

Uma Metaheurística Genética Não Convencional e *Ant Colony Systems* para resolver o Problema do Caixeiro Viajante com Grupamentos

PARAMAHANSA WERNER POLO VIEYRA

Dissertação apresentada ao Curso de Pós-Graduação
em Ciência da Computação da Universidade Federal
Fluminense, como requisito parcial para obtenção
do Grau de Mestre. Área de Concentração:
Inteligência Artificial e Otimização Combinatória.

Orientador: Prof. Dr. Luiz Satoru Ochi.

NITERÓI
1999

A minha família com amor,
e ao Senhor, meu melhor amigo

Agradecimentos

Para mim é um enorme prazer registrar a minha gratidão as pessoas que me apoiaram durante o curso de mestrado.

Ao professor Luiz Satoru Ochi, pela sua amizade e orientação como pesquisador e pessoa humana. Por ter sido tão paciente e sempre ter acreditado no meu trabalho. Esta tese tem a sua marca, pois foi feita dentro daquilo que pude aprender sobre sua visão e supervisão neste trabalho em conjunto.

Esta tese teve uma grande incentivadora, uma das pessoas que mais me apoiaram em todos os momentos. Mesmo naqueles momentos mais difíceis. Possivelmente sem sua ajuda meu trabalho não teria sido concluído. Gilda Dahis eis aqui o resultado do seu crédito, de seu apoio, e de sua amizade.

A minha mãe, a Katherine Verástegui e ao meu filho Guisseppe Polo, seres queridos que me apoiaram sempre nos meus estudos, e que sempre facilitaram as coisas nas horas mais difíceis.

*A meu amigo Marco Antônio Xaves Valentim e ao pessoal da turma *.93 do curso de informática: Dalessandro Vianna, Patricia Garces Rabelo, Geiza Hammasaki, Bazilio, Armis Moska, Edson, Robertinha, Francine, Erika, Adriana Bechara, Adriana Mendez, Junior, Fernando (o vovô da turma) pelo grande apoio dado para superar os muitos obstáculos e tentar concluir este trabalho e pela valiosa grande amizade.*

Agradeço a todos que me deram apoio neste duro caminho, em especial ao pessoal do PROTEM e aos companheiros da minha turma que sobreviveram no mestrado: Lino, Marcelo Módica, Marcelo Lisboa, Ricardo Pinheiro (... linux forever!!!), José Claudio, Márcia, Estela, Eduardo e a você Jaqueline (nunca esquecerei os nossos papos) e a lembrança dos outros que de alguma forma e em algum momento me apoiaram: Marcus Vinicius, Sônia, Denilton, Demerson. Aos novos companheiros do mestrado Luciene, Daniela e Waldir, gracias, sejam perseverantes e não deixem que seus esforços sejam envãos, depois de tudo, até que não é tão di ...

Agradeço ao prof. Carlos Martinhon e a todos os professores da Pós-Graduação, agradeço também ao Carlinhos, à Angela e ao pessoal da secretaria do Departamento de Ciências da Computação da UFF.

Resumo

Problemas freqüentemente encontrados, são os problemas combinatórios. Muitos deles consistem da busca, sobre um dado grafo, de um conjunto de arestas que satisfaçam certas condições. Um desses problemas é o Problema do Caixeiro Viajante (*TSP – Traveling Salesman Problem*), que consiste na determinação do ciclo hamiltoniano de custo mínimo sobre um dado grafo.

Uma das várias extensões do TSP é o Problema do Caixeiro Viajante com Grupamentos (*CTSP – Clustered Traveling Salesman Problem*), onde o conjunto de vértices é particionado em subconjuntos ou grupamentos onde todos os vértices de cada grupamento devem ser visitados contiguamente. O CTSP é um problema NP-Árduo, o que limita o uso exclusivo de técnicas exatas; por isso, a maioria dos trabalhos encontrados na literatura apresentam procedimentos aproximados ou heurísticas para sua solução.

Atualmente, muitos dos métodos computacionais para a solução de problemas complexos estão inspirados em processos naturais. Estes métodos surgem da interpretação de mecanismos usados pela natureza (tais como: evolução, adaptação e memória) como princípios para a solução inteligente de problemas.

Dois métodos inspirados na natureza e que tem sido independentemente aplicados com sucesso na solução do TSP e de outros problemas, são os Algoritmo Genéticos e o Sistemas de Colônia de Formigas (*ACS - Ant Colony Systems*), os quais compartilham a característica de trabalhar com uma população de indivíduos.

Como os Algoritmos Genéticos determinam rapidamente regiões promissoras no espaço de busca, eles tem sido freqüentemente aplicados em combinação com métodos de busca local ou de refinamento da melhor solução encontrada. Por outro lado, o *Ant Colony System* é um método que tem sido aplicado com sucesso a distintos problemas, porém, devido ao fato de que a atualização da trilha de feromônios e a emulação de

algumas características comportamentais das formigas consomem um excessivo tempo computacional, este método só tem tido sucesso quando aplicado a instâncias pequenas. O que faz necessário o desenvolvimento de técnicas para superar esta desvantagem.

Neste trabalho, apresentamos uma forma de combinar estes dois métodos, numa metaheurística que usa um Algoritmo Genético e um Ant Colony System. Nosso principal objetivo é melhorar o tempo computacional requerido por um *Ant Colony System* e comparar os efeitos do uso de uma lista de cidades candidatas no ACS com o esquema de inicialização do ACS proposto neste trabalho quando tentamos resolver instâncias grandes do CTSP.

Nesta metaheurística, propomos um algoritmo genético não convencional que introduz uma nova forma de manipulação da população; que consiste na criação de partições na população para selecionar tanto os indivíduos que entrarão no processo de reprodução quanto os indivíduos que serão substituídos pelos filhos.

Abstract

Frequently found problems, are the combinatorial problems. Many of them consist of the search, on a given graph, for a set of edges satisfying certain conditions. One of these problems is the Traveling Salesman Problem (TSP), which consists of the determination of the minimum cost Hamiltonian cycle in the graph.

One of the several extensions of TSP is the Clustered Traveling Salesman Problem (CTSP), where the vertex set is partitioned into subsets or "clusters" of vertices that must be visited contiguously. This problem is a NP-Hard problem, it limits the exclusive use of exact techniques. Therefore, most of the works found in the literature present approximate procedures or heuristics for its solution.

Nowadays, many of the computational methods for the solution of complex problems are inspired in natural processes. These methods appear from the interpretation of mechanisms used by the nature (as the evolution, the adaptation and the memory) as principles for the intelligent solution of problems.

Two bio-inspired methods, that have been applied independently with success in the solution of this and other problems, are the Genetic Algorithms and the Ants Colony Systems which share the characteristic of working with a population of individuals.

As the genetic algorithms determine quickly promising areas in the search space, they are frequently used with local search or refinement of the best-found solution methods. On the other hand, the Ant Colony System is a method that has been applied with success to different problems. However, due to the fact that the updating of the pheromone trail and the emulation of some behavioral characteristics take a meaningful computational time, this method have succeeded only when it was applied to small instances. Hence, it became necessary to develop ways to overcome these disadvantages. The Ant Colony System is a slow learning system that becomes more efficient as it acquires knowledge.

In this work, we present a metaheuristic that uses a Genetic Algorithm and an Ant Colony System, where is showed a form of transmission of the knowledge acquired from the genetic algorithm to the ant colony. The principal objective is the improvement of the total computational time used for the Ant Colony System and the to compare the effects of candidates' list used for the colony in the ACS with the initialization scheme for the ACS proposed in this work when it is applied to big instances of the Clustered Traveling Salesman Problem.

In this metaheuristic, the genetic algorithm introduces a non-conventional form of manipulation of the population, which consist in performing partitions into the populations to select both the individuals who will be reproduced and those who will be substituted by the offsprings.

- Índice -

ÍNDICE DE FIGURAS	iv
CAPÍTULO 1: Introdução	1
1.1 Organização da Monografia	3
1.2 Heurísticas	4
1.3 Metaheurísticas	5
1.4 Exploração x Exploração	6
1.5 Alguns métodos clássicos de otimização	7
1.5.1 Busca Aleatória	7
1.5.2 Métodos do Gradiente	8
1.5.3 Busca Iterativa	9
CAPÍTULO 2: O Problema do Caixeiro Viajante	10
2.1 Formulações Matemáticas para o <i>TSP</i>	11
2.2 Heurísticas para o <i>TSP</i>	15
2.2.1 Heurística do Vizinho Mais Próximo	15
2.2.2 Heurística de Christófidis	17
2.2.3 Método das K-Substituições ou K-Optimal	19
2.2.4 A Heurística <i>GENIUS</i>	21
CAPÍTULO 3: O Problema do Caixeiro Viajante com Grupamentos	23
3.1 Formulação Matemática do <i>CTSP</i>	25
3.2 Variações do <i>CTSP</i>	27
3.3 Algoritmos para solucionar o <i>CTSP</i> e suas variações	28
CAPÍTULO 4: Agentes Inteligentes	31
4.1 O que é um agente?	31
4.2 Propriedades dos Agentes	33
4.3 Seleção de Ações e Aprendizado	35
4.3.1 Reflexões sobre a Seleção de Ações	35
4.3.2 Reflexões sobre o Aprendizado através da experiência	35
4.4 Selecionando o Foco de Aprendizado	36
CAPÍTULO 5: Uma Breve Introdução ao Aprendizado Mediante Reforços	38
5.1 O Modelo de Um Agente no Aprendizado Mediante Reforços	40
5.1.1 A Política	41
5.1.2 A Função de Reforços	42
5.1.3 A Função de Valor	43
5.1.4 O Ambiente	46
5.1.5 A Interface Agente-Ambiente	46
5.2 Exploração x Exploração no RL	49
5.3 Modelos de Conduta Ótima	49
5.4 Aproximando a Função de Valor	52
5.4.1 A Essência da Programação Dinâmica	52
5.4.2 <i>Value Iteration</i>	55
5.4.3 <i>Q-Learning</i>	57

CAPÍTULO 6: Uma Introdução ao Ant Colony Systems	60
6.1 Um Modelo do Comportamento de uma Formiga Real	63
6.2 <i>Ant Colony System</i> : Um Modelo Computacional com Formigas Artificiais	65
6.3 <i>Ant Colony System</i> aplicado ao <i>TSP</i>	66
6.3.1 A Regra de Atualização Local	68
6.3.2 A Regra de Atualização Global	68
6.4 <i>Ant Colony System</i> aplicado ao <i>CTSP</i>	71
6.4.1 Modificações nos parâmetros do Algoritmo	71
6.4.2 Inicialização da Matriz de Feromônios	72
6.4.3 Adaptação na memória	72
6.4.4 Escolha da Cidade Inicial	72
6.4.5 Escolha da próxima cidade	73
6.4.5.1 Escolha da próxima cidade dentro de um mesmo grupamento	73
6.4.5.2 Escolha da próxima cidade em um outro grupamento	74
CAPÍTULO 7: Uma Introdução aos Algoritmos Genéticos	76
7.1 O Modelo dos Algoritmos genéticos	77
7.2 Princípios Básicos	79
7.2.1 A Codificação	79
7.2.2 A Função de Aptidão	81
7.3 As fases de um Algoritmo Genético	82
7.3.1 A Seleção	82
7.3.1.1 Seleção Proporcional à Aptidão (<i>Proportional Selection</i>)	83
7.3.1.2 Seleção por Mérito (<i>Rank Selection</i>)	83
7.3.1.3 Seleção por Competição (<i>Tournament Selection</i>)	84
7.3.1.4 Seleção por Elitismo (<i>Elitist Selection</i>)	84
7.3.2 A Reprodução	84
7.3.2.1 A Recombinação	85
7.3.2.2 A Mutação	86
7.3.3 A Substituição	87
7.4 Aspectos Práticos	88
7.4.1 Definição dos parâmetros nos Algoritmos Genéticos	88
7.4.2 Inicialização - População inicial	88
7.4.3 Estratégias de Evolução	89
7.4.4 Inserção de Genes na População	89
7.5 AGs e outros métodos	90
CAPÍTULO 8: O Algoritmo Genético Proposto	91
8.1 Descrição do Algoritmo Genético Proposto	93
8.1.1 Representação de uma solução e Criação da População Inicial	93
8.1.2 Evolução da População	93
8.1.2.1 Particionamento da População	94
8.1.2.2 Cálculo da aptidão para a escolha dos pais para a próxima geração	95
8.1.2.3 Escolha dos pais para a próxima geração	97
8.1.2.4 Aplicação do operador de recombinação e geração dos novos filhos	100
8.1.2.5 Cálculo da aptidão para determinar os indivíduos a serem substituídos	100
8.1.2.6 Escolha dos indivíduos a serem substituídos	102
8.1.2.7 Substituição dos cromossomos selecionados pelos novos filhos	103

8.1.3 Verificação do Critério de Convergência da População	104
8.1.4 Avaliação do Critério de Parada do Algoritmo	104
8.1.5 Inserção de Genes na População	104
8.2 Aplicação do Algoritmo Genético Proposto ao <i>CTSP</i>	105
8.2.1 Representação de uma solução	105
8.2.2 Inicialização da População de Indivíduos	106
8.2.3 Recombinação dos genes	108
8.2.3.1 O operador de recombinação <i>ERX</i> (<i>Edge Recombination Crossover</i>)	108
8.2.3.2 O operador de recombinação <i>ERX</i> adaptado ao <i>CTSP</i>	112
8.2.4 Inserção de Genes na População	113
8.2.4.2 A Mutação	114
CAPÍTULO 9: Algoritmo Genético Proposto com Ant Colony Systems para solucionar o CTSP	115
9.1 Parâmetros do algoritmo	116
9.2 A estrutura do algoritmo	116
CAPÍTULO 10: Resultados de Testes e Conclusões	119
10.1 Resultados de Testes	119
10.1.1 Avaliação dos resultados obtidos pela metaheurística proposto	119
10.1.2 Experiências	122
10.2 Conclusões	125
10.3 Trabalhos Futuros	126
APÊNDICE A: Uma Visão Histórica Simplificada do Aprendizado Mediante Reforços	127
BIBLIOGRAFIA	133

- Índice de Figuras-

Figura 1.1: Um algoritmo de programação matemática aplicado a um problema multimodal.	7
Figura 1.2: O Problema do hill climbing.	8
Figura 2.1: Exemplo de uma solução do <i>TSP</i> .	10
Figura 2.2: Grafo multiestágio construído a partir de uma instância do <i>TSP</i> .	13
Figura 2.3: Grafo multiestágio a partir do qual foi feito um modelo para o <i>TSP</i> .	13
Figura 2.4: Solução do <i>TSP</i> utilizando-se a Heurística do Vizinho Mais Próximo.	16
Figura 2.5: Solução do <i>TSP</i> , utilizando-se a heurística de Christófides.	18
Figura 2.6: Método 2-Optimal.	20
Figura 2.7: Os dois tipos de inserção GENI.	22
Figura 3.1: Exemplo de uma instância para o CTSP.	24
Figura 3.2: Particionamento da matriz de custos.	26
Figura 4.1: A interação entre a estrutura, a tarefa e o ambiente de um organismo influenciam sua conduta.	32
Figura 4.2: Existem muitas outras aplicações para agentes.	37
Figura 5.1: Componentes principais de um Agente no Aprendizado Mediante Reforços.	41
Figura 5.2: Função de valor para uma política aleatória.	45
Figura 5.3: a) função de valor ótima. b) política ótima para a função ilustrada na figura 5.3a.	45
Figura 5.4: O modelo padrão de Aprendizado Mediante Reforços..	47
Figura 5.5: Representação da relação entre um agente e seu ambiente através de um diálogo fictício.	47
Figura 5.6: Uma rede de estados e ações para a comparação de modelos de conduta ótima.	51
Figura 5.7: Uma cadeia de Markov simples.	54
Figura 5.8: Cadeia de Markov simples sem estado terminal.	55
Figura 5.9: Escopo de uma atualização da aproximação do valor do estado x_t .	56
Figura 5.10: Uma rede de estados e ações para a ilustração do problema enfrentado pelo Value Iteration.	58
Figura 5.11: Escopo de uma atualização da aproximação do Q-Value do estado x_t .	59
Figura 6.1: A influência dos Feromônios sobre as formigas reais, na escolha do menor caminho.	62
Figura 6.2: Modelo do comportamento de uma formiga real .	64
Figura 6.3: Evaporação e Difusão da Trilha de Feromônios.	65
Figura 6.4: O Algoritmo de Colônia de Formigas para o <i>TSP</i> .	70
Figura 6.5: A construção de uma lista de cidades candidatas à cidade inicial.	73
Figura 6.6: A construção de listas de grupamentos adjacentes.	74
Figura 7.1: A estrutura de um algoritmo genético.	77
Figura 7.2: Comparação entre conceitos da Evolução Natural e o modelo de Algoritmos Genéticos.	79
Figura 7.3: Reprodução através do Operador crossover clássico.	85
Figura 7.4: Um exemplo genérico de mutação (mutação simples).	87
Figura 8.1: Estrutura básica do algoritmo genético proposto.	92
Figura 8.2: Na população ordenada são criadas partições.	95
Figura 8.3: Algoritmo para Cálculo da aptidão para a escolha dos pais para a próxima geração.	97
Figura 8.4: Algoritmo para escolher os pais da próxima geração.	99

Figura 8.5: Algoritmo para o cálculo da aptidão de substituição dos cromossomos de cada partição.	101
Figura 8.6: Algoritmo para a seleção dos cromossomos a serem substituídos.	103
Figura 8.7: A representação de uma solução para o CTSP na aplicação do algoritmo.	106
Figura 8.8: Geração de um cromossomo para a população inicial.	107
Figura 8.9: Uma instância do TSP.	108
Figura 8.10: Listas de cidades adjacentes construídas a partir de P1 e P2.	109
Figura 8.11: Cidades candidatas para a próxima escolha.	109
Figura 8.12: Continuação do processo de geração do filho F.	110
Figura 8.13: Ciclos Hamiltonianos representados por P1, P2 e F.	110
Figura 8.14: Geração do filho F com através do operador ERX otimizado.	111
Figura 8.15: Ciclos Hamiltonianos representados por P1, P2 e F.	112
Figura 8.16: A diversificação da população.	114
Figura 8.17: A mutação.	114
Figura 9.1: O algoritmo Híbrido Proposto.	117
Figura 9.2: Uma representação do algoritmo Híbrido Proposto.	117
Figura 10.1: Valores dos parâmetros de nossa metaheurística utilizados nos testes.	120
Figura 10.2: Tabela comparativa dos custos das melhores soluções obtidas pela metaheurística de Rabelo et. al, pela heurística H2 de Gendreu e pela metaheurística proposta neste trabalho.	121
Figura 10.3: Tempos de CPU consumidos pela Metaheurística de Rabelo et. al, pela heurística H2 de Gendreu e pela metaheurística proposta neste trabalho	122
Figura 10.4: Tabela comparativa dos custos das melhores soluções obtidas por nossa metaheurística com e sem listas de cidades candidatas acopladas à colônia de formigas.	123
Figure 10.5: Gráfico mostrando os valores médios das soluções obtidas pelas duas versões da Metaheurística Proposta (MP) e da colônia de formigas pura (ACS puro), ambas com e sem listas de cidades candidatas.	124
Figure 10.6: Gráfico mostrando os tempos totais médios consumidos pelas duas versões da Metaheurística Proposta (MP) e da colônia de formigas pura (ACS puro), ambas com e sem listas de cidades candidatas.	125

Capítulo 1

Introdução

Uma parcela importante dos problemas tecno-científicos consiste em questões de busca; isto é, dada uma certa coleção de elementos, deseja-se encontrar nela um ou mais elementos que atendam a algumas condições previamente especificadas. Alguns destes problemas possuem elevado nível de complexidade computacional (NP-Completo, NP-Árduo¹) e têm sido um desafio constante para os pesquisadores de diversas áreas. Particularmente, nas áreas de Otimização, Pesquisa Operacional, Ciência da Computação, Matemática e Engenharia; freqüentemente, nos defrontamos com problemas altamente combinatórios, muitos dos quais consistem na busca, sobre um dado grafo, de um conjunto de arestas que satisfaçam a certas condições. Encontrar uma solução ótima em um tempo computacional aceitável para estes problemas, em muitos casos, só é possível para pequenas instâncias; e mesmo assim, as vezes o problema apresenta tantas dificuldades que fica-se satisfeito com alguma solução que atenda a todas as restrições impostas.

Apesar do grande desenvolvimento teórico que se tem observado no estudo de novos métodos para problemas complexos, este avanço tem sido, em alguns casos, pouco produtivo para a solução de problemas reais. Em parte, a responsabilidade talvez possa ser atribuída aos próprios pesquisadores destas áreas, que em seus trabalhos acadêmicos têm priorizado o estudo de casos particulares de modelos reais. Por exemplo, um dos problemas de otimização mais estudados é o Problema do Caixeiro Viajante (*TSP - Traveling Salesman Problem*), cuja maioria de aplicações em problemas práticos, envolve um elevado número de restrições adicionais, como por exemplo o Problema de gerar rotas para uma frota de veículos que pode ser composto de: centenas de veículos heterogêneos ao invés de um só veículo como no *TSP*; restrições de *scheduling* do tipo:

¹ Vide [CORMEN, 90] para um estudo sobre classes de problemas.

time-windows, precedências e tempo limite para concluir rotas; restrições de capacidade de cada veículo ao invés da capacidade ilimitada *TSP*; dados não determinísticos tais como: velocidade dos veículos, demanda de clientes, frequência de atendimento de cada cliente, etc.

Desta forma, apesar da importância inegável em abordar casos particulares, a falta de pesquisas em modelos mais próximos às situações reais criou uma distância considerável entre algumas pesquisas acadêmicas e as necessidades do mercado atual. Em particular, os métodos de otimização exata, sempre se caracterizaram pela rigidez de seus modelos matemáticos representados através de seus teoremas, dificultando a representação de situações reais cada vez mais complexas e dinâmicas. O problema desta falta de flexibilidade foi um pouco reduzido a partir do momento em que se passou a associar técnicas de otimização com ferramentas de Inteligência Artificial, mais especificamente, às ferramentas de busca heurística.

Muitos dos atuais métodos de busca estão inspirados na interpretação de mecanismos usados pela natureza (evolução, adaptação, memória e etc.) como princípios para a solução inteligente de problemas complexos. Esta interpretação inclui o processo de criar estruturas para o uso vantajoso do conhecimento prévio e o uso deste conhecimento para a obtenção de boas soluções.

Dois dos métodos inspirados na natureza e que têm sido aplicados independentemente com sucesso à solução do *TSP* e de outros problemas combinatórios, são os Algoritmos Genéticos [MICHALEWICZ, 92] e o *Ant Colony System* [GAMBARDELLA, 97A].

Neste trabalho apresentamos uma nova metaheurística que combina um algoritmo genético e o *Ant Colony System* para a solução do Problema do Caixeiro Viajante com Grupamentos (*CTSP – Clustered Traveling Salesman Problem*) [CHISMAN, 75], que é uma generalização do *TSP*.

1.1 Organização da Monografia

A seguir descrevemos o conteúdo de cada capítulo desta monografia:

Nas seções seguintes deste capítulo, serão descritos os conceitos de heurísticas, metaheurísticas, exploração e exploração, além de alguns métodos de busca.

O capítulo 2 descreve o problema do caixeiro viajante (*TSP*) e algumas das heurísticas desenvolvidas para resolver este problema. O capítulo 3 descreve nosso problema de interesse, o problema do caixeiro viajante com grupamentos (*CTSP* - *Clustered Traveling Salesman Problem*), uma generalização do *TSP*.

Nos capítulos 4 e 5, se apresenta os conceitos necessários para a leitura do capítulo 6 sobre *Ant Colony System*. As noções de Agentes, Comportamento Emergente (capítulo 4) e Aprendizado Mediante Reforços (capítulo 5) são introduzidas nestes dois capítulos.

O capítulo 6 descreve uma das metaheurísticas utilizadas neste trabalho: o *Ant Colony System* aplicado ao *TSP*. Na última seção deste capítulo, esta descrita nossa adaptação desta técnica para o *CTSP*.

O capítulo 7, aborda a outra metaheurística utilizada neste trabalho: os Algoritmos Genéticos.

No capítulo 8, apresentamos o nosso algoritmo genético, o qual introduz algumas novidades com relação aos algoritmos genéticos encontrados na literatura. Neste capítulo é mostrada uma aplicação deste algoritmo ao *CTSP*, incluindo uma adaptação do operador *ERX* utilizado para resolver instâncias do *TSP* [WHITLEY, 89].

O capítulo 9 descreve a metaheurística proposta, resultante de uma combinação entre o algoritmo genético para o *CTSP* e nossa adaptação do *Ant Colony System* para o *CTSP*.

O capítulo 10 apresenta as conclusões da pesquisa, incluindo resultados de testes, comparações com outros algoritmos e sugestões para trabalhos futuros.

A monografia apresenta também um apêndice. O apêndice A narra informações históricas sobre o desenvolvimento da Teoria dos Agentes Inteligentes.

1.2 Heurísticas

Os algoritmos heurísticos, ou simplesmente heurísticas, são métodos para solucionar problemas complexos de forma aproximada. Baseiam-se nas propriedades estruturais ou nas características das soluções dos problemas. Sua complexidade computacional é reduzida em relação à dos algoritmos exatos. Elas se caracterizam pela sua flexibilidade e têm como objetivo, encontrar soluções de boa qualidade num tempo computacional suportável. Contudo, as heurísticas isoladamente também possuem suas limitações, e a principal delas, é a deficiência histórica de nem sempre conseguirem superar as armadilhas de extremos (máximos ou mínimos) locais em Problemas de Otimização. Além disso, a falta de uma base teórica para os métodos heurísticos produz heurísticas muito especializadas. Ou seja, apesar de sua flexibilidade em incorporar novas situações, o seu desempenho pode oscilar muito quando são feitas modificações no problema analisado, mesmo que pequenas.

Os processos de resolução dos problemas de busca dependem fortemente do ambiente no qual a busca é feita e das condições a serem satisfeitas pela solução. De acordo com sua generalidade, os algoritmos de busca podem ser classificados como fracos, quando podem ser aplicados a uma gama variada de problemas, ou fortes, quando são projetados para aplicações mais específicas, fazendo uso de uma quantidade maior de informações disponíveis para tais situações. À medida que aumenta o conhecimento sobre o espaço de procura, especializam-se os algoritmos de busca para este espaço, com o aumento de eficiência e perda de generalidade. Assim, pode-se passar de algoritmos mais fracos para algoritmos mais fortes.

Outra classificação importante é segundo a inclusão ou não de algum componente probabilístico no algoritmo. Quando alguma decisão é tomada a partir de um valor gerado aleatoriamente, diz-se que o algoritmo é probabilístico e, em caso contrário, ele é dito determinístico. No primeiro caso, o gerador de números aleatórios é sempre inicializado com algum valor (fornecido pelo usuário, por exemplo) e assim, cada execução do algoritmo pode, em princípio, levar a um resultado diferente. À primeira vista, pode parecer estranho que um algoritmo com elementos probabilísticos possa ter alguma utilidade prática, já que, de uma maneira geral, nos acostumamos com algoritmos determinísticos eficientes em diversas áreas. Em muitas situações porém, algoritmos probabilísticos desempenham papel importante, por exemplo, quando muitos dados do problema não são previamente conhecidos, o uso de técnicas determinísticas torna-se mais difícil.

1.3 Metaheurísticas

A partir dos anos 80, surgiram alguns trabalhos sobre novos métodos heurísticos com ferramentas para tentar superar as limitações das heurísticas convencionais. A reunião dos modelos rígidos da otimização com os métodos flexíveis da busca heurística proporcionou o surgimento dos chamados Métodos Inteligentes Flexíveis, que procuram o desenvolvimento de métodos com uma certa rigidez matemática e com facilidades em incorporar novas situações e superar algumas deficiências apresentadas pelos métodos heurísticos. Estas técnicas são classificadas na literatura como Metaheurísticas; e são procedimentos que têm solucionado deficiências históricas dos algoritmos convencionais de busca heurística, como por exemplo, o problema de paradas prematuras em ótimos locais distantes da melhor solução em problemas de otimização.

