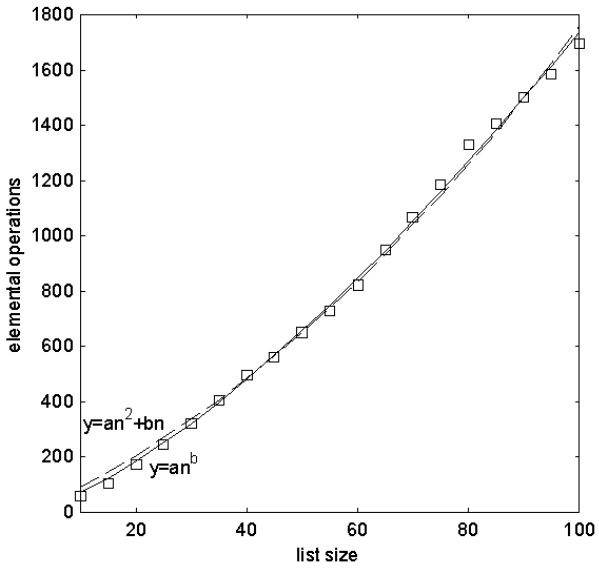

Figura 8.9: *Resultados obtidos por EAA aplicado a* ShellSort. *Os quadrados representam os dados experimentais, a linha tracejada é um ajuste a uma função do tipo* $an^2 + bn$, *e a linha sólida e um ajuste do tipo* an^b.

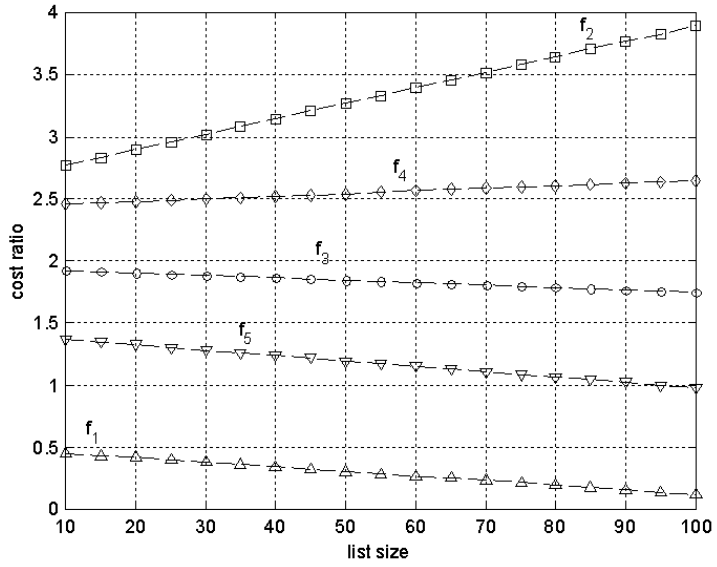

Figura 8.10: *Análise da tendência para os dados obtidos por EAA no caso de* ShellSort. *Uma interpolação linear do quociente entre os valores obtidos no experimento e algumas funções de teste.*

```
void QuickSortAux (int list[], int l, int r) {
     if (l<r) {
        int mid = (l+r)/2;
        int pivot = list[mid];
        int i = l+1;
        int j = r;
        list[mid] = list[r];
        list[r] = pivot;
        while (i<j) {
              while ((i<j) && (list[i]<=pivot))
                    i++;
              while ((i<j) && (list[j]>pivot))
                    j--;
              if (i<j) {
                    int temp = list[j];
                    list[j] = list[i];
                    list[i] = temp;
              }
        }
        list[r] = list[j];
        list[j] = pivot;
        QuickSortAux(list, l, i-1);
        QuickSortAux(list, j+1, r);
     }
}

void QuickSort (int list[]) {
     QuickSortAux(list, 0, list.length-1);
}
```

Figura 8.11: *Pseudocódigo do algoritmo de* QuickSort *usado nas experiências computacionais.*

tamanhos maiores que 50 são menos significativos e podem ser descartados. Um método do tipo semelhante ao *"moving-fit"* aplicada a *ShellSort* mostra uma oscilação do coeficiente exponencial 1,4729 (recorde o ótimo valor de 3/2). Isto é fácil de ver por uma inspeção visual das curvas, mas parece claro que técnicas estatísticas e ferramentas de *estimação robusta* teriam um papel central. Eles podem ser usados para dar um estimador extra aos resultados mais fidedignos que, no caso de ordenação, têm sido os resultados do GA para valores menores que n.

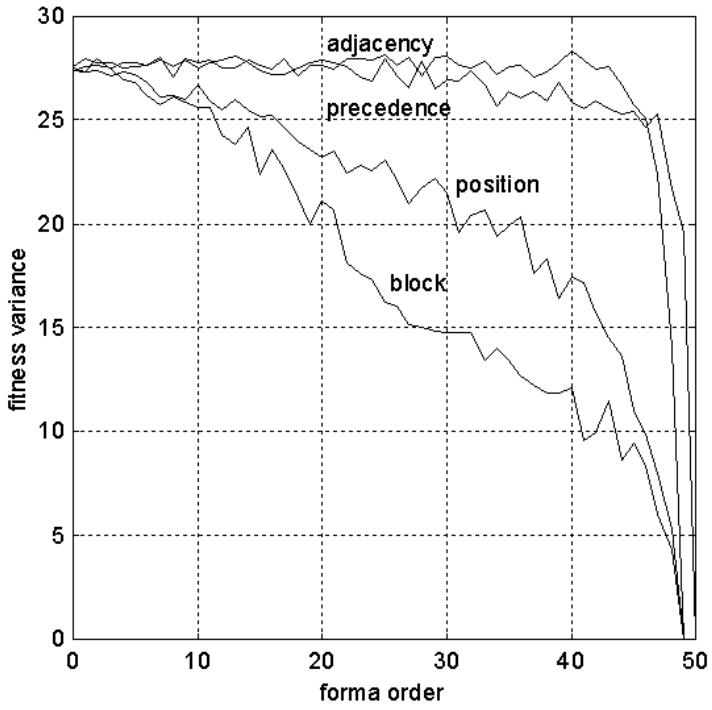

Figura 8.12: *Variação da função de aptidão em relação aos formae para o problema de* QuickSort-*ing.*

8.3 Conclusões

Um ponto de precaução a ser considerado na escolha de um GA como a máquina de busca subjacente deveria ser considerado. Selecionamos um GA, dentre outras técnicas evolutivas, devido ao fato de que constitui um código de alta reusabilidade. Além disso, é uma técnica familiar para muitos programadores em indústrias. Não obstante, podem ser usados outros tipos de busca sistemáticas ou estocásticas baseadas em população [273]; a filosofia geral da abordagem apresentada permanece a mesma. Em qualquer caso, a disponibilidade de ferramentas poderosas para projetar GAs, por exemplo Análise de Forma, faz estas técnicas uma escolha muito apropriada.

É claro que a metodologia de trabalho proposta promete bastante. Um trabalho futuro seria dirigido para analisar se também pode-se aplicar a outros problemas, como atacar algoritmos exatos baseado em estratégias de ramificação para problemas de otimização combinatória. Isto não significa que nosso trabalho está declarando o fim da *"era de provas"*. Ao contrário, pensamos e apoiamos fortemente que não há nada que possa substituir uma demonstração elegante que tem uma beleza peculiar quando relaciona um problema com uma análise de pior caso. Porém, esperamos que a análise evolutiva possa ser uma

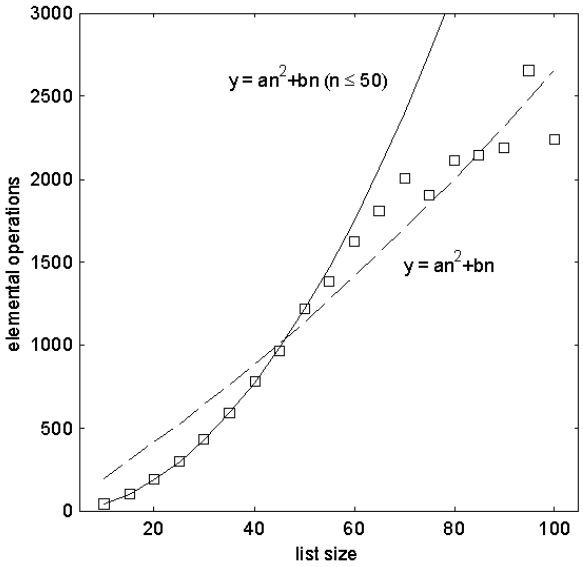

Figura 8.13: *Resultados obtidos por EAA aplicado a* QuickSort. *Os quadrados representam os dados experimentais, a linha tracejada é um ajuste a* $an^2 + bn$, *e a linha sólida e um ajuste a* $an^2 + bn$, *porém restrito a* $n \leq 50$.

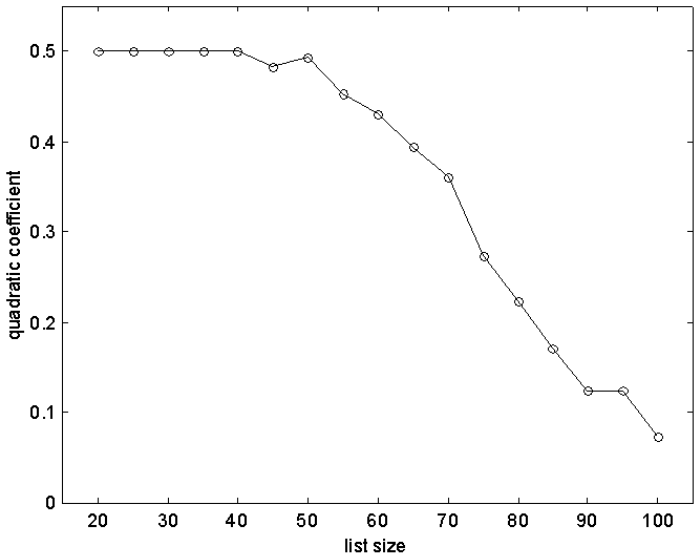

Figura 8.14: *Evolução do coeficiente quadrático na análise dos dados experimentais obtidos pelo EAA para* QuickSort.

contribuição positiva, particularmente quando outras abordagens que tentam uma análise de *caso-médio* ou *probabilística* constituem uma tarefa muito árdua para quem tem um interesse prático a resolver.

Capítulo 9

Conclusões e Direções Futuras

9.1 Direções Futuras

O futuro parece favorável para à Computação Evolutiva. Esta crença se fundamenta na combinação de vários fatores. Primeiro, os MAs (cada vez menos freqüentemente disfarçados sob diferentes nomes) estão mostrando um registro impressionante de boas implementações que provêem resultados excelentes em problemas práticos. Segundo, há razões para acreditar que algumas tentativas novas para fazer análises teóricas podem ser bem sucedidas. Isto inclui a análise de complexidade computational de pior caso e do caso médio, em procedimentos de recombinação. Terceiro, a natureza onipresente de sistemas distribuídos, como redes de estações de trabalho por exemplo, mais o paralelismo assíncrono inerente dos MAs e a existência de linguagens *WWW*-conscientes como Java; é uma combinação excelente para desenvolver *"frameworks"* orientados a objetos altamente portáveis e que permitem o reuso de algoritmos. Estes *frameworks* poderiam permitir aos usuários resolver subproblemas usando códigos comerciais ou softwares bem-testados de outros usuários que poderiam ser os especialistas nas outras áreas. Nosso trabalho atual no *MemePool Project* [126] vai nesta direção. Quarto, um importante e pioneiro grupo de MAs, o de *Scatter Search*, está desafiando o papel da recombinação randomizada. Assim, esperamos como uma reação saudável, ver novos tipos de MAs mais poderosos que misturam, de um modo mais apropriado, métodos exaustivos (truncados ou não) e métodos de busca sistemática.

9.1.1 Lições aprendidas

Em 1998, Applegate, Bixby, Cook e Chvatal estabeleceram novos resultados inovadores para o MIN TSP. De maneira bastante interessante, este trabalho apóia nossa visão de que os MAs terão um papel central como uma metodologia que resolve problemas combinatórios. Por exemplo, estes pesquisadores resolveram, na otimizalidade, uma instância do MIN TSP de 13.509 cidades

que corresponde a todas as cidades norte-americanas com população de mais de 500 pessoas [1]. A abordagem, de acordo com Bixby: *"...envolve idéias de combinatória poliédrica e otimização combinatória, programação linear e inteira, estrutura de dados e algoritmos, computação paralela, engenharia de software, análise numérica, teoria de grafos, e mais"*. A solução desta instância exigiu o uso de três AlphaServer 4100s Digital (com um total de 12 processadores) e um agrupamento de 32 PC Pentium-II. O cálculo completo levou aproximadamente três meses de tempo computacional. O código tem certamente mais que 1.000 páginas e está baseado em técnicas do estado-da-arte de uma ampla variedade de campos científicos.

A filosofia é certamente a mesma dos MAs, a sinergia de metodologias diferentes [270][272]. Porém, devido a um comentário de Johnson e McGeoch ([182], página 301), sugerimos previamente que a conexão com MAs pudesse ser mais forte, dado que há uma analogia com recombinação de *múltiplos pais*. Em um recente manuscrito não publicado, *"Finding Tours in the TSP"*, pelos mesmos autores (Bixby *et al.*), disponível no *web site* por eles mantido, esta suspeita foi agora confirmada. Eles apresentam resultados da execução de um algoritmo ótimo para resolver o problema MIN WEIGHTED HAMILTONIAN CYCLE em um subgrafo formado pela união de 25 rotas do método aleatorizado denominado *"Chained Lin-Kernighan"*. A metodologia acha a solução ótima constantemente para as instâncias de MIN TSP originais com até 4.461 cidades. Eles também tentaram aplicar esta idéia para uma instância com 85.900 cidades (a maior instância na `TSPLIB`) e desta experiência eles se convenceram que o método também trabalha bem para tais instâncias grandes.

A metodologia deles pode ser classificada possivelmente como o MA mais complexo já construído para um determinado problema de otimização combinatória. Um dos desafios atuais é desenvolver algoritmos mais simples que alcançam estes resultados impressionantes.

A metodologia de usar *"Chained Lin-Kernighan"* como algoritmo base de busca local para produzir uma coleção de soluções de excelente qualidade, seguido pelo método de recombinação dinástico ótimo que os autores nomearam *"tour merging"*, deu uma rota subótima de só 0,0002 % sobre a rota ótima para a instância de 13.509 cidades. Levamos isto em conta como uma prova clara dos benefícios de MAs. Mais trabalho é necessário para o desenvolvimento de boas estratégias para o projeto de *algoritmos meméticos completos*, i.e. esses que sistemática e sinergicamente usam métodos randomizados e determinísticos e podem provar otimalidade da solução final.

9.1.2 Complexidade computacional dos problemas de recombinação

Uma nova direção interessante para pesquisa teórica surgiu depois da introdução de duas classes de complexidade computacional, a classe PMA dos *"Polynomial Merger Algorithms"* e seu análogo não restrito, a classe uPMA. Notamos que a recombinação dinástica ótima, como era o procedimento de combinação de tours discutido anteriormente, pode levar a resolver outro problema NP-difícil

[1] See: `http://www.crpc.rice.edu/CRPC/newsArchive/tsp.html`

de um tamanho menor. Para tentar evitar a intratabilidade do novo subproblema gerado, buscamos achar casos interessantes com pior caso de complexidade de tempo polinomial para outros subproblemas relacionados, contanto que eles também sejam úteis para a meta primária de resolver o problema de otimização original. Tendo estas idéias em mente, discutimos as classes de complexidade uPMA e PMA.

A pesquisa atual está sendo conduzida para identificar problemas, e seus procedimentos de recombinação associados, podendo demostrar-se que são membros de PMA ou uPMA. Também é esperado que depois de algumas tentativas em problemas desafiadores, reduções e completude dentro dessas classes possam ser corretamente definidas.

9.1.3 Resultados que exploram algoritmos FPT

Uma nova área interessante de pesquisa pode ser estabelecida unindo adequadamente os resultados da teoria de tratabilidade com parâmetro fixo e o desenvolvimento de algoritmos de recombinação. Esta é uma área que vai em ambas as direções. Em uma direção, algoritmos eficientes, (i.e. tempo polinomial), de parâmetro fixo podem ser usados como uma ferramenta *"fora da caixa"* para criar procedimentos de recombinação eficientes. Eles podem ser usados como procedimentos de recombinação dinástica ótima ou algoritmos do tipo *k-merger*. Na direção oposta, visto que os Algoritmos Meméticos são tipicamente projetados para lidar bem com instâncias grandes e escalam bem com o tamanho delas, usar ambas as técnicas juntas pode produzir *MAs completos* que permitem estender os benefícios da tratabilidade de parâmetro fixo. De uma perspectiva de engenharia de software, a combinação é perfeita para reuso de softwares e algoritmos.

Para ilustrar esse tópico, discutimos um dos mais emblemáticos problemas FPT, o k-VERTEX COVER. Em geral, algoritmos FPT eficientes são baseados nas técnicas de *redução a um núcleo de problema* e *árvores de busca limitadas*. Para entender essas técnicas, o leitor pode checar o método de Chen, Kanj, e Yia. Este método pode resolver a versão parametrizada de VERTEX COVER no tempo $O(kn + 1,271^k k^2)$.

A definição da classe FPT permite $f(k)$ crescer arbitrariamente rápido (veja a definição de $f(k)$ e a classe FPT na Seção 6.2). Do lado positivo, um método geral novo para melhorar algoritmos FPT que executam no tempo $O(q(k)\beta^k + p(n))$ (onde p e q são polinomiais e β é uma constante pequena) tem sido desenvolvido por Niedermeier e Rossmanith [290]. Seus métodos intercalam a redução para o núcleo de problema e métodos de árvores de busca limitadas. Se é o caso que β^k é o tamanho da árvore de busca limitada e $q(k)$ é o tamanho do núcleo do problema, a nova técnica é capaz de eliminar o fator $q(k)$ permitindo ao algoritmo ser $O(\beta^k + p(n))$. Outro problema que pertence à classe FPT é:

HITTING SET FOR SIZE THREE SETS

Instância: Uma coleção C de subconjuntos de tamanho três de um conjunto finito S.

Parâmetro: Um inteiro positivo k.

Questão: Há um subconjunto $S' \subseteq S$, com $|S'| \leq k$ que permite a S' conter no mínimo um elemento de cada subconjunto de C ?