1.4 Exploração x Exploração

Qualquer algoritmo eficiente de otimização aproximada deve usar duas técnicas para tentar encontrar um extremo global: a **exploração** para pesquisar áreas novas e desconhecidas, no espaço de busca; e a **exploração** para fazer uso do conhecimento adquirido nos pontos previamente visitados afim de encontrar pontos melhores. Assim, um algoritmo de busca deverá ter a melhor combinação possível de suas capacidades de exploração e exploração.

Para exemplificar essas duas características desejáveis, mas em geral, conflitantes num algoritmo de busca, considere o problema enfrentado por uma companhia de petróleo. Numa primeira fase, ela deve ser capaz de identificar dentre diversas regiões - seu espaço de busca - as áreas mais promissoras no que tange à existência de petróleo de qualidade e em quantidade. É a etapa de exploração. Mas isso não é tudo. Uma vez localizada uma região promissora, é necessário, de maneira eficiente, retirar a maior quantidade possível de petróleo utilizando o mínimo possível de recursos. Esta etapa recebe o nome de exploração.

Em otimização, uma boa capacidade de exploração do espaço de busca é requerida se é desejado encontrar um extremo (máximo ou mínimo) global, ou pelo menos, uma boa solução. Algoritmos clássicos aproximados da programação matemática são projetados freqüentemente para problemas unimodais, isto é, com apenas um extremo local (e portanto global), sendo essencialmente voltados ao aspecto da exploração. Quando aplicados a um problema multimodal, sua convergência se dará para um extremo local, raramente a solução global, que dependerá do ponto de partida utilizado, conforme ilustrado na figura 1.1.

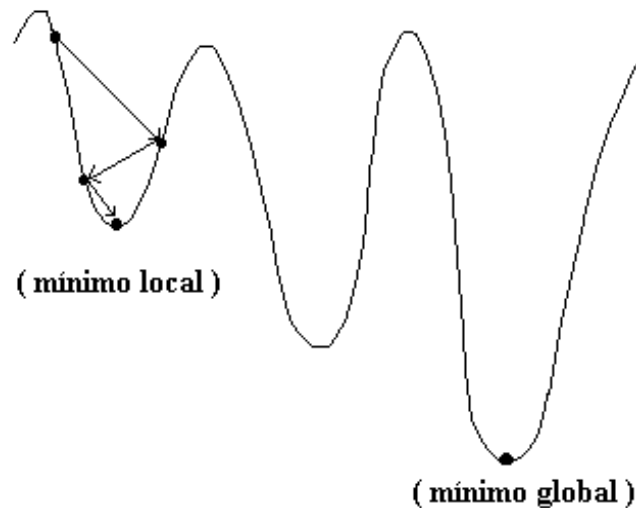


Figura 1.1: Um algoritmo de programação matemática aplicado a um problema multimodal.

1.5 Alguns métodos clássicos de otimização

A seguir apresentamos alguns métodos heurísticos clássicos utilizados para solucionar problemas de otimização:

- Busca Aleatória
- Método do Gradiente
- Busca Iterativa

Estes métodos serão apresentados para exemplificar métodos heurísticos e para apresentar algumas das deficiências de tais métodos.

1.5.1 Busca Aleatória

Busca Aleatória ou Enumerativa é um método de “força bruta” para funções difíceis. Pontos do espaço de busca são escolhidos aleatoriamente, ou de um modo sistemático e suas aptidões são avaliadas. Este método pouco inteligente é raramente usado de forma isolada pois consome um tempo computacional extremamente elevado.

1.5.2 Métodos do Gradiente

Desenvolveram-se diferentes métodos para otimizar funções contínuas “bem comportadas”, os quais utilizam o gradiente da função objetivo para guiar a direção da busca. Ainda que esta técnica seja utilizada para resolver problemas de otimização contínua; ela é útil para mostrar claramente outra deficiência dos métodos heurísticos: a de ficar presos em um ponto extremo.

Tais métodos são geralmente referenciados como Métodos de Subida (*hill climbing*) em problemas de maximização. Uma das características destes métodos é que eles se comportam bem com funções que possuem um único ótimo (*funções unimodais*). Mas, com funções com muitos pontos extremos (*funções multimodais*), eles apresentam o problema de ficarem presos no primeiro ponto extremo encontrado (pois nesse ponto o gradiente é zero), mas este ponto poderia não ser um extremo global.

Desse modo, uma vez atingido um extremo local nenhum progresso pode ser feito. Um exemplo, é apresentado na figura 1.2, cujo gráfico mostra uma função que está sendo maximizada. A busca de subida (*hill climbing*) começa no ponto X, escolhido aleatoriamente. Fazendo-se uso da informação dada pelo gradiente (crescimento da função), movimentos de subida são realizados até que o ponto B é alcançado e a busca termina. Porém, pontos de maior valor, como A e C, podem não ser encontrados.

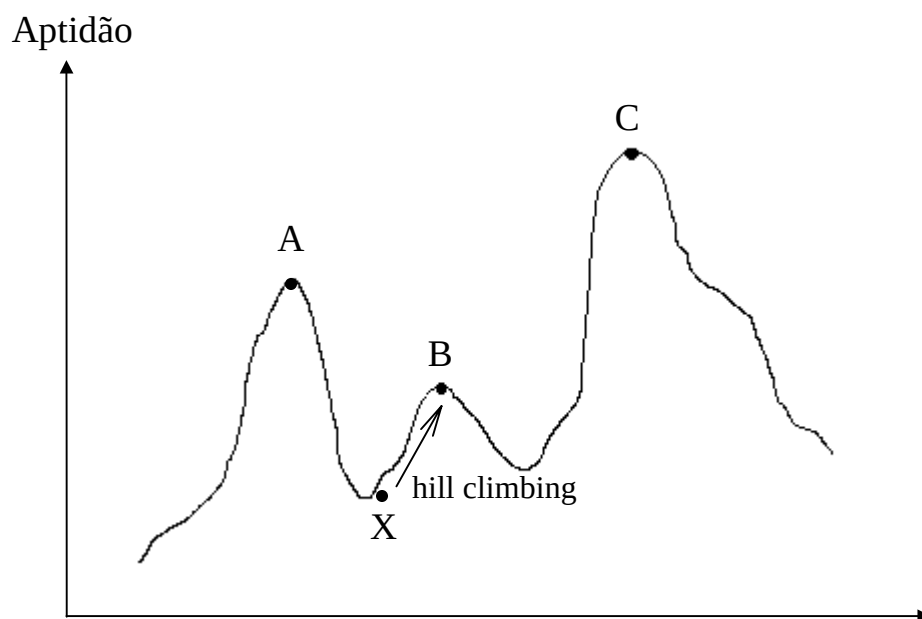


Figura 1.2: O Problema do hill climbing.

1.5.3 Busca Iterativa

A busca aleatória e a busca pelo gradiente podem ser combinadas para obter uma busca de subida (hill climbing) iterativa. Uma vez que um ponto extremo é atingido, a busca de subida (hill climbing) é recomeçada, mas a partir de um outro ponto de partida, escolhido aleatoriamente. Esta técnica possui as vantagens de ser simples e de se comportar bem quando a função que está sendo otimizada não possui muitos extremos locais.

Uma busca puramente aleatória é boa em exploração, mas não faz exploração nenhuma; enquanto que, um método de subida é bom em exploração, mas realiza muito pouca exploração. A combinação destas duas estratégias pode ser bastante efetiva, mas é difícil saber qual o melhor balanceamento.

Capítulo 2

O Problema do Caixeiro Viajante

Neste capítulo, é apresentada a descrição do clássico Problema do Caixeiro Viajante (*TSP - Traveling Salesman Problem*) através da formulação de programação linear e algumas heurísticas para a sua solução aproximada.

O *TSP* pode ser descrito do seguinte modo: dado um conjunto de n cidades, com distâncias conhecidas entre cada par de cidades, um caixeiro deve partir de uma cidade origem, visitar todas as outras cidades de seu território apenas uma única vez e então, retornar à cidade de origem, de tal forma que a distância percorrida seja mínima.

Matematicamente, o problema pode ser descrito da seguinte forma: seja $G = (N, M)$, um grafo completo simétrico, com $|N| = n$, onde N é o conjunto de vértices que representam as cidades, M é o conjunto de arestas (i, j) que representam o percurso entre duas cidades i e j e $c_{ij} \geq 0$ é o custo associado à aresta (i, j) . O Problema do Caixeiro Viajante consiste em determinar o Ciclo Hamiltoniano¹ de Custo Mínimo. A figura 2.1a apresenta um grafo representando cidades. Uma solução para o *TSP* (um ciclo hamiltoniano) neste grafo é mostrada na figura 2.1b.

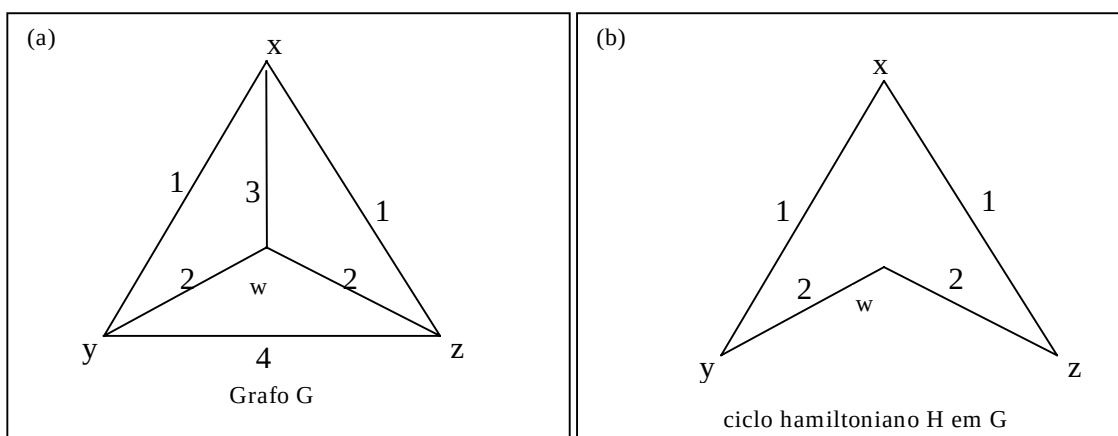


Figura 2.1: Exemplo de uma solução do *TSP*.

¹ Ver [BONDY, 76] para mais detalhes sobre ciclos hamiltonianos.

2.1 Formulações Matemáticas para o *TSP*

Em 1954, Dantzig, Fulkerson e Johnson propuseram a formulação matemática para o *TSP* [DANTZIG, 54] mostrada abaixo:

$$x_{ij} = \begin{cases} 1, & \text{se o caixeiro viaja da cidade } i \text{ para a cidade } j. \\ 0, & \text{caso contrário.} \end{cases}$$

$c_{ij} \geq 0$, é custo de percorrer diretamente o arco (i,j) , onde $c_{ii} = \infty$ para todo $i \in N$. Neste texto supõe-se sempre custos não negativos.

O modelo de programação matemática para o problema pode ser assim formulado:

$$(TSP) \quad \text{Minimizar} \quad z = \sum_{i=1}^n \sum_{j=1}^n c_{ij} x_{ij} \quad (2.1)$$

$$\text{sujeito à : } \sum_{i=1}^n x_{ij} = 1, \quad \forall j \in N \quad (2.2)$$

$$\sum_{j=1}^n x_{ij} = 1, \quad \forall i \in N \quad (2.3)$$

$$\sum_{i \in S} \sum_{j \in N/S} x_{ij} \geq 1, \quad \forall S \subset N, S \neq \emptyset \quad (2.4)$$

$$x_{ij} \in \{0,1\}, \quad \forall (i,j) \in M \quad (2.5)$$

As restrições (2.2) garantem que para cada cidade j chega somente um arco, enquanto que as (2.3) garantem que de cada cidade i sai somente um arco. As restrições (2.4) garantem que deve existir pelo menos um arco ligando o conjunto S ao seu complementar N/S , evitando assim a formação de subciclos.

Uma outra formulação do *TSP* pode ser obtida, se uma estrutura especial de grafo é considerada. Na formulação a seguir o *TSP* é visto como um problema que se assemelha ao problema de caminho mínimo.

Para explicar esta formulação, é necessário definir o conceito de grafo multiestágio. Um grafo multiestágio $G=(V,E)$ é um grafo direcionado no qual os vértices são particionados em $k \geq 2$ conjuntos disjuntos $V_i, 1 \leq i < k$. Cada conjunto de vértices V_i representa um estágio. Os estágios extremos V_1 e V_k possuem apenas um vértice cada, s e d , respectivamente. s é denominado origem e d destino. Neste grafo, se o arco $(u,v) \in E$, então, $u \in V_i$ e $v \in V_{i+1}$. Em outras palavras, de um vértice só saem arcos para vértices do estágio sucessor ao seu. [Horowitz, 96]

Constrói-se, a partir de uma dada instância do TSP, um grafo $G'(N',A')$ multiestágio, da seguinte maneira: o grafo conterá $n+1$ estágios intermediários, cada um deles contendo n vértices (as n cidades do TSP). Os vértices extremos, s (origem) e d (destino) são vértices artificiais. O vértice s possui um arco para cada um dos n vértices do segundo estágio. Da mesma forma, cada um dos vértices do estágio intermediário $n+1$ têm um arco para o vértice d . Cada vértice de um estágio intermediário terá um arco para cada um dos vértices do estágio seguinte, excetuando-se o vértice que corresponde à mesma cidade que ele. No total, sairão $n-1$ arcos de cada vértice intermediário, excetuando-se os vértices do estágio $n+1$. Note que neste grafo não existem arcos conectando dois nós que representem a mesma cidade.

Os custos associados aos arcos que saem de s são todos zero, assim como os custos associados aos arcos que chegam a d . Assim, no modelo eles não serão considerados. Como cada arco que conecta vértices intermediários representa a conexão entre duas cidades da instância do TSP sendo tratada, o custo de um arco que conecta dois destes vértices é o próprio custo definido pela instância do problema.

A figura 2.2, ilustra o grafo multiestágio que seria construído a partir de uma instância do TSP com n cidades.

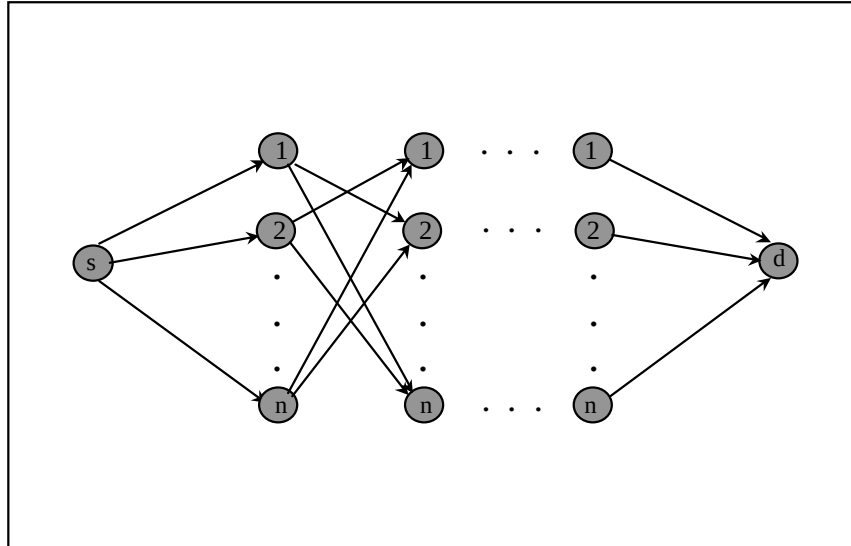


Figura 2.2: Grafo Multiestágio construído a partir de uma instância do TSP.

Adicionalmente, chamaremos de inter-estágio as arestas que estão entre dois estágios consecutivos. Agora podemos definir o significado das variáveis y_{ijt} :

$$y_{ijt} = \begin{cases} 1, & \text{se se viaja da cidade } i \text{ para a cidade } j \text{ no inter_estágio } t. \\ 0, & \text{em caso contrario} \end{cases}$$

Onde o grafo sobre o qual se faz o modelamento, é o seguinte:

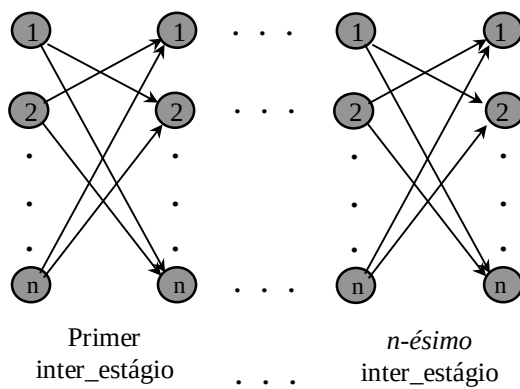


Figura 2.3: Grafo Multiestágio a partir do qual é feito o modelo para o TSP. Note que as cidades s e d tem sido retiradas. Note, também, que este grafo possui n inter-estágios e $n+1$ estágios.

Nesta formulação a variável de decisão é redefinida da seguinte forma:

$$x_{ij} = \sum_{t=1}^{n-1} y_{ijt} , \quad \forall i, j \in N \quad (2.6)$$

$$(TSP) \quad \text{Minimizar} \quad y = \sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n \sum_{t=1}^{n-1} c_{ij} * y_{ijt} \quad (2.7)$$

sujeito à:

$$\sum_{t=1}^n \sum_{\substack{i=1 \\ i \neq j}}^n y_{ijt} = 1, \quad \forall j \in N \quad (2.8)$$

$$\sum_{t=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n y_{ijt} = 1, \quad \forall i \in N \quad (2.9)$$

$$\sum_{i=1}^n \sum_{\substack{j=1 \\ j \neq i}}^n y_{ijt} = 1, \quad t = 1, \dots, n \quad (2.10)$$

$$\sum_{\substack{i=1 \\ i \neq j}}^n (y_{ijt} - y_{ji(t+1)}) = 0 \quad \begin{array}{l} t = 1, \dots, n-1; \\ \forall j \in N \end{array} \quad (2.11)$$

$$y_{ijt} \in \{0,1\}, \quad \begin{array}{l} t = 1, \dots, n; \\ \forall i, j \in N \end{array} \quad (2.12)$$

As restrições (2.8) são equivalentes às restrições (2.2), enquanto que, as restrições (2.9) são equivalentes às restrições (2.3). As restrições (2.11) juntamente com as restrições (2.10) são as restrições de conectividade. Observe que o total de restrições é de $n^2 + 2n$, além das restrições (2.12) que são as restrições de integralidade.

A formulação anterior, foi desenvolvida por nós durante este trabalho; porém, além destas formulações matemáticas para o *TSP*, existem outras. Algumas podem ser encontradas em [LAGEVIN, 90].

2.2 Heurísticas para o *TSP*

O *TSP* é um problema NP-Árduo [KARP 72], [GARY & JOHNSON 79]. Este fato justifica o uso de métodos aproximativos para resolvê-lo. As heurísticas são as ferramentas mais utilizadas, principalmente como ponto de partida para a elaboração de soluções para problemas complexos. Existem diversas heurísticas para a solução do *TSP*. Nas próximas seções serão abordadas aquelas que, de alguma forma estão relacionadas com este trabalho.

2.2.1 Heurística do Vizinho Mais Próximo

Esta heurística consiste em escolher um vértice inicial (aleatoriamente) e a partir daí, selecionar sempre a cidade (vértice) mais próxima daquela escolhida mais recentemente, eliminando-a do conjunto de possíveis viagens, até que não haja opções remanescentes. Neste ponto a viagem (grafo) está completa. A complexidade deste algoritmo é da ordem $O(n^2)$.

Na figura 2.4, mostra-se um exemplo deste método. A figura 2.4a apresenta um grafo para o qual deseja-se resolver o *TSP*. As figuras 2.4b a 2.4f mostram a seqüência de arestas escolhidas de acordo com a Heurística do Vizinho Mais Próximo.

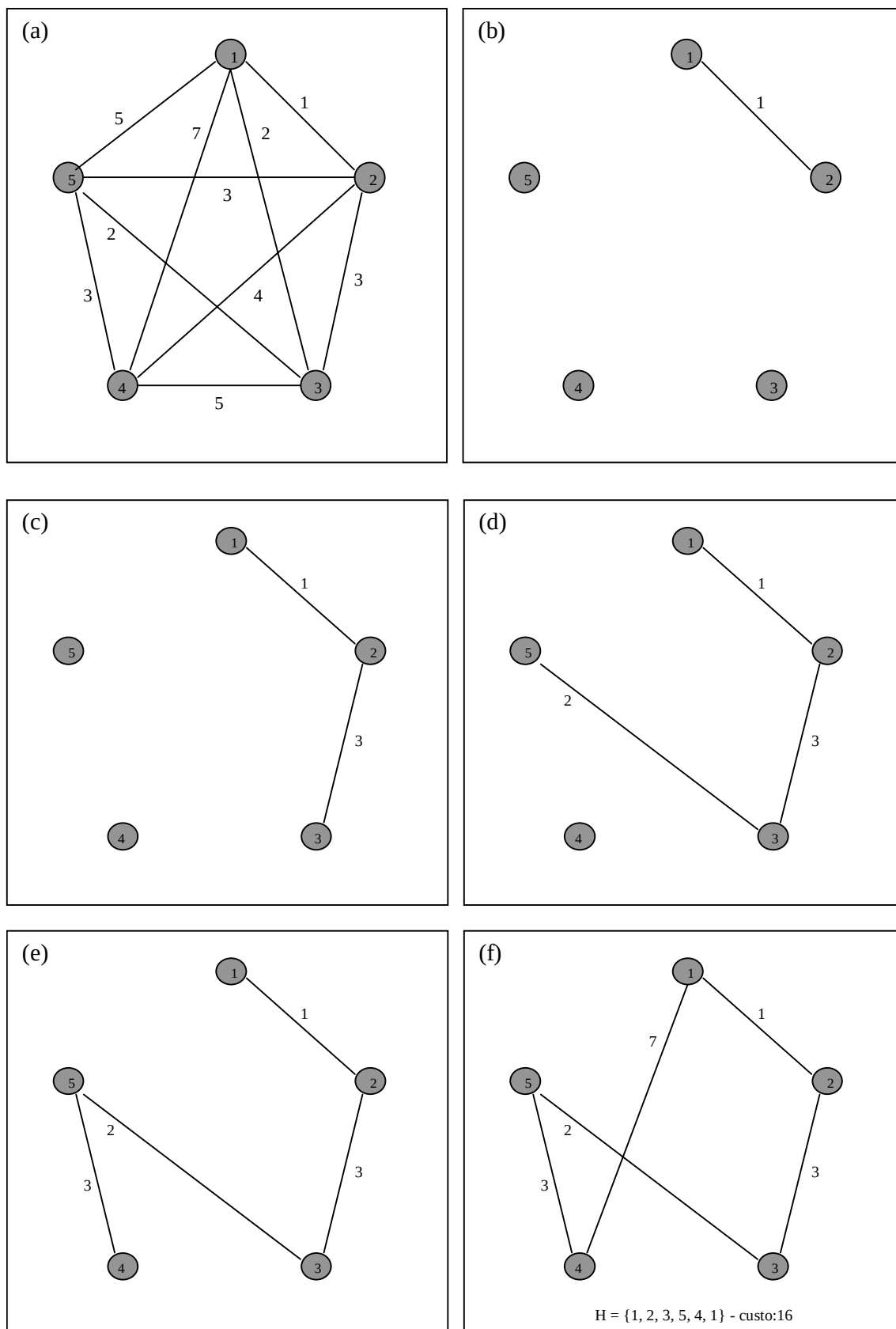


Figura 2.4: Solução do *TSP* utilizando-se a Heurística do Vizinho Mais Próximo.

2.2.2 Heurística de Christófides

A heurística de Christófides [CHRISTÓFIDES, 76] parte do princípio de que existe uma relação fundamental entre uma árvore geradora de custo mínimo e uma solução do *TSP*. Acredita-se que esta relação possa possibilitar a elaboração de heurísticas com bom desempenho.

A transformação proposta por essa heurística, passa pela obtenção de um ciclo Euleriano² a partir da árvore geradora mínima antes de chegar ao ciclo Hamiltoniano. Um ciclo Euleriano é aquele no qual todas as arestas são visitadas uma única vez durante uma rota.

Esta heurística utiliza o conceito de Emparelhamento Mínimo³. Este emparelhamento pode ser encontrado por um algoritmo de ordem $O(n^3)$.

Seja o grafo $G = (N, A)$, com $|N| = n$. Para resolver o *TSP* representado por G , a heurística de Christófides consiste dos seguintes passos:

- a) construção da árvore geradora mínima T de G [da ordem de $O(n^2)$].
- b) construção do Emparelhamento Mínimo M em $G_0 = (V_0, M_0)$, onde G_0 é o grafo completo induzido pelos vértices de grau ímpar de T e $M \subset M_0$ [da ordem de $O(n^3)$].
- c) obtenção do Ciclo Euleriano no grafo $G' = (N, T \cup M)$, aplicando-se em seguida operações de “atalho” para obter o ciclo hamiltoniano H [da ordem de $O(n)$].

Para exemplificar, considere agora o grafo G da figura 2.5a. Seja T sua árvore geradora mínima (figura 2.5b). Como em um grafo Euleriano todo vértice tem grau par, nossa preocupação deve estar voltada para os vértices 1, 3, 5 e 6 de T que possuem grau ímpar. O número de vértices de grau ímpar em T será sempre par, visto que a soma dos graus de todos os vértices deve ser par (cada aresta conecta dois vértices). Uma maneira de se obter um grafo Euleriano é acrescentar a T , as arestas do Emparelhamento Mínimo M , calculado sobre o grafo completo $G_0(V_0, M)$ induzido pelos vértices de grau ímpar de T , isto é, $V_0 = \{1, 3, 5, 6\}$. Digamos, então, que $M = \{(1,5), (3,6)\}$ seja o Emparelhamento Mínimo (figura 2.5c), temos então:

² Vide [BONDY, 76] para detalhes sobre ciclos eulerianos.

³ Para maior informação sobre o que são os emparelhamentos e os emparelhamentos mínimos, vide [BONDY, 76]

$$G_0 = (V_0, M), \quad V_0 = \{1, 3, 5, 6\}, \quad M = \{(1,5), (3,6)\}$$

Não é difícil deduzir que $G' = (N, T \cup M)$, é o grafo Euleriano de menor comprimento dentre os demais que contém a árvore geradora T (figura 2.3e). De G' constrói-se um ciclo Euleriano $E = \{1, 5, 4, 6, 3, 6, 2, 1\}$ e, em seguida, aplica-se uma operação de “atalho” (substituição de $(3,6)$ e $(6,2)$ por $(3,2)$), obtendo-se a rota $H = \{1, 5, 4, 6, 3, 2, 1\}$ (figura 2.5e).

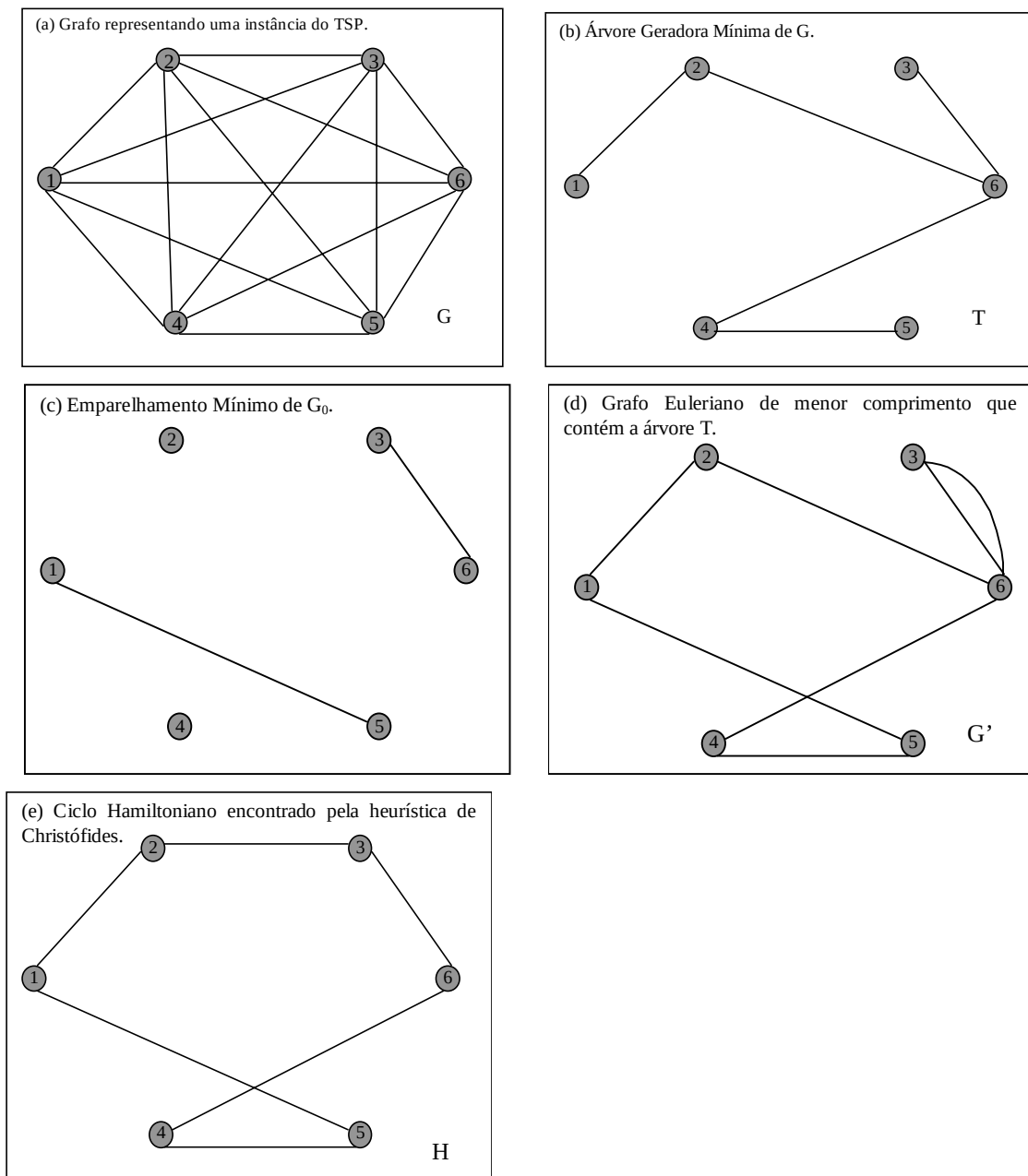


Figura 2.5: Solução do TSP, utilizando-se a heurística de Christófidis.

2.2.3 Método das K-Substituições ou *K-Optimal*

Quando já temos uma solução viável, o refinamento desta solução pode ser efetuado através de técnicas conhecidas como *k-optimal* [LIN, 73].

Este algoritmo tenta fazer recombinações entre k arestas não adjacentes de uma solução inicial, sendo o melhor ciclo Hamiltoniano obtido ao final do processo denominado *k-optimal*.

Em linhas gerais, o método segue os seguintes passos:

- a) Partindo-se de uma solução inicial H , remover k arestas não adjacentes da solução H , obtendo uma solução inviável H' .
- b) construir todas as soluções viáveis contendo H' .
- c) escolher a melhor dentre as soluções obtidas no passo (b) e continuar o processo até satisfazer um critério de parada pré-estabelecido.

Suponhamos que o ciclo inicial consista do seguinte conjunto de n arestas (não direcionadas) $H = \{x_1, x_2, \dots, x_n\}$ na ordem $x_1, x_2, \dots, x_n$. Para $k = 2$, seja $X = \{x_i, x_j\}$ um conjunto de k arestas não adjacentes em H , trocaremos X por um conjunto de k arestas $Y = \{y_p, y_q\}$, $y_i \notin Y$, de modo a formar uma solução viável do *TSP*. Se isto implicar em diminuição do custo total, $H' = (H - X) \cup Y$ será encarada como uma rota mais qualificada. As trocas das arestas devem obedecer às seguintes regras, no caso particular onde $k = 2$:

- a) Duas arestas x_i e x_j não podem ser adjacentes pois, neste caso, não é possível reconectar o ciclo de outro modo.
- b) Uma vez que o conjunto X tenha sido escolhido, o conjunto Y é determinado. Há $n(n-3)/2$ possíveis rotas H' para um determinado H original. Para cada uma dessas rotas vamos denotar o Ganho como:

Ganho = Custo total (H) - Custo total (H') = $W(x_i) + W(x_j) - W(y_p) - W(y_q)$, onde $W(a)$ é o peso da aresta a .

A cada passo, examinaremos uma das $n(n-3)/2$ rotas H' , se o ganho obtido for maior do que zero matemos esta solução parcial. Em caso contrário, a descartamos e utilizamos a solução parcial obtida no passo anterior. Continuamos com o processo até que o Ganho seja 0 (nulo), ou seja, até que não seja mais possível melhorar a solução. Então, teremos encontrado a solução *2-Optimal*.

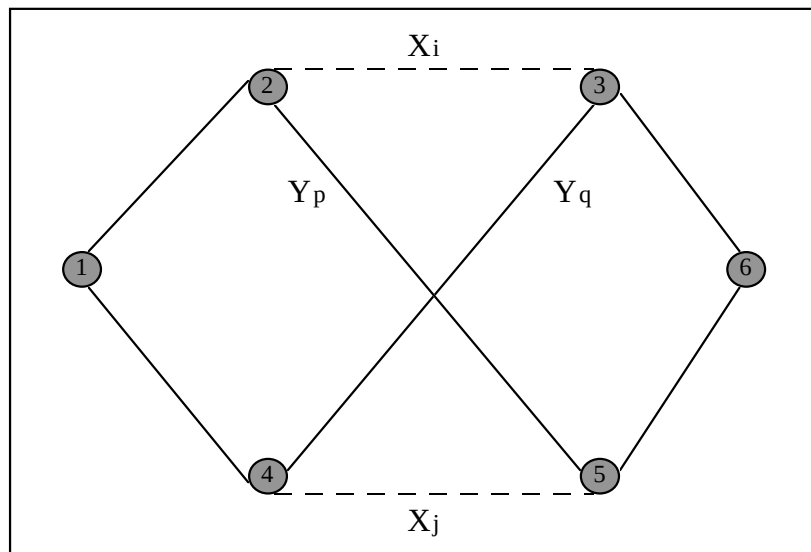


Figura 2.6: Método *2-Optimal*.

2.2.4 A Heurística GENIUS

Outra heurística para o *TSP*, conhecida na literatura como “GENIUS”, foi proposta por Gendreau [GENDREAU, 92]. O método inclui conceitos das heurísticas *k-optimal* e de inserção⁴ e é visto atualmente como uma das melhores heurísticas convencionais para o *TSP*. Esta heurística consiste de uma fase de construção do *tour* (GENI), seguida por uma fase de otimização (US). A heurística pode ser brevemente descrita da seguinte forma: Começando com três vértices arbitrários, GENI gradualmente constrói um *tour* sobre todos os vértices mediante a inserção de um vértice ainda não incluído, a cada passo. A inserção de um vértice só acontece entre dois de seus p vizinhos mais próximos sobre o *tour* atual; cada re-inserção é acompanhada por uma re-otimização local do *tour*. Seja $N_h(v)$ o conjunto de h vértices sobre o *tour* parcialmente construído, mais próximos a v . Se o *tour* tem menos do que h vértices, então todos seus vértices estão sob $N_h(v)$. A partir de agora, para uma dada orientação do *tour*, v_{i+1} denotará o sucessor de v_i sobre o *tour* e v_{i-1} denotará seu predecessor e P_{ik} denotará o conjunto de vértices sobre o caminho que vai de v_i até v_k . Existem dois tipos de inserção de vértices GENI, considerados para um vértice v , os quais estão ilustrados na figura 2.7 e serão descritos a seguir.

Tipo 1:

Escolher $v_r, v_s \in N_h(v)$ e $v_p \in N_h(v_{r+1}) \cap P_{sr} \setminus \{v_r, v_s\}$. Eliminar (v_r, v_{r+1}) , (v_s, v_{s+1}) e (v_p, v_{p+1}) ; inserir (v_r, v) , (v_{r+1}, v_p) e (v_{s+1}, v_{p+1}) .

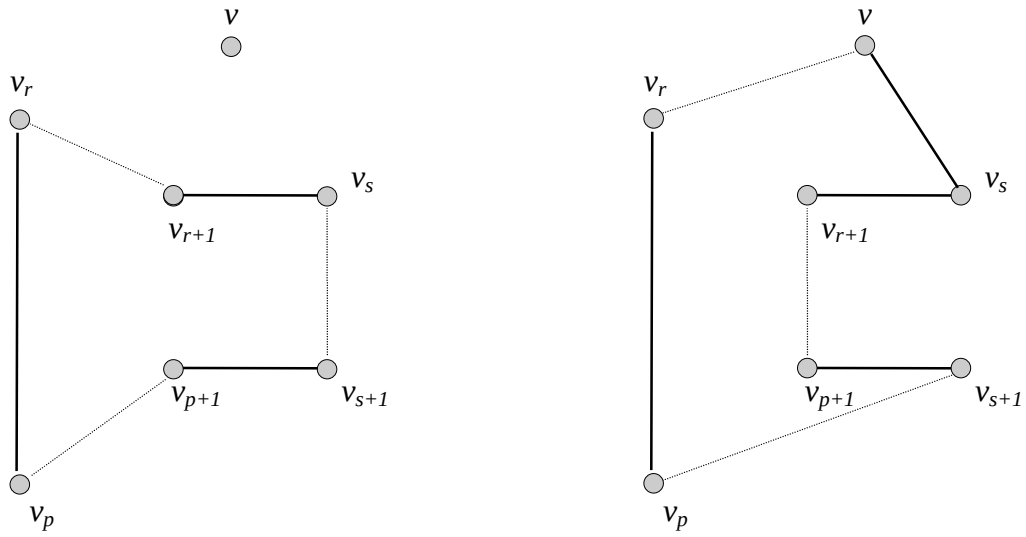
Tipo 2:

Escolher $v_r, v_s \in N_h(v)$, $v_p \in N_h(v_{r+1}) \cap P_{sr} \setminus \{v_s, v_{s+1}\}$ e $v_q \in N_h(v_{s+1}) \cap P_{rs} \setminus \{v_r, v_{r+1}\}$. Eliminar (v_s, v_{r+1}) , (v_{q-1}, v_q) , (v_s, v_{s+1}) e (v_{p-1}, v_p) ; inserir (v_r, v) , (v, v_s) , (v_q, v_{s+1}) , (v_{p-1}, v_{q-1}) e (v_{r+1}, v_p) .

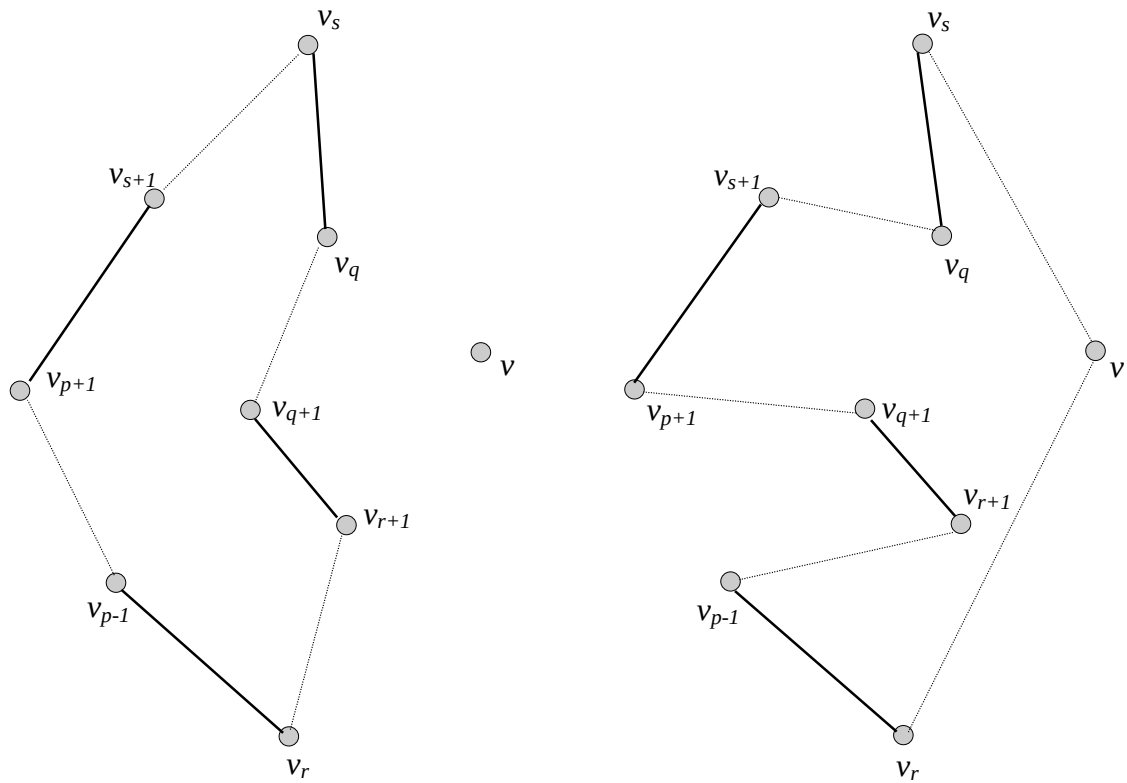
Para um dado v , a melhor inserção é obtida considerando-se arestas e trocas viáveis de v_r, v_s, v_p e v_q , os dois tipos de inserção e os dois tipos de orientação do *tour*. Inserir os n vértices do *tour* usando o GENI requer $O(nh^4 + n^2)$ operações. Na fase US de otimização, cada nó (vértice) é, por sua vez, removido do *tour* através da operação reversa do GENI

⁴ Vide [CAMPELLO, 94] para uma descrição da heurística de inserção.

e o nó é logo re-inserido no *tour* usando GENI. O procedimento acaba quando não é mais possível fazer melhoras mediante a remoção e re-inserção de qualquer nó.



(a). Inserção de tipo 1.



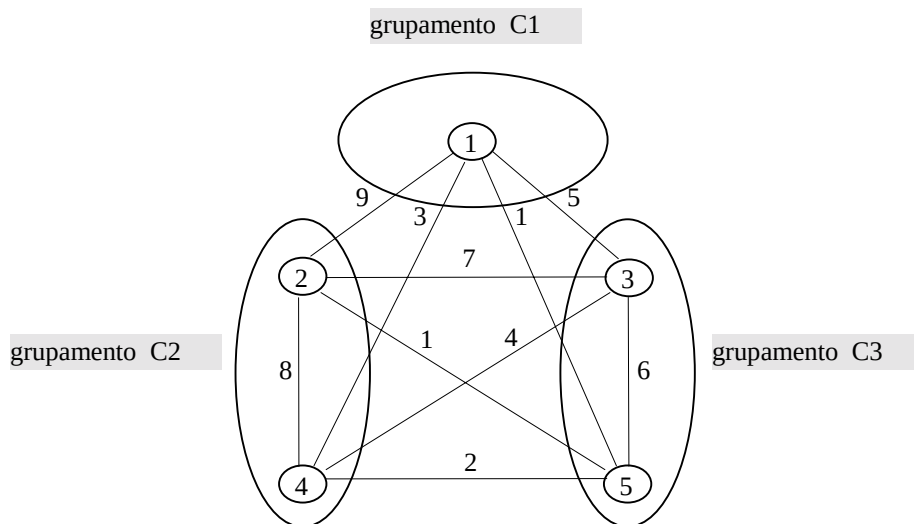
(b). Inserção de tipo 2.

Figura 2.7: Os dois tipos de inserção GENI.

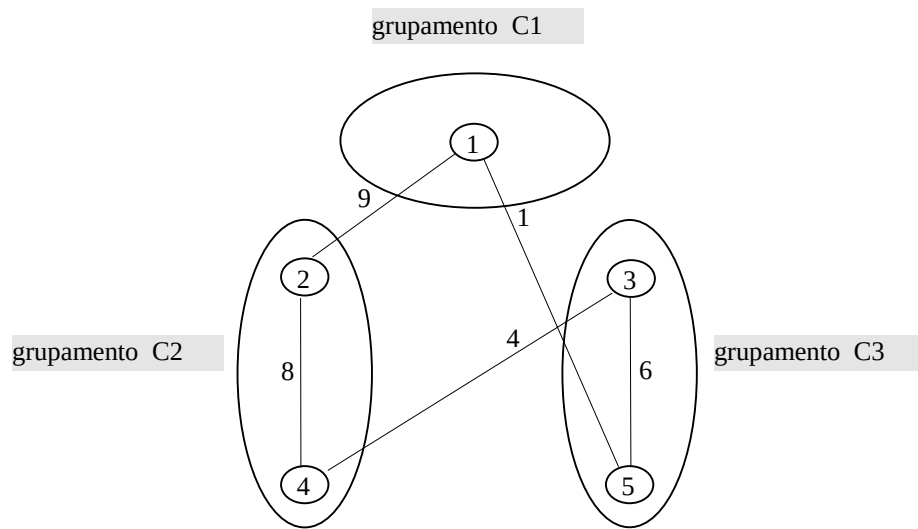
Capítulo 3

O Problema do Caixeiro Viajante com Grupamentos

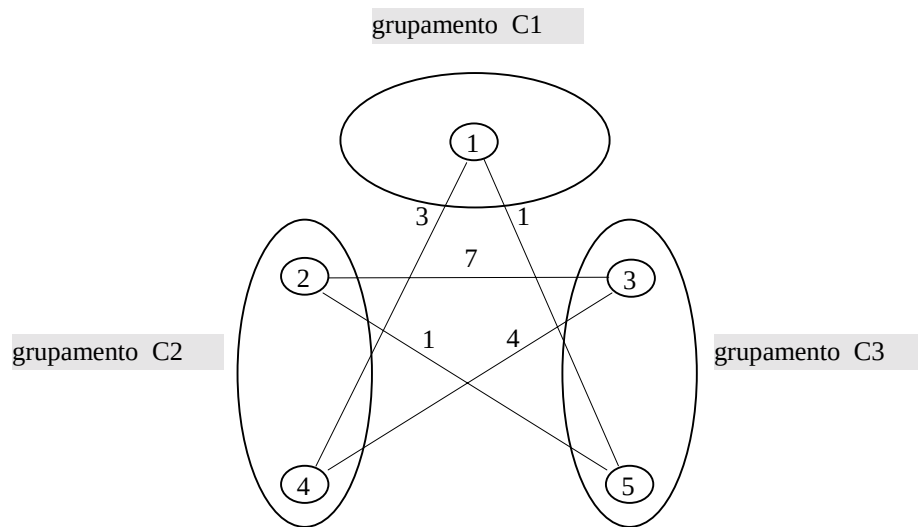
Uma variação do Problema do Caixeiro Viajante (*TSP*), conhecida na literatura como Problema do Caixeiro Viajante com Grupamentos (*CTSP* - *Clustered Traveling Salesman Problem*), foi proposta por Chisman [CHISMAN, 75]. Nesta variação o conjunto de n cidades do *TSP* é particionado em k subconjuntos denominados grupamentos, e o problema agora consiste em encontrar o menor caminho que visite todas as cidades de forma tal que as cidades pertencentes a um mesmo grupamento sejam visitadas contiguamente.



(a). Instância do *CTSP* com $n=5$ cidades e três grupamentos C1, C2, C3. ($k=3$)



(b). Uma Solução viável para a instância em (a). Note que as cidades dentro de cada grupamento são visitadas contiguamente dentro de cada grupamento. Neste caso o custo total é igual a 28.



(c). O caminho hamiltoniano apresentado nesta figura constitui uma solução viável para o *TSP*; mas ela não é uma solução viável para o *CTSP*, pois as cidades dentro de cada grupamento não são visitadas contiguamente. Observe, que uma solução ótima para o *TSP* sempre terá um valor menor ou igual que uma solução ótima para o *CTSP*, devido à restrição imposta pelos grupamentos.

Figura 3.1: Exemplo de uma instância para o *CTSP*.

3.1 Formulação Matemática do CTSP

Formalmente, Chisman [CHISMAN, 75], define o CTSP da seguinte forma: Dado um grafo completo e sem arestas múltiplas $G=(N, M)$, onde $N = \{1, \dots, n\}$ é o conjunto nós e $M = \{(i,j) / i,j \in N, i \neq j\}$ é o conjunto de arestas. Em adição também, são dadas partições $C_1, \dots, C_k$, $1 < k < n$, do conjunto de nós, onde cada subconjunto C_h ($h=1, \dots, k$) é chamado de grupamento.

Seja, $c_{ij} \geq 0$ o custo associado à aresta $(i,j) \in M$. Um ciclo no CTSP é dito viável se ele é não possui arestas múltiplas e passa por cada grupamento, visitando contiguamente todas as cidades dentro do grupamento, uma única vez. Então, o CTSP consiste em encontrar o ciclo viável T cujo custo global $\sum_{(i,j) \in T} c_{ij}$ seja mínimo.

Uma formulação matemática para o CTSP pode ser obtida definindo inicialmente:

$$x_{ij} = \begin{cases} 1, & \text{se o caixeiro viaja da cidade } i \text{ para a cidade } j. \\ 0, & \text{caso contrário.} \end{cases}$$

O modelo de programação matemática para o problema pode ser assim formulado:

$$(CTSP) \quad \text{Minimizar } Z = \sum_{i=1}^n \sum_{j=1}^n c_{ij} * x_{ij} \quad (3.1)$$

sujeito à :

$$\sum_{i=1}^n x_{ij} = 1, \quad \forall j \in N \quad (3.2)$$

$$\sum_{j=1}^n x_{ij} = 1, \quad \forall i \in N \quad (3.3)$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \quad \forall S \subset N, |S| \geq 1 \quad (3.4)$$

$$\sum_{i \in C_h} \sum_{j \in C_h} x_{ij} = |C_h| - 1, \quad \forall C_h \subset N, |C_h| \geq 1 \quad (3.5)$$

onde $h = 1, 2, \dots, k$ e C_h é um grupamento.

$$x_{ij} \in \{0, 1\}, \quad \forall (i,j) \in M \quad (3.6)$$

As restrições (3.2) garantem que para cada cidade j chega somente um arco, enquanto em (3.3) garantem que de cada cidade i sai somente um arco. As restrições (3.4) indicam que os sub-ciclos não são permitidos; note que esta é uma forma alternativa de expressar as restrições de conexidade (2.4). Por outro lado, também é necessário que exista um caminho hamiltoniano que visite todos os nós de cada grupamento; e, dado que um caminho hamiltoniano visita η vértices com somente $(\eta-1)$ arestas, as restrições (3.5) dizem que um caminho hamiltoniano de longitude $|C_k|-1$ deve ir através dos $|C_k|$ nós do grupamento k . A longitude de um caminho é determinada pelo número de arestas que compõem o caminho.

Como no *TSP*, no *CTSP* pode-se ter que $c_{ij} = c_{ji}$. Neste caso o *CTSP* se denomina *CTSP* simétrico. Quando $c_{ij} \neq c_{ji}$ o *CTSP* se denomina *CTSP* assimétrico.

Também no *CTSP*, a matriz de custos $C=[c_{ij}]$, associada ao problema pode ser particionada em submatrizes para identificar os grupamentos, como mostrado a seguir:

$$C = \begin{bmatrix} G_{11} & L_{12} & \cdots & L_{1K} \\ L_{12} & G_{22} & & L_{2K} \\ \vdots & & \ddots & \vdots \\ L_{K1} & L_{K2} & \cdots & G_{KK} \end{bmatrix}$$

Figura 3.2: Particionamento da matriz de custos.

Cada submatriz G_{ii} , na figura 3.2, contém os custos associados às arestas que conectam nós dentro do mesmo grupamento i ; e cada submatriz L_{ij} contém os custos associados às arestas que conectam os nós do grupamento i aos nós do grupamento j .

Adicionalmente, uma aresta será chamada de **aresta local** se ela pertence a G_{ii} , $1 \leq i \leq k$ ou de **aresta não-local**, caso contrário.

Em seu artigo, Chisman mostrou que o *CTSP* pode ser transformado no *TSP* mediante a adição de uma constante muito grande Γ ($\Gamma \gg \max\{c_{ij}\}$) em cada aresta que conecta dois grupamentos ou mediante a subtração de Γ , de cada aresta que conecta dois vértices dentro do mesmo grupamento [CHISMAN, 75]. Após esta transformação, qualquer

algoritmo para solucionar o *TSP* pode ser utilizado sobre o *CTSP*. Segundo Chisman, a razão para que esta transformação funcione é que, devido ao incremento no valor das arestas entre grupamentos, passa a ser mais atrativo escolher arestas que unem vértices que estão dentro do mesmo grupamento, do que escolher arestas que unem vértices que estão em grupamentos distintos; logo uma solução ótima para o *TSP*, conterá exatamente k arestas conectando grupamentos distintos, ou seja, o mínimo possível destas arestas. Porém, esta transformação não preserva a taxa de erro [GENDREAU, 97]; por exemplo, um algoritmo 2-aproximado para o *TSP* não é 2-aproximado para o *CTSP*.

3.2 Variações do *CTSP*

O *CTSP*, tem sofrido algumas variações; por exemplo [ANILY, 98], [GENDREAU, 96], [GENDREAU, 97], [LOKIN, 78] estudam o caso no qual existem relações de precedência entre os grupamentos e, [ARKIN, 97] o caso no qual é definido um nó inicial para cada grupamento. Em geral, no *CTSP*, se espera que a ordem de visitas entre os grupamentos não seja predefinida, porém, quando esta ordem é predefinida, o problema se reduz a encontrar a rota ótima dentro dos grupamentos e obter em cada grupamento a melhor entrada e saída, segundo a ordem de visita entre eles. Para o caso geral, por outro lado, se faz necessário encontrar simultaneamente a ordem de visita dentro de cada grupamento e a ordem de visita entre os grupamentos, pois estes problemas não são independentes.

O caso no qual a ordem de visitas entre os grupamentos não é predefinida é chamado de Problema do Caixeiro Viajante com Grupamentos Irrestrito (*FCTSP - Free Clustered Traveling Salesman Problem*) [CHISMAN, 75], [JONGENS, 85]. Quando a ordem de visitas entre os grupamentos é predefinida, é denominado Problema do Caixeiro Viajante com Grupamentos Restrito (*RCTP - Restricted Clustered Traveling Salesman Problem*) [GENDREAU, 96] ou Problema do Caixeiro Viajante com Grupamentos Ordenado (*OCTSP - Ordered Clustered Traveling Salesman Problem*) [ANILY, 98]; e existem vários trabalhos na literatura a respeito desta variação de *CTSP* ([ANILY, 98], [ARKIN, 97], [GENDREAU, 97], entre outros.)

Uma cidade isolada, isto é, que não está num grupamento de duas ou mais cidades pode ser vista como caso especial, limite, de um grupamento com somente uma cidade (ver figura 3.1). Um caso especial, do *CTSP*, bastante estudado pela sua aplicação ao problema de roteamento de veículos [GENDREAU, 97], é quando o número de grupamentos $k=2$. Denominado *Traveling Salesman Problem with Backhauls*, o problema consiste de dois grupamentos L e B (*linehaul customers* e *backhaul customers*, respectivamente) e possivelmente um nó, v_0 , adicional (o depósito); e o objetivo é determinar o ciclo hamiltoniano de custo mínimo que começando em v_0 , visite contiguamente os nós de L e depois os nós de B.