O problema pode ser visto como uma generalização do problema VERTEX COVER para hipergrafos, onde há um hiperaresta entre três vértices (ao invés de somente dois como em VERTEX COVER), e a tarefa é encontrar um subconjunto mínimo de vértices cobrindo todas as arestas. Recentemente, Niedermeier e Rossmanith apresentaram um algoritmo que resolve este problema no tempo $O(2,311^k + n)$ [288]. Deixamos como um exercício para o leitor a tarefa de encontrar operadores de recombinação que tomam vantagens deste algoritmo. Também deixamos como uma interessante questão para futuras pesquisas o seguinte, *"é possível usar o $O(2.311^k + n)$ para construir um algoritmo k-merger para o problema de otimização* MIN HITTING SET FOR SIZE THREE SETS *que satisfaz o critério exigido pela definição da classe PMA ?"*

9.1.4 Adição de conhecimento negativo

Durante os últimos cinco anos, a adição de conhecimento dependente do problema, para melhorar metodologias baseadas em população, receberam um interesse renovado entre investigadores de MAs [323, 3, 132, 149, 29, 44, 345]. Os primeiros artigos [145, 229, 353, 270, 272] tinham enfatizado a relevância deste assunto, como também Lawrence Davis, entre seus numerosos artigos, também apoiou esta idéia quando ainda não havia um consenso, e muitos investigadores eram céticos sobre o seu papel central.

Porém, como muitas outras coisas, a adição de conhecimento dependente do problema, e dependente da instância, pode ser tratada de modos diferentes. Hoje, é razoável continuar achando modos novos para incorporar *conhecimento negativo* em MAs. Também é um desafio como encontrar modos de extrair (ou podemos dizer melhor, induzir), "conhecimento negativo" da população corrente de soluções. Citamos M. Minsky [266]:

> *"Tendemos a pensar em conhecimento em termos positivos – e em especialistas como pessoas que sabem o que fazer. Mas um modo 'negativo' de aparentarem competência é, simplesmente, jamais cometer um erro."*

Isto é o que as heurísticas fazem melhor ao reduzir o esforço computational mudando radicalmente o espaço de busca, assim certamente não há nenhum paradoxo aqui. A maioria das heurísticas, porém, tem um "sesgo positivo". Por exemplo, em [308] podemos ler:

> *"A estratégia desenvolvida por Lin [230] para o TSP é obter diversos ótimos locais e então identificar as arestas que são comuns em todos. Isto é então fixado, reduzindo de essa maneira o tempo de encontrar mais ótimos locais. Esta idéia é desenvolvida mais adiante em [231] e [138]."*

É bastante curioso que, embora esta estratégia básica tenha estado presente por mais de três décadas, a metodologia de otimizar e recombinar é dificilmente

aceito por alguns investigadores indeferentemente de toda evidência empírica dos benefícios de MAs que exploram, usando recombinação, a correlação de soluções altamente otimizadas [200] [270] [279] [272] [37] [257].

Já reconhecido por R. Hofmann em 1993, há uma certa assimetria curiosa nesta estratégia. Tendemos a pensar em arestas comuns como pedaços de informação "positiva" a ser passada entre gerações. Porém, depois de algumas gerações de um MA, uma população altamente evoluída (contudo ainda diversa), poderia estar constantemente evitando certas arestas. *O que a população de agentes está "sugerindo" sobre estas arestas ?* Por exemplo, dada uma aresta, podemos definir dois tours como sendo equivalentes se ambos contêm a aresta, mas também podem ser definidos equivalentes *se ambos não têm*. Hofmann reconheceu esta dualidade e veio com a denominação de *formae de aresta negativo* para a classe de equivalência induzida posteriormente que descreve o conjunto de rotas que não compartilham esta aresta em particular (veja [170], página 35).

Neste ponto, é importante distingüir conhecimento de *crença*. Formalmente, conhecimento é geralmente definido como uma *crença verdadeira*, por exemplo "agente a conhece ϕ se a acredita em ϕ e ϕ é verdadeiro". Então uma maneira de adicionar conhecimento aos agentes é realmente tentar *provar* fatos que os agentes acreditam.

Por exemplo, se estamos tentando resolver uma instância euclidiana planar do MIN TSP, é sempre verdadeiro que a rota ótima não tem duas arestas que se cruzam. Este é um fato verdadeiro que pode ser provado com um teorema. Visto que movimentos *2-opt* podem remover cruzamentos de arestas eficazmente com a ajuda de estruturas de dados e *range queries*, todos os agentes de MAs para o MIN TSP geralmente têm incorporado este conhecimento nos métodos de busca local. Porém, também pode ser o caso de que outros predicados envolvendo mais de duas arestas possam ser demonstrados verdadeiros em todas as circunstâncias para o tour ótimo (i.e. um predicado que sempre é verdade para todas as instâncias consideradas). Contudo, é possível que não seja fácil achar métodos eficientes de busca local associados. Dito isto, pressentimos que dado um problema de otimização com só uma função objetivo, poderíamos usar métodos de MAs multi-objetivos usando como funções objetivo auxiliares o número de violações no conjunto de propriedades que a solução ótima deveria ter. Pesquisa nesta direção também está sendo desenvolvida.

9.1.5 Algoritmos Meméticos Guiados

Em [308] podemos ler:

> *"Como é comum com algumas heurísticas, pode-se também argumentar com exatamente a idéia oposta. Uma vez que estes aspectos sejam detectados, eles são proibidos ao invés de fixos. Esta justificativa é a seguinte: Se tememos que o ótimo local está escapando, isto poderia ser porque nossa heurística está "enganada" por estes aspectos tentadores. Sua proibição poder-nos-ia colocar finalmente na direção certa do ótimo global. Isto é chamado negação em [350]."*

Também pressentimos que futuras gerações de MAs trabalharão em pelo menos dois níveis e duas escalas de tempo. Durante a escala de curto prazo, um conjunto de agentes estaria procurando no espaço de busca associado ao problema. Isto é o que a maioria dos MAs faz hoje. Porém, uma escala de longo prazo *adaptaria os algoritmos* associados com os agentes. Aqui englobamos todos eles, as estratégias de busca de cada agente, os operadores de recombinação, etc. Por exemplo, um MA que usa *Busca Local Guiada* para uma estratégia adaptativa foi proposto em [172]. Operadores de recombinação com diferentes comportamentos (*rebelde*, *bajulador*, e *conciliador*) foram introduzidos em [34]. Achamos que estas novas direções promissoras necessitam ser investigadas, fornecendo mecanismos de recombinação adaptativos que tomam informações do progresso da população corrente. Além disso, é intrigante a idéia de usar uma combinação de procedimentos de busca determinísticos e sistemáticos com um MA estocástico. A combinação de ambos ajudaria a explorar sub-regiões do espaço de busca sistematicamente e *garantir* uma cobertura que atualmente só é exercida por algum procedimento randômico que visam diversificar a busca (como os denominados procedimentos de *mutação pesados*).

9.1.6 Lógical Modal para MAs e Busca com Crenças

Como uma conseqüência lógica dos possíveis panoramas a que os MAs podem levar, é razoável afirmar que esquemas mais complexos que evoluem soluções, agentes, como também representações, serão logo implementados. Alguns investigadores teóricos da Ciência de Computação desconsideram heurísticas e meta-heurísticas pois elas não são coordenadas como um paradigma formal. Porém, suas realizações são bem-reconhecidas. De [225]:

> *"Explicar e predizer o impressionante sucesso empírico de alguns destes algoritmos é hoje uma das fronteiras desafiadoras da Teoria da Computação."*

Este comentário é até mesmo mais pertinente para MAs visto que eles geralmente apresentam resultados até melhores que métodos de agente único. Embora meta-heurísticas sejam extremamente poderosas na prática, concordamos que um problema com a tendência atual de pesquisa aplicada é que se permite a introdução crescente de heurísticas mais complexas, infelizmente na maioria das vezes parametrizada por números escolhidos *"ad-hoc"*. Além disso, algumas meta-heurísticas, como algumas implementações de *"sistemas de colônia de formigas"*, podem basicamente ser vistas como tipos particulares de MAs. Este é o caso, se permitesse às "formigas" usar *Branch-and-Bound* ou métodos de *Busca Local*. Além disso, estes métodos para recombinação distribuída de informação (ou crenças) tem alguns pontos em comum com os chamados *"blackboard systems"* [110], como foi reconhecido no passado, contudo quase não está sendo mencionado na literatura atual de metaheurísticas.

Para ilustrar como a Busca com Crenças pode trabalhar em um MA, usaremos um exemplo ilustrativo. Vamos supor que a fórmula $\mathbf{B}_a^i \phi$ tem o seguinte significado *"agente i acredita com força no mínimo a que ϕ é verdadeiro"*, tal que os valores de força a estão restritos a serem números racionais em [0,1].

Vamos supor que aceitamos como um axioma que por ψ ser verdadeiro podemos deduzir $\mathbf{B}_1^i\psi$ para todo i. Agora vamos supor que nossos agentes estão tentando resolver um MIN TSP e que a instânica particular sendo considerada é euclidiana e de duas dimensões. Seja ϕ_k a proposição que representa *"a aresta e_k está presente no tour ótimo"* e seja $\chi_{k,l}$ verdadeira se as arestas e_k e e_l cruzam-se, e falso caso contrário. Pode ser provado que para tal tipo de instância *"se as arestas e_k e e_l se cruzam entre si, então e_k e e_l não podem estar ambas presentes no tour ótimo"*. Então podemos assumir que isto é conhecido por todos os agentes, e pelo axioma anterior podemos deduzir que o agente 2 agora acredita que $\mathbf{B}_1^2(\chi_{k,l} \rightarrow \neg(\phi_k \wedge \phi_l))$. Agora vamos supor que o agente 1 acredita, com força 0,4, que *"ou a aresta e_k ou e_l, mas não ambas, está presente no tour ótimo"*. Representaremos isto como $\mathbf{B}_{0,4}^1\phi_{k,l}$. Não entraremos na discussão de como cada agente alcançou suas crenças e tomaremos isto como um fato. Agora vamos supor que outro agente acredita, em um nível 0,7 que $\chi_{k,l} \rightarrow \phi_{k,l}$, então escrevemos $\mathbf{B}_{0,7}^3(\chi_{k,l} \rightarrow \phi_{k,l})$. Note que este tipo de suposição confunde nosso senso comum, dado que em geral não vemos qualquer relação entre o fato de duas arestas que se cruzam e que podemos deduzir que, como uma conseqüência, uma delas poderia estar presente no tour ótimo. Contudo, note que o agente 3 acredita nesta relação (em um nível 0,7) para *um par particular* de arestas e_k e e_l. Agora, o que podemos dizer sobre *a crença distribuída* do grupo de três agentes ? Como podemos combinar esta informação?

De acordo com $\mathbf{PL}_n^{\otimes}$, uma lógica epistêmica de múltiplos agentes recentemente introduzida por Boldrin e Saffiotti [38], as opiniões partilhadas por um conjunto de n agentes diferentes podem ser *recombinadas* em uma crença distribuída. Usando $\mathbf{PL}_n^{\otimes}$ podemos deduzir $\mathbf{D}_{0,82}\phi_{k,l}$. A crença distribuída sobre a proposição $\phi_{k,l}$ é mais forte que qualquer crença individual sobre ele, e é até mais forte do que se adquiriria se qualquer agente acreditasse nos três fatos.

9.1.7 Uso de regras de inferência

De acordo com $\mathbf{PL}_n^{\otimes}$ temos os seguintes axiomas e regras de inferência, onde ϕ e ψ são fórmulas de $\mathcal{L}$; a, b e c são números racionais em [0,1]; e $i = 1, \ldots, n$. A lógica tem cinco axiomas:

(A0)	Axioma 0	Todas as proposições tautológicas	
(A1)	Axioma 1	$\mathbf{B}_0^i \perp$	
(A2)	Axioma 2	$\mathbf{B}_a^i(\phi \rightarrow \psi) \rightarrow (\mathbf{B}_b^i\phi \rightarrow \mathbf{B}_c^i\psi)$	$c \leq \min\{a, b\}$
(A3)	Axioma 3	$\mathbf{D}_a(\phi \rightarrow \psi) \rightarrow (\mathbf{D}_b\phi \rightarrow \mathbf{D}_c\psi)$	$c \leq \min\{a, b\}$
(A4)	Axioma 4	$(\wedge_{i=1}^n \mathbf{B}_{a_i}^i\phi) \rightarrow \mathbf{D}_c\phi$	$c = \oplus_{i=1}^n a_i$

e três regras de inferência:

(MP)	*Modus Ponens*	de ϕ e $\phi \rightarrow \psi$ se deduz ψ
(NEC)	Necessitação	de ϕ se deduz $\mathbf{B}_1^i\phi$
(US)	Substituição Uniforme	

A fórmula ϕ é dita ser um teorema de $\mathbf{PL}_n^{\otimes}$ escrito $\vdash \phi$, se ϕ é obtida a partir de **A0-A4** por um número finito de aplicações de **MP**, **NEC** e substituições uniformes (**US**). Então, se $\Gamma \subseteq \mathcal{L}$ escreveremos $\Gamma \vdash \phi$ para significar $\vdash (\wedge_{\phi \in \Gamma}) \rightarrow \phi$.

Proposição: Dado $\Gamma = \{(a), (b), (c)\}$ (veja abaixo) então $\Gamma \vdash (\mathbf{D}_{0,82}\phi_{k,l} \wedge \mathbf{D}_{0,82}(\chi_{k,l} \rightarrow \phi_{k,l}))$.

(a)$\mathbf{B}^1_{0,4}\phi_{k,l}$,
(b)$\chi_{k,l}$,
(c)$\mathbf{B}^3_{0,7}(\chi_{k,l} \rightarrow \phi_{k,l})$.

Prova:

(1)$\phi_{k,l} \rightarrow (\chi_{k,l} \rightarrow \phi_{k,l})$	**A0**
(2)$\mathbf{B}^1_1(\phi_{k,l} \rightarrow (\chi_{k,l} \rightarrow \phi_{k,l}))$	**NEC**, 1
(3)$\mathbf{B}^1_1(\phi_{k,l} \rightarrow (\chi_{k,l} \rightarrow \phi_{k,l}))$ $\rightarrow (\mathbf{B}^1_{0,4}\phi_{k,l} \rightarrow \mathbf{B}^1_{0,4}(\chi_{k,l} \rightarrow \phi_{k,l}))$	**A2**, **US**
(4)$\mathbf{B}^1_{0,4}\phi_{k,l} \rightarrow \mathbf{B}^1_{0,4}(\chi_{k,l} \rightarrow \phi_{k,l})$	**MP**, 2, 3
(5)$\mathbf{B}^1_{0,4}(\chi_{k,l} \rightarrow \phi_{k,l})$	**MP**, (a), 4
(6)$\bot\rightarrow (\chi_{k,l} \rightarrow \phi_{k,l})$	**A0**
(7)$\mathbf{B}^2_1(\bot\rightarrow (\chi_{k,l} \rightarrow \phi_{k,l}))$	**NEC**, 6
(8)$\mathbf{B}^2_0 \bot$	**A1**
(9)$\mathbf{B}^2_1(\bot\rightarrow (\chi_{k,l} \rightarrow \phi_{k,l}))$ $\rightarrow (\mathbf{B}^2_0 \bot\rightarrow \mathbf{B}^2_0(\chi_{k,l} \rightarrow \phi_{k,l}))$	**A2**, **US**
(10)$\mathbf{B}^2_0 \bot\rightarrow \mathbf{B}^2_0(\chi_{k,l} \rightarrow \phi_{k,l})$	**MP**, 7, 9
(11)$\mathbf{B}^2_0(\chi_{k,l} \rightarrow \phi_{k,l})$	**MP**, 8, 10
(12)$\mathbf{B}^2_0(\chi_{k,l} \rightarrow \phi_{k,l}) \rightarrow (\mathbf{B}^2_1\chi_{k,l} \rightarrow \mathbf{B}^2_0\phi_{k,l})$	**A2**, **US**
(13)$\mathbf{B}^2_1\chi_{k,l} \rightarrow \mathbf{B}^2_0\phi_{k,l}$,	**MP**, 11, 12
(14)$\mathbf{B}^2_1\chi_{k,l}$	**NEC**, (b)
(15)$\mathbf{B}^2_0\phi_{k,l}$	**MP**, 14, 13
(16)$\mathbf{B}^3_1\chi_{k,l}$	**NEC**, (b)
(17)$\mathbf{B}^3_{0,7}(\chi_{k,l} \rightarrow \phi_{k,l}) \rightarrow (\mathbf{B}^3_1\chi_{k,l} \rightarrow \mathbf{B}^3_{0,7}\phi_{k,l})$	**A2**, **US**
(18)$\mathbf{B}^3_1\chi_{k,l} \rightarrow \mathbf{B}^3_{0,7}\phi_{k,l}$	**MP**, (c), 17
(19)$\mathbf{B}^3_{0,7}\phi_{k,l}$	**MP**, (c), 18
(20)$\mathbf{D}_{0,82}(\chi_{k,l} \rightarrow \phi_{k,l})$	**A4**, 5, 11, (c)
(21)$\mathbf{D}_{0,82}\phi_{k,l}$	**A4**, (a), 15, 19

Como outro exercício interessante, nos colocamos o seguinte: Seja ϕ_k (ϕ_l) o predicado *"a aresta e_k (respectivamente, e_l) está presente no tour ótimo"*. A tarefa é então deduzir, de acordo com $\mathbf{PL}_n^{\otimes}$, quais são as crenças distribuídas sobre as arestas e_k e e_l, se em adição aos três agentes anteriores há também dois outros agentes, tais que eles têm como crenças próprias: $\mathbf{B}^4_{0,54}\phi_k$, $\mathbf{B}^4_{0,27}\phi_l$, e $\mathbf{B}^5_{0,63}\phi_k$, $\mathbf{B}^5_{0,15}\phi_l$.