Aplicações conhecidas do *CTSP* são o *automated warehouse routing* [CHISMAN, 75], [Lokin, 78] e o planejamento da produção [LOKIN, 78]. Outras aplicações são aquelas que envolvem sistemas de serviços onde os clientes tem serviços de prioridades diferentes [GENDREAU, 96].

Existe na literatura um problema ainda mais geral, do qual o *CTSP* apresenta-se como um caso particular. Este problema denominado Problema do Caixeiro Viajante Generalizado (*GTSP - Generalized Traveling Salesman Problem*) é também uma variante do *TSP*, no qual, o conjunto de vértices ou nós são particionados em grupamentos e o caixeiro deve visitar ao menos um nó de cada grupamento [NOON, 91].

3.3 Algoritmos para solucionar o *CTSP* e suas variações

Como o *CTSP* é uma generalização do *TSP* (O *TSP* é o *CTSP* com $k=1$) o problema é NP-Árduo; o que limita o uso exclusivo de técnicas exatas. Desta forma, a maioria dos trabalhos encontrados na literatura abordam procedimentos aproximados ou heurísticas para a sua solução.

Em 1963, um algoritmo aproximado eficaz para resolver o *TSP* foi desenvolvido por [LITTLE, 63]; e, em 1970, Held e Karp [HELD, 70] [HELD, 71] introduziram um método de relaxação lagrangeana para o *TSP*. Lokin [LOKIN, 78] reporta que o algoritmo de Little

[LITTLE, 63] em combinação com a transformação de Chisman é bastante ineficiente quando aplicado ao *CTSP*. A razão para isto, segundo Lokin, é, que os algoritmos de ramificação e cotação (*branch and bound*) dependem muito da estrutura da matriz de custos associada ao problema.

Mais tarde, Jongens e Volgenant [JONGENS, 85] fizeram uma adaptação do algoritmo baseados na forma lagrangeana de Volgenant e Jonker [JONKER, 84], [VOLGENANT, 82], [VOLGENANT, 83]. Esta adaptação resolve de forma exata o *FCTSP* simétrico, porém, não é possível resolver com ela, o *CTSP* assimétrico ou com relações de precedência.

Para o *OCTSP* simétrico, em 1996 Gendreau et al. [GENDREAU, 96], apresentaram um algoritmo 2-aproximado; mais tarde, Shoshana Anily [ANILY, 97], apresentou um algoritmo 5/3-aproximado com uma complexidade da ordem de $O(n^3)$; este algoritmo é uma adaptação da heurística de Christófidis. Arkin [ARKIN, 97], provê um algoritmo 3.5-aproximado e, mais recentemente, Guttman-Beck, *et al.* [GUTTMAN-BECK, 97] desenvolveram um algoritmo 2.75-aproximado. Para o caso especial onde $k=2$, Gendreau, Hertz e Laporte [GENDREAU, 97] desenvolveram um algoritmo 3/2-aproximado.

No trabalho de Gendreau, Laporte e Potvin [GENDREAU, 96] foram apresentadas três heurísticas para o *CTSP*, duas delas para o *RCTSP*. Dentre as três heurísticas, destacamos a heurística H2, que utiliza a transformação proposta por Chisman e a heurística GENIUS desenvolvida por Gendreau, Hertz e Laporte [GENDREAU, 92], para solucionar o *TSP* resultante. Em termos de metaheurísticas, Potvin e Guertin [POTVIN, 95] propuseram um algoritmo genético para o *RCTSP* (heurística H3), onde o processo de reprodução é feito em dois níveis: a nível dos grupamentos e a nível dos vértices. A nível de vértices, é utilizada a heurística genética *ERX* (*Genetic Edge Recombination*), operador desenvolvido por Whitley, Starkweather e Fuquay [WHITLEY, 89]. Para a geração de uma seqüência de visitas das cidades pertencentes a cada grupamento, utilizam-se operadores clássicos para a geração de cromossomos filhos, de forma a manter a ordem de visitas preestabelecida dos grupamentos.

Resultados computacionais, listados pelos autores, mostram uma superioridade da heurística H3 sobre as demais, em termos de qualidade de solução. Porém H3 é muito

mais lenta que a H2. Outro ponto importante a respeito de H3, é o fato de que esta obtém melhores resultados para casos especiais como aqueles em que há um grande número de grupamentos, com poucas cidades em cada um deles.

Num trabalho recente Rabelo, Ochi e Maculan [RABELO, 96], propuseram um algoritmo genético para o *CTSP*, onde para a etapa da reprodução aplica-se uma adaptação do operador *ERX*, a qual é utilizada tanto para a determinação das seqüências entre as cidades quanto para a determinação da seqüência entre os grupamentos. Adicionalmente, um processo de refinamento (*2-optimal*) é aplicado sobre os melhores cromossomos. Os resultados por eles apresentados mostraram-se melhores que os resultados da heurística H2, tanto no valor da solução obtida quanto no tempo computacional empregado.

Neste trabalho, nos consideramos o *CTSP* simétrico sem ordem de precedência nenhuma.

Capítulo 4

Agentes Inteligentes

Segundo Wooldridge e Jennings [WOOLDRIDGE, 94], uma maneira de definir Inteligência Artificial (IA) é como o sub-campo da ciência da computação que visa construir agentes que exibam aspectos de comportamento inteligente. Esta visão nos mostra como a noção de agente é central em IA. Entretanto, surpreendentemente, os pesquisadores deram pouca importância a este tema até o meio da década de 80. A partir daí, o interesse por agentes vem florescendo em várias áreas, tanto de pesquisa como comerciais. Podemos citar: Comunicação de Dados, Concorrência de Sistemas, Robótica e até mesmo Interação Homen-Máquina.

4.1 O que é um agente?

Segundo [Foner, 94], um agente é um sistema que tenta cumprir um conjunto de metas enquanto está situado num ambiente complexo e dinâmico.

Um agente pode ser composto por Hardware e estar situado em um mundo físico ou pode ser composto por software e rodar em um computador, neste caso, pode estar situado em um mundo simulado ou interagir como um parceiro de outras entidades, possivelmente outros agentes.

Um agente deve ser capaz de sentir o ambiente e executar ações para modificá-lo ou modificar sua própria posição neste ambiente, de modo tal que isso favoreça a ele mesmo ou contribua para as ações que são cruciais para a sua existência.

A conduta de um agente é determinada pela interação de três fatores, representados na figura 4.1: o ambiente no qual o agente opera, a tarefa para a qual o agente foi projetado e de como ele projetado.

As metas dos agentes podem ser de vários tipos. Dentre estes podemos citar: metas de homeostase (por exemplo, um robô não pode deixar sua bateria descarregar), recompensas a serem maximizadas, punições a serem minimizadas, e etc.

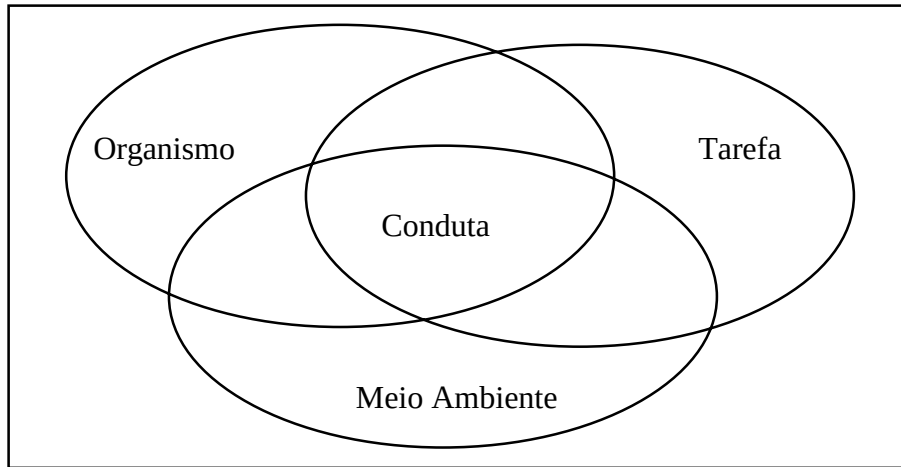


Figura 4.1: A interação entre a estrutura, a tarefa e o ambiente de um organismo influenciam sua conduta.

Pesquisadores têm oferecido uma variedade de definições sobre agentes. Duas das quais são:

"Os Agentes são sistemas computacionais que habitam algum sistema dinâmico complexo, sentindo e agindo autonomamente no seu ambiente, e tentando realizar um conjunto de metas ou tarefas para as quais foram criados."

The Maes Agent [MAES, 94A]

"Os Agentes Inteligentes continuamente realizam três ações: percebem as condições dinâmicas no ambiente; agem para afetar as condições no ambiente; e raciocinam para interpretar o que percebem, solucionar problemas, fazer inferências e determinar ações."

The Hayes-Roth Agent [HAYES-ROTH, 90]

Para Wooldridge e Jennings, alguns pesquisadores (particularmente os que trabalham na área de IA), geralmente entendem por "agente" um sistema de computador que é conceitualizado ou implementado usando conceitos que são normalmente aplicados a

humanos. Por exemplo, é bastante comum em IA caracterizar um agente usando noções *mentais*, tais como conhecimento, crença, intenção, e obrigação [SHOHAM, 93].

4.2 Propriedades dos Agentes

Vários autores identificam diversas propriedades de agentes [WOOLDRIDGE, 95B] [FONER, 94]:

- **Autonomia:** Possuir autonomia significa "ser relativamente independente". Os agentes operam sem a intervenção direta de seres humanos ou de algum outro "ser", e possuem de certo modo, algum controle sobre suas ações e estado interno [WOOLDRIDGE, 95B]; e em suma: agentes operam de forma independente] [FONER, 94]. Os autores também consideram a autonomia de agentes como se ela fosse subdividida nas seguintes propriedades:

1. **Orientação a metas ou Pro-atividade:** agentes aceitam requisições de alto nível indicando o que um humano quer e são responsáveis por decidir como e aonde satisfazer as requisições.
2. **Racionalidade:** assume-se que um agente agirá de modo a alcançar suas metas, não obedecendo cegamente a comandos e podendo pedir explicações, modificar requisições ou recusar-se a satisfazê-las quando não vão de encontro a suas metas.
3. **Flexibilidade:** as ações de um agente não são pré-programadas, ele pode escolher dinamicamente que ações executar e em que seqüência, em resposta ao estado do seu ambiente externo.
4. **Início automático:** ao contrário de programas padrão que são invocados pelo usuário, agentes podem sentir mudanças no seu ambiente e decidir quando agir.

- **Adaptatividade:** agentes podem evoluir com o tempo, presumivelmente através do aprendizado, adaptando-se a mudanças do seu ambiente. Podem se configurar automaticamente de acordo com a preferência de seus usuários (baseados em experiências anteriores).
- **Comunicatividade ou Habilidade social:** os agentes interagem com outros agentes (possivelmente humanos) através de algum tipo de linguagem para comunicação entre agentes; para obter informação ou para pedir ajuda para alcançar suas metas.
- **Reatividade:** os agentes percebem seu ambiente (o qual pode ser o mundo físico, um usuário numa interface gráfica, uma coleção de outros agentes, a *INTERNET*, ou talvez uma combinação de tudo isto), e respondem de forma rápida.
- **Mobilidade:** um agente pode movimentar-se em volta de uma rede eletrônica, transportando-se de uma máquina para outra, possivelmente por diferentes arquiteturas de sistemas e plataformas.
- **Caráter:** Um agente tem uma personalidade bem definida e com crenças, além de um estado emocional.
- **Veracidade:** assume-se que agentes não comunicarão informações falsas voluntariamente [GALLIERS, 88].
- **Benevolência:** assume-se que um agente não tem metas conflituosas, e que cada agente, portanto, sempre tentará fazer o que se espera dele [ROSENCEIN, 85].
- **Continuidade Temporal:** um agente é um processo que roda continuamente.

Uma discussão de várias diferentes propriedades de um agente, visando defini-los formalmente pode ser encontrada em [GOODWIN, 93].

4.3 Seleção de Ações e Aprendizado

Dadas as especificações anteriores de autonomia e adaptatividade, existem duas questões fundamentais relacionadas a agentes autônomos e adaptativos:

- Seleção de Ações (autonomia)
- Aprendizado através da experiência (adaptatividade)

Estas questões figuram entre os muitos problemas relacionados a agentes, ainda mal resolvidos.

Em Maes [MAES, 94B] encontramos interessantes reflexões sobre estes dois pontos chave.

4.3.1 Reflexões sobre a Seleção de Ações

- Como o mecanismo de seleção de ação funciona?
- Como um agente aprende? Que hipóteses ele pode criar e como ele decide quais são relevantes?
- Qual é a sua estratégia experimental? Como ele balanceia a exploração (realizar a tarefa da maneira que considera ótima de acordo com o seu conhecimento atual) e a exploração (realizar a tarefa de maneira sub-ótima na tentativa de descobrir uma estratégia nova e melhor)?

4.3.2 Reflexões sobre o Aprendizado através da experiência

- O aprendizado deveria ser incremental.
- O agente deve tender para o aprendizado do que é relevante para o alcance de suas metas.
- O aprendizado deve ser capaz de lidar com um mundo não-determinístico, onde coisas imprevisíveis podem ocorrer.
- O aprendizado não deve ser supervisionado, o agente deve aprender autonomamente.

- Deve ser possível fornecer um conhecimento inicial para o agente, para que ele não comece do nada, especialmente em situações onde o conhecimento prévio é facilmente acessível.

4.4 Selecionando o Foco de Aprendizado

Segundo [FONER, 94], para se projetar um agente e determinar suas características (sensores de entrada, estrutura interna, etc.) é necessário considerar, além do mundo com o qual interage e suas metas desejadas, a maneira como o agente vai aprender: seletivamente ou indiscriminadamente. Muitos métodos de aprendizado foram propostos para agentes autônomos, em particular, Aprendizado Mediante Reforços (que será descrito no capítulo seguinte) [SUTTON, 90] [KAELBLING, 93], Sistemas Classificadores [HOLLAND, 86] [WILSON, 85], Aprendizes de Modelo de Ação [DRESCHER, 91] [MAES, 92] e métodos híbridos [SUTTON, 90] [BOOKER, 88]. [FONER, 94] menciona a importância de se assegurar a correção, a relevância e a completude das informações aprendidas:

Correção: “As coisas aprendidas são realmente verdadeiras no mundo?”

Relevância: “O agente está aprendendo o suficiente?”. Para agentes que possuem metas, a relevância das informações aprendidas é questão fundamental.

Completude: “O que o agente aprende é útil para que ele cumpra suas metas? Este aprendizado evita que ele tente aprender coisas inúteis a estas metas?”

Muitos algoritmos tentam aprender tudo o que sabem sobre o ambiente, sem seletividade. Desta forma acabam aprendendo muitas informações irrelevantes e quanto mais se aprende menor fica a velocidade do algoritmo, até que ele se torna extremamente lento.

Em geral, criaturas reais usam mecanismos de foco de atenção, dentre outras seletividades perceptuais e cognitivas, para guiar o seu aprendizado. Focalizando aspectos importantes da sua experiência e memória atuais, elas diminuem dramaticamente o carregamento perceptual e cognitivo necessário para aprender sobre o

seu ambiente e tomar decisões sobre o que fazer [ALONIMONOS, 93]. Entretanto, existem muitas outras aplicações para agentes. Por exemplo, para resolver problemas de otimização, como o realizado neste trabalho, através da simulação do comportamento de uma colônia de formigas¹.

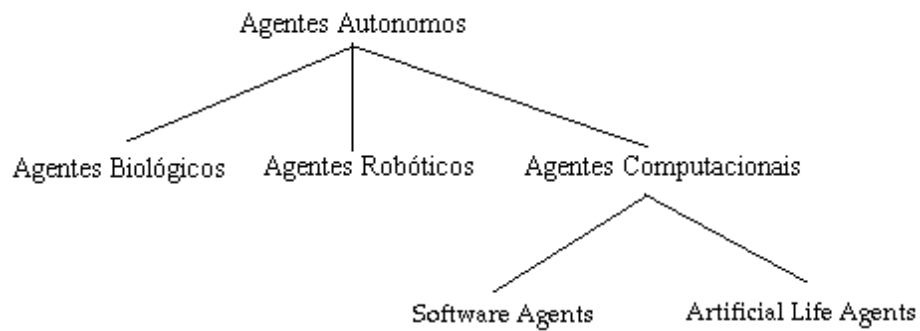


Figura 4.2: Existem muitas outras aplicações para agentes.

¹ Vide capítulo 6.

Capítulo 5

Uma Breve Introdução ao Aprendizado Mediante Reforços

A idéia de que nós aprendemos interagindo com nosso ambiente é, provavelmente, a primeira que nos ocorre quando pensamos na natureza da aprendizagem. Quando uma criança brinca, move seus braços, ou olha ao seu redor, ela mantém uma conexão sensor-motora direta com seu ambiente. Exercitar esta conexão produz uma riqueza de informações do tipo "causa e efeito" sobre conseqüências de ações e sobre o que fazer para alcançar metas. Esta interação é indubitavelmente um contribuinte importante para a criança que está desenvolvendo uma noção do seu ambiente e de seu próprio papel nele. A experiência permanece como mestre poderoso, conforme a criança cresce, vira um adolescente e depois um adulto, mesmo quando a natureza da interação muda significativamente com o passar do tempo. Se estamos aprendendo a dirigir um carro ou a manter uma conversa, estamos intensamente atentos a como nosso ambiente responde ao que fazemos e, desse modo, procuramos influenciar seu comportamento. Aprender por meio da interação é a idéia fundamental que está por trás de quase todas as teorias de aprendizagem. E foi a partir desta idéia que surgiu uma aproximação computacional para o aprendizado, o Aprendizado Mediante Reforços (*RL - Reinforcement Learning*). Ela tem raízes particularmente ricas na psicologia da aprendizagem animal, a partir da qual deriva-se seu nome.

O *RL* aparece desde os primeiros dias da cibernética e nos primeiros trabalhos em estatística, psicologia, neurociência e ciências da computação. Nos últimos cinco ou dez anos, tem gerado rapidamente um interesse crescente entre as comunidades das áreas de Inteligência Artificial e Aprendizado de Máquinas. Este interesse é, em parte, impulsionado pelo desafio de se projetar sistemas inteligentes que devem operar nos ambientes dinâmicos do mundo real. Por exemplo, fazer robôs ou “agentes” robóticos, mais autônomos, ou seja, menos confiantes em condições cuidadosamente controladas,

requer métodos para a tomada de decisões que sejam efetivos na presença de incerteza, e que possam satisfazer restrições de tempo. Sob estas condições, o aprendizado parece ser essencial para se alcançar um comportamento qualificado, e é sob estas condições que o Aprendizado Mediante Reforços pode ter vantagens significantes sobre outros tipos de Aprendizado.

Há muitos outros problemas que os computadores poderiam resolver se existisse o "software" apropriado. Sistemas de controle de vôos para aviões, sistemas de manufatura automatizados, entre outros, apresentam problemas de controle não-lineares difíceis. Muitos destes problemas são atualmente insolúveis, não porque os atuais computadores sejam lentos ou possuam pouca memória, mas, simplesmente, porque é muito difícil determinar o que o programa deve fazer. Se um computador pudesse aprender a resolver um problema por tentativa-e-erro, isto seria de grande valor prático.

O *RL* é um método baseado em "tentativa e erro" (variação e seleção - *busca*) e aprendizagem (associação - *memória*). Por isso, todos os algoritmos de Aprendizado Mediante Reforços requerem uma combinação sinérgica particular de busca e memória. A busca é requerida para achar ações boas, e a memória é requerida para lembrar quais ações trabalharam bem em quais situações no passado. A sinergia surge porque a busca provê a informação que deve ser lembrada e a memória faz com que a busca fique mais fácil e mais rápida. O Aprendizado Mediante Reforços envolve o sistemático *caching*¹ dos resultados da busca de forma que a busca futura possa ser mais eficiente, ou talvez, até mesmo eliminada.

Para resolver com êxito problemas que nenhuma disciplina poderia abordar individualmente, o Aprendizado Mediante Reforços combina duas disciplinas: a Programação Dinâmica [BELLMAN, 57] e o Aprendizado Supervisionado [GATHERCODE, 98].

¹ Os melhores resultados são armazenados de forma a serem recuperados mais rapidamente que os resultados de outras buscas, pois assume-se que serão procurados com mais frequência em buscas futuras, conforme o aprendizado evolui.

- A **Programação Dinâmica** é um campo da matemática que tem sido tradicionalmente usado para otimização e controle. Porém, a Programação Dinâmica tradicional está limitada pelo tamanho e complexidade dos problemas que pode resolver.
- **Aprendizado Supervisionado** (*Supervised Learning*) é um método geral para treinar um aproximador de funções parametrizadas, tal como uma rede neural, para representar funções. Porém, o Aprendizado Supervisionado requer amostras de pares entrada-saída da função que está sendo aprendida. Em outras palavras, o aprendizado supervisionado requer um conjunto de perguntas com suas respectivas respostas corretas.

5.1 O Modelo de Um Agente no Aprendizado Mediante Reforços

Na forma padrão do modelo de Aprendizado Mediante Reforços um agente interage com seu ambiente. Esta interação acontece da seguinte maneira: o agente percebe o ambiente e, baseado no seu estado, escolhe uma ação à realizar. A ação, de alguma forma, modifica o ambiente e este comunica a mudança ao agente através de sinais escalares de reforço, colocando-o em uma nova situação, na qual terá que tomar uma nova decisão. Em geral, ações podem ser qualquer decisão que o agente deseja aprender a tomar e situações podem ser qualquer coisa a partir da qual possa medir-se a utilidade de tê-las tomado.

No *RL*, o conceito de meta é modelado através destes sinais especiais chamados de **recompensa**. A meta do agente é maximizar a soma das recompensas que recebe, desde um dado estado “inicial” até um estado “terminal” prefixado. Isto significa não apenas as recompensas imediatas, mas a recompensa total recebida ao longo de todo o processo.

Formalmente, o modelo consiste de:

- ♦ um conjunto discreto de estados do ambiente S
- ♦ um conjunto discreto de ações do agente A , onde $A(s)$ é o conjunto de ações que podem ser tomadas no estado s ; e
- ♦ um conjunto de sinais de reforço escalares; tipicamente $\{0,1\}$, ou números reais.

Um agente, no Aprendizado Mediante Reforços, geralmente consiste de quatro componentes básicos: uma política, uma função de reforço (ou de recompensa), uma função de valor e um modelo do ambiente. Esta estrutura está ilustrada na figura 5.1. Os componentes de um agente no Aprendizado Mediante Reforços serão descritos nas próximas seções.

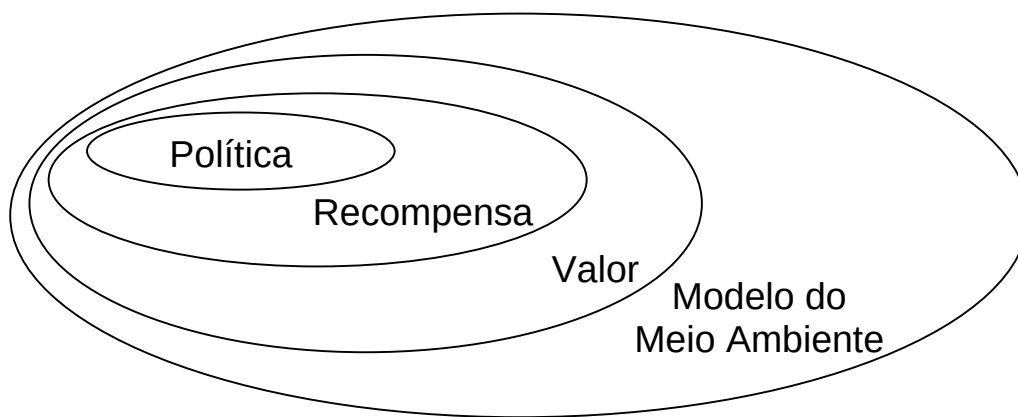


Figura 5.1: Componentes principais de um Agente no Aprendizado Mediante Reforços.

5.1.1 A Política

A **política** é a função para a tomada de decisões do agente, especifica que ação ele deve tomar em cada uma das situações em que pode se encontrar. Em psicologia, isto corresponderia ao conjunto de regras de associação estímulo-resposta. Como é sugerido pela figura 5.1, a política é a essência de um agente no Aprendizado Mediante Reforços, pois ela sozinha é suficiente para definir completamente o comportamento do agente. Os outros componentes servem para mudar e melhorar a política, mas ela é o principal determinante do comportamento e do desempenho do agente.

5.1.2 A Função de Reforços

A **função de reforços (ou de recompensa)** qualifica as decisões do agente, definindo quais são boas e quais são ruins. Na verdade, esta é a função exata dos reforços futuros que o agente busca maximizar. O sistema de *RL* aprende um mapeamento de situações para ações mediante interações de tentativa-e-erro com um ambiente dinâmico. Isto acontece porque existe um mapeamento entre cada par estado/ação e um reforço. Em outras palavras, após uma ação ter sido realizada num dado estado, o agente recebe algum reforço (recompensa) na forma de um valor escalar. Conhecendo os reforços associados a cada par estado/ação ele é capaz de aprender a realizar ações que maximizem a soma dos reforços recebidos desde algum estado inicial até o estado terminal.

As recompensas ou reforços são imediatos e são definidos de forma a representar as características do problema enfrentado. Portanto, a função de recompensa deve ser necessariamente fixa. Pode, porém, ser usada como base para mudar a política. Por exemplo, se uma ação selecionada pela política é seguida por uma baixa recompensa, então a política pode ser mudada para selecionar alguma outra ação naquela situação, no futuro.

Assim, os métodos de *RL* especificam como tentativas e erros sistemáticos resultam em mudanças na política. E é desta forma que o agente pode aprender a atingir sua meta.

É trabalho do projetista de sistemas de *RL*, definir uma função de reforços que reflita apropriadamente as metas do agente. Embora possam ser definidas funções de reforços complexas, existem três classes notáveis destas funções freqüentemente usadas para construir funções de reforço que definam apropriadamente as metas desejadas: *Pure Delayed Reward and Avoidance Problems*, Tempo Mínimo até a Meta e Jogos.

- ***Pure Delayed Reward and Avoidance Problems***: nesta classe de funções todos os reforços são zero exceto no estado terminal. Neste estado o sinal do reforço escalar

indica se este é um estado meta (um prêmio) ou um estado que deveria ser evitado (uma penalidade).

- **Tempo Mínimo até a Meta:** as funções de reforços desta classe fazem com que o agente realize ações que gerem o menor caminho ou trajetória até um estado meta.
- **Jogos:** até agora assumimos que o agente aprendiz sempre tenta maximizar a função de reforço. Nem sempre isto é verdade. Ele pode facilmente aprender a minimizar a função de reforço da mesma maneira que aprende a maximizá-la. Isto pode ser necessário quando o reforço é uma função de recursos limitados e o agente tem que aprender a conservar estes recursos enquanto tenta alcançar uma meta (por exemplo, um avião que executa uma manobra enquanto procura conservar o máximo de combustível possível). Uma função de reforço alternativa seria usada no contexto de um ambiente de jogo, quando há dois ou mais jogadores com metas adversárias. Em um cenário de jogo, o sistema de *RL* pode aprender a gerar comportamento ótimo para os jogadores ao achar o *maximin*, *minimax*, ou ponto de sela da função de reforço.

5.1.3 A Função de Valor

Considerando-se que a função de recompensa indica que ações são boas, de forma imediata, a **função de valor** especifica quais são potencialmente boas, pelo fato de que ela prediz a recompensa que pode vir a ser obtida a partir de uma ação. A diferença entre **recompensa** e **valor** é crítica para o Aprendizado Mediante Reforços. Por exemplo, quando jogamos xadrez, dar xeque mate no oponente é associado com uma grande recompensa, mas ganhar sua rainha é associado com um valor alto. A primeira define a verdadeira meta do agente -ganhar o jogo- enquanto que a segunda só prediz a verdadeira meta.

As recompensas organizadas para serem entregues a um agente do *RL*, devem representar o que realmente se quer que ele faça. Em particular, o sinal de reforço não é forma de dar ao agente conhecimento prévio de como fazer o que queremos que ele faça. Por exemplo, um agente que joga xadrez deveria ser recompensado somente se ele ganha, não por alcançar submetas tais como tomar as peças do seu oponente ou ganhar o

controle do centro do tabuleiro. Se este tipo de submetas são incluídas nas recompensas, o agente pode achar um modo para alcançá-las sem alcançar a meta real; por exemplo, levar as peças do oponente mesmo à custas de perder o jogo. Assim, a função de recompensa é o modo como comunicamos ao agente o que queremos que ele faça e não como queremos que ele o faça.

Aprender o valor dos estados, ou dos pares estado/ação, é o passo crítico nos métodos de Aprendizado Mediante Reforços. A questão é: como o agente aprende a escolher boas ações, ou ainda, como podemos medir a utilidade de uma ação? Primeiramente, observemos a relação existente entre a política e a função do valor. Como foi dito, uma **política** determina qual a ação a realizar em cada estado, uma política é um mapeamento de estados a ações; por outro lado, o **valor de um estado** é definido como a soma dos reforços recebidos quando se começa nesse estado e se segue alguma política fixa até um estado terminal. A política ótima, portanto, deveria ser aquele mapeamento de estados para ações, tal que, quando um agente começa em um estado arbitrário e realiza ações até que um estado terminal seja atingido, a soma dos reforços recebidos é maximizada. Sob esta definição, o valor de um estado é dependente da política. A função de valor é um mapeamento dos estados aos valores dos estados.

Um exemplo de uma função de valor pode ser visto através de um processo de decisão de Markov (*MDP – Markov Decision Process*²) [CHUNG, 83] simples. O espaço de estados pode ser visualizado mediante uma grade 4x4. Cada quadrado representa um estado (ver figura 5.2). A função de reforços é -1 sempre (isto é, os agentes recebem um reforço de -1 em cada transição). Há 4 ações possíveis em cada estado: norte, sul, leste, oeste. Os estados meta são o canto superior esquerdo e o canto inferior direito. A função de valor para uma política aleatória é mostrada na figura 5.2. Para cada estado, esta política, aleatoriamente, escolhe uma das quatro possíveis ações. Os números nos estados representam seu valor esperado. Neste exemplo, quando começamos no canto

² um *MDP* consiste de um conjunto de estados X ; um conjunto de estados iniciais S o qual é um subconjunto de X ; um conjunto de ações A ; uma função de reforço R , onde $R(x, a)$ é o reforço imediato esperado obtido quando se toma a ação a no estado x ; e um modelo de ações P , onde $P(x'|x, a)$ dá a probabilidade de que a realização da ação a no estado x levará ao estado x' . Nota: A escolha da ação deve ser dependente apenas do estado atual x . Se ações ou estados anteriores afetam a atual escolha da ação, então o processo de decisão não é de Markov.

superior esquerdo e seguimos uma política aleatória, na média teremos 22 transições à outros estados antes que o estado terminal seja atingido.

0	-14	-20	-22
-14	-18	-22	-20
-20	-22	-18	-14
-22	-20	-14	0

Figura 5.2: Função de valor para uma política aleatória.

A função de valor ótima, é mostrada na figura 5.3a. Novamente, começando no canto inferior esquerdo, calculamos a soma dos reforços quando realizamos uma política ótima (a política que maximiza a soma dos reforços), o valor deste estado é -3, pois com somente 3 transições a partir dele, atingimos um estado terminal. Se a função de valor ótima fosse dada *a priori*, então seria uma tarefa trivial extrair a política ótima. Para isto, basta que comecemos em qualquer estado e simplesmente escolhamos a ação que maximiza o reforço imediato recebido. Em outras palavras, podemos realizar uma busca primeiro em largura de um nível de profundidade sobre as ações para encontrar a ação que maximiza a premiação imediata. A política ótima para a função mostrada na figura 5.3a está ilustrada na figura 5.3b.

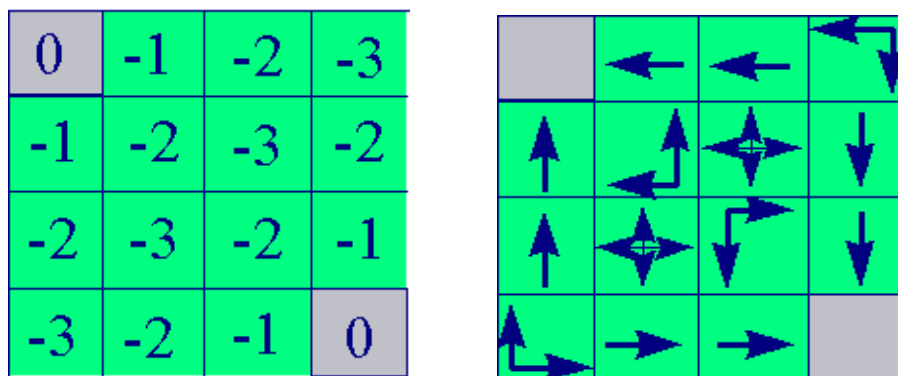


Figura 5.3: a) função de valor ótima. b) política ótima para a função ilustrada na figura 5.3a.

5.1.4 O Ambiente

O quarto e último componente principal de um agente no Aprendizado Mediante Reforços é um modelo do seu ambiente ou mundo externo, isto é, algo que imite o comportamento do ambiente de alguma forma. Por exemplo, dada uma situação e uma ação, o modelo poderia prever o próximo estado resultante e a próxima recompensa. O modelo do meio ambiente é desenhado como o maior componente na figura 5.1 pois podemos esperar que ele ocupe o maior espaço de armazenamento. Se há $|S|$ estados e $|A|$ ações, então um modelo completo do meio ambiente ocupará um espaço proporcional a $|S| \times |S| \times |A|$, pois ele mapeia pares estado-ação para distribuições de probabilidade sobre estados, fornecendo a probabilidade de cada possível estado resultante de cada ação tomada em cada estado. As funções de recompensa e de valor mapeiam estados para números reais, e, assim, serão no máximo de tamanho $|S|$, enquanto uma política, a qual mapeia estados para ações, será no máximo de tamanho $|S| \times |A|$.

Cada sistema de *RL* aprende um mapeamento de situações a ações mediante interações de tentativa-e-erro com um ambiente dinâmico. Este ambiente deve ser ao menos parcialmente observável pelo sistema de Aprendizado Mediante Reforços. Informações podem entrar no sistema através de leituras de sensores, descrições simbólicas ou, possivelmente, situações mentais (por exemplo, a situação de sentir-se perdido). As ações podem ser de baixo nível (por exemplo, voltagem para um motor), alto nível (por exemplo, aceitar um trabalho oferecido), ou ainda mentais (por exemplo, prestar atenção).

5.1.5 A Interface Agente-Ambiente

Como dito anteriormente, o Aprendizado Mediante Reforços consiste em aprender um mapeamento de situações para ações afim de atingir uma meta. Para isto, o *RL* é uma abstração simplificada de um agente interagindo com seu meio ambiente. No modelo padrão, um agente *B* está conectado a seu ambiente via percepção e ação, como representado na figura 5.4. A cada intervalo de tempo discreto ou iteração t , o agente recebe como entrada o estado atual do ambiente, $s_t \in S$, e a partir desta informação,

escolhe uma ação $a_t \in A(s_t)$ para gerar como saída. A ação muda o estado do ambiente e, um passo de tempo depois (em parte como consequência da ação tomada) o valor desta transição de estado é comunicado ao agente através de uma *signal escalar de reforço* $r_{t+1} \in R$; chegando-se a uma nova situação (ou estado do ambiente) s_{t+1} .

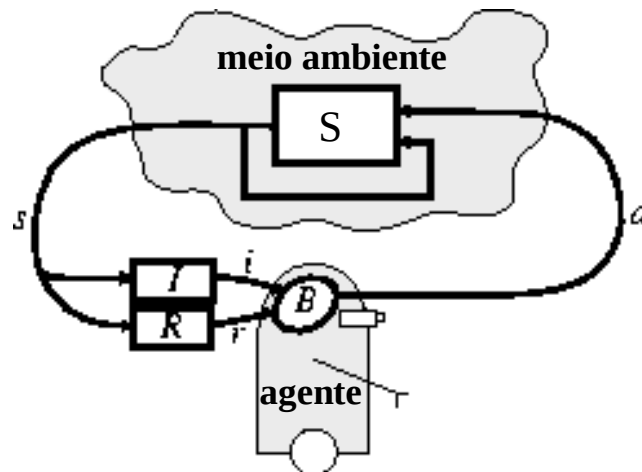


Figura 5.4: O modelo padrão de Aprendizado Mediante Reforços. Na figura, foi incluída uma função de entrada I, a qual determina como o agente vê o estado do ambiente. Nós assumimos que esta é a função identidade (isto é, o agente percebe o exato estado do ambiente); porém, o ambiente poderia também ser considerado como parcialmente observável.

Para exemplificar melhor, o seguinte diálogo tenta expressar intuitivamente a relação entre o agente e seu ambiente:

Ambiente:	Você está no estado 65. Você tem 4 ações possíveis.
Agente:	Eu tomarei a ação 2.
Ambiente:	Você recebeu uma premiação de 7 unidades. Agora você está no estado 15. Você tem 2 ações possíveis.
Agente:	Eu tomarei a ação 1.
Ambiente:	Você recebeu uma premiação de -4 unidades. Agora você está no estado 65. Você tem 4 ações possíveis.
Agente:	Eu tomarei a ação 2.
Ambiente:	Você recebeu uma premiação de 5 unidades. Agora você está no estado 44. Você tem 5 ações possíveis.
.	.
.	.

Figura 5.5: Representação da relação entre um agente e seu ambiente através de um diálogo fictício.

Em geral, espera-se que o ambiente seja não-determinístico; isto é, que a tomada de uma mesma ação no mesmo estado em duas ocasiões diferentes possa resultar em estados seguintes diferentes e/ou valores de reforço diferentes (como acontece no exemplo acima). Porém, assume-se que o ambiente é estacionário; isto é, que as probabilidades de se fazer uma transição de estado ou de se receber um sinal de reforço específico não mudam com o tempo.

O Aprendizado Mediante Reforços difere em vários aspectos da maioria dos problemas de Aprendizado Supervisionado. A diferença mais importante é que não precisa de pares de entrada/saída. Ao invés disso, depois que o agente escolhe uma ação, recebe uma recompensa imediata e atinge um subsequente estado do sistema, mas não é informado de qual ação teria sido melhor para seus objetivos. Desta forma, é necessário que o agente acumule experiência útil acerca dos possíveis estados do sistema, ações, transições e recompensas para que possa agir otimamente.

A **interface** entre o agente e seu ambiente não deve ser vista como o limite físico do corpo de um robô ou animal. Normalmente, o limite é mais íntimo do que isso para o agente. Por exemplo, os motores e engrenagens mecânicas de um robô e seu hardware para percepção, poderiam ser considerados parte do ambiente em lugar de parte do agente. Semelhantemente, se aplicamos o mesmo argumento para uma pessoa ou animal, os músculos, esqueleto e órgãos sensores como olhos e orelhas, deveriam ser considerados parte do ambiente. As recompensas também são, presumivelmente, computadas dentro dos corpos físicos dos sistemas de aprendizado, tanto naturais quanto artificiais, porém, são consideradas externas ao agente do Aprendizado Mediante Reforços.

A regra geral é que qualquer coisa que não pode ser mudada arbitrariamente pelo agente do *RL* é considerada externa a ele e assim, parte do ambiente. Observemos, também, que não assumimos que tudo no ambiente é desconhecido pelo agente, assumimos apenas que não é totalmente controlável por ele. Assim, por exemplo, o agente frequentemente saberá, com grande aproximação, como suas recompensas são computadas em função de suas ações e das situações nas quais elas são executadas. Porém, a computação da

recompensa sempre é considerada externa ao agente, porque define o problema enfrentado por ele e, assim, deve estar além de sua habilidade, para mudá-la arbitrariamente. De fato, em alguns casos, o agente pode saber tudo acerca de seu ambiente e, ainda assim, enfrentar um difícil problema de Aprendizado Mediante Reforços, da mesma maneira que é possível saber exatamente como um quebra-cabeça como o cubo de Rubik trabalha e, ainda assim, estar impossibilitado de resolvê-lo. Logo o limite entre o agente e seu ambiente representa o limite do que o agente controla, não do que ele conhece.

5.2 Exploração x Exploração no *RL*

Um dos desafios que surgem no aprendizado mediante reforços, e não em outros tipos de aprendizagem, é o balanceamento entre exploração e exploração. Para obter muita recompensa, um agente do Aprendizado Mediante Reforços prefere realizar as ações que tentou no passado e achou efetivas na produção de recompensas. Mas para descobrir quais ações são essas, ele tem que selecionar ações que não tentou antes. O agente tem que explorar o que já conhece para obter recompensas, mas ao mesmo tempo, tem que explorar para fazer uma melhor escolha de suas ações no futuro. O dilema é que nem a exploração nem a exploração podem ser seguidas exclusivamente sem falhar à tarefa. O agente tem que tentar uma variedade de ações e progressivamente favorecer aquelas que pareçam ser melhores

5.3 Modelos de Conduta Ótima

Antes de pensar em algoritmos para que um agente aprenda a agir otimamente, temos que decidir qual será nosso modelo de otimalidade. Em particular temos de especificar como o agente levará em conta o futuro nas decisões que fará acerca de da ação que deve tomar no presente. Há três modelos que tem sido utilizados na maioria dos trabalhos nesta área: o **modelo de horizonte finito**, o **modelo de horizonte infinito com descontos** e o **modelo de recompensa média**.

O **modelo de horizonte finito** (*finit-horizon model*) é o mais imediato; em um dado momento no tempo, o agente deverá otimizar sua recompensa esperada para os seguintes h passos, isto é, neste modelo não se considera o que acontecerá após os h passos. A recompensa esperada é dada por:

$$E\left(\sum_{t=0}^h r_t\right) \quad (5.1)$$

onde, r_t representa o reforço escalar recebido no passo t .

Este modelo pode ser usado de duas maneiras. Na primeira, o agente terá uma política não-estacionária; isto é, sua política muda com o tempo. No primeiro passo ele realizará aquela ação indicada como ótima de passo h . Esta é definida como sendo a melhor ação disponível dado que ainda se tem h passos restantes nos quais executar ações e obter premiações. No passo seguinte será realizada a ação ótima de passo $(h-1)$, e assim por diante, até que, finalmente, seja realizada a ação ótima de passo 1 e o agente termine os h passos. Na segunda abordagem do modelo, o agente faz o chamado *receding-horizontal control*, realizando sempre a ação ótima de passo h . O agente sempre age segundo uma mesma política, mas o valor de h limita seu campo de visão para a escolha de sua ação.

O modelo de horizonte finito nem sempre é o mais apropriado. Aliás, em muitos casos, não é conhecido *a priori* o “tempo de vida exato” do agente (número de passos que executará).

O **modelo de horizonte infinito com descontos** (*infinite-horizon discounted model*) leva em consideração as recompensas recebidas pelo agente ao longo de todo o processo, mas as que serão recebidas no futuro são geometricamente descontadas segundo um fator de desconto γ , (onde $0 \leq \gamma < 1$). Formalmente, a recompensa esperada é dada por:

$$E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \quad (5.2)$$

Pode-se interpretar γ de várias formas. Ele pode ser visto como uma taxa de interesse, a probabilidade de viver outro passo, ou como um truque matemático para limitar a soma

infinita. Este modelo é conceitualmente similar à *receding-horizon controle*, porém, o modelo de descontos é matematicamente mais tratável que o modelo de horizonte finito.

O **modelo de recompensa média** apresenta outro critério de otimalidade, no qual o agente realizará as ações que otimizem sua recompensa média durante todo o processo. Neste modelo, a recompensa média esperada a ser otimizada é dada por:

$$\lim_{h \rightarrow \infty} E\left(\frac{1}{h} \sum_{t=0}^h r_t\right) \quad (5.3)$$

Uma política como esta é referida como uma política de ganho ótimo. Ela pode ser vista como o caso limite do modelo de horizonte infinito com descontos, com o fator de desconto aproximado à 1 [14]. Um dos problemas com este critério é que não há modo de se distinguir entre duas políticas, uma na qual se ganha uma grande quantidade de recompensas na fase inicial e outra na qual isto não acontece.

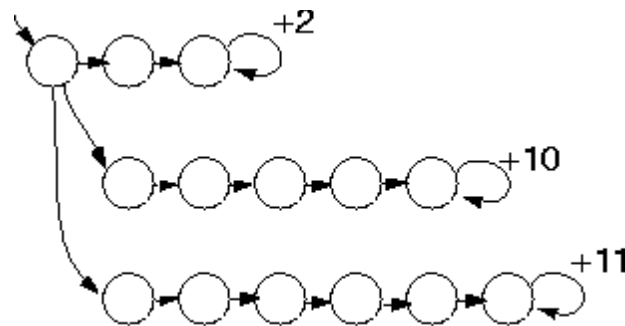


Figura 5.6: Uma rede de estados e ações para a comparação de modelos de conduta ótima.

A figura 5.6 contrasta estes modelos de otimalidade mediante um ambiente onde a troca do modelo de otimalidade muda a política ótima. Neste exemplo, círculos representam estados do ambiente e setas são as transições entre estados. Há somente uma ação a ser escolhida em cada estado, exceto no estado inicial, o qual está no topo da figura, à esquerda e está marcado com uma seta de entrada. Todas as recompensas são zero exceto onde indicado. Sob um modelo de horizonte finito com $h=5$, as três ações produzem respectivamente recompensas de +6.0, +0.0, e +0.0, assim a primeira ação deveria ser escolhida. Sob um modelo de horizonte infinito com descontos com $\gamma=0.9$, as três escolhas produzem +16.2, +59.0, e +58.5, respectivamente. Assim a segunda ação deveria ser escolhida. Sob o modelo de recompensa média, a terceira ação deveria

ser escolhida, pois isso permite uma recompensa média de +11.0, enquanto que a primeira e a segunda permitiriam uma recompensa média de 2 e 10, respectivamente.

5.4 Aproximando a Função de Valor

O Aprendizado Mediante Reforços é uma técnica difícil de aplicar devido ao fato de que os sistemas de aprendizado (*learning systems*) podem realizar uma ação e não ser capazes de determinar se a ação foi boa ou má. Por exemplo, poderia-se dar o controle de um simulador de voo de um avião, a um programa que esteja aprendendo a pilotar e evitar batidas. O programa terá que tomar muitas decisões a cada segundo e então, após ter tomado milhares de decisões, o avião pode bater. O que o sistema pode aprender desta experiência? Qual de todas as decisões foi a responsável pela batida? Determinar a punição (ou a recompensa) de cada ação é o que faz do Aprendizado Mediante Reforços uma técnica complexa. Surpreendentemente, existe uma solução para este problema, ela é baseada na Programação Dinâmica, a qual envolve dois princípios básicos:

Primeiro, se uma ação a_e causa alguma coisa errada, isso acontecerá imediatamente (tal como a queda do avião); então, o sistema aprende a não mais realizar a ação a_e na mesma situação. Portanto, qualquer ação que o sistema realize um milissegundo antes do avião bater, deve ser evitada no futuro. Mas este princípio não serve para analisar as outras ações tomadas anteriormente e que não ocasionaram um desastre imediato.

De acordo com o segundo princípio, se todas as ações em uma determinada situação conduzem a um mal resultado; então, esta situação deve ser evitada. Este, é um princípio bastante útil, pois, o sistema de aprendizado agora pode aprender sem bater.

5.4.1 A Essência da Programação Dinâmica

Inicialmente, a aproximação da função de valor ótima é pobre. Em outras palavras, o mapeamento de estados a valores dos estados não é válido. O objetivo primário do

aprendizado é encontrar o mapeamento correto. Uma vez que isto é completado, a política ótima pode ser facilmente extraída.

Sejam: $V^*(x_t)$ a função de valor ótima, onde x_t é o vetor de estado; $V(x_t)$ a aproximação da função de valor; e $\gamma \in [0,1]$ um fator de desconto que permite que seja dada uma maior importância aos reforços imediatos que aos reforços futuros.

Em geral, $V(x_t)$ será inicializada com valores aleatórios que não conterão informação acerca da função de valor ótima $V^*(x_t)$. Isto significa que a o valor da função de aproximação num dado estado é igual ao valor verdadeiro da função de valor ótima neste estado mais algum erro de aproximação, como expressado na equação (5.4)

$$V(x_t) = e(x_t) + V^*(x_t) \quad (5.4)$$

onde $e(x_t)$ é o erro na aproximação do valor do estado ocupado no instante t . Da mesma maneira, a aproximação do valor do estado alcançado depois de realizada alguma ação no instante t é o valor verdadeiro do estado ocupado no instante $t+1$ mais algum erro de aproximação, como expressado na equação (3.5)

$$V(x_{t+1}) = e(x_{t+1}) + V^*(x_{t+1}) \quad (5.5)$$

Como estabelecido anteriormente, o valor do estado x_t para a política ótima é a soma dos reforços obtidos, quando começamos no estado x_t e realizamos ações ótimas até que um estado terminal seja atingido. Devido à esta definição, existe uma relação simples entre os valores de estados sucessivos x_t e x_{t+1} . Esta relação é definida pela equação de Bellman mostrada na equação (3.6). O fator de desconto γ é usado para decrementar exponencialmente o peso dos reforços recebidos no futuro e $r(x_t)$ representa o reforço recebido no estado x_t .

$$V^*(x_t) = r(x_t) + \gamma V^*(x_{t+1}) \quad (5.6)$$

A aproximação $V(x_t)$ também relaciona valores de estados sucessivos, como mostrado na equação (5.7). Substituindo-se $V(x_t)$ e $V(x_{t+1})$ nesta equação, pelos lados direitos das equações (5.4) e (5.5), respectivamente, obtemos a equação (5.8), a qual mostra foi expandida na equação (5.9).

$$V(x_t) = r(x_t) + \gamma V(x_{t+1}) \quad (5.7)$$

$$e(x_t) + V^*(x_t) = r(x_t) + \gamma(e(x_{t+1}) + V^*(x_{t+1})) \quad (5.8)$$

$$e(x_t) + V^*(x_t) = r(x_t) + \gamma e(x_{t+1}) + \gamma V^*(x_{t+1}) \quad (5.9)$$

Subtraindo-se a equação (5.6) da equação (5.9), obtemos a relação entre os erros de estados sucessivos, expressa na equação (5.10)

$$e(x_t) = \gamma e(x_{t+1}) \quad (5.10)$$

O significado desta relação pode ser visto usando uma cadeia de Markov simples, como a mostrada na figura 5.7. Na figura, o estado T é o estado terminal. O valor verdadeiro deste estado é conhecido *a priori*. Em outras palavras, o erro de aproximação do estado T, $e(T)$, é zero por definição. Uma analogia pode ser o estado no qual um míssil finalmente acerta ou erra seu objetivo. O valor verdadeiro deste estado é conhecido: +1 se acerta, -1 se erra.

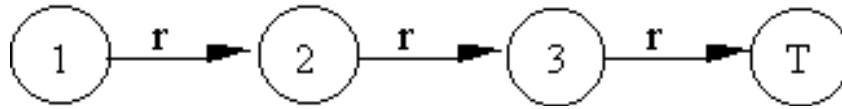


Figura 5.7: Uma cadeia de Markov simples.

O processo de aprendizado é o processo de encontrar uma aproximação $V(x_t)$ que faz as equações (5.6) e (5.10) verdadeiras para todos os estados x_t . Se o erro de aproximação no estado 3 é menor que o erro no estado T, por um fator de γ , então o erro de aproximação no estado 3 deve ser zero, pois, por definição, o erro no estado T é zero. Se a equação (5.10) é verdadeira para todo x_t , então, o erro de aproximação em cada estado x_t é necessariamente zero, isto é, $V(x_t) = V^*(x_t)$ para todo x_t .

A importância do fator de desconto γ pode ser vista através de uma outra cadeia de Markov simples, que não possui um estado terminal (figura 5.8). Usando um argumento similar ao utilizado sobre a cadeia da figura 5.7, pode-se ver que quando a equação (5.10) é satisfeita o erro de aproximação deve ser zero para todos os estados. A única diferença entre o exemplo anterior e este caso é que o erro em qualquer estado deve ser

um fator de γ^6 menor que ele mesmo (pois há 6 estados no ciclo). Portanto, a equação (5.10) só pode ser satisfeita se o erro de aproximação é zero em cada estado.

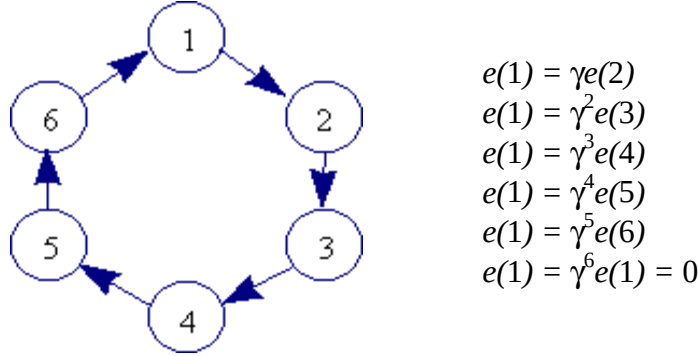


Figura 5.8: Cadeia de Markov simples sem estado terminal.

Por conseguinte, como dito anteriormente, o processo de aprendizado é o processo de encontrar uma solução para a equação (5.7) para todos os estados x_t (a qual também é uma solução para a equação (5.10)).

Nas seções 5.4.2 e 5.4.3 serão apresentados algoritmos aproximadores da função de valor.

5.4.2 Value Iteration

Value Iteration é um algoritmo iterativo para aproximar o valor dos estados. Se assumirmos que a função de aproximação usada para representar V^* é uma tabela de pesquisa (cada estado tem um elemento correspondente na tabela, o qual representa o valor aproximado do estado), então podemos encontrar a função de valor ótima, percorrendo o espaço de estados, atualizando o valor de cada estado de acordo com a equação (5.11) até que seja realizada uma varredura no espaço de estados, na qual não haja nenhuma mudança nos valores (os valores dos estados terão convergido).

$$\Delta w_t = \max_u \{r(x_t, u) + \gamma V(x_{t+1})\} - V(x_t) \quad (5.11)$$

Na equação (5.11) u é a ação realizada no estado x_t que ocasiona a transição para estado x_{t+1} , e $r(x_t, u)$ é o reforço recebido quando é realizada a ação u no estado x_t .

A figura 5.9 descreve o escopo de uma única atualização da aproximação do valor de x_t . Neste exemplo específico, há duas ações possíveis no estado x_t , e cada uma destas ações leva a um diferente estado sucessor x_{t+1} .