Para resolver a questão precisamos dos seguintes teoremas de $\mathbf{PL}_n^{\otimes}$:

(TH1) $(\mathbf{B}^i_a\phi \wedge \mathbf{B}^i_b\psi) \rightarrow \mathbf{B}^i_c(\phi \vee \psi)$ $\quad c = \max\{a, b\}$
(TH2) $(\mathbf{B}^i_a\phi \wedge \mathbf{B}^i_b\psi) \rightarrow \mathbf{B}^i_c(\phi \wedge \psi)$ $\quad c = \min\{a, b\}$

Se os agentes estão tentando resolver uma instância euclidiana do MIN TSP de 2 dimensões, então temos também os *axiomas dependentes da instância* ou **IDA**s.

$$\textbf{IDA1} \quad \chi_{k,l} \rightarrow \neg(\phi_k \wedge \phi_l)$$

Além disso, também conhecemos que:

$$\textbf{IDA2} \quad \phi_{k,l} \equiv ((\phi_k \rightarrow \neg\phi_l) \wedge (\neg\phi_k \rightarrow \phi_l))$$

Mostraremos então o que pode-se inferir nesse caso.

Proposition: *Dado* $\Gamma' = \{(a), \ldots, (g)\}$ *(veja abaixo) então* $\Gamma' \vdash \mathbf{D}_{0,96934}\phi_{k,l}$.

(a)$\mathbf{B}^1_{0,4}\phi_{k,l}$
(b)$\chi_{k,l}$
(c)$\mathbf{B}^3_{0,7}(\chi_{k,l} \rightarrow \phi_{k,l})$
(d)$\mathbf{B}^4_{0,54}\phi_k$
(e)$\mathbf{B}^4_{0,27}\phi_l$
(f)$\mathbf{B}^5_{0,63}\phi_k$
(g)$\mathbf{B}^5_{0,15}\phi_l$

Prova:

(1)$\mathbf{B}^4_1(\chi_{k,l} \rightarrow \neg(\phi_k \wedge \phi_l))$	**IDA1**, **NEC**
(2)$\mathbf{B}^4_1(\chi_{k,l} \rightarrow (\phi_k \rightarrow \neg\phi_l))$	**A0**
(3)$\mathbf{B}^4_1\chi_{k,l}$	(b), **NEC**
(4)$\mathbf{B}^4_1(\chi_{k,l} \rightarrow (\phi_k \rightarrow \neg\phi_l)) \rightarrow (\mathbf{B}^4_1\chi_{k,l} \rightarrow \mathbf{B}^4_1(\phi_k \rightarrow \neg\phi_l))$	**A2**, **US**
(5)$\mathbf{B}^4_1\chi_{k,l} \rightarrow \mathbf{B}^4_1(\phi_k \rightarrow \neg\phi_l)$	**MP**, 2, 4
(6)$\mathbf{B}^4_1(\phi_k \rightarrow \neg\phi_l)$	**MP**, 3, 5
(7)$\mathbf{B}^4_1(\phi_k \rightarrow \neg\phi_l) \rightarrow (\mathbf{B}^4_{0,54}\phi_k \rightarrow \mathbf{B}^4_{0,54}(\neg\phi_k))$	**A2**
(8)$\mathbf{B}^4_{0,54}\phi_k \rightarrow \mathbf{B}^4_{0,54}(\neg\phi_l)$	**MP**, 6, 7
(9)$\mathbf{B}^4_{0,54}(\neg\phi_l)$	**MP**, (d), 8

analogamente podemos deduzir,

(18)$\mathbf{B}^4_{0,27}(\neg\phi_k)$ **MP**, (e), 17
(27)$\mathbf{B}^5_{0,63}(\neg\phi_l)$ **MP**, (f), 26
(36)$\mathbf{B}^5_{0,15}(\neg\phi_k)$ **MP**, (g), 35

e agora usaremos um dos teoremas,

(37)$(\mathbf{B}^4_{0,54}\phi_k \wedge \mathbf{B}^4_{0,27}\phi_l) \rightarrow \mathbf{B}^4_{0,54}(\phi_k \vee \phi_l)$	**TH1**, **US**
(38)$\mathbf{B}^4_{0,54}(\phi_k \vee \phi_l)$	**MP**, (d), (e), 37
(39)$(\mathbf{B}^5_{0,63}\phi_k \wedge \mathbf{B}^5_{0,15}\phi_l) \rightarrow \mathbf{B}^5_{0,63}(\phi_k \vee \phi_l)$	**TH1**, **US**
(40)$\mathbf{B}^5_{0,63}(\phi_k \vee \phi_l)$	**MP**, (f), (g), (39)
(41)$(\mathbf{B}^4_{0,54}(\neg\phi_l) \wedge \mathbf{B}^4_{0,27}(\neg\phi_k)) \rightarrow \mathbf{B}^4_{0,54}(\neg\phi_k \vee \neg\phi_l)$	**TH1**, **US**
(42)$\mathbf{B}^4_{0,54}(\neg\phi_k \vee \neg\phi_l)$	**MP**, 9, 18, 41
(43)$(\mathbf{B}^5_{0,63}(\neg\phi_l) \wedge \mathbf{B}^5_{0,15}(\neg\phi_k)) \rightarrow \mathbf{B}^5_{0,63}(\neg\phi_k \vee \neg\phi_l)$	**TH1**, **US**
(44)$\mathbf{B}^5_{0,63}(\neg\phi_k \vee \neg\phi_l)$	**MP**, 27, 36, 43

se combinamos a informação usando o outro teorema,

$(45)(\mathbf{B}^4_{0,54}(\phi_k \vee \phi_l)) \wedge (\mathbf{B}^4_{0,54}(\neg\phi_k \vee \neg\phi_l))$	
$\rightarrow \mathbf{B}^4_{0,54}((\phi_k \vee \phi_l) \wedge (\neg\phi_k \vee \neg\phi_l))$	**TH2**, **US**
$(46)\mathbf{B}^4_{0,54}((\phi_k \vee \phi_l) \wedge (\neg\phi_k \vee \neg\phi_l))$	**MP**, 38, 42, 45
$(47)\mathbf{B}^4_{0,54}\phi_{k,l}$	**A0**
$(48)(\mathbf{B}^5_{0,63}(\phi_k \vee \phi_l)) \wedge \mathbf{B}^5_{0,63}(\neg\phi_k \vee \neg\phi_l))$	
$\rightarrow \mathbf{B}^5_{0,63}((\phi_k \vee \phi_l) \wedge (\neg\phi_k \vee \neg\phi_l))$	**TH2**, **US**
$(49)\mathbf{B}^5_{0,63}((\phi_k \vee \phi_l) \wedge (\neg\phi_k \vee \neg\phi_l))$	**MP**, 40, 44, 48
$(50)\mathbf{B}^5_{0,63}\phi_{k,l}$	**A0**
$(51)\mathbf{B}^3_{0,7}(\chi_{k,l} \rightarrow \phi_{k,l}) \rightarrow (\mathbf{B}^3_1\chi_{k,l} \rightarrow \mathbf{B}^3_{0,7}\phi_{k,l})$	**A2**, **US**
$(52)\mathbf{B}^3_1\chi_{k,l} \rightarrow \mathbf{B}^3_{0,7}\phi_{k,l}$	**MP**, (c), 51
$(53)\mathbf{B}^3_1\chi_{k,l}$	**NEC**
$(54)\mathbf{B}^3_{0,7}\phi_{k,l}$	**MP**, 53, 52
$(55)\mathbf{B}^2_1(\bot\rightarrow \phi_{k,l})$	**A0**, **NEC**
$(56)\mathbf{B}^2_1(\bot\rightarrow \phi_{k,l}) \rightarrow (\mathbf{B}^2_0 \bot\rightarrow \mathbf{B}^2_0\phi_{k,l})$	**A2**, **US**
$(57)\mathbf{B}^2_0 \bot\rightarrow \mathbf{B}^2_0\phi_{k,l}$	**MP**, 55, 56
$(58)\mathbf{B}^2_0 \bot$	**A1**
$(59)\mathbf{B}^2_0\phi_{k,l}$	**MP**, 58, 57
$(60)\mathbf{D}_{0,96934}\phi_{k,l}$	**A4**, (a), 47, 50, 54, 59

Notamos que, devido à computação da crença distribuída do conjunto de agentes em um MA, *é agora possível usar esta informação para guiar algoritmos construtivos*. Dito isto, um MA baseado em uma Busca com Crenças pode também tirar proveito de heurísticas construtivas já disponíveis na literatura. Além disso, métodos de busca exaustiva podem priorizar algumas decisões pendentes baseando-se na informação que é distribuidamente acreditada. Isto pode também permitir uma busca paralela por um conjunto de agentes, permitindo aos agentes terem muitas alternativas ao invés de procedimentos de busca *em profundidade* (*"depth-first"*) ou *em amplitude* (*"best-first"*) geralmente usados.

De nenhuma maneira afirmamos que $\mathbf{PL}^{\otimes}_n$ é a lógica definitiva que deve ser usada para guiar MAs baseados em Busca com Crenças. Mencionamos isto, pois $\mathbf{PL}^{\otimes}_n$ é relacionado com lógicas multi-modais de crença parcial e pode ser o caso que algumas outras formas para conectivos são mais apropriadas que a T-norma proposta para recombinar informação. Porém, $\mathbf{PL}^{\otimes}_n$ já incorpora características muito interessantes que gostaríamos de destacar e que não temos notado em outras lógicas de crença. Primeiro, permite *raciocínio epistêmico aninhado*, i.e. uma expressão como $\mathbf{B}^i_a\mathbf{D}_b\phi$ pode ser interpretada como *agente i acredita com força a que ϕ é distribuidamente acreditado em um com força b"*. Isto é muito interessante pois algumas heurísticas *ad-hoc* para operadores de recombinação genéricos, como os comportamentos *rebelde*, *conciliador*, e *bajulador* [34], podem ser interpretados em termos de um raciocínio epistémico aninhado. Segundo, a negação é tipicamente modal pois acarreta o conceito de ausência de informação. Como observado por Boldrin e Saffiotti [38], isto contrasta com a metodologia algébrica de outras lógicas onde a negação representa informação positiva em algumas fórmulas "ortogonais". Novamente,

isto é bem ilustrado com o MIN TSP , nosso exemplo favorito. Uma crença forte de que um subconjunto de $O(n)$ arestas estão na solução ótima *não significa necessariamente* que o conjunto de $O(n^2)$ arestas restantes não devam estar na solução ótima. A Lógica Modal parece ter um papel interessante neste respeito. Finalmente, de acordo com Boldrin e Saffiotti [38], $\mathbf{PL}_n^{\otimes}$ pode ser estendido para também incluir um conjunto de operadores epistêmicos D^G, com G sendo um subconjunto de agentes. O significado intencional disto é que eles combinarão a crença distribuída de subconjuntos dos agentes. Este trabalho está em progresso e pode ser importante para dirigir a recombinação de crenças em MAs de estrutura hierárquica tal como proposto em [34].

9.1.8 O *MemePool Project*: nosso lugar de encontro para reuso de algoritmos e de conhecimento

Como dissemos antes, acreditamos que estas diferentes direções de pesquisa podem ser combinadas e podem realizar uma notável contribuição quando usadas juntas.

Um interessante exemplo destes novos MAs, é um recente trabalho de Lamma, Pereira, e Riguzzi [218] aplicado para o campo de diagnóstico de circuitos digitais. Em sua metodologia, eles diferenciam entre genes e *"memes"*. Este último grupo codifica as crenças entre os agentes e suas suposições. Usando uma técnica baseada em lógica, eles modificam os memes de acordo com o nível em que as crenças presentes são contraditórias por restrições de integridade que expressam observações e leis.

Cada agente mantém uma população de cromossomos e encontra uma solução para o problema de revisão de crenças por meio de um algoritmo genético. Um operador *"lamarckiano"* é usado para modificar um cromossomo usando mutações dirigidas para a revisão de crenças, orientadas segundo derivações lógicas. Como conseqüência, um cromossomo satisfará um número maior de restrições. A evolução provida pelos operadores *"darwinianos"*, permite aos agentes melhorarem os cromossomos ganhando parte da experiência de outros agentes. É central a esta metodologia o operador lamarckiano chamado apropriadamente *"Learn"*. Ele pega um cromossomo e produz um cromossomo revisado como saída. Para conseguir isto, elimina alguns caminhos de derivação que alcançam alguma contradição.

Surpreendentemente (e aqui observamos a possibilidade de usar a teoria de *tratabilidade de parámetro fixo*), a aprendizagem é obtida encontrando um *"hitting set"* que não é necessariamente mínimo. Os autores deixam claro este ponto dizendo que: *"um 'hitting set' gerado a partir destes conjuntos suportes não é necessariamente um conjunto de remoção de contradições e portanto não é uma solução para o problema de revisão de crença."* Os autores não devem estar cientes do algoritmo exato $O(2,311^k + n)$ para MIN HITTING SET FOR SIZE THREE SETS. Eles poderiam até ser capazes de usá-lo de alguma maneira, mas o assunto é anedótico no momento. O que é importante observar é que esses algoritmos podem ser usados *"fora da caixa"* se um *"framework"* de algoritmos apropriados, baseado em um sistema de códigos fonte abertos, fosse criado.

Por outro lado, observamos como resultados de Programação Lógica e de revisão de crenças podem ajudar ao estado atual das meta-heurísticas. A situação atual, em que os pesquisadores propoẽm novos nomes para as mesmas técnicas básicas, e a maioria das contribuições é somente a adição de novos parâmetros para guiar a busca, é uma direção de busca fútil. Pode ser possível que MAs guiados por Busca por Crenças podem ser uma ferramenta válida para ajudar a sistematizar o cenário. Em particular, a discussão seria baseada em qual *lógica* de múltiplos agentes funciona melhor ao invés de qual *parâmetro* trabalha melhor para problemas específicos ou instâncias. Finalmente, esperamos convencer pesquisadores de Programação Lógica para começar a tratar destes aspectos, tendo a tarefa de guiar MAs para otimização combinatória de larga escala, ao invés de só resolver problemas ilustrativos pequenos, que geralmente são apresentados nos artigos. O projeto *MemePool*[2] parece ser um bom cenário para um trabalho internacional interdisciplinar deste tipo. Se funcionará ou não depende mais de nossa habilidade de trabalhar cooperativamente, do que qualquer outra coisa.

[2] `http://www.densis.fee.unicamp.br/~moscato/memepool.html`

Bibliografia

[1] H.S. Abdinnour. A hybrid heuristic for the uncapacitated hub location problem. *European Journal of Operational Research*, 106(2-3):489–99, 1998.

[2] K.A. Abrahamson, R.G. Downey, and M.R. Fellows. Fixed-parameter tractability and completeness IV: On completeness for W[P] and PSPACE analogues. *Annals of Pure and Applied Logic*, 73(3):235–276, Jun. 1995.

[3] C.C. Aggarwal, J.B. Orlin, and R.P. Tai. Optimized crossover for the independent set problem. *Operations Research*, 45(2):226–234, 1997.

[4] J. Aguilar and A. Colmenares. Resolution of pattern recognition problems using a hybrid genetic/random neural network learning algorithm. *Pattern Analysis and Applications*, 1(1):52–61, 1998.

[5] A.V. Aho, D.S. Johnson, R.M. Karp, S.R. Kosaraju, C.C. McGeogh, C.H. Papadimitriou, and P. Pevzner. Theory of computing: Goals and directions. Technical Report CSE-96-03-03, University of Washington, 1996. Available at the following URL: `ftp://ftp.cs.washington.edu/tr/1996/03/UW-CSE-96-03-03.PS.Z`.

[6] J. Alber, H.L. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory (SWAT'2000) Bergen, Norway, July 2000*, volume 1851 of *Lecture Notes in Computer Science*, pages 97–110. Springer, 2000. expanded version submitted to *Algorithmica*.

[7] D. Aldous and U. Vazirani. "Go with the winners" algorithms. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pages 492–501, Los Alamitos, CA, 1994. IEEE.

[8] P. Alimonti. New local search approximation techniques for maximum generalized satisfiability problems. *Information Processing Letters*, 57(3):151–158, February 1996.

[9] S. Amin. Simulated jumping. *Annals of Operations Research*, 86:23–38, 1999.

[10] M.F. Arguello, T.A. Feo, and O. Goldschmidt. Randomized methods for the number partitioning problem. *Computers & Operations Research*, 23(2):103–111, Feb. 1996.

[11] S. Arora and C. Lund. Hardness of approximations. In D.S. Hochbaum, editor, *Approximation Algorithms for NP-Hard Problems*, chapter 10, pages 399–446. PWS, 1995.

[12] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation algorithms. In *Proceedings of the 33rd IEEE Symposium on the Foundations of Computer Science*, pages 13–22, 1992.

[13] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *Journal of the ACM*, 45(3):501–555, May 1998.

[14] A. Augugliaro, L. Dusonchet, and E. Riva-Sanseverino. Service restoration in compensated distribution networks using a hybrid genetic algorithm. *Electric Power Systems Research*, 46(1):59–66, 1998.

[15] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation: Combinatorial optimization problems and their approximability properties.* Springer Verlag, November 1999.

[16] G. Ausiello, A. D'Atri, and M. Protasi. Structure preserving reductions among convex optimization problems. *J. Comput. System Sci.*, 21:136–153, 1980.

[17] G. Ausiello and M. Protasi. Local search, reducibility and approximability of NP-optimization problems. *Information Processing Letters*, 54(2):73–79, 1995.

[18] R. Axelrod and W.D. Hamilton. The evolution of cooperation. *Science*, 211(4489):1390–1396, 1981.

[19] K. Aygun, D.S. Weile, and E. Michielssen. Design of multilayered periodic strip gratings by genetic algorithms. *Microwave and Optical Technology Letters*, 14(2):81–85, 1997.

[20] T. Bäck and F. Hoffmeister. Adaptive search by evolutionary algorithms. In W. Ebeling, M. Peschel, and W. Weidlich, editors, *Models of Selforganization in Complex Systems*, number 64 in Mathematical Research, pages 17–21. Akademie-Verlag, 1991.

[21] Th. Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York, 1996.

[22] Th. Bäck, D.B. Fogel, and Z. Michalewicz. *Handbook of Evolutionary Computation.* Oxford University Press, New York NY, 1997.

[23] B.S. Baker. Approximation algorithms for NP-complete problems on planar graphs. In *Proc. 24th Ann. IEEE Symp. on Foundations of Comput. Sci.*, pages 265–273, 1983.

[24] E. Balas and P. Toth. Branch-and bound methods. In E.L. Lawler, J.K. Lentra, A.H.G. Rinnoy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem*. John Wiley and Sons, New York, 1985.

[25] R. Balasubramanian, M.R. Fellows, and V. Raman. An improved fixed-parameter algorithm for vertex cover. *Information Processing Letters*, 65(3):163–168, 1998.

[26] M.J. Bayley, G. Jones, P. Willett, and M.P. Williamson. Genfold: A genetic algorithm for folding protein structures using NMR restraints. *Protein Science*, 7(2):491–499, 1998.

[27] C. Bazgan, M. Santha, and Zs. Tuza. On the approximation of finding (a)nother Hamiltonian cycle in cubic Hamiltonian graphs. In *Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science*, volume 1373, pages 276–286, 1998. Lecture Notes in Computer Science.

[28] J. Beasley and P.C. Chu. A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):393–404, 1996.

[29] J. Beasley and P.C. Chu. A genetic algorithm for the multidimensional knapsack problem. *Journal of Heuristics*, 4:63–86, 1998.

[30] B. Becker and R. Drechsler. Ofdd based minimization of fixed polarity Reed-Muller expressions using hybrid genetic algorithms. In *Proceedings IEEE International Conference on Computer Design: VLSI in Computers and Processor*, pages 106–110, Los Alamitos, CA, 1994. IEEE.

[31] J.L. Bentley. Experiments on traveling salesman heuristics. In *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 91–99, January 1990.

[32] J. Berger, M. Salois, and R. Begin. A hybrid genetic algorithm for the vehicle routing problem with time windows. In R.E. Mercer and E. Neufeld, editors, *Advances in Artificial Intelligence. 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, pages 114–127, Berlin, 1998. Springer-Verlag.

[33] P. Berman and G. Schnitger. On the complexity of approximating the independent set problem. In *Proc. 6th Ann. Symp. on Theoretical Aspects of Comput. Sci.*, pages 256–268. Springer-Verlag, 1989. Lecture Notes in Comput. Sci. 349.

[34] R. Berretta and P. Moscato. The number partitioning problem: An open challenge for evolutionary computation ? In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 261–278. McGraw-Hill, 1999.

[35] H. Bodlaender, R. Downey, M. Fellows, M. Hallett, and H.T. Wareham. Parameterized complexity analysis in computational biology. *Computer Applications in the Biosciences*, 11:49–57, 1995.

[36] H.L. Bodlaender. Treewidth: Algorithmic techniques and results. In *Proceedings of the 22nd International Symposium on Mathematical Foundations of Computer Science, MFCS'97*, volume 1295, pages 29–37, Berlin, Germany, 1997. Springer-Verlag. Lecture Notes in Computer Science.

[37] K.D. Boese. Cost versus Distance in the Traveling Salesman Problem. Technical Report TR-950018, UCLA CS Department, 1995.

[38] L. Boldrin and A. Saffiotti. A modal logic for merging partial belief of multiple reasoners. *Journal of Logic and Computation*, 9(1):81–103, 1999. Online at `http://www.aass.oru.se/~asaffio/`.

[39] M. Bonet, M. Steel, T. Warnow, and S. Yooseph. Better methods for solving parsimony and compatibility. In Sorin Istrail, Pavel Pevzner, and Michael Waterman, editors, *Proceedings of the 2nd Annual International Conference on Computational Molecular Biology (RECOMB-98)*, pages 40–49, New York, March22–25 1998. ACM Press.

[40] A.H.W. Bos. Aircraft conceptual design by genetic/gradient-guided optimization. *Engineering Applications of Artificial Intelligence*, 11(3):377–382, 1998.

[41] D. Brown, C. Huntley, and A. Spillane. A Parallel Genetic Heuristic for the Quadratic Assignment Problem. In J. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 406–415. Morgan Kaufmann, 1989.

[42] D. Bruschi, D. Joseph, and P. Young. A structural overview of NP optimization problems. In *Proc. Optimal Algorithms*, volume 401, pages 205–231. Springer-Verlag, 1989. Lecture Notes in Comput. Sci.

[43] T.N. Bui and B.R. Moon. Genetic algorithm and graph partitioning. *IEEE Transactions on Computers*, 45(7):841–855, 1996.

[44] T.N. Bui and B.R. Moon. GRCA: A hybrid genetic algorithm for circuit ratio-cut partitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(3):193–204, 1998.

[45] L.S. Buriol, P.M. França, and P. Moscato. A New Memetic Algorithm for the Asymmetric Traveling Salesman Problem. Submitted to *Journal of Heuristics* (October/1999), under revision.

[46] L.S. Buriol, P.M. França, and P. Moscato. New heuristic results for the asymmetric traveling salesman problem. Abstract accepted to *4th International Conference on Operations Research Optimization. Havana, March 6-10*. Organized by Universidad de La Habana and Humboldt-Universität zu Berlin.

[47] L.S. Buriol, P.M. França, and P. Moscato. Algoritmo Memético para o Problema do Caixeiro Viajante. In *Proceedings of XXXI SOBRAPO - Simpósio Brasileiro de Pesquisa Operacional, Juiz de Fora, MG, Brasil, 20-22 de Octobre*, pages 1015–1028. Sociedade Brasileira de Pesquisa Operacional, 1999.

[48] E. K. Burke, D. G. Elliman, and R. F. Weare. A hybrid genetic algorithm for highly constrained timetabling problems. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 605–610. Morgan Kaufmann, San Francisco, CA, 1995.

[49] E.K. Burke, K.S. Jackson, J.H. Kingston, and R.F. Weare. Automated timetabling: The state of the art. *The Computer Journal*, 40(9):565–571, 1997.

[50] E.K. Burke and J.P. Newall. A phased evolutionary approach for the timetable problem: An initial study. In *Proceedings of the ICONIP/ANZIIS/ANNES '97 Conference*, pages 1038–1041, Dunedin, New Zealand, 1997. Springer-Verlag.

[51] E.K. Burke and J.P. Newall. A multi-stage evolutionary algorithm for the timetable problem. *IEEE Transactions on Evolutionary Computation*, 3(1):63–74, 1999.

[52] E.K. Burke, J.P. Newall, and R.F. Weare. A memetic algorithm for university exam timetabling. In E.K. Burke and P. Ross, editors, *The Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 241–250. Springer-Verlag, 1996.

[53] E.K. Burke, J.P. Newall, and R.F. Weare. Initialisation strategies and diversity in evolutionary timetabling. *Evolutionary Computation*, 6(1):81–103, 1998.

[54] E.K. Burke and A.J. Smith. A memetic algorithm for the maintenance scheduling problem. In *Proceedings of the ICONIP/ANZIIS/ANNES '97 Conference*, pages 469–472, Dunedin, New Zealand, 1997. Springer-Verlag.

[55] E.K. Burke and A.J. Smith. A memetic algorithm to schedule grid maintenance. In *Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation, Vienna: Evolutionary Computation and Fuzzy Logic for Intelligent Control, Knowledge Acquisition and Information Retrieval*, pages 122–127. IOS Press 1999, 1999.

[56] E.K. Burke and A.J. Smith. A multi-stage approach for the thermal generator maintenance scheduling problem. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1085–1092, Washington D.C., 1999. IEEE.

[57] J.F. Buss and J. Goldsmith. Nondeterminism within P. *SIAM Journal on Computing*, 22:560–572, 1993.

[58] S. Cadieux, N. Tanizaki, and T. Okamura. Time efficient and robust 3-D brain image centering and realignment using hybrid genetic algorithm. In *Proceedings of the 36th SICE Annual Conference*, pages 1279–1284. IEEE, 1997.

[59] L. Cai and J. Chen. On fixed-parameter tractability and approximability of NP optimization problems. *J. Comput. Syst. Sci.*, 54:465–474, 1997.

[60] L. Cai, J. Chen, R.G. Downey, and M.R. Fellows. On the parameterized complexity of short computation and factorization. *Archives for Math. Logic*, 36:321–337, 1997.

[61] K. Cameron and J. Edmonds. Existentially polytime theorems. *DIMACS Series Discrete Mathematics and Theoretical Computer Science*, pages 83–99, 1990.

[62] G. Carpaneto, M. DellAmico, and P. Toth. Exact solution of large-scale, asymmetric traveling salesman problems. *ACM Transactions on Mathematical Software*, 21(4):394–409, 1995.

[63] J. Carrizo, F.G. Tinetti, and P. Moscato. A computational ecology for the quadratic assignment problem. In *Proceedings of the 21st Meeting on Informatics and Operations Research*, Buenos Aires, 1992. SADIO.

[64] S. Cavalieri and P. Gaiardelli. Hybrid genetic algorithms for a multiple-objective scheduling problem. *Journal of Intelligent Manufacturing*, 9(4):361–367, 1998.

[65] M. Cesati and L. Trevisan. On the efficiency of polynomial time approximation schemes. *Information Processing Letters*, 64:165–171, 1997.

[66] N. Chaiyaratana and A.M.S. Zalzala. Hybridisation of neural networks and genetic algorithms for time-optimal control. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 389–396, Washington D.C., 1999. IEEE.

[67] L.D. Chambers, editor. *Practical Handbook of Genetic Algorithms: Vol.1 Applications*. CRC Press, Boca Ratón FL, 1995.

[68] J. Chen, I.A. Kanj, and W. Jia. Vertex cover: further observations and further improvements. In *Proc. 25th Int. Worksh. Graph-Theoretic Concepts in Computer Science*, number 1665 in Lecture Notes in Computer Science, pages 313–324. Springer-Verlag, 1999.

[69] R. Cheng and M. Gen. Parallel machine scheduling problems using memetic algorithms. In *1996 IEEE International Conference on Systems, Man and Cybernetics. Information Intelligence and Systems*, volume 4, pages 2665–2670, New York, NY, 1996. IEEE.

[70] R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms. II. Hybrid genetic search strategies. *Computers & Industrial Engineering*, 37(1-2):51–55, 1999.

[71] R.W. Cheng and M. Gen. Parallel machine scheduling problems using memetic algorithms. *Computers & Industrial Engineering*, 33(3–4):761–764, 1997.

[72] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical report, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, 1976.

[73] P.C. Chu and J. Beasley. A genetic algorithm for the generalised assignment problem. *Computers & Operations Research*, 24:17–23, 1997.

[74] D.E. Clark and D.R. Westhead. Evolutionary algorithms in computer-aided molecular design. *Journal of Computer-aided Molecular Design*, 10(4):337–358, 1996.

[75] H.G. Cobb and J.J. Grefenstette. Genetic algorithms for tracking changing environments. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 529–530, San Mateo, CA, 1993. Morgan Kaufmann.

[76] E.G. Coffman, D.S. Johnson, G.S. Lueker, and P.W. Shor. Probabilistic analysis of packing and related partitioning problems. *Statistical Science*, 8(1):40–47, 1993.

[77] P.E. Coll, G.A. Durán, and P. Moscato. On worst-case and comparative analysis as design principles for efficient recombination operators: A graph coloring case study. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 279–294. McGraw-Hill, 1999.

[78] D. Costa. An evolutionary tabu search algorithm and the NHL scheduling problem. *INFOR*, 33(3):161–178, 1995.

[79] D. Costa, N. Dubuis, and A. Hertz. Embedding of a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *Journal of Heuristics*, 1(1):105–128, 1995.

[80] C. Cotta. A study of hybridisation techniques and their application to the design of evolutionary algorithms. *AI Communications*, 11(3-4):223–224, 1998.

[81] C. Cotta. A study of hybridisation techniques and their application to the design of evolutionary algorithms. *AI Communications*, 11(3-4):223–224, 1998.

[82] C. Cotta, E. Alba, and J.M. Troya. Utilising dynastically optimal forma recombination in hybrid genetic algorithms. In A.E. Eiben, Th. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature V*, volume 1498 of *Lecture Notes in Computer Science*, pages 305–314. Springer-Verlag, Berlin, 1998.

[83] C. Cotta, E. Alba, and J.M. Troya. Stochastic reverse hillclimbing and iterated local search. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1558–1565, Washington D.C., 1999. IEEE.

[84] C. Cotta and J.M. Troya. Genetic forma recombination in permutation flowshop problems. *Evolutionary Computation*, 6(1):25–44, 1998.

[85] C. Cotta and J.M. Troya. A hybrid genetic algorithm for the 0-1 multiple knapsack problem. In G.D. Smith, N.C. Steele, and R.F. Albrecht, editors, *Artificial Neural Nets and Genetic Algorithms 3*, pages 251–255, Wien New York, 1998. Springer-Verlag.

[86] C. Cotta and J.M. Troya. On the influence of the representation granularity in heuristic forma recombination. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *ACM Symposium on Applied Computing 2000*, pages 433–439. ACM Press, 2000.

[87] C. Cotta and J.M. Troya. Using a hybrid evolutionary-A* approach for learning reactive behaviors. In S. Cagnoni et al., editors, *Real-World Applications of Evolutionary Computation*, volume 1803 of *Lecture Notes in Computer Science*, pages 347–356, Edinburgh, 15-16 April 2000. Springer-Verlag.

[88] T. Crain, R. Bishop, W. Fowler, and K. Rock. Optimal interplanetary trajectory design via hybrid genetic algorithm/recursive quadratic program search. In *Ninth AAS/AIAA Space Flight Mechanics Meeting*, pages 99–133, Breckenridge CO, 1999.

[89] P. Crescenzi and V. Kann. A compendium of NP optimization problems. Technical Report SI/RR-95/02, Dipartimento di Scienze dell'Informazione, Università di Roma "La Sapienza", 1995. The list is updated continuously. The latest version is available on the web as `http://www.nada.kth.se/~viggo/problemlist/`.

[90] P. Crescenzi and A. Panconesi. Completeness in approximation classes. *Inform. and Comput.*, 93(2):241–262, 1991.

[91] T. Dandekar and P. Argos. Identifying the tertiary fold of small proteins with different topologies from sequence and secondary structure using the genetic algorithm and extended criteria specific for strand regions. *Journal of Molecular Biology*, 256(3):645–660, 1996.

[92] B. DasGupta and B. Hammer. Hardness of approximation of the loading problem for multi-layered feedforward neural nets. Technical Report 99-60, DIMACS, November 29 1999.

[93] Y. Davidor and O. Ben-Kiki. The interplay among the genetic algorithm operators: Information theory tools used in a holistic way. In R. Männer and B. Manderick, editors, *Parallel Problem Solving From Nature II*, pages 75–84, Amsterdam, 1992. Elsevier Science Publishers B.V.

[94] S. Davies and S. Russell. NP-completeness of searches for smallest possible feature sets. In R. Greiner and D. Subramanian, editors, *AAAI Symposium on Intelligent Relevance*, pages 41–43, New Orleans, 1994. AAAI Press.

[95] L. Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold Computer Library, New York, 1991.

[96] R. Dawkins. *The Selfish Gene*. Clarendon Press, Oxford, 1976.

[97] P. de Causmaecker, G. van den Berghe, and E.K. Burke. Using tabu search as a local heuristic in a memetic algorithm for the nurse rostering problem. In *Proceedings of the Thirteenth Conference on Quantitative Methods for Decision Making*, pages abstract only, poster presentation, Brussels, Belgium, 1999.

[98] P.A. de Souza, R. Garg, and V.K. Garg. Automation of the analysis of Mossbauer spectra. *Hyperfine Interactions*, 112(1–4):275–278, 1998.

[99] D.M. Deaven and K.O. Ho. Molecular-geometry optimization with a genetic algorithm. *Physical Review Letters*, 75(2):288–291, 1995.

[100] D.M. Deaven, N. Tit, J.R. Morris, and K.M. Ho. Structural optimization of Lennard-Jones clusters by a genetic algorithm. *Chemical Physics Letters*, 256(1–2):195–200, 1996.

[101] N. Dellaert and J. Jeunet. Solving large unconstrained multilevel lot-sizing problems using a hybrid genetic algorithm. *International Journal of Production Research*, 38(5):1083–1099, 2000.

[102] J.R. Desjarlais and T.M. Handel. New strategies in protein design. *Current Opinion in Biotechnology*, 6(4):460–466, 1995.

[103] R. Doll and M.A. VanHove. Global optimization in LEED structure determination using genetic algorithms. *Surface Science*, 355(1-3):L393–L398, 1996.

[104] R. Dorne and J.K. Hao. A new genetic local search algorithm for graph coloring. In A.E. Eiben, Th. Bäck, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature V*, volume 1498 of *Lecture Notes in Computer Science*, pages 745–754, Berlin, 1998. Springer-Verlag.

[105] R.G. Downey and M.R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, Aug 1995.

[106] R.G. Downey and M.R. Fellows. Parameterized computational feasibility. In *P. Clote, J. Remmel (eds.): Feasible Mathematics II*, pages 219–244. Boston: Birkhäuser, 1995.

[107] R.G. Downey and M.R. Fellows. *Parameterized Complexity*. Springer-Verlag, New York, NY, USA, 1997.

[108] R.G. Downey, M.R. Fellows, and U. Stege. Computational Tractability: The View From Mars. *Bulletin of the European Association for Theoretical Computer Science*, 69:73–97, October 1999.

[109] R.G. Downey, M.R. Fellows, and U. Stege. Parameterized Complexity: A framework for systematically confronting computational intractability. In *Contemporary Trends in Discrete Mathematics: Form DIMACS to DIMATIA to the future*, AMS-DIMACS Proceedings Series, pages 49–99. AMS, 1999.

[110] R. Englemore and T. Morgan (eds.). *Blackboard Systems*. Addison-Wesley, 1988.

[111] C. Ersoy and S.S. Panwar. Topological design of interconnected LAN/MAN networks. *IEEE Journal on Selected Areas in Communications*, 11(8):1172–1182, 1993.

[112] J. Fang and Y. Xi. A rolling horizon job shop rescheduling strategy in the dynamic environment. *International Journal of Advanced Manufacturing Technology*, 13(3):227–232, 1997.

[113] M.R. Fellows. On the complexity of vertex set problems. Technical report, University of New Mexico, 1988.

[114] M.R. Fellows and M.A. Langston. Nonconstructive advances in polynomial-time complexity. *Information Processing Letters*, 28:157–162, 1987/88.

[115] F.F. Ferreira and J.F. Fontanari. Probabilistic analysis of the number partitioning problem. *Journal of Physics A: Math. Gen*, pages 3417–3428, 1998.

[116] C.N. Fiechter. A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics*, 51:243–267, 1994.

[117] M. Fischetti and S. Martello. Worst-case analysis of the differencing method for the partition problem. *Mathematical Programming*, 37(1):117–120, 1987.