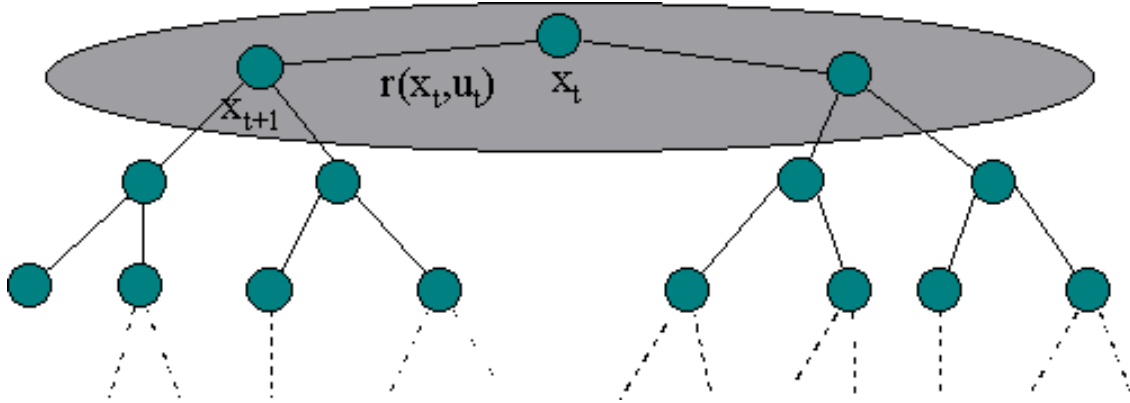


Figura 5.9: Escopo de uma atualização da aproximação do valor do estado x_t .

Na atualização do *Value Iteration*, deve-se encontrar primeiro a ação que retorna o valor máximo. O único modo para completar isto é, de fato, executar uma ação e calcular a soma dos reforços recebidos e o valor aproximado (possivelmente descontado) do estado sucessor $V(x_{t+1})$. Isto deve ser feito para todas as ações u num dado estado x_t , e isso não é possível sem um modelo da dinâmica do sistema. Por exemplo, no caso em que um robô deve tomar uma decisão sobre um caminho a seguir, não é possível escolher um caminho, observar o estado sucessor, e então retornar para o estado inicial afim de explorar os resultados da próxima ação disponível. Ao invés disso, o robô precisa realizar estas ações numa simulação e observar os resultados. Então, baseado nos resultados da simulação, o robô pode escolher a ação que resulta no valor máximo.

Podemos notar que o lado direito da equação 5.11 é, simplesmente, a diferença dos dois lados da equação de Bellman definida na equação 5.7, considerando-se que generalizamos esta equação para adequá-la ao Processo de Decisão de Markov (múltiplas ações possíveis num dado estado ao invés de uma única ação possível como nas cadeias de Markov). Esta expressão é a Residual de Bellman, e é formalmente definida pela equação 5.12.

$$e(x_t) = \max_u \{r(x_t, u) + \gamma V(x_{t+1})\} - V(x_t) \quad (5.12)$$

$e(x_t)$ é a função de erro definida pela Residual de Bellman sobre todo o espaço de estados. Cada atualização (equação 5.11) reduz o valor de $e(x_t)$ e conforme o número de atualizações tende ao infinito, $e(x_t)$ tende a 0. Quando $e(x_t) = 0$, a equação 5.7 é satisfeita, $V(x_t) = V^*(x_t)$ e o aprendizado se completa.

5.4.3 Q-Learning

O Q-Learning [WATKINS, 1989], é uma extensão do *Value Iteration* que resolve o seguinte problema por ele enfrentado:

Um Processo de Decisão de Markov Determinístico é aquele no qual as transições entre os estados são determinísticas (um ação realizada no estado x_t sempre conduz ao mesmo estado sucessor x_{t+1}). Alternativamente, em um Processo de Decisão de Markov Não-determinístico, uma função de distribuição de probabilidade define um conjunto de potenciais estados sucessores, para uma ação num dado estado. Se o *MDP* é não-determinístico, então **value iteration** requer que seja encontrada a ação que retorna o máximo valor esperado (a soma do reforço com a integral sobre todos os possíveis estados sucessores para a ação). Por exemplo, para encontrar o valor esperado do estado sucessor associado com uma dada ação, devemos realizar esta ação um número infinito de vezes, calculando a integral sobre os valores de todos os possíveis estados sucessores para esta ação. O motivo pelo qual isto é necessário, pode ser explicado analisando-se a figura 5.10. Observe que há duas possíveis ações no estado x . Cada ação retorna um reforço de 0. A ação u_1 causa a transição para um de dois possíveis estados com igual probabilidade. O mesmo acontece para a ação u_2 . Os valores dos estados sucessores são 0 e 1 para ambas as ações. O *Value Iteration* requer que o valor do estado x seja dado pela expressão

$$V(x_t) = r(x_t, u) + \max_u \{V(x_{t+1})\} \quad (5.13)$$

Tomando-se um número infinito de amostras de estados sucessores para a ação u_1 , poderia-se calcular que o valor esperado atual é 0.5. O mesmo acontece para a ação u_2 . Portanto, o valor do estado x é 0.5. Porém, se ingenuamente executarmos o *Value Iteration* neste *MDP*, tomando um exemplo simples do estado sucessor associado com

cada ação ao invés da integral, então o valor de x poderia convergir para 0.75. Esta, claramente, é a resposta errada.

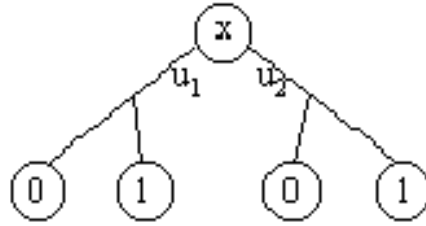


Figura 5.10: Uma rede de estados e ações para a ilustração do problema enfrentado pelo *Value Iteration*.

Teoricamente, aplicar *Value Iteration* no contexto dos MDPs não-determinísticos é possível. Porém, na prática, é computacionalmente impossível calcular as integrais necessárias sem conhecimento extra ou algum grau de modificação. *Q-Learning* resolve o problema de se ter que pegar o máximo sobre um conjunto de integrais.

Melhor que encontrar um mapeamento dos estados para os seus valores (como no *Value Iteration*), *Q-Learning* encontra um mapeamento dos pares estado/ação para os valores (chamados *Q-values*). Ao invés de ter uma função de valor associada, *Q-Learning* faz uso da *Q-function*. Em cada estado, há um *Q-value* associado com cada ação. O *Q-value* de uma ação é a soma dos reforços recebidos (possivelmente descontados) quando realizamos a ação e então seguimos uma dada política a partir deste ponto.

No contexto do *Q-Learning*, o valor de um estado é definido como o máximo *Q-value* no estado dado. Dada esta definição, é fácil derivar a equação de Bellman (equação 5.7) equivalente para o *Q-Learning*.

$$Q(x_t, u_t) = r(x_t, u_t) + \gamma \max_{u_{t+1}} \{Q(x_{t+1}, u_{t+1})\} \quad (5.14)$$

Q-Learning difere do *Value Iteration* pelo fato de que este não requer que em um dado estado cada ação seja realizada e que os valores esperados dos estados sucessores sejam calculados. Enquanto o *Value Iteration* realiza uma atualização que é análoga a uma busca em amplitude de um nível, *Q-Learning* toma uma amostra de passo-único em uma rodada do método de Monte-Carlo. Este processo está ilustrado na figura 5.11.

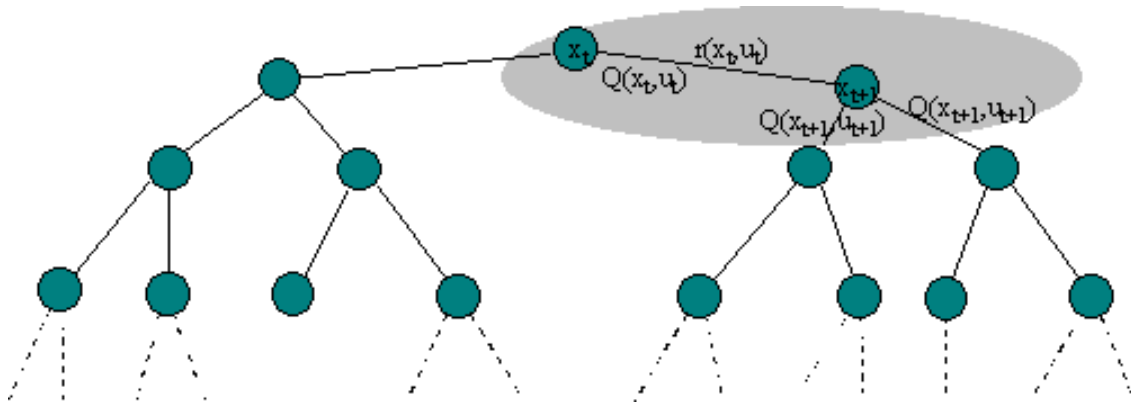


Figura 5.11: Escopo de uma atualização da aproximação do Q-Value do estado x_t .

A equação de atualização na figura 5.11 é válida quando é usada uma tabela de pesquisa para representar a *Q-function*. O *Q-value* de uma ação em um determinado estado é uma previsão da soma dos reforços que um agente recebe quando realiza esta ação neste estado e segue a política dada. Para atualizar esta previsão $Q(x_t, u_t)$, deve-se realizar a ação associada u_t , ocasionando-se uma transição para o estado seguinte x_{t+1} e retornando-se um reforço escalar $r(x_t, u_t)$. Então, somente é necessário encontrar o *Q-value* máximo no novo estado para ter toda a informação necessária para revisar a previsão (*Q-value*) associada com a ação recentemente realizada. *Q-Learning* não requer o cálculo da integral sobre todos os possíveis estados sucessores no caso em que a transição entre estados seja não-determinística. A razão disto é que uma única amostra de um estado sucessor para uma dada ação é uma estimativa equilibrada³ do valor esperado para o estado sucessor. Em outras palavras, após muitas atualizações, o *Q-value* associado com uma determinada ação, convergirá para a soma esperada de todos os reforços recebidos quando esta ação é realizada e seguimos a política ótima, a partir deste ponto.

Além dos algoritmos para a aproximação da função de valor ótima apresentados neste capítulo, existem vários outros. Alguns podem ser encontrados em [SUTTON, 98].

O *Q-Learning* nos interessa em especial pois foi neste algoritmo que se baseou nossa adaptação do *Ant Colony System*, que será descrita no capítulo 6.

³ O erro esperado nesta estimativa é zero.

Capítulo 6

Uma Introdução ao *Ant Colony Systems*

No mundo animal, podem ser encontrados vários casos de sistemas sociais integrados por indivíduos com poucas capacidades quando comparados com sua complexa conduta coletiva. Isto é observado em diferentes estágios de desenvolvimento da vida animal: bactérias [SHAPIRO, 88], formigas [GOSS, 90], minhocas [FITZGERALD, 88], moluscos e larvas. Além disso, o mesmo processo causal que origina esta conduta é em grande parte conservado em espécies de maior hierarquia, tais como: peixes, pássaros e mamíferos. Estas espécies, fazem uso de diferentes meios de comunicação, adaptados a diversas situações, mas que, essencialmente, permitem o mesmo padrão de conduta.

Isto sugere que os mecanismos de comunicação naturais mostraram ser mecanismos de evolução extremamente efetivos e merecem ser analisados quando se tenta alcançar uma meta semelhante a de realizar tarefas complexas por meio de atividades distribuídas sobre sistemas massivamente paralelos.

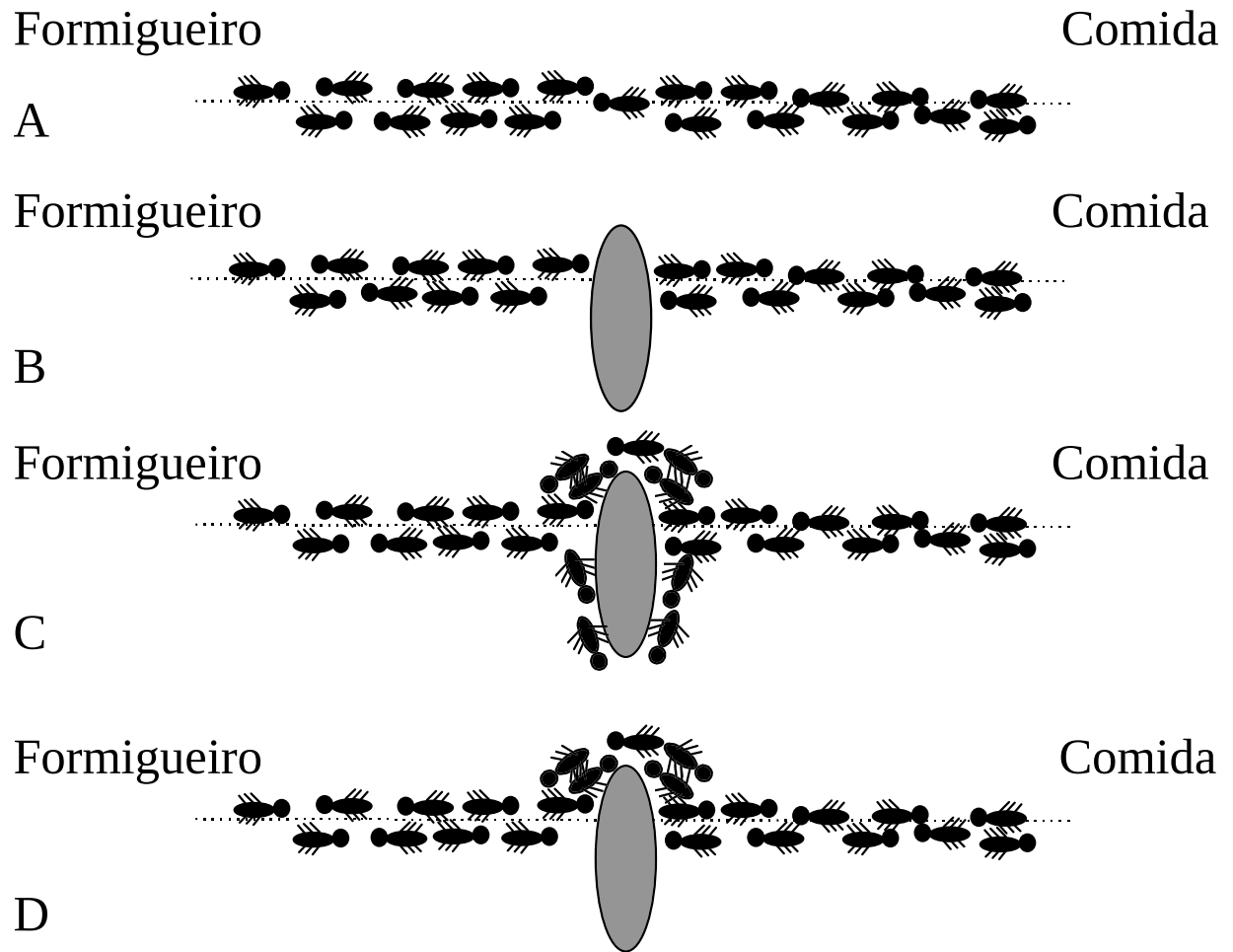
Um dos mais estudados casos de atividades distribuídas são as chamadas colônias de formigas [DENEBOURG, 83]. Mesmo quando uma única formiga possui capacidades simples, a conduta de toda a colônia de formigas é altamente estruturada. E isto é o resultado de interações coordenadas. Mas, devido ao fato de que as possibilidades de comunicação entre as formigas são limitadas, as interações devem estar baseadas em fluxos de informação muito simples. Aqui, o problema de interesse é: “como animais quase cegos agem para estabelecer a rota mais curta desde sua colônia até uma fonte de alimento?”.

No caso das formigas, o meio usado para comunicar aos indivíduos informações sobre caminhos percorridos e para decidir qual caminho escolher, consiste em uma trilha de feromônios. As formigas depositam uma certa quantidade (variável) de feromônios enquanto caminham e cada formiga prefere seguir uma trilha rica em feromônios. Esta

conduta elementar das formigas reais¹ pode ser usada para explicar como elas encontram o menor caminho que reconecta uma linha interrompida pela aparição de um obstáculo inesperado sobre a trilha do caminho inicial. De fato, uma vez que um obstáculo aparece, aquelas formigas que se encontram frente ao obstáculo não podem seguir a trilha de feromônios e assim elas têm que escolher entre ir para a esquerda ou para direita. Nesta situação, nós podemos supor que a metade das formigas escolha ir para a esquerda e a outra metade escolha a direita. Pode-se esperar uma situação similar do outro lado do obstáculo. É interessante observar, que aquelas formigas que escolheram, por sorte, o menor caminho ao redor do obstáculo, reconstituirão mais rapidamente a trilha de feromônios interrompida, em comparação com aquelas que escolheram o caminho maior. Assim, o menor caminho receberá uma maior quantidade de feromônios por unidade de tempo e em breve um maior número de formigas o escolherão. Devido a esse processo de retroalimentação positiva (autocatalítico ou alelomimético), todas as formigas rapidamente escolherão o menor caminho. O aspecto mais interessante deste processo autocatalítico é que, encontrar o menor caminho ao redor do obstáculo, parece ser uma propriedade emergente² da interação entre a forma do obstáculo e a conduta distribuída das formigas. Embora todas as formigas se movimentem à mesma velocidade e depositem uma trilha de feromônios aproximadamente à mesma razão, elas tardam mais para contornar obstáculos pelo seu maior lado do que pelo menor, o que faz com que a trilha de feromônios se acumule mais rapidamente sobre o lado menor. E a preferência das formigas, por trilhas com maiores níveis de feromônios, faz esta acumulação ser ainda mais rápida sobre o lado menor.

¹ Formigas reais é como chamaremos estes insetos da natureza.

² **Emergencia** ou comportamento emergente é um conceito ainda não definido formalmente; porem, supõe a capacidade de producir comportamentos, não planejados, a partir da interação dos componentes do sistema observado. Vide [Holland, 98] numa tentativa por formalizar este conceito.



(A) Formigas reais seguem o caminho entre o formigueiro e a fonte de comida. (B) Um obstáculo aparece no caminho: as formigas escolhem entre ir para a direita ou ir para esquerda com igual probabilidade. (C) Feromônios são depositados mais rapidamente sobre o menor caminho. (D) Chega um momento onde todas as formigas escolhem o menor caminho.

Figura 6.1: A influência dos Feromônios sobre as formigas reais, na escolha do menor caminho

Atualmente, muitos dos métodos computacionais para a solução de problemas complexos baseiam-se em processos naturais como o que acabamos de descrever.

Ant Colony Optimization (ACO), é um conjunto de técnicas inspiradas no comportamento das colônias de formigas, que tem sido aplicadas com êxito a problemas de otimização combinatória.

O primeiro sistema a utilizar *ACO* foi o *Ant System (AS)*, proposto por Marco Dorigo na sua Tese de Doutorado [DORIGO, 92]. O AS foi aplicado ao Problema do Caixeiro Viajante e ao Problema de Alocação Quadrática.

Recentemente, desenvolveram-se várias extensões do paradigma do AS. Uma destas é o *Ant-Q*, uma forma híbrida de AS com *Q-Learning* [GAMBARDELA, 97B], uma técnica baseada no Aprendizado Mediante Reforços³. Entretanto, o *Ant-Q* difere do *Q-Learning* pelo fato de que, enquanto as aplicações típicas do *Q-Learning* empregam um único agente que explora um espaço de estados, *Ant-Q* emprega um conjunto de agentes que cooperam entre si, mediante o intercâmbio de informações.

Outra extensão mais recente é, o sistema baseado em colônia de formigas *Ant Colony System* (ACS) uma variante do *Ant-Q*, onde foi introduzida uma maneira diferente de atualizar a experiência acumulada pelas formigas artificiais [DORIGO, 97]. Estas duas extensões têm sido aplicadas com grande sucesso ao Problema do Caixeiro Viajante Simétrico e Assimétrico. Nos interessa, particularmente, uma das aplicações do *Ant Colony System* ao TSP [GAMBARDELLA, 97A], na qual baseou-se a colônia de formigas utilizada na metaheurística proposta no capítulo 9. Neste capítulo, descrevemos esta aplicação e as adaptações que foram feitas em nosso trabalho.

6.1 Um Modelo do Comportamento de uma Formiga Real

Segundo [Nakamura, 95], uma formiga pode comportar-se de quatro modos diferentes, descritos a seguir:

- ↪ **busca**, a formiga procura uma fonte de alimentos.
- ↪ **atração**, a formiga é atraída para uma trilha pelos feromônios.
- ↪ **rastro**, a formiga segue uma trilha de feromônios.
- ↪ **transporte**, a formiga transporta um pedaço de alimento para seu ninho.

O modo **busca** é o modo inicial. As formigas se movimentam aleatoriamente neste modo. Quando uma formiga em qualquer modo encontra uma fonte de alimento, ela muda para o modo **transporte**, no qual carrega um pouco do alimento de volta para o ninho da sua colônia. O alimento pode estar em vários lugares. As formigas que estão no modo

³ Vide capítulo 5.

transporte, liberam uma certa quantidade de feromônios sobre seu caminho, o que logo mais se torna uma trilha de feromônios. Uma formiga no modo **transporte** retorna para o modo **busca** quando alcança seu ninho. O modelo de comportamento de uma formiga real está ilustrado na figura 6.2.

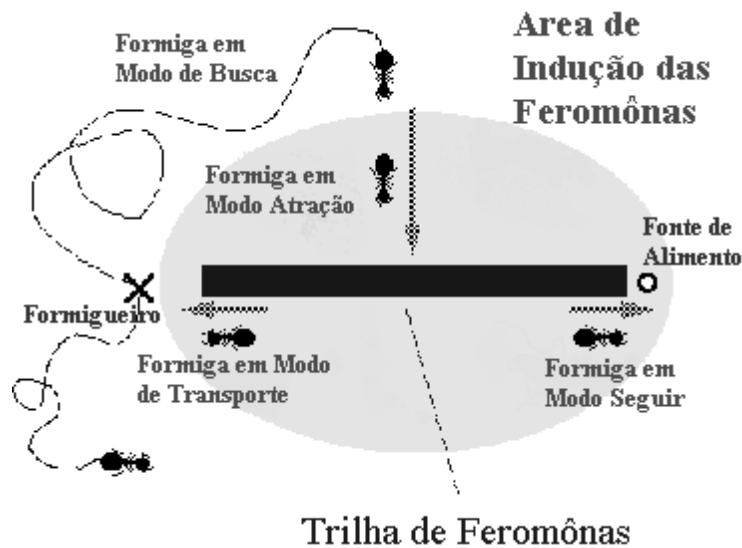


Figura 6.2: Modelo do comportamento de uma formiga real .

A trilha de feromônios se evapora e difunde, o que produz uma atmosfera de feromônios (Figura 6.3). Quando uma formiga no modo **busca**, atravessa esta atmosfera de feromônios, ela muda para o modo **atração**, no qual ela é induzida a se movimentar para posições com maior densidade de feromônios. Se a atmosfera de feromônios desaparece antes que a formiga encontre a trilha, ela volta para o modo **busca**.

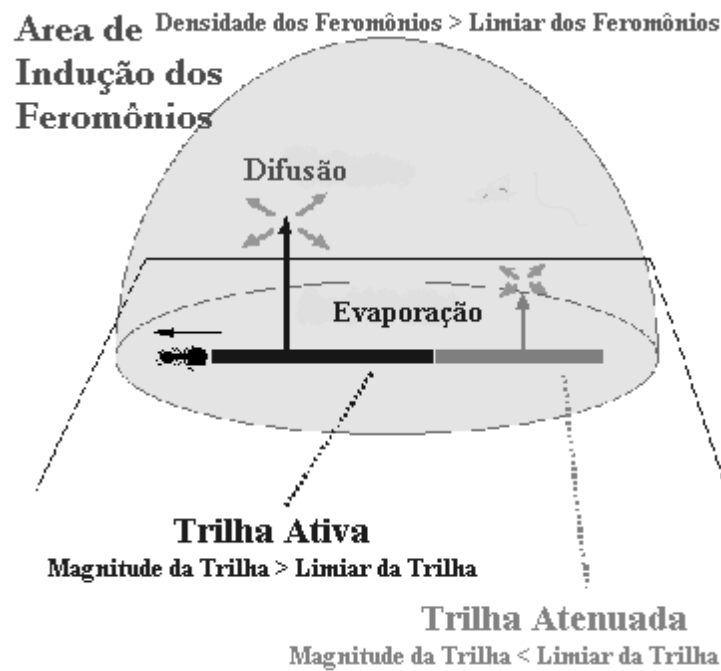


Figura 6.3: Evaporação e Difusão da Trilha de Feromônios.

Quando uma formiga no modo *busca* ou *atração* encontra uma trilha, ela muda para o modo *rastro*, no qual segue uma trilha de feromônios em sentido contrário ao seu ninho. Se a formiga em modo *rastro* não encontra alimento no final da trilha, ela volta para o modo de *busca*.

6.2 *Ant Colony System*: Um Modelo Computacional com Formigas Artificiais

Em nosso modelo, uma formiga artificial é um agente que se movimenta de um nó a outro sobre um dado grafo. Há três comportamentos provenientes da conduta natural das formigas que são transferidos à nossa colônia de formigas artificiais:

- (i) a preferência por caminhos com maior nível de feromônios;
- (ii) uma maior razão de crescimento da quantidade de feromônios sobre caminhos mais curtos;
- (iii) a trilha é o meio de comunicação entre as formigas.

A formiga artificial escolhe o próximo nó a visitar, usando uma função probabilística que depende da quantidade de feromônios acumulada sobre cada aresta e de um valor

heurístico, que será determinado em função do peso de cada aresta. Formigas artificiais, probabilisticamente, preferem nós que estão conectados por arestas com grande quantidade de feromônios e que se encontram mais próximos. Inicialmente, m formigas artificiais são colocadas sobre nós aleatoriamente escolhidos. A cada passo elas se movimentam para novos nós e modificam a trilha de feromônios sobre a aresta usada (isto é chamado de **atualização local da trilha**). Quando todas as formigas completarem um ciclo, a formiga que fez o menor ciclo modifica os valores associados às arestas de seu ciclo (isto é chamado de **atualização global da trilha**), mediante a adição de uma quantidade de feromônios sobre a trilha, a qual é inversamente proporcional ao peso total da mesma. Então, reinicia-se o processo, com os valores das arestas atualizados.

6.3 *Ant Colony System* aplicado ao TSP

Apresentamos a seguir o algoritmo de ACS na sua aplicação ao Problema do Caixeiro Viajante como descrito por Gambardella [GAMBARDELA, 97A].

Quando o ACS é aplicado ao Problema do Caixeiro Viajante, as formigas artificiais estão dotadas com algumas capacidades adicionais não existentes nas formigas reais, mas que têm-se apresentado como uma vantagem na sua aplicação ao TSP: formigas artificiais podem determinar sua distância para uma cidade e, além disso, estão providas de uma memória de trabalho usada para armazenar as cidades já visitadas.

Seja k um agente cuja tarefa é visitar todas as cidades e voltar para a cidade inicial percorrendo a menor distância possível. Associado ao agente k , existe uma lista $J_k(r)$ de cidades não visitadas (isto é equivalente a dizer que o agente k lembra de todas as cidades que ele já visitou), onde r é a cidade atual. Um agente k situado na cidade r , se movimenta para a cidade s usando a seguinte regra, chamada regra para escolher uma ação *pseudo-aleatória-proporcional* (ou regra de transição de estado), descrita a seguir:

$$s = \begin{cases} \arg \max_{u \in J_k(r)} \{ [\tau(r, s)] \cdot [\eta(r, s)]^\beta \}, & \text{se } q \leq q_0 \quad (\text{exploração}) \\ S & , \text{ caso contrario} \quad (\text{exploração}) \end{cases} \quad (6.1)$$

onde:

- $\tau(r,s)$, é um peso ≥ 0 associado ao arco (r,s) que no ACS representa os feromônios deixados pelas formigas. Durante a evolução do algoritmo estes valores vão mudando afim de tentar indicar o quão útil seria mover-se para a cidade s quando se está na cidade r .
- $\eta(r,s)$, é uma função heurística⁴ que mede a utilidade de ir para a cidade s quando se está na cidade r . Em nosso caso, $\eta(r,s)$ é a inversa da distância entre as cidades r e s .
- O parâmetro β pondera a importância relativa da função heurística. Assim, ele estabelece a sensibilidade à distância
- q é um valor escolhido aleatoriamente com probabilidade uniforme em $[0,1]$.
- q_0 ($0 \leq q_0 \leq 1$) é um parâmetro do algoritmo. Ele determina a importância relativa da exploração contra a exploração na fórmula (6.1). Um valor pequeno de q_0 permitirá uma alta probabilidade de fazer uma escolha aleatória, o que favorecerá a exploração.
- S representa a cidade escolhida aleatoriamente de acordo com a distribuição de probabilidade mostrada na fórmula (6.2), a qual expressa a probabilidade com a qual um agente na cidade r escolhe ir para a cidade s .