[118] M. Fischetti and P. Toth. A polyhedral approach to the asymmetric traveling salesman problem. *Management Science*, 43(11):1520–1536, 1997.

[119] M. Fischetti, P. Toth, and D. Vigo. A Branch-and-Bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research*, 42:846–859, 1994.

[120] C. Fleurent and J.A. Ferland. Genetic and hybrid algorithms for graph coloring. *Annals of Operations Research*, 63:437–461, 1997.

[121] P.M. França, J.D. Gupta, A.S. Mendes, P. Moscato, and K.J. Veltink. Novos resultados para o problema de flowshop com famílias de tarefas e tempos de preparação. In *Proceedings of XXXI SOBRAPO - Simpósio*

Brasileiro de Pesquisa Operacional, Juiz de Fora, MG, Brasil, 20-22 de Octobre. Sociedade Brasileira de Pesquisa Operacional, 1999.

[122] P.M. França, J.D. Gupta, A.S. Mendes, P. Moscato, and K.J. Veltink. Metaheuristic Approaches for the Pure Flowshop Manufacturing Cell Problem. In *PMS2000 - 7th International Workshop On Project Management And Scheduling, Osnabrck, Germany*, April 2000. extended abstract.

[123] P.M. França, A.S. Mendes, and P. Moscato. A Memetic Algorithm for the Total Tardiness Single Machine Scheduling Problem. Accepted in *European Journal of Operations Research*.

[124] P.M. França, A.S. Mendes, and P. Moscato. Algoritmos Meméticos e o Sequenciamento em Máquina Simples com Setup Times e Restrições de Datas de Entrega. In *XXX SOBRAPO - Simpósio Brasileiro de Pesquisa Operacional, Curitiba, PR, Brasil, 25-27 de Novembro*, pages 315–316. Sociedade Brasileira de Pesquisa Operacional, 1998. extended abstract.

[125] P.M. França, A.S. Mendes, and P. Moscato. Memetic algorithms to minimize tardiness on a single machine with sequence-dependent setup times. In *Proceedings of the 5th International Conference of the Decision Sciences Institute, Athens, Greece*, pages 1708–1710, 1999.

[126] P.M. França, P. Moscato, F. Müller, A.S. Mendes, and L.S. Buriol. O Projeto MemePool: Um Framework para Otimização Combinatória. In *XXX SOBRAPO - Simpósio Brasileiro de Pesquisa Operacional, Curitiba, PR, Brasil, 25-27 de Novembro*, pages 20–21. Sociedade Brasileira de Pesquisa Operacional, 1998. extended abstract.

[127] B. Freisleben and P. Merz. A Genetic Local Search Algorithm for Solving Symmetric and Asymmetric Traveling Salesman Problems. In *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, Nagoya, Japan*, pages 616–621. IEEE Press, 1996.

[128] B. Freisleben and P. Merz. New Genetic Local Search Operators for the Traveling Salesman Problem. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature IV*, volume 1141 of *Lecture Notes in Computer Science*, pages 890–900. Springer, 1996.

[129] R.T. Fu, K. Esfarjani, Y. Hashi, J. Wu, X. Sun, and Y. Kawazoe. Surface reconstruction of Si (001) by genetic algorithm and simulated annealing method. *Science Reports of The Research Institutes Tohoku University Series A-Physics Chemistry And Metallurgy*, 44(1):77–81, Mar. 1997.

[130] B.L. Garcia, P. Mahey, and L.J. LeBlanc. Iterative improvement methods for a multiperiod network design problem. *European Journal of Operational Research*, 110(1):150–165, 1998.

[131] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, San Francisco, 1979.

[132] M. Gen and R. Cheng. *Genetic Algorithms and Engineering Design*. Wiley Series in Engineering Design and Automation. John Wiley & Sons (Sd), 1997.

[133] M. Gen, K. Ida, and L. Yinzhen. Bicriteria transportation problem by hybrid genetic algorithm. *Computers & Industrial Engineering*, 35(1-2):363–366, 1998.

[134] I.P. Gent and T. Walsh. Analysis of heuristics for number partitioning. *Computational Intelligence*, 14(3):430–451, 1998.

[135] F. Glover, G. Gutin, A. Yeo, and A. Zverovich. Construction heuristics and domination analysis for the asymmetric TSP. In J.S. Vitter and C.D. Zaroliagis, editors, *Proceedings of the Algorithm Engineering: 3rd International Workshop, London, UK, Lecture Notes in Computer Science Vol. 1668*, pages 85–94. Springer-Verlag, 1999.

[136] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, Massachusetts, USA, 1997.

[137] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, Massachusetts, 1989.

[138] A.J. Goldstein and A.B. Lesk. Common feature techniques for discrete optimization. Comp. Sci. Tech. Report 27, Bell. Tel. Labs, March 1975.

[139] M. Gorges-Schleuter. ASPARAGOS: An asynchronous parallel genetic optimization strategy. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 422–427. Morgan Kaufmann Publishers, 1989.

[140] M. Gorges-Schleuter. Explicit Parallelism of Genetic Algorithms through Population Structures. In H.-P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature*, pages 150–159. Springer-Verlag, 1991.

[141] M. Gorges-Schleuter. *Genetic Algorithms and Population Structures - A Massively Parallel Algorithm*. PhD thesis, University of Dortmund, Germany, 1991.

[142] M. Gorges-Schleuter. Asparagos96 and the traveling salesman problem. In Th. Bäck, Z. Michalewicz, and X. Yao, editors, *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation, Indianapolis, USA*, pages 171–174. IEEE Press, 1997.

[143] J. Gottlieb. Permutation-based evolutionary algorithms for multidimensional knapsack problems. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *ACM Symposium on Applied Computing 2000*, pages 408–414. ACM Press, 2000.

[144] J. Gottlieb and T. Kruse. Selection in evolutionary algorithms for the traveling salesman problem. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *ACM Symposium on Applied Computing 2000*, pages 415–421. ACM Press, 2000.

[145] J.J. Grefenstette. Incorporating Problem Specific Knowledge into Genetic Algorithms. In L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, Research Notes in Artificial Intelligence, pages 42–60. Morgan Kaufmann Publishers, 1987.

[146] M. Grigni, E. Koutsoupias, and C.H. Papadimitriou. An approximation scheme for planar graph TSP. In *Proc. 36th Ann. IEEE Symp. on Foundations of Comput. Sci.*, pages 640–645, Milwaukee, Wisconsin, 23–25 October 1995.

[147] P. Grim. The undecidability of the spatialized prisoner's dilemma. *Theory and Decision*, 42(1):53–80, 1997.

[148] J.B. Grimbleby. Hybrid genetic algorithms for analogue network synthesis. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1781–1787, Washington D.C., 1999. IEEE.

[149] J.R. Gunn. Sampling protein conformations using segment libraries and a genetic algorithm. *Journal of Chemical Physics*, 106(10):4270–4281, 1997.

[150] M. Guotian and L. Changhong. Optimal design of the broadband stepped impedance transformer based on the hybrid genetic algorithm. *Journal of Xidian University*, 26(1):8–12, 1999.

[151] G. Gutin and A. Yeo. Polynomial approximation algorithms for the TSP and the QAP with a factorial domination number. *Discrete Applied Mathematics*, 1999. to appear.

[152] O.C.L. Haas, K.J. Burnham, and J.A. Mills. Optimization of beam orientation in radiotherapy using planar geometry. *Physics in Medicine and Biology*, 43(8):2179–2193, 1998.

[153] O.C.L. Haas, K.J. Burnham, J.A. Mills, C.R. Reeves, and M.H. Fisher. Hybrid genetic algorithms applied to beam orientation in radiotherapy. In *Fourth European Congress on Intelligent Techniques and Soft Computing Proceedings*, volume 3, pages 2050–2055, Aachen, Germany, 1996. Verlag Mainz.

[154] A.B. Hadj-Alouane, J.C. Bean, and K.G. Murty. A hybrid genetic/optimization algorithm for a task allocation problem. *Journal of Scheduling*, 2(4), 1999.

[155] E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. In *Proceedings of the 11th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 329–337. ACM-SIAM, 2000.

[156] S.P. Harris and E.C. Ifeachor. Automatic design of frequency sampling filters by hybrid genetic algorithm techniques. *IEEE Transactions on Signal Processing*, 46(12):3304–3314, 1998.

[157] W.E. Hart and R.K. Belew. Optimizing an arbitrary function is hard for the genetic algorithm. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 190–195, San Mateo CA, 1991. Morgan Kaufmann.

[158] B. Hartke. Global geometry optimization of clusters using genetic algorithms. *Journal of Physical Chemistry*, 97(39):9973–9976, 1993.

[159] R. Hassin and N. Meggido. Approximation algorithms for hitting objects with straight lines. *Discrete Applied Mathematics*, 30:29–42, 1991.

[160] J. Håstad. Recent results in hardness of approximation. In Erik M. Schmidt and Sven Skyum, editors, *Algorithm Theory—SWAT '94: 4th Scandinavian Workshop on Algorithm Theory*, volume 824 of *Lecture Notes in Computer Science*, pages 231–239, Aarhus, Denmark, 6–8 July 1994. Springer-Verlag.

[161] J. Håstad. Some optimal inapproximability results. In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 1–10. ACM, 1997.

[162] M. Hifi. A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems. *Journal of the Operational Research Society*, 48(6):612–622, 1997.

[163] R. Hirsch and C.C. Mullergoymann. Fitting of diffusion-coefficients in a 3-compartment sustained-release drug formulation using a genetic algorithm. *International Journal of Pharmaceutics*, 120(2):229–234, 1995.

[164] K.M. Ho, A.A. Shvartsburg, B.C. Pan, Z.Y. Lu, C.Z. Wang, J.G. Wacker, J.L. Fye, and M.F. Jarrold. Structures of medium-sized silicon clusters. *Nature*, 392(6676):582–585, 1998.

[165] C.A.R. Hoare. Partition (algorithm 63), quicksort (algorithm 64), and find (algorithm 65). *Communications of the ACM*, 4(7):321–322, 1961.

[166] S. Hobday and R. Smith. Optimisation of carbon cluster geometry using a genetic algorithm. *Journal of The Chemical Society-Faraday Transactions*, 93(22):3919–3926, 1997.

[167] D.S. Hochbaum. *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, USA, 1997.

[168] R.J.W. Hodgson. Memetic algorithms and the molecular geometry optimization problem. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 625–632, Piscataway, NJ, 2000. IEEE Service Center.

[169] R. Hofmann. Parallel evolutionary trajectories. Research Report SS92-11, Edinburgh Parallel Computing Centre, 1992.

[170] R. Hofmann. Examinations on the algebra of genetic algorithms. Master's thesis, Technische Universität München, Institut für Informatik, 1993.

[171] D.R. Hofstadter. Computer tournaments of the prisoners-dilemma suggest how cooperation evolves. *Scientific American*, 248(5):16–23, 1983.

[172] D. Holstein and P. Moscato. Memetic algorithms using guided local search: A case study. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 235–244. McGraw-Hill, 1999.

[173] E. Hopper and B. Turton. A genetic algorithm for a 2d industrial packing problem. *Computers & Industrial Engineering*, 37(1-2):375–378, 1999.

[174] M. Hulin. An optimal stop criterion for genetic algorithms: A bayesian approach. In Th. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 135–143, San Mateo, CA, 1997. Morgan Kaufmann.

[175] O. H. Ibarra and C. E. Kim. Fast approximation for the knapsack and sum of subset problems. *J. ACM*, 22(4):463–468, 1975.

[176] T. Ichimura and Y. Kuriyama. Learning of neural networks with parallel hybrid GA using a Royal Road function. In *1998 IEEE International Joint Conference on Neural Networks*, volume 2, pages 1131–1136, New York, NY, 1998. IEEE.

[177] W.R. Jih and Y.J. Hsu. Dynamic vehicle routing using hybrid genetic algorithms. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 453–458, Washington D.C., 1999. IEEE.

[178] D.S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, 9:256–278, 1974.

[179] D.S. Johnson. The NP-completeness column: An ongoing guide: The many faces of polynomial time. *Journal of Algorithms*, 8(2):285–303, 1987.

[180] D.S. Johnson. Local optimization and the traveling salesman problem. In *Proc. ICALP-90*, pages 446–461. Springer-Verlag, 1990. Lecture Notes in Comput. Sci. 443.

[181] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II: Graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991.

[182] D.S. Johnson and L.A. McGeoch. The traveling salesman problem: A case study. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 215–310. Wiley, Chichester, 1997.

[183] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis. How easy is local search ? *Journal of Computers and System Sciences*, 37:79–100, 1988.

[184] D.R. Jones and M.A. Beltramo. Solving partitioning problems with genetic algorithms. In R.K Belew and L.B. Booker, editors, *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 442–449, San Mateo, CA, USA, 1991. Morgan Kaufmann.

[185] G. Jones, P. Willett, R.C. Glen, A.R. Leach, and R. Taylor. Development and validation of a genetic algorithm for flexible docking. *Journal of Molecular Biology*, 267(3):727–748, 1997.

[186] T.C. Jones. *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, University of New Mexico, 1995.

[187] M. Jünger, G. Reinelt, and G. Rinaldi. The traveling salesman problem. In M. Ball, T. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Handbooks in Operations Research and Management Sciences: Networks.* North-Holland, 1995.

[188] V. Kann. *On the Approximability of NP-complete Optimization Problems.* PhD thesis, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, 1992. NADA report TRITA-NA-9206.

[189] B.M. Kariuki, H. Serrano-Gonzalez, R.L. Johnston, and K.D.M. Harris. The application of a genetic algorithm for solving crystal structures from powder diffraction data. *Chemical Physics Letters*, 280(3-4):189–195, 1997.

[190] N. Karmarkar and R.M. Karp. The differencing method of set partitioning. Report UCB/CSD 82/113, University of California, Berkeley, CA, USA, 1982.

[191] N. Karmarkar, R.M. Karp, G.S. Lueker, and A.M. Odlyzko. Probabilistic analysis of optimum partitions. *Journal of Applied Probability*, 23(3):626–645, 1986.

[192] R.M. Karp. Reducibility among combinatorial problems. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum, New York NY, 1972.

[193] R.M. Karp. Probabilistic analysis of partitioning algorithms for the traveling salesman problem in the plane. *Math. Operational Research*, 2:209–224, 1977.

[194] R.M. Karp. A patching algorithm for the nonsymmetric traveling salesman problem. *SIAM Journal on Computing*, 8:561–573, 1979.

[195] I.E. Kassotakis, M.E. Markaki, and A.V. Vasilakos. A hybrid genetic approach for channel reuse in multiple access telecommunication networks. *IEEE Journal on Selected Areas in Communications*, 18(2):234–243, 2000.

[196] K. Katayama, H. Hirabayashi, and H. Narihisa. Performance analysis for crossover operators of genetic algorithm. *Transactions of the Institute of Electronics, Information and Communication Engineers*, J81D-I(6):639–650, 1998.

[197] J. Kececioglu and J. Pecqueur. Computing maximum-cardinality matchings in sparse general graphs. In *Proceedings of the 2nd Workshop on Algorithm Engineering*, pages 121–132, 1998.

[198] S. Khanna and R. Motwani. Towards a syntactic characterization of PTAS. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, pages 329–337, Philadelphia, Pennsylvania, 1996. ACM.

[199] T.S. Kim and G.S. May. Intelligent control of via formation by photosensitive BCB for MCM-L/D applications. *IEEE Transactions on Semiconductor Manufacturing*, 12:503–515, 1999.

[200] S. Kirkpatrick and G. Toulouse. Configuration space analysis of traveling salesman problems. *J. Physique*, 46:1277–1292, 1985.

[201] J. Knowles and D. Corne. Optimisation Method. European Patent EP00305549.8, 2000. Date filed: 30th June.

[202] J.D. Knowles and D.W. Corne. Benchmarking a New Memetic Algorithm Framework for Pareto Multiobjective Optimization, 2000. Submitted to *Knowledge and Information Systems*.

[203] J.D. Knowles and D.W. Corne. A Comparison of Diverse Aproaches to Memetic Multiobjective Combinatorial Optimization. In Annie S. Wu, editor, *Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program*, pages 103–108, 2000.

[204] J.D. Knowles and D.W. Corne. M-PAES: A Memetic Algorithm for Multiobjective Optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC00)*, pages 325–332, Piscataway, NJ, 2000. IEEE Press. Nominated by CEC Program Committe to be extended and submitted to Knowledge and Information Systems (KAIS) Journal.

[205] J.D. Knowles and D.W. Corne. Multiobjective Approaches to the Adaptive Distributed Database Management Problem. In D. Corne, editor, *Proceedings of the SAB/PPSN Workshops (Evolutionary Computation in Telecommunications Workshop*, 2000.

[206] J.D. Knowles, D.W. Corne, and M.J. Oates. On the Assessment of Multiobjective Approaches to the Adaptive Distributed Management Problem. In *Proceedings of the Sixth International Conference of Parallel Problem Solving From Nature (PPSN VI)*, pages 869–878, Berlin, 2000. Springer-Verlag.

[207] J.D. Knowles, M.J. Oates, and D.W. Corne. Advanced Multiobjective Evolutionary Algorithms Applied to two Problems in Telecommunications. *BT Technology Journal*, 18(4):51–65, October 2000.

[208] R. Korf. From approximate to optimal solutions: A case study of number partitioning. In C.S. Mellish, editor, *Proceedings of 14th Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada*, pages 266–272, San Mateo, CA, 1995. Morgan Kauffman.

[209] R. Korf. A complete anytime algorithm for number partitioning. *Artificial Intelligence*, 1998. to appear.

[210] M.W. Krentel. The complexity of optimization problems. *J. Comput. System Sci.*, 36:490–509, 1988.

[211] M.W. Krentel. Structure in locally optimal solutions. In *Proc. 30th Ann. IEEE Symp. on Foundations of Comput. Sci.*, pages 216–221, 1989.

[212] K. Krishna and M. Narasimha-Murty. Genetic k-means algorithm. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 29(3):433–439, 1999.

[213] K. Krishna, K.R. Ramakrishnan, and M.A.L. Thathachar. Vector quantization using genetic k-means algorithm for image compression. In *1997 International Conference on Information, Communications and Signal Processing*, volume 3, pages 1585–1587, New York, NY, 1997. IEEE.

[214] R.M. Krzanowski and J. Raper. Hybrid genetic algorithm for transmitter location in wireless networks. *Computers, Environment and Urban Systems*, 23(5):359–382, 1999.

[215] A. Kuehlmann, A. Srinivasan, and D.P. LaPotin. Verity - a formal verification program for custom CMOS circuits. *IBM Journal of Research and Development*, 39:149–165, 1995.

[216] M. Laguna and P. Laguna. Applying Tabu Search to the 2-dimensional Ising spin glass. *International Journal of Modern Physics C - Physics and Computers*, 6(1):11–23, 1995.

[217] M. Laguna and P. Moscato. Capítulo 3: Algoritmos genéticos. In B.A. Diaz, editor, *Las Nuevas Técnicas Heurísticas y las Redes Neuronales*, pages 67–103. Ed. Paraninfo, Madrid, 1996.

[218] E. Lamma, L.M. Pereira, and F. Riguzzi. Multi-agent logic aided lamarckian learning. Technical Report DEIS-LIA-00-004, Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna (Italy), 2000. LIA Series no. 44, submitted for publication.

[219] E. Landree, C. Collazo-Davila, and L.D. Marks. Multi-solution genetic algorithm approach to surface structure determination using direct methods. *Acta Crystallographica Section B - Structural Science*, 53:916–922, 1997.

[220] G. Laporte. The traveling salesman problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(2):231–247, 1992.

[221] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley-Interscience, Chichester, 1985.

[222] G.A. Lazar, J.R. Desjarlais, and T.M. Handel. De novo design of the hydrophobic core of ubiquitin. *Protein Science*, 6(6):1167–1178, 1997.

[223] C.Y. Lee. Genetic algorithms for single machine job scheduling with common due date and symmetric penalties. *Journal of the Operations Research Society of Japan*, 37(2):83–95, 1994.

[224] D. Levine. A parallel genetic algorithm for the set partitioning problem. In I.H. Osman and J.P. Kelly, editors, *Meta-Heuristics: Theory & Applications*, pages 23–35. Kluwer Academic Publishers, 1996.

[225] H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, 1998.

[226] F. Li, R. Morgan, and D. Williams. Economic environmental dispatch made easy with hybrid genetic algorithms. In *Proceedings of the International Conference on Electrical Engineering*, volume 2, pages 965–969, Beijing, China, 1996. Int. Acad. Publishers.

[227] L.P. Li, T.A. Darden, S.J. Freedman, B.C. Furie, B. Furie, J.D. Baleja, H. Smith, R.G. Hiskey, and L.G. Pedersen. Refinement of the NMR solution structure of the gamma-carboxyglutamic acid domain of coagulation factor IX using molecular dynamics simulation with initial Ca2+ positions determined by a genetic algorithm. *Biochemistry*, 36(8):2132–2138, 1997.

[228] C.F. Liaw. A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Oprational Research*, 124(1):28–42, 2000.

[229] G. E. Liepins and M. R. Hilliard. Greedy Genetics. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 90–99, Cambridge, MA, July 1987. Lawrence Erlbaum Associates.

[230] S. Lin. Computer solutions of the traveling salesman problem. *Bell System Technical Journal*, 10:2245–2269, December 1965.

[231] S. Lin and B.W. Kernighan. An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2):498–516, 1973.

[232] S. E. Ling. Integrating genetic algorithms with a prolog assignment program as a hybrid solution for a polytechnic timetable problem. In *Parallel Problem Solving from Nature II*, pages 321–329. Elsevier Science Publisher B. V., 1992.

[233] A. Linhares. State-space search strategies gleaned from animal behavior: a traveling salesman experiment. *Biological Cybernetics*, 78:167–173, 1998.

[234] A. Linhares and J.R.A. Torreão. Microcanonical optimization applied to the traveling salesman problem. *International Journal of Modern Physics C*, 9:133–146, 1998.

[235] D.M. Lorber and B.K. Shoichet. Flexible ligand docking using conformational ensembles. *Protein Science*, 7(4):938–950, 1998.

[236] S.J. Louis, X. Yin, and Z.Y. Yuan. Multiple vehicle routing with time windows using genetic algorithms. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 1804–1808, Washington D.C., 1999. IEEE Neural Network Council - Evolutionary Programming Society - Institution of Electrical Engineers.

[237] G.S. Lueker. A note on the average-case behaviour of a simple differencing method for partitioning. *Operations Research Letters*, 6(6):285–287, Dec. 1987.

[238] G.S. Lueker. Exponentially small bounds on the expected optimum of the partition and subset sum problems. *Random Structures & Algorithms*, 12(1):51–62, Jan. 1998.

[239] A.L. MacKay. Generalized crystallography. *THEOCHEM-Journal of Molecular Structure*, 336(2–3):293–303, 1995.

[240] J. Maddox. Genetics helping molecular-dynamics. *Nature*, 376(6537):209–209, 1995.

[241] B. Manderick, M. de Weger, and P. Spiessens. The genetic algorithm and the structure of the fitness landscape. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 143–150, San Mateo CA, 1991. Morgan Kaufmann.

[242] A. Mariano, M.G. Norman, and P. Moscato. Using L-systems to generate arbitrarily large instances of the euclidean traveling salesman problem with known optimal tours. In *XXVII Simposio Brasileiro de Pesquisa Operacional*, Rio de Janeiro, 1995. Sociedade Brasileira de Pesquisa Operacional.

[243] A. Mariano, M.G. Norman, and P. Moscato. Arbitrarily large planar ETSP instances with known optimal tours. *Pesquisa Operacional*, 15(1,2):89–96, jun./dez 1995.

[244] K.E. Mathias and L.D. Whitley. Noisy function evaluation and the delta coding algorithm. In *Proceedings of the SPIE–The International Society for Optical Engineering*, pages 53–64, 1994.

[245] A.C.W. May and M.S. Johnson. Protein-structure comparisons using a combination of a genetic algorithm, dynamic-programming and least-squares minimization. *Protein Engineering*, 7(4):475–485, 1994.

[246] A.S. Mendes, P.M. França, and P. Moscato. Fitness Landscapes for the Total Tardiness Single Machine Scheduling Problem. Accepted at *ICANNGA 2001 - V International Conference on Artificial Neural Networks and Genetic Algorithms, Prague, Czech Republic, April, 2001*.

[247] A.S. Mendes, P.M. França, and P. Moscato. Fuzzy-Evolutionary Algorithms Applied to Scheduling Problems. In *Proceedings of POM2000 - First World Conference on Production and Operations Management, Sevilla, Spain*, 2000.

[248] A.S. Mendes, F.M. Muller, P.M. França, and P. Moscato. Comparing meta-heuristic approaches for parallel machine scheduling problems with sequence-dependent setup times. In *Proceedings of the 15th International Conference on CAD/CAM Robotics & Factories of the Future*, volume 1, pages 1–6, Aguas de Lindoia, Brasil, 1999.

[249] L.D. Merkle, G.B. Lamont, G.H. Jr. Gates, and R. Pachter. Hybrid genetic algorithms for minimization of a polypeptide specific energy model. In *Proceedings of 1996 IEEE International Conference on Evolutionary Computation*, pages 396–400, New York, NY, 1996. IEEE.

[250] S. Mertens. Phase transition in the number partitioning problem. *Physical Review Letters*, 81(20):4281–4284, 1998.

[251] P. Merz and B. Freisleben. A Genetic Local Search Approach to the Quadratic Assignment Problem. In T. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 465–472, San Mateo, CA, 1997. Morgan Kaufmann.

[252] P. Merz and B. Freisleben. Genetic Local Search for the TSP: New Results. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, pages 159–164. IEEE Press, 1997.

[253] P. Merz and B. Freisleben. Fitness Landscapes, Memetic Algorithms and Greedy Operators for Graph Bi-Partitioning. Technical Report 98-01, University of Siegen, Germany, 1998. to appear in Evolutionary Computation.

[254] P. Merz and B. Freisleben. Memetic Algorithms and the Fitness Landscape of the Graph Bi-partitioning Problem. In A.E. Eiben, T. Back, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving from Nature V*, volume 1498 of *Lecture Notes in Computer Science*, pages 765–774. Springer-Verlag, 1998.

[255] P. Merz and B. Freisleben. On the Effectiveness of Evolutionary Search in High–Dimensional NK-Landscapes. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 741–745. IEEE Press, 1998.

[256] P. Merz and B. Freisleben. A comparion of Memetic Algorithms, Tabu Search, and ant colonies for the quadratic assignment problem. In *Proceedings of the 1999 Congress on Evolutionary Computation, Washington D.C.*, pages 2063–2070, Piscataway, NJ, 1999. IEEE Service Center.

[257] P. Merz and B. Freisleben. Fitness landscapes and memetic algorithm design. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 245–260. McGraw-Hill, 1999.

[258] P. Merz and B. Freisleben. Fitness landscapes, memetic algorithms, and greedy operators for graph bipartitioning. *Evolutionary Computation*, 8(1):61–91, 2000.

[259] J.C. Meza, R.S. Judson, T.R. Faulkner, and A.M. Treasurywala. A comparison of a direct search method and a genetic algorithm for conformational searching. *Journal of Computational Chemistry*, 17(9):1142–1151, 1996.

[260] M. Mignotte, C. Collet, P. Pérez, and P. Bouthemy. Hybrid genetic optimization and statistical model based approach for the classification of shadow shapes in sonar imagery. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(2):129–141, 2000.

[261] D.L. Miller and J.F. Pekny. Exact solution of large asymmetric traveling salesman problems. *Science*, 251:754–761, 1991.

[262] D.M. Miller, H.C. Chen, J. Matson, and Q. Liu. A hybrid genetic algorithm for the single machine scheduling problem. *Journal of Heuristics*, 5(4):437–454, 1999.

[263] S.T. Miller, J.M. Hogle, and D.J. Filman. A genetic algorithm for the *ab initio* phasing of icosahedral viruses. *Acta Crystallographica Section D - Biological Crystallography*, 52:235–251, 1996.

[264] L. Min and W. Cheng. Identical parallel machine scheduling problem for minimizing the makespan using genetic algorithm combined with simulated annealing. *Chinese Journal of Electronics*, 7(4):317–321, 1998.

[265] X.G. Ming and K.L. Mak. A hybrid hopfield network-genetic algorithm approach to optimal process plan selection. *International Journal of Production Research*, 38(8):1823–1839, 2000.

[266] M. Minsky. Negative expertise. *International Journal of Expert Systems*, 7(1):13–19, 1994.

[267] A. Monfroglio. Hybrid genetic algorithms for a rostering problem. *Software – Practice and Experience*, 26(7):851–862, 1996.

[268] A. Monfroglio. Hybrid genetic algorithms for timetabling. *International Journal of Intelligent Systems*, 11(8):477–523, 1996.

[269] A. Monfroglio. Timetabling through constrained heuristic search and genetic algorithms. *Software – Practice and Experience*, 26(3):251–279, 1996.

[270] P. Moscato. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA, 1989.

[271] P. Moscato. On genetic crossover operators for relative order preservation. C3P Report 778, California Institute of Technology, Pasadena, CA 91125, 1989.

[272] P. Moscato. An Introduction to Population Approaches for Optimization and Hierarchical Objective Functions: The Role of Tabu Search. *Annals of Operations Research*, 41(1-4):85–121, 1993.

[273] P. Moscato. Memetic algorithms: A short introduction. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 219–234. McGraw-Hill, 1999.

[274] P. Moscato and M.F. Nobrega da Costa. A Prolog-based expert system to benefit from reencodings and reductions between NP-optimization problems. In *Anais do XIX SOBRAPO - Simpósio Brasileiro de Pesquisa Operacional, Salvador, BA, Brasil*, page 201, Rio de Janeiro, 1997. Sociedade Brasileira de Pesquisa Operacional. extended abstract.

[275] P. Moscato and B.A. Diaz. Capítulo 2: Recocido simulado. In B.A. Diaz, editor, *Las Nuevas Técnicas Heurísticas y las Redes Neuronales*, pages 37–66. Ed. Paraninfo, Madrid, 1996.

[276] P. Moscato and M. G. Norman. A Memetic Approach for the Traveling Salesman Problem Implementation of a Computational Ecology for Combinatorial Optimization on Message-Passing Systems. In M. Valero, E. Onate, M. Jane, J. L. Larriba, and B. Suarez, editors, *Parallel Computing and Transputer Applications*, pages 177–186, Amsterdam, 1992. IOS Press.

[277] P. Moscato and M.G. Norman. On the performance of heuristics on finite and infinite fractal instances of the euclidean traveling salesman problem. *INFORMS Journal on Computing*, 10(2):121–132, 1998.

[278] P. Moscato and F. Tinetti. Blending heuristics with a population-based approach: A memetic algorithm for the traveling salesman problem. Report 92-12, Universidad Nacional de La Plata, C.C. 75, 1900 La Plata, Argentina, 1992.

[279] H. Mühlenbein. Evolution in Time and Space – The Parallel Genetic Algorithm. In Gregory J.E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337. Morgan Kaufmann Publishers, 1991.

[280] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution Algorithms in Combinatorial Optimization. *Parallel Computing*, 7:65–88, 1988.

[281] T. Murata and H. Ishibuchi. Performance evaluation of genetic algorithms for flowshop scheduling problems. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 2, pages 812–817, New York, NY, 1994. IEEE.

[282] T. Murata, H. Ishibuchi, and H. Tanaka. Genetic algorithms for flowshop scheduling problems. *Computers & Industrial Engineering*, 30(4):1061–1071, 1996.

[283] M. Musil, M.J. Wilmut, and N.R. Chapman. A hybrid simplex genetic algorithm for estimating geoacoustic parameters using matched-field inversion. *IEEE Journal of Oceanic Engineering*, 24(3):358–369, 1999.

[284] Y. Nagata and Sh. Kobayashi. Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In Th. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms, East Lansing, EUA*, pages 450–457, San Mateo, CA, 1997. Morgan Kaufmann.

[285] M. Nakamaru, H. Matsuda, and Y. Iwasa. The evolution of social interaction in lattice models. *Sociological Theory and Methods*, 12(2):149–162, 1998.

[286] M. Nakamaru, H. Nogami, and Y. Iwasa. Score-dependent fertility model for the evolution of cooperation in a lattice. *Journal of Theoretical Biology*, 194(1):101–124, 1998.

[287] R. Niedermeier. Some prospects for efficient fixed parameter algorithms. In B. Rovan, editor, *Theory and Practice of Informatics, Seminar on Current Trends in Theory and Practice of Informatics*, volume 1521, pages 168–185. Springer-Verlag, 1998.

[288] R. Niedermeier and P. Rossmanith. An efficient fixed parameter algorithm for 3-hitting set. Technical Report WSI-99-18, Universität Tübingen, Wilhelm-Schickard-Institut für Informatik, 1999. Technical Report, Revised version accepted in *Journal of Discrete Algorithms*, August 2000.

[289] R. Niedermeier and P. Rossmanith. Upper bounds for vertex cover futher improved. In *STACS '99*, volume 1563, pages 561–570. Springer-Verlag, 1999.

[290] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73:125–129, 2000.

[291] J.A. Niesse and H.R. Mayne. Global geometry optimization of atomic clusters using a modified genetic algorithm in space-fixed coordinates. *Journal of Chemical Physics*, 105(11):4700–4706, 1996.

[292] A.L. Nordstrom and S. Tufekci. A genetic algorithm for the talent scheduling problem. *Computers & Operations-Research*, 21(8):927–940, 1994.

[293] M.G. Norman and P. Moscato. A competitive and cooperative approach to complex combinatorial search. Technical Report Caltech Concurrent Computation Program, Report. 790, California Institute of Technology, Pasadena, California, USA, 1989. expanded version published at the Proceedings of the 20th Informatics and Operations Research Meeting, Buenos Aires (20th JAIIO), Aug. 1991, pp. 3.15–3.29.

[294] M.G. Norman and P. Moscato. The euclidean traveling salesman problem and a space-filling curve. *Chaos, Solitons and Fractals*, 6:389–397, 1995.

[295] A.G.N. Novaes, J.E.S. De-Cursi, and O.D. Graciolli. A continuous approach to the design of physical distribution systems. *Computers & Operations Research*, 27(9):877–893, 2000.

[296] M.A. Nowak and K. Sigmund. Evolution of indirect reciprocity by image scoring. *Nature*, 393(6685):573–577, 1998.

[297] P. Orponen and H. Mannila. On approximation preserving reductions: Complete problems and robust measures. Technical Report C-1987-28, Department of Computer Science, University of Helsinki, 1987.

[298] P. Osmera. Hybrid and distributed genetic algorithms for motion control. In V. Chundy and E. Kurekova, editors, *Proceedings of the Fourth International Symposium on Measurement and Control in Robotics*, pages 297–300, 1995.

[299] R. Ostermark. A neuro-genetic algorithm for heteroskedastic time-series processes: empirical tests on global asset returns. *Soft Computing*, 3(4):206–220, 1999.

[300] R. Ostermark. Solving a nonlinear non-convex trim loss problem with a genetic hybrid algorithm. *Computers & Operations Research*, 26(6):623–635, 1999.

[301] R. Ostermark. Solving irregular econometric and mathematical optimization problems with a genetic hybrid algorithm. *Computational Economics*, 13(2):103–115, 1999.

[302] E. Ozcan and C.K. Mohan. Steady state memetic algorithm for partial shape matching. In V.W. Porto, N. Saravanan, and D. Waagen, editors, *Evolutionary Programming VII*, volume 1447 of *Lecture Notes in Computer Science*, pages 527–236. Springer, Berlin, 1998.

[303] L. Ozdamar. A genetic algorithm approach to a general category project scheduling problem. *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, 29(1):44–59, 1999.

[304] M.N. Pacey, X.Z. Wang, S.J. Haake, and E.A. Patterson. The application of evolutionary and maximum entropy algorithms to photoelastic spectral analysis. *Experimental Mechanics*, 39(4):265–273, 1999.

[305] B. Paechter, A. Cumming, M.G. Norman, and H. Luchian. Extensions to a Memetic timetabling system. In E.K. Burke and P. Ross, editors, *The Practice and Theory of Automated Timetabling*, volume 1153 of *Lecture Notes in Computer Science*, pages 251–265. Springer Verlag, 1996.

[306] B. Paechter, R.C. Rankin, and A. Cumming. Improving a lecture timetabling system for university wide use. In E.K. Burke and M. Carter, editors, *The Practice and Theory of Automated Timetabling II*, volume 1408 of *Lecture Notes in Computer Science*, pages 156–165. Springer Verlag, 1998.