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)] \cdot [\eta(r,s)]^\beta}{\sum_{z \in J_k(r)} [\tau(r,z)] \cdot [\eta(r,z)]^\beta} , & \text{if } s \in J_k(r) \\ 0 , & \text{caso contrario} \end{cases} \quad (6.2)$$

Esta regra de transição de estado favorecerá as transições para os nós conectados por arcos curtos e com uma alta quantidade de feromônios. A fórmula (6.1), mostra como uma transição pode explorar o conhecimento acumulado acerca do problema (na forma de diferentes quantidades de feromônios sobre os arcos) ou explorar novos arcos (a exploração é orientada para arcos curtos com alta quantidade de feromônios).

⁴ Pode-se observar que, no ACS, o único conhecimento relacionado com o TSP é esta função que representa a inversa das distâncias entre as cidades r e s .

6.3.1 A Regra de Atualização Local

Enquanto, constróem uma solução (isto é, um *tour*) para o *TSP*, as formigas visitam os arcos e mudam sua quantidade de feromônios pela aplicação da seguinte regra de atualização local:

$$\tau(r,s) = (1-\rho) \cdot \tau(r,s) + \rho \cdot \tau_0 \quad (6.3)$$

onde, τ_0 ($\tau_0 < 1$) é um parâmetro, uma constante muito pequena⁵ que será o valor inicial dos feromônios em cada arco, e ρ ($0 < \rho < 1$) é outro parâmetro, que indica a velocidade a qual se evaporam os feromônios sobre a trilha.

6.3.2 A Regra de Atualização Global

Uma vez que cada formiga completou sua solução (visita a todas as cidades), cada um dos arcos (r,s) existentes na menor solução encontrada, terá sua quantidade de feromônio acumulada modificada pela aplicação da seguinte fórmula:

$$\tau(r,s) = (1-\alpha) \cdot \tau(r,s) + \alpha \cdot (L_{best_iter})^{-1} \quad (6.4)$$

onde, α ($0 < \alpha < 1$) é um parâmetro de algoritmo, que em certa forma indica o atrativo do alimento no final da trilha. α estabelece a sensibilidade à trilha.

A atualização global da trilha fornece uma alta quantidade de feromônios à menor solução. Isto é similar ao esquema do Aprendizado Mediante Reforços, no qual as melhores soluções obtêm um reforço elevado (como acontece, por exemplo, nos Algoritmos Genéticos).

São dois os métodos mais usados para escolher a formiga que fará a atualização global: ***iteration-best*** e ***global-best updating***⁶. No método de ***iteration-best***, o agente escolhido é aquele que faz o menor *tour* na atual iteração do algoritmo; enquanto que, no método

⁵ [GAMBARDELLA, 97A] sugere empregar $(n \cdot L_{nn})^{-1}$, onde L_{nn} é o custo do *tour* produzido pela heurística do vizinho mais próximo (ou por qualquer outra heurística) e n é o número de cidades.

⁶ Vide [GAMBARDELLA, 97A] para maior informação.

de ***global-best updating***, o agente escolhido é aquele que fez o menor *tour* desde o início do algoritmo.

O pseudocódigo do algoritmo é apresentado na figura 6.4 e está descrito a seguir:

A Fase 1, é a fase de inicialização, na qual um valor inicial τ_0 é dado para os τ -valores, e cada agente k é colocado numa cidade r_{k1} escolhida de acordo com alguma política. O conjunto $J_k(r_{k1})$ das cidades ainda não visitadas também é inicializado.

Na Fase 2, é executado um *loop*, no qual, a cada passo, cada um dos m agentes faz um movimento e os valores $\tau(r,s)$ são atualizados segundo a fórmula 6.3; o *loop* é repetido até que cada agente tenha completado um *tour* e voltado a sua cidade de origem.

Na Fase 3, é determinada a longitude do menor *tour* feito por um dos agentes, L_{best} . Esta é usada para calcular os *delayed reinforcement* $(L_{best})^{-1}$. Então, os valores de $\tau(r,s)$ são novamente atualizados através da fórmula 6.4.

Finalmente, a Fase 4, verifica se a condição de parada foi satisfeita, e se este não for o caso, o algoritmo volta para a Fase 2.

Usualmente, a condição de parada é alcançada após um número de ciclos prefixado, ou quando não se obtém melhoras após um dado número de ciclos.

Algoritmo:**FASE 1:** Fase de Inicialização

Para cada $\text{par}(r,s)$, fazer $\tau(r,s) := \tau_0$ Fim-para
Para ($k := 1$ até m), fazer
 Seja r_{k1} a cidade de início para o agente k
 /* $J_k(r_{k1})$ é o conjunto de cidades a ser visitadas pelo agente k desde a cidade r_{k1} */
 $J_k(r_{k1}) := \{1, \dots, n\} - r_{k1}$
 $r_k := r_{k1}$ /* r_k é a cidade onde o agente k está localizado */
Fim-para

FASE 2: Esta é a fase, na qual o agente constrói seu *tour*. O *tour* do agente k é armazenado em $Tour_k$

Para ($i := 1$ até N), fazer
 Se ($i < N$), então
 Para ($k := 1$ até m), fazer
 Escolha uma nova cidade s_k de acordo com as fórmulas (6.1) e (6.2)
 Se ($i < N-1$), então $J_k(s_k) := J_k(r_k) - s_k$ fim-se
 Se ($i = N-1$), então $J_k(s_k) := J_k(r_k) - s_k + r_{k1}$ fim-se
 $Tour_k(i) := (r_k, s_k)$
 Fim-para
 senão
 /* Neste ciclo todos os agentes regressam a sua cidade inicial r_{k1} */
 Para ($k := 1$ até m), fazer
 $s_k := r_{k1}$
 $Tour_k(i) := (r_k, s_k)$
 Fim-para
 Fim-se
 /* Em esta fase é feita a atualização local da trilha de feromônios e os τ -valores são atualizados usando a fórmula (6.3) */
 Para ($k := 1$ até m), fazer
 $\tau(r_k, s_k) := (1-\rho)\tau(r_k, s_k) + \rho\tau_0$
 $r_{kl} := s_k$ /* Nova cidade para o agente */
 Fim-para

Fim-para**FASE 3:** Nesta fase os *delayed reinforcement* e os τ -valores são atualizados

Para ($k := 1$ até m), fazer
 Calcular L_k /* L_k é a longitude do *tour* feito pelo agente k */
 Fim-para
 Calcular o melhor $L_{\text{best-iter}}$
 /* Atualizar as arestas sob o $L_{\text{best-iter}}$ usando a fórmula (4) */
 Para cada aresta (r,s) , fazer
 $\tau(r_k, s_k) := (1-\alpha)\tau(r_k, s_k) + \alpha(L_{\text{best-iter}})^{-1}$
 Fim-para

FASE 4:

Se ($\text{End_condition} = \text{TRUE}$), então
 Imprimir (menor L_k)
 senão
 Voltar para a FASE 2.
 Fim-se

Figura 6.4: O Algoritmo de Colônia de Formigas para o TSP.

6.4 *Ant Colony System* aplicado ao *CTSP*

Como dito no capítulo 3, poderíamos usar a transformação proposta por Chisman para transformar o *CTSP* no *TSP* e assim, aplicar o *Ant Colony System* proposto por Gambardella e Dorigo [GAMBARDELLA, 97A] para resolvê-lo. Porém, em nosso trabalho, decidimos usar um caminho diferente, pois uma dificuldade que o ACS apresenta é que, aplicar esta técnica a problemas consistentes na composição de dois ou mais problemas interdependentes é uma tarefa bastante difícil e o *CTSP* é um exemplo deste tipo de problema, pois determinar a ordem de visita dentro de cada grupamento e determinar a ordem de visita entre grupamentos são problemas interdependentes.

Como já foi dito, nosso interesse é desenvolver uma forma de combinar um algoritmo genético com uma colônia de formigas artificiais. Para isso modificamos o ACS de Gambardella e Dorigo para o *TSP*, de modo a poder usá-lo em combinação com um algoritmo genético afim de resolver o *CTSP*.

A seguir descreveremos as modificações que foram feitas no ACS de Gambardella e Dorigo. As demais características do *Ant Colony System* aplicado ao *TSP* foram mantidas em nossa adaptação para o *CTSP*. Convém mencionar que utilizamos o método ***iteration best*** para realizar a Atualização Global da trilha de feromônios.

6.4.1 Modificações nos parâmetros do Algoritmo

No ACS descrito na seção anterior τ_0 é um parâmetro do algoritmo que depende da instância que está sendo resolvida. Na verdade, τ_0 costuma ser uma aproximação grossa à solução da instância do problema. Em nossa modificação, ao invés do algoritmo receber este parâmetro, são passadas para ele, uma ou mais soluções⁷ para a instância que está sendo resolvida. O valor de τ_0 é obtido, calculando-se a média dos valores destas soluções.

⁷Na metaheurística desenvolvida neste trabalho, as soluções recebidas pelo ACS são as melhores soluções obtidas pelo algoritmo genético executado previamente.

6.4.2 Inicialização da Matriz de Feromônios

Como no algoritmo descrito na seção 6.3, cada elemento da matriz de feromônios é inicializado com o valor de τ_0 . Após isto, com o intuito de aproveitar o conhecimento presente nas soluções recebidas como parâmetro, nós realizamos uma atualização global (como a descrita anteriormente) sobre cada trilha representada por cada solução.

6.4.3 Adaptação na memória

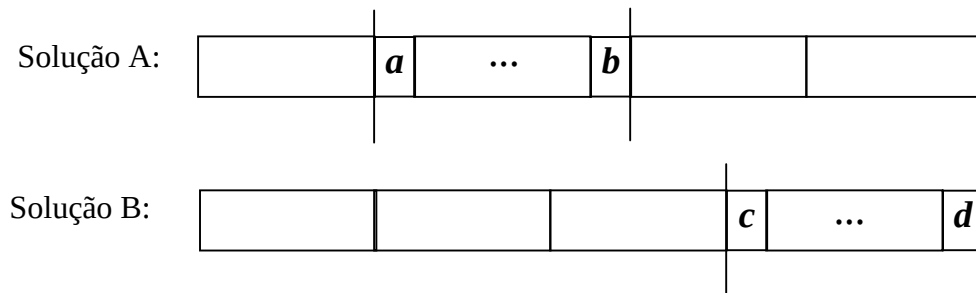
A memória de cada formiga guarda as cidades que ainda não foram visitadas. Ela foi implementada através de uma lista, que inicialmente contém todas as cidades, divididas em grupos correspondendo aos grupamentos do *CTSP*. Conforme uma cidade é visitada ela é retirada desta lista.

6.4.4 Escolha da Cidade Inicial

Um dos principais obstáculos no *CTSP* para o uso de uma colônia de formigas para resolve-o, é a determinação da cidade na qual uma formiga deve começar seu tour. Isto, é pelo fato, de que não se conhece *a priori* quais são as cidades de entrada e saída de cada grupamento. Devido a que a formiga deve visitar todas as cidades de todos os grupamentos, a cidade de início do tour pode estar em qualquer grupamento e devido a que, as cidades dentro de cada grupamento devem ser visitadas contiguamente, a cidade de início do tour pode ser uma das cidades de entrada (ou saída) de qualquer grupamento. Por esta razão, em nosso algoritmo, cada formiga começa seu *tour* numa cidade do grupamento com o menor número de cidades. Para escolher a cidade inicial, cria-se uma lista onde são incluídas somente uma das cidades extremo⁸ do menor grupamento em cada solução. Então, cada formiga escolhe aleatoriamente uma destas cidades para iniciar todos os seus *tours*. Um exemplo de construção de uma lista de cidades candidatas à cidade inicial está ilustrado na figura 6.5. Note que, para que o algoritmo encontre uma solução ótima, pelo menos uma das cidades contidas na lista de

⁸ Numa solução qualquer, uma cidade extremo é uma cidade de entrada ou da saída de um grupamento.

cidades candidatas à cidade inicial, deve ser uma cidade extremo para o menor grupamento numa solução ótima.



Suponhamos que a, b, c e d sejam as cidades extremo no menor grupamento dadas pelas soluções A e B passadas ao algoritmo. Então, a lista de cidades candidatas à cidade inicial poderia ser uma das seguintes: $\{a, c\}$, $\{a, d\}$, $\{b, c\}$ e $\{b, d\}$

Figura 6.5: A construção de uma lista de cidades candidatas à cidade inicial.

6.4.5 Escolha da próxima cidade

A escolha da próxima cidade a visitar é um tanto diferente devido a existência dos grupamentos. Em nosso caso, existem duas situações possíveis para a escolha da próxima cidade a ser visitada. Se no grupamento no qual a formiga se encontra, ainda houverem cidades por visitar, ela deve escolher uma cidade no mesmo grupamento. Em caso contrário, a próxima cidade a ser visitada deve ser de um outro grupamento, ainda não visitado. As próximas seções descrevem a maneira como a próxima cidade é escolhida em cada uma destas situações.

6.4.5.1 Escolha da próxima cidade dentro de um mesmo grupamento

Nesta situação, a escolha da próxima cidade é feita usando-se a regra definida pela equação (6.1), porém, neste caso, $J_k(r)$ é a lista de cidades não visitadas do grupamento atual.

6.4.5.2 Escolha da próxima cidade em um outro grupamento

Quando uma formiga tiver visitado todas as cidades de um grupamento, ela escolherá, se houver, um novo grupamento dentre aqueles ainda não visitados. Esta escolha também é feita segundo a regra definida na equação (6.1), mas $J_k(r)$ é uma lista de cidades não visitadas especial, construída da seguinte maneira:

Na fase de inicialização do ACS é construída para cada grupamento, uma lista de grupamentos adjacentes, a partir das seqüências de grupamentos obtidas nas soluções iniciais recebidas pelo algoritmo. Um exemplo da construção destas listas está ilustrado na figura 6.6.

Solução A:	g_1	g_2	g_3	g_4	g_5
Solução B:	g_1	g_4	g_5	g_3	g_2

Sejam g_1, g_2, g_3, g_4 e g_5 os cinco grupamentos de cidades. A partir das soluções A e B, são construídas as seguintes listas de grupamentos adjacentes para cada grupamento:

$$\begin{aligned}
 g_1 &\rightarrow g_2, g_4, g_5 \\
 g_2 &\rightarrow g_1, g_3 \\
 g_3 &\rightarrow g_2, g_4, g_5 \\
 g_4 &\rightarrow g_1, g_3, g_5 \\
 g_5 &\rightarrow g_1, g_3, g_4
 \end{aligned}$$

Figura 6.6: A construção de listas de grupamentos adjacentes.

A partir da lista de grupamentos adjacentes ao grupamento atual escolhem-se os grupamentos que ainda não foram visitados. $J_k(r)$ será agora, a lista composta pelas cidades extremo destes grupamentos. Esta forma de proceder, criando listas de adjacencias, para a escolha do proximo grupamento a visitar obedece ao mesmo criterio empregado pelo operador genético ERX; o qual será descrito no capítulo 8.

Note que as soluções recebidas como parâmetro devem ser tais, que as permutações dos grupamentos permitam a obtenção da solução ótima ou de uma boa solução. Isto se

deve ao fato de que as listas de grupamentos adjacentes são construídas a partir de tais permutações e são utilizadas ao longo de todo o algoritmo, sem sofrer modificações, limitando assim, a variedade de permutações que podem ser obtidas, à vizinhança definida por estas.

Capítulo 7

Uma Introdução aos Algoritmos Genéticos

Inicialmente propostos pelo Professor John Holland [HOLLAND, 75] da Universidade de Michigan, os Algoritmos Genéticos (AGs) são métodos heurísticos adaptativos que podem ser utilizados para resolver problemas de busca e otimização. A idéia inicial de Holland foi tentar imitar algumas etapas do processo de evolução natural das espécies incorporando-as a um algoritmo computacional.

Na natureza, os indivíduos competem entre si por recursos tais como: comida, água e moradia. Adicionalmente, os animais da mesma espécie normalmente antagonizam-se para obter um parceiro (parceira). Aqueles indivíduos que têm mais êxito terão, provavelmente, um número maior de descendentes e portanto, maiores probabilidades de que seus genes sejam propagados ao longo de sucessivas gerações. A combinação de informações de pais bem adaptados, e a propagação destas informações num de seus descendente, pode produzir muitas vezes um novo indivíduo melhor adaptado às características do seu meio do que qualquer de seus antepassados.

Os processos genéticos dos organismos biológicos foram estudados por Charles Darwin o que o levou a formular os princípios de “seleção natural” e “sobrevivência do mais apto”, estabelecidos pela primeira vez em seu livro "Origem das Espécies". Emulando os processos definidos por estes princípios, os AGs são capazes de apresentar soluções para problemas do mundo real.

Os AGs trabalham a partir de um conjunto (**população**) de representações de uma solução (ou **indivíduos** denominados **cromossomos**), onde cada representação é algum tipo de **codificação** (normalmente uma cadeia de bits ou de caracteres) adequada para o dado problema. Para cada indivíduo atribui-se uma pontuação de adaptação, dependendo do quão boa é a sua resposta ao problema. Aos mais adaptados é dada a

oportunidade de se reproduzirem mediante *cruzamentos* com outros indivíduos da população, produzindo descendentes com características de ambos os pais. Sendo assim, os membros menos adaptados possuem poucas chances de serem selecionados para a reprodução e tendem a desaparecer.

Uma nova população de possíveis soluções é gerada a partir da seleção dos melhores indivíduos da geração atual e o cruzamento entre eles. Esta nova geração contém uma proporção mais alta das características possuídas pelos melhores membros da geração anterior. Desta forma, através de várias gerações, as características boas tendem a ser propagadas sobre toda a população, misturando-se com outras.

Favorecendo o cruzamento dos indivíduos melhor adaptados, é possível percorrer as áreas mais promissoras do espaço de busca. Se o algoritmo genético tiver sido modelado corretamente, a população terá mais chances de convergir a uma solução ótima para o problema analisado.

7.1 O Modelo dos Algoritmo genéticos

Em geral, todo o processo de geração de soluções de um Algoritmo Genético pode ser estruturado em duas etapas, como mostrado na figura 7.1.

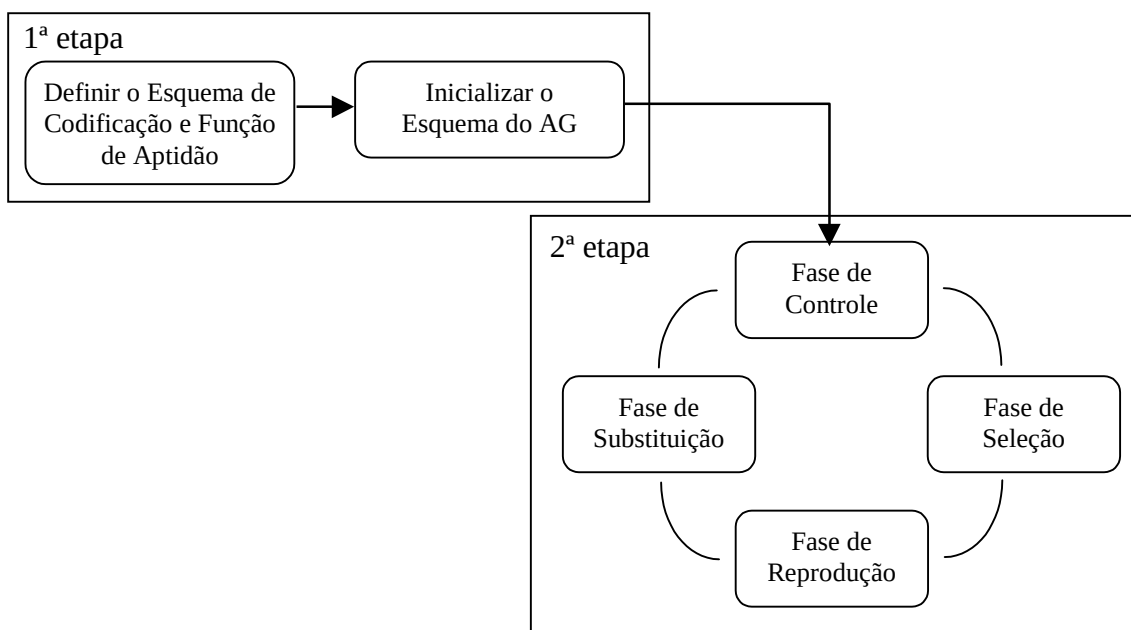


Figura 7.1: A estrutura de um algoritmo genético.

Na primeira etapa, a qual é estática, um esquema de codificação e uma função de aptidão apropriada capturam o objetivo principal do algoritmo. Também, nesta etapa, são definidas algumas restrições para o tipo de problema dado. Além disso, os parâmetros estáticos do algoritmo são inicializados, como por exemplo: tamanho da população, estrutura e esquema de comunicação. A especificação dos operadores e estratégias a serem aplicadas na etapa dinâmica também são definidos neste momento.

A etapa dinâmica, está dividida em quatro fases as quais são iterativamente aplicadas para produzir novas populações e simular o processo de evolução natural, até que um dado critério de parada seja atendido.

- ◆ **Fase de Seleção:** Nesta fase são selecionados para a reprodução, indivíduos da atual geração.
- ◆ **Fase de Reprodução:** Na fase de reprodução de um AG, dois ou mais indivíduos selecionados como pais (cromossomos pais) são combinados para a geração de novas populações de indivíduos (cromossomos filhos), através de processos de **mutação** e/ou **recombinação**. Os procedimentos que implementam estes processos são denominados "**operadores genéticos**".
- ◆ **Fase de Substituição:** Os novos indivíduos são avaliados segundo uma dada função de aptidão. A partir dos valores desta função, decide-se quais destes filhos serão inseridos na nova população e quais dos antigos indivíduos serão excluídos.
- ◆ **Fase de Controle:** Nesta fase, são realizadas atualizações requeridas pelo algoritmo e se verifica se o critério de parada foi satisfeito.

Cada iteração da etapa dinâmica é conhecida como geração. A primeira geração deste processo é uma população de indivíduos gerados aleatoriamente. A partir daí, os

operadores genéticos em união com a medida de adaptação, atuam para melhorar a população.

Na prática, pode-se implementar este modelo utilizando matrizes de bits ou caracteres para representar os cromossomos. Operações simples de bits permitem efetuar o cruzamento, a mutação e outras operações. Embora, uma grande quantidade de pesquisa tenha sido feita em cadeias de tamanho variável e em outras estruturas, a maior parte do trabalho com AGs tem sido realizada em cadeias de tamanho fixo. Por isso enfatiza-se este aspecto e a necessidade de codificar a solução como uma cadeia de caracteres.

A tabela da figura 7.2 faz um mapeamento entre os conceitos de evolução natural e sua equivalência no modelo de Algoritmos Genéticos.

Evolução Natural	Algoritmos Genéticos
Indivíduo	Uma solução para o problema
População	Conjunto de soluções
Aptidão	Qualidade de uma solução
Cromossomo	Codificação de uma solução
Gene	Parte de uma codificação de uma solução
Recombinação/Mutação	Operadores de busca
Seleção Natural	Reuso de boas (sub-)soluções

Figura 7.2: Comparação entre conceitos da Evolução Natural e o modelo de Algoritmos Genéticos.

7.2 Princípios Básicos

7.2.1 A Codificação

As características do AG que o relacionam com um dado problema são a codificação e a função de aptidão.

Se a solução de um problema pode ser representada por um conjunto de valores (conhecidos como *genes*), estes podem ser unidos para formar uma cadeia de valores

ainda maior (**cromossomo**). Este processo é chamado de **codificação**. Por exemplo, se temos o problema de maximizar uma função de três variáveis, $F(x,y,z)$, poderia-se representar cada variável por um número binário de 10 bits, obtendo-se um cromossomo de 30 bits de comprimento e três genes.

Em genética este conjunto representado por um cromossomo em particular é conhecido como **genótipo**, que contem a informação necessária para construir um organismo, conhecido como **fenótipo**. Estes mesmos termos aplicam-se a AGs; por exemplo, se queremos projetar uma ponte, o conjunto de parâmetros especificados no projeto é o genótipo e a construção final é o fenótipo. A adaptação de cada indivíduo depende de seu fenótipo, o qual pode ser inferido de seu genótipo, isto é, pode ser calculado a partir do cromossomo, utilizando-se a função de aptidão.

Existem vários aspectos relacionados com a codificação de um problema:

- ◆ Deve-se utilizar o menor alfabeto possível para representar os parâmetros; normalmente se utilizam dígitos binários, isto é, 0 e 1.
- ◆ As variáveis que representam os parâmetros do problema devem ser discretizadas para que possam ser representadas como cadeias de bits. Deve ser utilizada resolução suficiente para assegurar que a saída tem um nível de precisão adequado. Assume-se que a discretização é representativa da função objetivo.
- ◆ A maior parte dos problemas tratados com AGs são não lineares e muitas vezes existem dependências entre as variáveis que compõem a solução. Esta interação é denominada "**epistasia**" e é necessário considerá-la, para representar o problema adequadamente.
- ◆ O tratamento dos **genótipos** inválidos deve ser considerado para o desenvolvimento da codificação. Por exemplo, suponhamos que

sejam necessários 1200 valores para representar uma variável, isto requer pelo menos 11 bits, mas estes codificam um total de 2048 possibilidades, "sobrando" 848 seqüências de bits não necessárias. A estas seqüências pode ser dado o valor de adaptação zero, ou um valor real, ou ainda, os cromossomos inválidos podem ser eliminados.

7.2.2 A Função de Aptidão

Dado um cromossomo, a função de aptidão procura atribuir-lhe um valor numérico de adaptação, o qual deve ser proporcional à "utilidade" ou "habilidade" do indivíduo representado. A função de aptidão deve ser tal, que, através de sucessivas amostras obtidas ao longo do processo evolutivo, forneça indicações sobre a localização do ótimo. Por exemplo, uma função de aptidão que retorna um valor unitário para a única solução do problema (como a combinação que abre um cofre) e zero para qualquer outra tentativa é incapaz de guiar a busca. A busca se torna completamente cega e todos os algoritmos são equivalentes a uma busca aleatória: encontrar a solução é uma questão de sorte. Adicionalmente, a função de aptidão deve ser rápida, pois será aplicada para cada indivíduo da população nas sucessivas gerações, e portanto, uma grande parte do tempo de execução do algoritmo genético será empregado para a execução desta função.

A função de aptidão é a responsável pelo processo de seleção dos indivíduos e deve indicar a qualidade de cada indivíduo (cromossomo) na população. Para o caso de otimização, ela está intimamente ligada à função objetivo que se deseja minimizar ou maximizar. No caso de maximização, por exemplo, podemos obter a função de aptidão a partir da função objetivo $g(x)$, da seguinte forma:

$$f(x) = g(x) + C$$

onde C é uma constante introduzida nos casos em que o processo de seleção exigir que a aptidão seja sempre positiva. À medida que a população evolui, os indivíduos passam a ter aptidões cada vez mais semelhantes e pode ser necessário aumentar a *pressão de*

seleção adotando-se uma nova função de aptidão, a partir da função objetivo, da seguinte forma:

$$f(x) = h(g(x))$$

com uma função h conveniente. Fica claro que não há necessidade de se incluir derivadas da função objetivo e, conseqüentemente, um AG pode ser utilizado também – sem modificação alguma – para otimizar funções não-diferenciáveis.