[307] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[308] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1982.

[309] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. In *Proc. Twentieth Ann. ACM Symp. on Theory of Comp.*, pages 229–234. ACM, 1988.

[310] C.H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. System Sci.*, 43:425–440, 1991.

[311] C.H. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the V-C dimension. In Jacobo Allender, Eric; Beigel, Richard; Cai, Jin-yi; Feigenbaum, Joan; Homer, Steven; Lange, Klaus-Jörn; Lutz, Jack; Mahaney, Stephen; Straubing, Howard; Torán, editor, *Proceedings of the 8th Annual Conference on Structure in Complexity Theory (SCTC '93)*, pages 12–18, San Diego, CA, USA, May 1993. IEEE Computer Society Press.

[312] C.H. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the V-C dimension. *Journal of Computer and System Sciences*, 53(2):161–170, October 1996.

[313] M. Peinado and T. Lengauer. Parallel "go with the winners algorithms" in the LogP Model. In IEEE Computer Society Press, editor, *Proceedings of the 11th International Parallel Processing Symposium*, pages 656–664. Los Alamitos, California, 1997.

[314] J.G. Peters and L. Rudolph. Parallel approximation schemes for subset sum and knapsack problems. *Acta Inf.*, 24:417–432, 1987.

[315] J.Y. Potvin. The traveling salesman problem: A neural network perspective. *ORSA Journal on Computing*, 5:328–348, 1993.

[316] D.K. Pratihar, K. Deb, and A. Ghosh. Fuzzy-genetic algorithms and mobile robot navigation among static obstacles. In *Proceedings of the 1999 Congress on Evolutionary Computation*, pages 327–334, Washington D.C., 1999. IEEE.

[317] N. Pucello, M. Rosati, G. D'Agostino, F. Pisacane, V. Rosato, and M. Celino. Search of molecular ground state via genetic algorithm: Implementation on a hybrid SIMD-MIMD platform. *International Journal of Modern Physics C*, 8(2):239–252, 1997.

[318] W.J. Pullan. Structure prediction of benzene clusters using a genetic algorithm. *Journal of Chemical Information and Computer Sciences*, 37(6):1189–1193, 1997.

[319] D. Quagliarella and A. Vicini. Hybrid genetic algorithms as tools for complex optimisation problems. In P. Blonda, M. Castellano, and A. Petrosino, editors, *New Trends in Fuzzy Logic II. Proceedings of the Second Italian Workshop on Fuzzy Logic*, pages 300–307, Singapore, 1998. World Scientific.

[320] N.J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–205, 1991.

[321] N.J. Radcliffe. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10:339–384, 1994.

[322] N.J. Radcliffe and P.D. Surry. Fitness Variance of Formae and Performance Prediction. In L.D. Whitley and M.D. Vose, editors, *Proceedings of the Third Workshop on Foundations of Genetic Algorithms*, pages 51–72, San Francisco, 1994. Morgan Kaufmann.

[323] N.J. Radcliffe and P.D. Surry. Formal Memetic Algorithms. In T. Fogarty, editor, *Evolutionary Computing: AISB Workshop*, volume 865 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Berlin, 1994.

[324] G.R. Raidl and B.A. Julstron. A weighted coding in a genetic algorithm for the degree-constrained minimum spanning tree problem. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *ACM Symposium on Applied Computing 2000*, pages 440–445. ACM Press, 2000.

[325] E. Ramat, G. Venturini, C. Lente, and M. Slimane. Solving the multiple resource constrained project scheduling problem with a hybrid genetic algorithm. In Th. Bäck, editor, *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 489–496, San Francisco CA, 1997. Morgan Kaufmann.

[326] R.C. Rankin. Automatic timetabling in practice. In *Practice and Theory of Automated Timetabling. First International Conference. Selected Papers*, pages 266–279, Berlin, 1996. Springer-Verlag.

[327] M.L. Raymer, P.C. Sanschagrin, W.F. Punch, S. Venkataraman, E.D. Goodman, and L.A. Kuhn. Predicting conserved water-mediated and polar ligand interactions in proteins using a k-nearest-neighbors genetic algorithm. *Journal of Molecular Biology*, 265(4):445–464, 1997.

[328] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog Verlag, Stuttgart, 1973.

[329] C. Reeves. Hybrid genetic algorithms for bin-packing and related problems. *Annals of Operations Research*, 63:371–396, 1996.

[330] C. Reich. Simulation of imprecise ordinary differential equations using evolutionary algorithms. In J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, editors, *ACM Symposium on Applied Computing 2000*, pages 428–432. ACM Press, 2000.

[331] G. Reinelt. TSPLIB - a traveling salesman library. *ORSA Journal on Computing*, 3:376–384, 1991.

[332] M.A. Ridao, J. Riquelme, E.F. Camacho, and M. Toro. An evolutionary and local search algorithm for planning two manipulators motion. In A.P. Del Pobil, J. Mira, and M. Ali, editors, *Tasks and Methods in Applied Artificial Intelligence*, volume 1416 of *Lecture Notes in Computer Science*, pages 105–114. Springer-Verlag, Berlin Heidelberg, 1998.

[333] C.F. Ruff, S.W. Hughes, and D.J. Hawkes. Volume estimation from sparse planar images using deformable models. *Image and Vision Computing*, 17(8):559–565, 1999.

[334] W. Ruml. Stochastic approximation algorithms for number partitioning. Technical Report TR-17-93, Harvard University, Cambridge, MA, USA, 1993. available via ftp://das-ftp.harvard.edu/techreports/tr-17-93.ps.gz.

[335] W. Ruml, J.T. Ngo, J. Marks, and S.M. Shieber. Easily searched encodings for number partitioning. *Journal of Optimization Theory and Applications*, 89(2):251–291, 1996.

[336] S.M. Sait and H. Youssef. *VLSI Design Automation: Theory and Practice.* McGraw-Hill Book Co. (copublished by IEEE), Europe, 2000.

[337] A. Sakamoto, X.Z. Liu, and T. Shimamoto. A genetic approach for maximum independent set problems. *IEICE Transactions on Fundamentals of Electronics Communications and Computer Sciences*, E80A(3):551–556, 1997.

[338] V. Schnecke and O. Vornberger. Hybrid genetic algorithms for constrained placement problems. *IEEE Transactions on Evolutionary Computation*, 1(4):266–277, 1997.

[339] H.-P. Schwefel. Evolution strategies: A family of non-linear optimization techniques based on imitating some principles of natural evolution. *Annals of Operations Research*, 1:165–167, 1984.

[340] R. Sedgewick. *Algorithms in C++*. Addison-Wesley, 1992.

[341] M. Serna, L. Trevisan, and F. Xhafa. The (parallel) approximability of non-Boolean satisfiability problems and restricted integer programming. *Lecture Notes in Computer Science*, 1373:488, 1998.

[342] M. Serna and F. Xhafa. On parallel versus sequential approximations. Technical Report LSI-95-18-R, Universitat Politècnica de Catalunya, 1995.

[343] A. Shamir. On the cryptocomplexity of knapsack systems. In *Proceedings of the 11th Annual ACM Symposium on Theory of Computing*, pages 118–129, New York, 1979. Association for Computing Machinery.

[344] K. Shankland, W.I.F. David, and T. Csoka. Crystal structure determination from powder diffraction data by the application of a genetic algorithm. *Zeitschrift Fur Kristallographie*, 212(8):550–552, 1997.

[345] K. Shankland, W.I.F. David, T. Csoka, and L. McBride. Structure solution of ibuprofen from powder diffraction data by the application of a genetic algorithm combined with prior conformational analysis. *International Journal of Pharmaceutics*, 165(1):117–126, 1998.

[346] A. Shapira. An exact performance bound for an $O(m + n)$ time greedy matching procedure. *Electronic Journal of Combinatorics*, 4(1):R25, 1997.

[347] D.L. Shell. A high-speed sorting procedure. *Communications of the ACM*, 2:30–32, 1959.

[348] G. Sorkin. *Theory and Practice of Simulated Annealing on Special Energy Landscapes*. Ph.d. thesis, University of California at Berkeley, Berkeley, CA, USA, 1992.

[349] D. Srinivasan, R.L. Cheu, Y.P. Poh, and A.K.C. Ng. Development of an intelligent technique for traffic network incident detection. *Engineering Applications of Artificial Intelligence*, 13(3):311–322, 2000.

[350] K. Steiglitz and P. Weiner. Some improved algorithms for computer solution of the traveling salesman problem. In *Proceedings of the Sixth Allerton Conference on Circuit and System Theory*, pages 814–821, Urbana, Illinois, 1968.

[351] R.H. Storer, S.W. Flanders, and S.D. Wu. Problem space local search for number partitioning. *Annals of Operations Research*, 63:465–487, 1996.

[352] T. Stützle and M. Dorigo. ACO algorithms for the traveling salesman problem. In K. Miettinen, M. Mäkelä, P. Neittaanmäki, and J. Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science:*

Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications. John Wiley & Sons, 1999.

[353] J.Y. Suh and D. Van Gucht. Incorporating Heuristic Information into Genetic Search. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 100–107, Cambridge, MA, July 1987. Lawrence Erlbaum Associates.

[354] P.D. Surry and N.J. Radcliffe. Inoculation to initialise evolutionary search. In T.C. Fogarty, editor, *Evolutionary Computing: AISB Workshop*, number 1143 in Lecture Notes in Computer Science, pages 269–285. Springer-Verlag, 1996.

[355] G. Syswerda. Uniform crossover in genetic algorithms. In J.D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, San Mateo, CA, 1989. Morgan Kaufmann.

[356] T. Taguchi, T. Yokota, and M. Gen. Reliability optimal design problem with interval coefficients using hybrid genetic algorithms. *Computers & Industrial Engineering*, 35(1–2):373–376, 1998.

[357] K.Y. Tam and R.G. Compton. GAMATCH - a genetic algorithm-based program for indexing crystal faces. *Journal of Applied Crystallography*, 28:640–645, 1995.

[358] M. Tanaka and M. Yamada. Solving multiobjective time tabling problem by using requests. *Transactions of Systems, Control and Information*, 12(2):90–97, 1999.

[359] A.G. Thomason. Hamilton cycles and uniquely edge-colourable graphs. *Annals of Discrete Mathematics*, 3:259–268, 1978.

[360] A.P. Topchy, O.A. Lebedko, and V.V. Miagkikh. Fast learning in multilayered networks by means of hybrid evolutionary and gradient algorithms. In *Proceedings of International Conference on Evolutionary Computation and its Applications*, pages 390–398, June 1996.

[361] L. Trevisan and F. Xhafa. The parallel complexity of positive linear programming. *Parallel Processing Letters*, 8(4):527–533, 1998.

[362] A.J. Urdaneta, J.F. Gómez, E. Sorrentino, L. Flores, and R. Díaz. A hybrid genetic algorithm for optimal reactive power planning based upon successive linear programming. *IEEE Transactions on Power Systems*, 14(4):1292–1298, 1999.

[363] A.H.C. vanKampen, C.S. Strom, and L.M.C Buydens. Lethalization, penalty and repair functions for constraint handling in the genetic algorithm methodology. *Chemometrics And Intelligent Laboratory Systems*, 34(1):55–68, 1996.

[364] T. Walters. Repair and brood selection in the traveling salesman problem. In A.E. Eiben, T. Back, M. Schoenauer, and H.-P. Schwefel, editors, *Proceedings of the Fifth International Conference on Parallel Problem Solving from Nature, Amsterdam, The Netherlands*, volume 1498 of *Lecture Notes in Computer Science*, pages 813–822. Springer-Verlag, 1998.

[365] L. Wang and J. Yen. Extracting fuzzy rules for system modeling using a hybrid of genetic algorithms and kalman filter. *Fuzzy Sets and Systems*, 101(3):353–362, 1999.

[366] J.P. Watson, S. Rana, L.D. Whitley, and A.E. Howe. The impact of approximate evaluation on the performance of search algorithms for warehouse scheduling. *Journal of Scheduling*, 2(2):79–98, 1999.

[367] R. Wehrens, C. Lucasius, L. Buydens, and G. Kateman. HIPS, A hybrid self-adapting expert system for nuclear magnetic resonance spectrum interpretation using genetic algorithms. *Analytica Chimica ACTA*, 277(2):313–324, May 1993.

[368] P. Wei and L.X. Cheng. A hybrid genetic algorithm for function optimization. *Journal of Software*, 10(8):819–823, 1999.

[369] X. Wei and F. Kangling. A hybrid genetic algorithm for global solution of nondifferentiable nonlinear function. *Control Theory & Applications*, 17(2):180–183, 2000.

[370] D.S. Weile and E. Michielssen. Design of doubly periodic filter and polarizer structures using a hybridized genetic algorithm. *Radio Science*, 34(1):51–63, 1999.

[371] D.B. West. Pair of adjacent Hamiltonian circuits with small intersection. *Studies in Applied Mathematics*, 59:245–248, 1978.

[372] R.P. White, J.A. Niesse, and H.R. Mayne. A study of genetic algorithm approaches to global geometry optimization of aromatic hydrocarbon microclusters. *Journal of Chemical Physics*, 108(5):2208–2218, 1998.

[373] D. Whitley. Using reproductive evaluation to improve genetic search and heuristic discovery. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 108–115, Cambridge, MA, July 1987. Lawrence Erlbaum Associates.

[374] D. Whitley, T. Starkweather, and D. Shaner. The traveling salesman and sequence scheduling: Quality solutions using genetic edge recombination. In L. Davis, editor, *Handbook of Genetic Algorithms*, pages 350–372. Van Nostrand Reinhold, 1991.

[375] P. Willett. Genetic algorithms in molecular recognition and design. *Trends in Biotechnology*, 13(12):516–521, 1995.

[376] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[377] J. Xiao and L. Zhang. Adaptive evolutionary planner/navigator for mobile robots. *IEEE Transactions on Evolutionary Computation*, 1(1):18–28, 1997.

[378] B. Yakir. The differencing algorithm LDM for partitioning: A proof of a conjecture of Karmarkar and Karp. *Mathematics of Operations Research*, 21(1):85–99, Feb. 1996.

[379] M. Yannakakis. The analysis of local search problems and their heuristics. In *Proc. 7th Ann. Symp. on Theoretical Aspects of Comput. Sci.*, pages 298–311. Springer-Verlag, 1990. Lecture Notes in Comput. Sci. 415.

[380] M. Yannakakis. Perspectives on database theory. In *Proceedings of the IEEE Symposium on the Foundations of Computer Science*, pages 224–246, 1995.

[381] M. Yannakakis. Computational complexity. In E.H.L. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 19–55. Wiley, Chichester, 1997.

[382] X. Yao. Finding approximate solutions to NP-hard problems by neural networks is hard. *Inform. Process. Lett.*, 41:93–98, 1992.

[383] X. Yao. Evolutionary artificial neural networks. *Int. Journal of Neural Systems*, 4(3):203–222, September 1993.

[384] I.C. Yeh. Hybrid genetic algorithms for optimization of truss structures. *Computer Aided Civil and Infrastructure Engineering*, 14(3):199–206, 1999.

[385] M. Yoneyama, H. Komori, and S. Nakamura. Estimation of impulse response of vocal tract using hybrid genetic algorithm-a case of only glottal source. *Journal of the Acoustical Society of Japan*, 55(12):821–830, 1999.

[386] C.R. Zacharias, M.R. Lemes, and A.D. Pino. Combining genetic algorithm and simulated annealing: a molecular geometry optimization study. *THEOCHEM-Journal of Molecular Structure*, 430(29–39), 1998.

[387] D. Zuckerman. On unapproximable versions of NP-complete problems. *SIAM Journal on Computing*, 25(6):1293–1304, 1996.

[388] M. Zwick, B. Lovell, and J. Marsh. Global optimization studies on the 1-d phase problem. *International Journal of General Systems*, 25(1):47–59, 1996.

Índice Remissivo

I_F, 39
$\mathbb{N}$, 16
$\mathbb{Q}$, 16
$\mathbb{R}$, 16
$\mathbb{R}^+$, 16
$\mathbb{Z}$, 16
$\mathbb{Z}^+$, 16
ln, 17
log, 17
2-*merger*
    como heurística randomizada, 141
N-point generalized crossover, 134
k-merger
    a partir de algoritmos FPT, 139
    papel nas classes uPMA e PMA, 131
k-opt
    algoritmo, 142
2-repair, 67
3-change move, 53
3-opt, 67
Asparagos96, 55
Belief Search, 176
Binomial Minimal Mutation, 27
Branch-and-Bound, 2, 54, 128, 130, 152, 176
Branch-and-Cut, 130
Busca Local Genética, 55
Directed Edge Repair, 56
Distance Preserving Crossover, 55, 64, 66, 141
Dynastical Optimal Recombination, 152
Edge Assembly Crossover, 30, 55
Enhanced Edge Recombination, 64
Forma Analysis, 157, 168
Formae Theory, 131, 152
Genetic Local Search, 55
    vs. *Asparagos96*, 55
Iterated Lin-Kerninghan, 2
Maximal Preservative Crossover, 55
MemePool Project, 181
Multiple Fragment-Nearest Neighbor Repair, 66
PocketPropagation
    ordenação em população hierárquica, 98
PocketPropagation, 63
Pockets
    como memórias, 97
Polynomial Local Search, 129, 131
Polynomial Merger Algorithms, 172
Problem Space Local Search, 85
RestartPop()
    apos crise de diversidade, 102
Scatter Search, 171
Simulated Annealing, 85, 130
    falha, 93
Simulated Jumping
    para o MIN ATSP, 54, 72
Strategic Arc Crossover, 56
Strategic Edge Crossover, 56, 64
Tabu Search, 85, 109
Uniform Block Crossover, 164
Uniform Crossover, 66
Uniform Cycle Crossover, 159–161
Uniform Nearest Neighbor Crossover, 66
annealed approximation, 92
best-first search, 180
bounded search trees, 173
branchwidth, 153
depth-first search, 180
don't look bits, 67
eptas

definição, 143
equal piles problem, 88
fast 3-opt, 56
fixed-parameter tractability, 138
para recombinação, 173
fptas, 7, 49
frameworks, 171, 181
greedy improvement, 105
hill climbing, 27
memes, 181
one-city-insertion, 70
one-point crossover, 134
ptas, 6, 48
random generate-and-test, 109
reduction to a problem kernel, 173
slack
em Two-Machine Scheduling (D), 89
soft brood selection, 56
tour merging
relação com DOR, 172
complexidade, 173
treewidth, 153
two-point crossover, 134
NP-completo
no sentido forte, 87
NPO
problema
definição, 39
solução factível
definição, 40
NPO PB, 50
OptP[$\log n$], 51
PO, 41
Min Vertex Cover, 137
k-Feature Set
problema, 145
reformulação, 146
k-Vertex Cover, 114
0-1 Multiple Knapsack Problem, 23, 24
Bin Packing, 87, 88
Clique
problemas $W[1]$-completos, 139
versão parametrizada, 139
Constrained Max 3-Dimensional Matching, 145
Constrained Max Chromatic Number, 144
Constrained Max Clique Cover, 144
Constrained Max Cut, 144
Constrained Max Directed Hamiltonian Circuit, 143
Constrained Max Steiner Tree, 145
Constrained Max Three-Dimensional Matching, 145
Constrained Max Undirected Hamiltonian Circuit, 143
Constrained Max Vertex Cover, 143
Dominating Set
problemas $W[2]$-completos, 140
versão parametrizada, 140
Exact Knapsack 0/1
versão de decisão
definição, 88
Feedback Vertex Set
versão parametrizada, 138
Hamiltonian Path, 87
Hitting Set For Size Three Sets
versão parametrizada, 115
definição, 173
Independent Dominating Set
definição, 140
problemas $W[2]$-completos, 140
versão parametrizada, 140
Integer Programming, 6
Knapsack, 88
Longest Cycle in Cubic Hamiltonian Graphs, 141
Max 3-Dimensional Matching
versão com restrições, 145
Max 3-SAT, 52
Max Chromatic Number
versão com restrições, 144
Max Clique Cover
versão com restrições, 144
Max Clique, 52, 139
heurística de Balas and Niehaus, 139
Max Cut, 6
versão com restrições, 144

MAX DIRECTED HAMILTONIAN CIRCUIT
versão com restrições, 143
MAX INDEPENDENT SET, 48, 51, 121, 139
PLANAR, 48
MAX KNAPSACK, 7
MAX LABEL COVER, 52
MAX SATISFIABILITY, 25
MAX STEINER TREE
versão com restrições, 145
MAX THREE-DIMENSIONAL MATCHING, 40, 51
versão com restrições, 145
MAX UNDIRECTED HAMILTONIAN CIRCUIT
versão com restrições, 143
MIN $0-1$ PROGRAMMING, 51
MIN ATSP
3-change move, 53
definição, 53
MIN GRAPH BI-PARITITIONING, 145
MIN GRAPH COLORING, 87
MIN GRAPH COLOURING, 93
MIN GRAPH PARTITIONING, 87
MIN GRAPH PARTITIONING, 93
MIN HITTING SET, 115
casos aproximáveis, 115
variação restrita, 115
MIN KNAPSACK
fptas, 49
MIN NUMBER PARTITIONING, 85
instâncias difíceis, 103
transição de fase, 103
MIN QUADRATIC ASSIGNMENT PROBLEM, 72
MIN SET COVER, 52
equivalente a MIN HITTING SET, 115
MIN TSP, 6, 51, 129, 130, 133, 140–142, 144, 171, 175, 181
MIN VERTEX COVER, 8, 40, 121
MIN WEIGHTED HAMILTONIAN CYCLE, 140, 172
OPEN-SHOP SCHEDULING, 88
PLANAR DOMINATING SET
versão parametrizada, 114
PRODUCTION PLANNING, 88
SATISFIABILITY, 87
SCHEDULING TO MINIMIZE WEIGHTED COMPLETION TIME, 88
TRAVELING SALESMAN PROBLEM, 87
TRIPARTITE MATCHING, 87
TWO-MACHINE SCHEDULING (D)
aproximabilidade, 90
versão de decisão
definição, 89
VERTEX COVER
generalização, 174
versão com restrições, 143
árvores de busca limitadas, 173
ótimo local, 26
NUMBER PARTITIONING
problema de decisão, 85, 86
1-minimal, 40

acoplamento antiferromagnético de Mattis, 110
algoritmo
k-opt, 142
CKK, 103
complexidade
de pior caso, 4
de Busca Local, 26
de Karmarkar-Karp, 87
costo esperado, 92
de tempo polinomial, 3
Korf
Karmarkar-Karp completo, 91
MERGE
para MAX CLIQUE, 139
pseudo-polinomial, 3, 86
randomizado, 9
resolvendo um problema
definição, 22
Simplex
para Programação Linear, 2
algoritmo guloso
como função de crescimento, 100
algoritmos
adaptação, 176
para problemas da classe FPT

paralelização, 139
polinomiais de fusão, 172
pseudo-polinomiais, 49
algoritmos evolutivos, 155
algoritmos genéticos
para revisão de crenças, 181
problemas difíceis, 85
algoritmos meméticos, 2
assincronismo inerente, 63
baseados em Busca Local, 56
completos, 152, 172
fixed-parameter tractability, 173
direções futuras, 171
em eletrônica e engenharia, 36
guiados, 176
usando Lógica Modal, 176
mais complexo já construído, 172
multi-objetivos, 175
projeto de proteinas, 37
análise
comparativa
para recombinação, 153
de dominância
para recombinação, 153
de Forma, 157
do pior caso, 155
heurística de Karmarkar-Karp, 92
usando formae, 157
aprendizado
por revisão de contradições, 181
aproximação
asintótica em p, 47
dentro de uma constante p
medida de aproximação, 45
em $p(n)$, 48
esquema de tempo completamente polinomial, 49
esquemas, 48
esquemas de tempo polinomial, 48
limitada, 47
aproximabilidade
de Two-Machine Scheduling (D), 90
medidas, 43
resultado negativo, 10
resultados negativos, 51
aptidão
função, 3
APX, 6
aspiração
critério de, 102
assíncrono
paralelismo de MAs, 171
auto-redutibilidade, 41, 121

bajulador
comportamento, 99
Balas and Niehaus
algoritmo MERGE, 139
BubbleSort
pseudocódigo, 158
Busca
em amplitude, 180
em profundidade, 180
busca
em árvores limitadas, 114
busca binária, 41, 121
Busca Exaustiva
guiada por crenças, 180
Busca Local, 24, 129
algoritmo de, 26
baseada em Busca Tabu, 101, 106
benefícios do uso esporádico, 109
para diversificação, 107
gulosa, 101
Busca Predatória, 77
Busca Tabu, 85, 109
Incremental, 79
para o Min ATSP, 54

cúbico
grafo, 141
caminho
alternado
definição, 135
aumentante, 136
definição, 135
características
compatíveis, 152
cego, 100

cegueira
na recombinação, 134
Christofides' Algorithm, 45
circuito hamiltoniano, 42
de menor peso, 140
melhor que o pior problema, 140
circuitos
k-limitados
definição, 116
circuitos hamiltonianos
disjuntos, 130
classe
NP, 5
APX, 6
FPT
definição, 114
método geral de aceleração, 114
FPTAS, 7
PLS, 131
PMA
definição, 132
PTAS, 6
uPMA, 139
definição, 131
clique
definição, 139
cobertura de vértices
definição, 9
minimal, 9
colônia de formigas
método para o MIN ATSP, 54
para o MIN ATSP, 72
complexidade
de pior caso, 4
complexidade computacional
recombinação, 173
comportamento
bajulador, 99
conciliador, 99
rebelde, 99
comportamentos
adaptação na recombinação, 176
na recombinação, 180
para recombinação, 99
computadores
MIMD, 139
conciliador
comportamento, 99
conhecimento
definição em sistemas multi-agentes, 175
negativo, 174
conjectura de West, 129, 130, 152
conjunto dominante
de vértices em grafos, 140
convergencia prematura
entropia, 33
correlação de ótimos locais, 66
crença, 175
distribuída, 177
criptografia, 89
crise de diversidade, 102
critério de aspiração, 102
Crossover
Distance Preserving, 55, 66
Edge Assembly, 55
Strategic Arc, 56
Strategic Edge, 56
Uniform Nearest Neighbor, 66
Uniform, 66
Maximal Preservative, 55
Uniform Block, 164
Uniform Cycle, 159–161
Uniforme, 29
crossover
Edge Assembly, 30
generalized N-point, 134
one-point, 134
two-point, 134
cego, 100
como recombinação cega em $O(N \log N)$, 31
definição, 134
uniforme, 100

dígrafo, 17
desafio de Sedgewick, 156, 161
dinástica ótima
recombinação, 173
diversidade
crise de, 102
diversificação, 152

DOR, 172
DPX, 55, 141

eletrônica
  aplicações de algoritmos meméticos, 36
emparelhamento, 46
  de máxima cardinalidade, 135
  de menor peso
    definição, 46
  definição, 135
  em grafos esparsos, 137
  maximal, 135
  parcial de formas
    problema, 145
  perfeito, 135
engenharia
  aplicações de algoritmos meméticos, 36
entropia de Shannon, 33
epistasis, 31
ergodicidade, 25
erro
  absoluto, 44
    constante garantido, 44
  relativo
    definição, 43
    normalizado, 44
espaço
  de busca, 23
  de genótipos, 96
esquema de aproximação
  em tempo completamente polinomial, 49
  em tempo polinomial, 48
esquemas de aproximação, 48
evolução cultural, 60
evolução da cooperação, 20
explorativo, 152

Física de sistemas desordenados, 111
fácil-difícil-fácil
  transição de fase, 103
fenótipo, 97
Forma
  definição, 157
formulação
  válida, 24

FPT
  método geral de aceleração, 114
FPTAS, 7
função
  de aptidão, 3
  de crescimento, 23, 24, 97
    como algoritmo guloso de construção, 24
  de factibilidade, 22
  guia, 23, 134
    definição, 25
  objetivo, 22, 131, 144

Garey e Johnson
  lista de problemas NP, 88
genótipo, 96
geração
  definição, 98
grafo, 17
  cúbico, 141
  dígrafo, 17
  emparelhamento
    de menor peso, 46
    definição, 46
  emparelhamento de máxima cardinalidade, 135
  esparso
    "branchwidth", 153
    "treewidth", 153
    emparelhamento, 137
  euleriano, 46
  hamiltoniano
    circuito de máximo comprimento, 141
  hipergrafo, 115, 174
  não dirigido
    máximo grau quatro, 141
    mínimo grau dois, 141
  restrição, 17
  tour euleriano, 46
  vértices
    conjunto dominante, 140
GRASP, 85, 87
gravitação, 96

heurística
  de Karp
    para o MIN TSP, 64

de Karmarkar-Karp, 87
costo esperado, 92
definição, 91
hipergrafos, 115, 174

inatividade
em Two-Machine Scheduling (D), 89
instância, 39
de um problema computacional, 21
definição, 21
intensificação, 152
Ising
vidro de spin 2-dimensional, 110

jogos deportivos
escalonamento
problemas de, 35

Karmarkar-Karp
algoritmo completo, 91
heurística, 87
análise do pior caso, 92
costo esperado, 92
definição, 91
Korf
algoritmo exato CKK, 103
algoritmo Karmarkar-Karp completo, 91

Lógica Modal
de crença parcial, 180
e algoritmos meméticos, 176
papel da negação, 181
para recombinação de crença, 176
líder, 60
em população hierárquica, 97
lógica epistêmica
papel em MAs, 177
Lin-Kernighan heuristic, 55
Lin-Kernighan local search, 130
linguagem
NP
representação básica, 144
parametrizada
definição, 143

máximo, 40
método poliédrico
para o Min ATSP, 54
mínimo, 40
míope
recombinação
definição, 134
macromutação, 133
maldição da dimensionalidade, 165
Mattis
acoplamento antiferromagnético, 110
maximal, 40
medida de aproximação, 45
dentro de uma constante p, 45
razão de desempenho, 45
respeitosa do custo, 45
medidas de aproximabilidade, 43
melhor razão de desempenho, 47
assintótico, 47
melhora gulosa, 101, 105
memórias
uso de *Pockets*, 97
memes, 181
MergeSort
pseudocódigo, 159
meta-heurísticas
estado atual, 182
microcanônica
otimização
meta-heurística, 79
MIMD
computadores, 139
minimal, 40
modal
negação, 180
modelos de ilhas, 60
multiobjetivo
otimização combinatória
problemas de, 35
mutação, 27
binomial mínima, 27
minimal, 100
para representação binária direta, 100
relação com melhora gulosa, 101

simple, 100
para representação binária direta, 100
representação binária direta, 104
representação ternária direta, 105

não aproximabilidadade, 10
negação
modal, 180
normalizado
erro relativo, 44

ordenação, 156
BubbleSort, 158
MergeSort, 159
QuickSort, 164
ShellSort, 160
orientação a objetos
frameworks, 171, 181
otimalidade
local
correlação, 66
medida de aproximabilidade, 43
otimização
microcanônica, 79
problema
maximização, 40
minimização, 40
resolver
definição, 40
otimização
com restrições, 133

paisagem
de aptidão, 93
definição, 26
paralelismo assíncrono, 171
perceptron binário
problema de aprendizado, 106
PMA
papel das restrições, 143
Polynomial Local Search, 28
Polynomial Merger Algorithms
definição, 132
população
estruturada
árvore ternária, 97
modelos de ilhas, 60, 97
hierárquica, 53
líder, 60, 97
subordinados, 60, 97
potencial dinástico
definição, 131
preparticionamento
como representação, 94
problema
equal piles, 88
NP otimização
definição, 39
NPO
função objetivo, 40
de revisão de crenças, 181
designação
como relaxação para o MIN ATSP, 54
instância, 39
otimização
$goal_F$, 40
MIN HITTING SET, 115
otimização combinatória
instância resolvida, 22
parameterizado
HITTING SET FOR SIZE THREE SETS, 173
VERTEX COVER, 173
perceptron binário
aprendizado, 106
problemas
$W[1]$-completos
CLIQUE, 139
$W[2]$-completos
DOMINATING SET, 140
INDEPENDENT DOMINATING SET, 140
NP-completos, 85
no sentido forte, 3
classe NP, 5
classe uPMA
definição, 131
combinatorial
definição, 22
confecção de tabelas de horários, 35, 144

contagem
definição, 22
de decisão, 5
papel da busca binária, 41
subotimalidade, 42
decisão
definição, 22
em hipergrafos, 174
enumeração
definição, 22
escalonamento de jogos deportivos, 35
multi-objetivos
para guiar MAs, 175
não aproximáveis, 143
otimização
definição, 22
otimização combinatória multiobjetivo, 35
parameterizados
k-Hitting Set For Size Three Sets, 115
k-Vertex Cover, 114
Planar Dominating Set, 114
definição, 113
tratabilidade de parâmetro fixo, 114
parametrizados
ptas implican FPT, 124
FPT uniforme, 123
relação com *fptas*, 123
transformação, 124
visão invariariante de FPT, 123
redução, 4
visão de computadores, 145
Programação Lógica, 182
Programação Linear, 2
Projeto *MemePool*, 171
pseudo-polinomial, 3
algoritmo, 86
pseudopolinomial
algoritmo, 49
PTAS, 6
pureza, 28

QuickSort
pseudocódigo, 164

raciocínio epistémico aninhado, 180
randomizado
algoritmo, 9
razão de desempenho
medida de aproximação, 45
melhor, 47
assintoticamente, 47
rebelde
comportamento, 99
recombinação, 28, 130, 171
k-merger, 131
Distance Preserving Crossover, 66
Enhanced Edge, 64
Multiple Fragment-Nearest Neighbor Repair, 66
Strategic Edge Crossover, 64
Uniform Crossover, 66
Uniform Nearest Neighbor Crossover, 66
a partir de algoritmos FPT, 139
análise
de caso promedio, 171
de pior caso, 171
assortment
definição, 152
cega, 29, 134
Crossover Uniforme, 29
cego, 144
classe PMA, 132
classe uPMA, 131
como transições generalizadas, 28
complexidade computacional, 173
comportamento
bajulador, 99
conciliador, 99
rebelde, 99
comportamentos, 99, 176, 180
adaptativos, 176
crença distribuída, 177
crossover
definição, 134
de algoritmos FPT, 139
de dois pontos, 56

dinástica ótima, 30, 113, 173
distance preserving, 141
e busca local
sinergia, 142
em população hierárquica, 98
heurística ou híbrida, 30
miopia, 134
multiples pais, 134
mutação implícita, 29
pai-simétrico, 135
papel da randomização, 171
patching
by forma completion, 30
pureza, 28, 134
respeito, 28, 99
definição, 152
sem respeito, 99
sortimento, 28
transmisão, 29, 99
transmitting
definição, 152
uniforme, 99
Recombinação Dinástica Ótima, 30
redes
de estações de trabalho, 171
neurais
para o MIN ATSP, 54
redução, 4
a um núcleo de problema, 114, 173
relação
de vizinhança, 23, 24
reparação gulosa, 99
com semente aleatória, 99
representação
binária direta
e mutação simple, 104
mutação minimal, 100
mutação simple, 100
de soluções, 86
direta, 96
preparticionamento, 94
restrições parametrizadas de Ruml, 95
ternária direta, 96
e mutação simple, 105
válida, 24
reprodução
crossover
cego, 100
uniforme, 100
respeito, 28, 99
respeitosa do custo
medida de aproximação, 45
resposta
definição, 22
restrição de um grafo, 17

Sedgewick
desafio, 156, 161
ShellSort
pseudocódigo, 161
seqüência de Knuth, 161
Simplex, 2
sinergia
da recombinação e a busca local, 142
sistemas desordenados
ground-state, 110
sistemas distribuídos, 171
solução
Current, 61
Pocket, 61
sortimento
fraco, 28
subordinados, 60
em população hierárquica, 97
subotimalidade, 42

T-norma, 180
tabelas de horários
confecção
problemas de, 35
Teoria dos Formae, 152
tour
euleriano, 46
transformações
entre problemas
parametrizada, 124
transição
de fase, 103
CKK e MIN NUMBER PARTITIONING, 103
mutação, 27
transições entre configurações, 24

transmisão
em recombinação, 99
generalizada, 99
tratabilidade de parâmetro fixo
definição, 114

UBX, 164
UCX, 159–161

vertices
de un grafo
conjunto dominante, 140
vidro de spin de Ising, 110
vizinhança de grupo, 133

West
conjectura de, 152