Em alguns casos a aptidão pode ter um caráter dinâmico (que varia ao longo do processo evolutivo), estocástico, impreciso e até mesmo subjetivo, como, por exemplo, no caso de algumas aplicações artísticas envolvendo pintura ou música.

7.3 As fases de um Algoritmo Genético

7.3.1 A Seleção

Depois de ter sido calculada a aptidão de cada indivíduo; a seleção é o processo pelo qual indivíduos (isto é, cromossomos) são escolhidos para se reproduzirem. Embora existam diversos métodos para escolher os pais, tem-se como regra favorecer aos indivíduos melhor adaptados, atribuindo-lhes uma probabilidade alta de serem escolhidos, a fim de emular a seleção natural. Após escolhidos os pais, seus cromossomos sofrem uma combinação e mudam.

Os métodos mais usados para a seleção de pais são¹:

1. Seleção Proporcional à Aptidão,
2. Seleção por Mérito,
3. Seleção por Competição,
4. Seleção por Elitismo.

Estes métodos serão descritos a seguir.

¹ Vide [BLICKLE, 95] para ver um estudo comparativo entre estes métodos.

7.3.1.1 Seleção Proporcional à Aptidão (*Proportional Selection*)

Este é o esquema de seleção usado com maior frequência nos AGs, também é conhecido como o Método da Roleta.

Seja:

f_i = Aptidão do indivíduo i .

$\bar{f} = \frac{1}{N} \sum_i f_i$ = a aptidão média da população, onde N é número de indivíduos.

Então, na Seleção Proporcional à Aptidão, o indivíduo i é selecionado para reproduzir-se com probabilidade

$$p_i = \frac{f_i}{\sum_i f_i} = \frac{f_i}{N \bar{f}}$$

As principais dificuldades deste método são a convergência prematura e a *estagnação*.

7.3.1.2 Seleção por Mérito (*Rank Selection*)

Neste método, os indivíduos são ordenados em ordem crescente de aptidão, isto é: $f_i > f_j$ para $i < j$. Então, para cada indivíduo é atribuída uma probabilidade p_i de seleção, a partir de uma dada distribuição de probabilidade, com a restrição de que $\sum_i p_i = 1$.

As distribuições típicas utilizadas neste esquema são:

- Linear: $p_i = a \times i + b$ (onde a e $b \in \mathbb{R}$ e $a < 0$)
- Exponencial Negativa: $p_i = a \exp(b \times i + c)$ (onde a, b e $c \in \mathbb{R}$)

As vantagens deste método consistem no fato de que ele pode evitar uma convergência prematura e a estagnação, além de não requerer uma aptidão explícita, isto é, para ordenar os indivíduos, só é necessária a habilidade para comparar duas soluções.

Suas desvantagens consistem no fato de ser necessário reordenar a população a cada iteração, e na dificuldade de sua análise teórica.

7.3.1.3 Seleção por Competição (*Tournament Selection*)

Para selecionar um indivíduo, é criado um grupo de N ($N \geq 2$) indivíduos escolhidos aleatoriamente. Então, o indivíduo com a maior aptidão no grupo é escolhido e os outros são descartados.

As vantagens deste método são: menor risco de convergência prematura e estagnação, não é necessário reordenar a população e não há a necessidade de uma aptidão explícita.

7.3.1.4 Seleção por Elitismo (*Elitist Selection*)

Ao menos uma cópia do melhor indivíduo da população é sempre passada para a próxima geração. A vantagem deste método, é o fato, de que a convergência é garantida, isto é, se um extremo global é descoberto, o algoritmo converge para ele. Uma desvantagem é o risco de que o algoritmo seja “enganado” por um extremo local, mas não global. Uma forma de superar esta desvantagem, é conservar um registro da melhor solução descoberta até o momento; e, no final do algoritmo, apresentar esta solução como resultado, ao invés da melhor da última geração.

7.3.2 A Reprodução

Durante a fase de reprodução de um algoritmo genético, os indivíduos escolhidos na fase de Seleção são combinados para formar descendentes para a próxima geração. O objetivo básico do processo de reprodução, é gerar novos cromossomos (cromossomos filhos) que possuam propriedades genéticas superiores às encontradas nos pais. Isto é feito através de processos de mutação e/ou recombinação que serão descritos nas próximas seções. Os procedimentos que implementam estes processos são denominados

"operadores genéticos". Duas propriedades são consideradas bastante desejáveis em um operador genético:

1. Quando os pais possuem uma característica comum, seus descendentes sempre devem herdar esta característica.
2. Quando os pais possuem características diferentes, deve ser possível que seus descendentes as herdem.

7.3.2.1 A Recombinação

Como o próprio nome indica, os operadores de recombinação visam promover a recombinação do material genético de pelo menos dois “pais” para a geração de um ou mais “filhos” que serão avaliados posteriormente. Tais operadores são também conhecidos na literatura como operadores de *crossover*, termo também oriundo da Genética.

Embora existam muitos operadores genéticos para a recombinação, um dos mais utilizados é o operador de *crossover* clássico. Neste tipo de cruzamento, dois cromossomos são escolhidos e são submetidos à operação de recombinação. Uma posição de cruzamento é sorteada e o material genético dos “pais” é recombinado conforme o esquema da figura 7.3, onde os “pais” P1 e P2 dão origem aos “filhos” F1 e F2 com a posição de cruzamento estando entre a quarta e a quinta posições dos cromossomos P1 e P2.

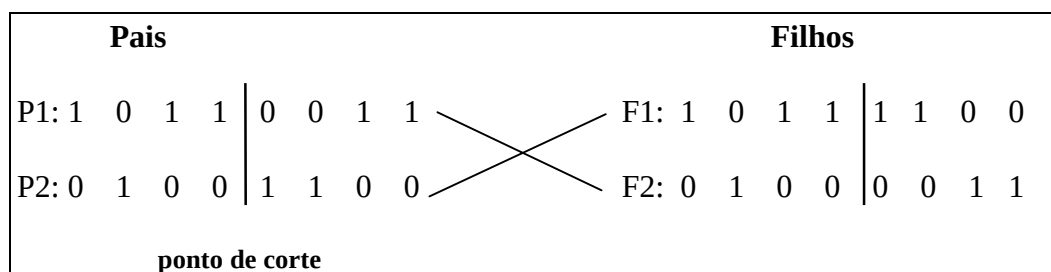


Figura 7.3: Reprodução através do Operador *crossover* clássico.

Alguns dos operadores de recombinação podem gerar soluções inválidas (alguns com menor frequência que outros). Soluções não-viáveis podem ser, eventualmente, úteis para manter e gerar diversidade na população, ainda que, isto possa ocasionar lentidão ou evitar que o algoritmo convirja. Uma possível estratégia para melhorar os resultados de qualquer operador de recombinação é a idéia de *reacasalamento* de filhos não-viáveis.

Um operador de recombinação muito utilizado em AGs construídos para solucionar o TSP é o operador *ERX* [WHITLEY, 89]. Este operador teve grande utilidade no desenvolvimento deste trabalho e será descrito no capítulo 8, que apresenta o algoritmo genético proposto e uma aplicação do mesmo ao CTSP.

7.3.2.2 A Mutação

A idéia de mutação também é oriunda da Genética e visa preservar alguma diversidade genética na população que vai se reduzindo à medida que a evolução avança e os indivíduos vão se tornando mais e mais semelhantes tendendo a uma solução que, nos casos de sucesso do AG, é uma boa solução para o problema. Além disso, a mutação proporciona uma forma de "exploração global" do espaço de busca, pois ela gera novos indivíduos, a serem avaliados, que podem revelar novas regiões promissoras no espaço de busca.

Normalmente, são introduzidos em um AG, um ou mais operadores de mutação, os quais são operadores unários que, operando sobre um dado elemento da população, produzem uma modificação que dá origem a um novo indivíduo que será avaliado posteriormente. Este tipo de operação é efetuada de maneira que a probabilidade p_m de ocorrência de mutação de um dado gene (cromossomo) seja geralmente pequena.

Um exemplo de mutação encontra-se na figura 7.4. Uma mudança é feita em um gene de C1 para gerar um novo cromossomo C2.

7.4 Aspectos Práticos

7.4.1 Definição dos parâmetros nos Algoritmos Genéticos

Geralmente, existem vários parâmetros a serem definidos para um AG. A maioria deles são específicos para o problema que está sendo tratado e não serão abordados nesta seção. Porém, dois parâmetros merecem especial atenção, pois estão presentes na maioria dos AGs: o tamanho da população de indivíduos e o número de iterações do algoritmo.

Testes experimentais mostram que quanto maior for a população, melhores serão os resultados alcançados. Mas, por outro lado, quanto maior a população, maior será também o tempo computacional exigido para cada iteração. O número de iterações também está ligado a este dilema: melhora da solução implica em maior consumo de tempo. Na prática, tem se usado o bom senso e o conhecimento prévio do problema para o controle dos parâmetros nos AGs.

7.4.2 Inicialização – População inicial

O processo de inicialização da população é muitas vezes realizado aleatoriamente utilizando-se um gerador de números aleatórios. A população de um AG é um conjunto de soluções viáveis (representadas na forma de cromossomos) para um determinado problema. O procedimento de gerar uma população inicial para um AG é muito simples para a maioria dos problemas. Normalmente, utilizam-se também, além dos processos aleatórios, algoritmos heurísticos simples para este fim. Por exemplo, pode se introduzir na população inicial um ou mais indivíduos “interessantes”, com soluções aproximadas conhecidas ou contendo algum tipo de informação prévia.

Cada execução de um AG pode ser replicada no computador, bastando para isto, fornecer a mesma “semente” para o algoritmo gerador de números aleatórios. Um valor diferente da semente dá origem a uma outra seqüência de números aleatórios e, conseqüentemente, uma nova população inicial. Para obter-se uma previsão do

comportamento de um AG em uma dada aplicação, é comum computar-se estatísticas, para um conjunto de resultados correspondentes a diferentes valores da semente.

7.4.3 Estratégias de Evolução

Na maioria dos casos, os algoritmos genéticos são inicializados com uma população gerada aleatoriamente; isto, porque é difícil determinar a priori quais as características (genes) boas e quais as características ruins. Durante o processo de evolução, muitos indivíduos são eliminados da população e isto faz com que as características que eles possuíam sejam perdidas durante o processo evolutivo, as quais, combinadas a outras poderiam talvez, produzir indivíduos ainda melhores. Isto, apresenta um dilema: convergência da população *versus* a necessidade de manter a diversidade de genes, pois conforme a convergência, os indivíduos começam a possuir as mesmas características. Após a convergência da população, esta geralmente estará composta por cópias de um mesmo indivíduo ou por indivíduos a partir dos quais não podem ser obtidos indivíduos melhores, neste instante o algoritmo terá encontrado um ponto extremo. As estratégias de evolução [BÄCK, 96] [SCHWEFEL, 99], são critérios para tentar enfrentar este problema. Elas foram inicialmente desenvolvidas por Rechenberg e por Schwefel em meados da década de 60 [RECHENBERG, 65] [SCHWEFEL, 65]. Adotar uma estratégia significa definir claramente como serão empregados os operadores de recombinação, de mutação e como será feita a seleção dos indivíduos que sobreviverão geração após geração, assim como a seleção dos indivíduos que participarão de cada processo de reprodução.

7.4.4 Inserção de Genes na População

Para tentar continuar com a busca após a população ter convergido, insere-se genes na população. Isto pode ser feito de muitas formas, por exemplo, [YURET, 96] apresentou uma metaheurística baseada nos algoritmos genéticos, na qual são selecionados os melhores cromossomos de cada geração e utilizados como entrada de uma heurística que gera um novo cromossomo o mais afastado possível dos melhores; com isto, ele visa explorar regiões ainda não exploradas e esgotar a maior quantidade de ótimos locais dentro do espaço de soluções. Pode-se, também, gerar uma nova população totalmente aleatória e recomençar todo o processo novamente, pode-se realizar mudanças nos genes de alguns indivíduos bons e criar uma nova população com isto, etc. Na

maioria dos casos, a geração dos genes a ser inseridos na população depende da direção que queremos dar à busca após a população ter convergido. Pode-se querer concentrar a busca numa região promissora, redirecionar a busca para regiões novas ou apenas recuperar o material genético perdido. Isto está intimamente ligado ao problema que está sendo tratado, ao conhecimento que se tem sobre ele e a estratégia de busca adotada.

7.5 AGs e outros métodos

Os AGs possuem quatro diferenças principais, se comparados com a maioria dos métodos de otimização e busca mais utilizados ou conhecidos:

- Trabalham sobre uma codificação das possíveis soluções (cromossomos) e não sobre as soluções propriamente ditas;
- Empregam uma população de soluções (cromossomos) a cada iteração e não uma solução a cada iteração;
- Empregam regras de transição probabilísticas, não determinísticas;
- Utilizam apenas a função objetivo (aptidão), não requerendo informações adicionais (derivadas, por exemplo) sobre a função a otimizar.

O potencial dos AGs provem do fato de que esta é uma técnica robusta, que pode lidar com êxito com uma ampla variedade de problemas, incluindo alguns que são difíceis de resolver por outros métodos. Os AGs não garantem que encontrarão a solução ótima do problema, mas geralmente encontram boas soluções em um intervalo de tempo suportável.

Capítulo 8

O Algoritmo Genético Proposto

Tradicionalmente, um dos principais aspectos no desenvolvimento de um algoritmo genético é a definição dos critérios de seleção, tanto dos indivíduos que serão os pais da próxima geração quanto daqueles que serão substituídos pelos filhos, durante o processo evolutivo da população. Entretanto, um problema é frequentemente enfrentado:

Durante o processo de evolução, muitos indivíduos são eliminados da população e isto faz com que as características que eles possuíam sejam perdidas durante o processo evolutivo, as quais, combinadas a outras poderiam talvez, produzir indivíduos ainda melhores. Por esta razão, torna-se necessário encontrar uma maneira de gerar novos filhos a partir dos melhores indivíduos, porém, sem eliminar características boas de indivíduos menos adaptados.

Para resolver este problema, o algoritmo proposto apresenta uma nova forma de administrar a população de indivíduos para a seleção de cromossomos, tanto para serem pais da próxima geração, quanto para serem substituídos pelos novos indivíduos. A solução proposta em nosso trabalho, consiste no particionamento da população. Partindo-se da suposição de que cromossomos que possuem uma maior quantidade de genes idênticos terão um valor objetivo mais próximo, espera-se que ao particionar-se a população de acordo com os valores objetivo dos indivíduos, cromossomos de partições diferentes possuam maior quantidade de genes diferentes do que cromossomos que pertencem a uma mesma partição. De fato, dois cromossomos com valores objetivo diferentes terão pelo menos um gene diferente.

As partições são utilizadas para que seja possível selecionar indivíduos que não teriam chances de se reproduzir em um algoritmo genético tradicional. Ao invés de selecionarmos somente os melhores indivíduos para a reprodução, selecionamos um

número pré-fixado de pais de cada partição. Da mesma forma, os filhos gerados são inseridos ordenadamente na população, podendo ser colocados em partições distintas.

Permitindo a reprodução de indivíduos com características inferiores, evitamos que seu material genético se perca e permitimos que combinados, possam originar indivíduos ainda melhores que os melhores da geração anterior.

Neste capítulo descrevemos o algoritmo genético proposto neste trabalho e aplicação deste algoritmo ao CTSP. Os passos deste algoritmo são ilustrados na figura 8.1 e serão descritos nas próximas seções.

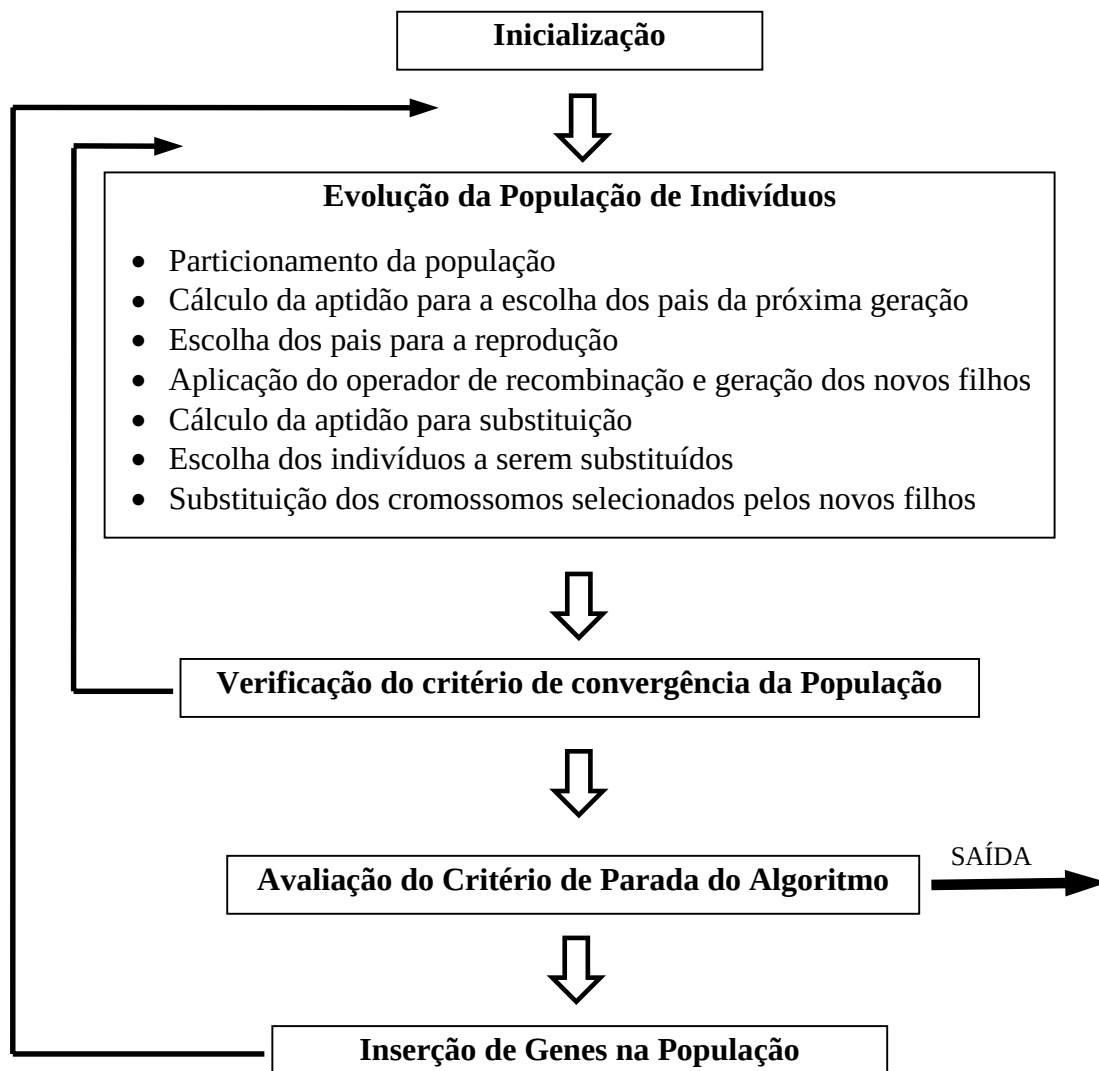


Figura 8.1: Estrutura básica do algoritmo genético proposto.

8.1 Descrição do Algoritmo Genético Proposto

A seguir serão descritos cada um dos passos do algoritmo genético proposto, mostrados na figura 8.1

8.1.1 Representação de uma solução e Criação da População Inicial

Como em toda metaheurística baseada em algoritmos evolutivos, estas características do algoritmo geral proposto são dependentes do problema que está sendo tratado. O problema a ser tratado é codificado de modo a fazer possível a aplicação do Algoritmo genético para resolvê-lo.

O número de indivíduos a serem criados para a população é indicado pelo parâmetro *tamanho_da_População*. O valor deste parâmetro deve ser grande o bastante para permitir boa diversidade entre cromossomos, no entanto, limitado superiormente para viabilizar o tempo computacional exigido.

8.1.2 Evolução da População

A etapa de evolução requer a escolha de subconjuntos de indivíduos da população. A seleção dos cromossomos pais para cruzamento é feita ponderando-se o seu grau de adaptabilidade (valor objetivo). Para a aplicação dos operadores de seleção, nós criamos partições dentro da população para a classificação dos cromossomos em níveis de aptidão. Para permitir que esta forma de cruzamento seja eficiente, temos a preocupação de manter os indivíduos da população ordenados pelos seus respectivos valores objetivos.

Depois da fase de reprodução, este mesmo método de seleção é aplicado para a escolha dos cromossomos a serem substituídos.

A seguir descreveremos em detalhe cada um dos passos da etapa de evolução.

8.1.2.1 Particionamento da População

O número de partições é determinado pelo parâmetro *numero_Partições*.

O particionamento da população é feito da seguinte forma:

- Calculamos a diferença entre os valores objetivo do pior e do melhor cromossomo da população

$$intervalo = pior_valor_objetivo - melhor_valor_objetivo$$

- Dividimos o *intervalo* pela quantidade de partições indicada pelo parâmetro *numero_Partições*

$$delta = \frac{intervalo}{numero_Particoes}$$

- Construímos as partições da seguinte forma:

Um cromossomo qualquer, na população, pertencera a *i*-ésima partição se

$$\boxed{\begin{array}{c} \text{Valor objetivo do} \\ \textbf{melhor} \text{ cromossomo} \\ + \\ (i-1) * delta \end{array}} \leq \boxed{\begin{array}{c} \text{Valor objetivo do} \\ \text{cromossomo} \end{array}} \leq \boxed{\begin{array}{c} \text{Valor objetivo do} \\ \textbf{melhor} \text{ cromossomo} \\ + \\ i * delta \end{array}}$$

Onde, $1 \leq i \leq numero_Particoes$

A figura 8.2 mostra uma população dividida em partições. Observe que nem todas as partições terão a mesma quantidade de elementos.

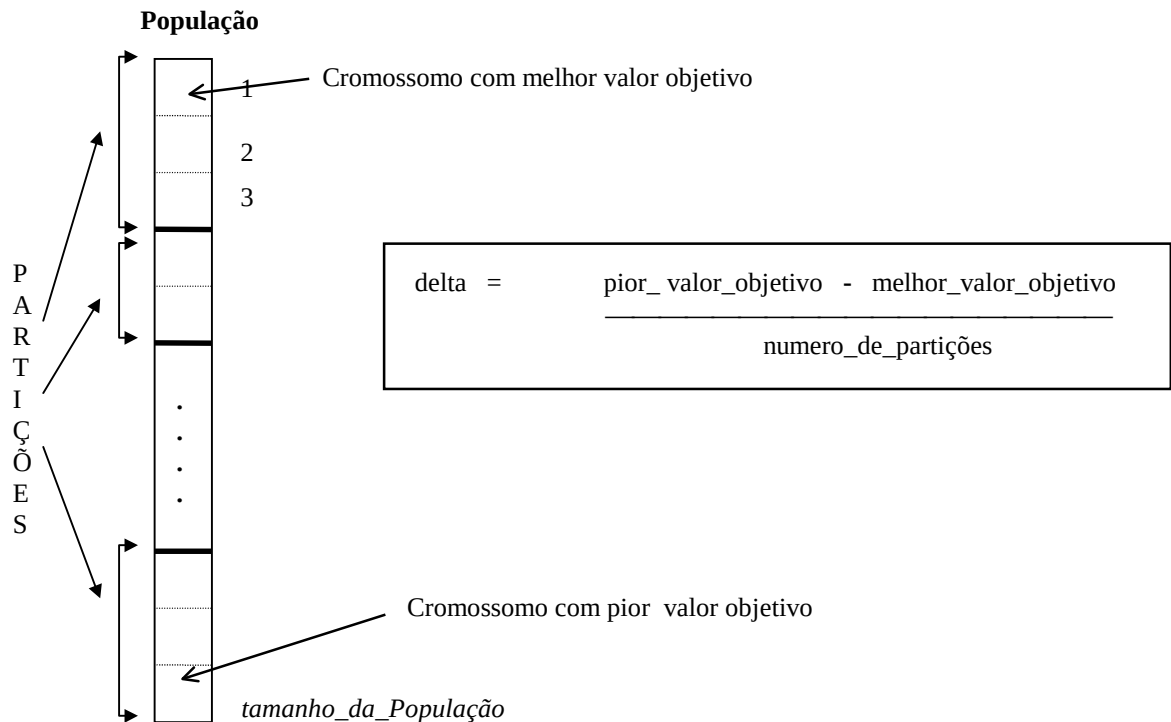


Figura 8.2: Na população ordenada são criadas partições.

8.1.2.2 Cálculo da aptidão para a escolha dos pais para a próxima geração

Depois que a população é particionada, são escolhidos, em função de suas aptidões, os indivíduos que serão os pais da próxima geração. Para calcular a aptidão dos indivíduos de uma partição, é usado um método baseado nos critérios de seleção proporcional e de seleção por elitismo¹. Os parâmetros do algoritmo, *tamanho_Elite_superior* e *peso_Elite_superior*, são usados para indicar o número de indivíduos de elite e a ponderação que eles receberão sobre suas aptidões, respectivamente.

Para cada partição na população, o cálculo da aptidão está descrito a seguir:

Sejam $v_1, v_2, \dots, v_E, \dots, v_k$ os valores objetivo dos k indivíduos de uma partição qualquer, na população, onde $v_1 \leq v_2 \leq \dots \leq v_E \leq \dots \leq v_k$ e $1, 2, \dots, E$ são os indivíduos de elite, ou seja, $tamanho_Elite_superior = E$, $0 \leq E$. Suponhamos, também, que

¹ Vide seção 7.3.1 que apresenta alguns dos critérios de seleção.

$peso_Elite_superior = p$, $0 \leq p \leq 1$. A figura 8.3 mostra o algoritmo utilizado para o cálculo da aptidão para a escolha dos pais para a próxima geração.

ALGORITMO:

Se ($k \geq 0$), então (... se houver elementos na partição)

Se ($E=0$) ou ($E \geq k$), então

(... se o tamanho da Elite superior for igual a 0, ou)

(... se o tamanho da Elite for maior que o número de elementos na partição)

(... então, sem ponderação para todos os indivíduos na partição)

$$total_objetivo = \sum_{i=1}^k v_i$$

$$ratio_v_i = \frac{total_objetivo}{v_i} ; \forall i = 1, \dots, k$$

$$total_ratio = \sum_{i=1}^k ratio_v_i$$

$$aptidao_v_i = \frac{ratio_v_i}{total_ratio} ; \forall i = 1, \dots, k$$

senão

Se ($peso_Elite_superior = 1$), então

(... se o peso da Elite superior for igual a 1)

(... então, distribuição somente para os elementos da Elite superior)

$$total_objetivo = \sum_{i=1}^E v_i$$

$$ratio_v_i = \frac{total_objetivo}{v_i} ; \forall i = 1, \dots, E$$

$$total_ratio = \sum_{i=1}^E ratio_v_i$$

$$aptidao_v_i = \frac{ratio_v_i}{total_ratio} ; \forall i = 1, \dots, E$$

$$aptidao_v_i = 0 ; \forall i = E, \dots, k$$

senão

(... distribuição com ponderação para todos na partição)

(... primeiro a Elite superior com ponderação p)

$$total_objetivo = \sum_{i=1}^E v_i$$

$$ratio_v_i = \frac{total_objetivo}{v_i} ; \forall i = 1, \dots, E$$

$$total_ratio = \sum_{i=1}^E ratio_v_i$$

$$aptidao_v_i = \frac{ratio_v_i * p}{total_ratio} ; \forall i = 1, \dots, E$$

(... agora os indivíduos que não são da Elite, com ponderação $1-p$)

$$\begin{aligned}
total_objetivo &= \sum_{i=E+1}^k v_i \\
ratio_v_i &= \frac{total_objetivo}{v_i} \quad ; \quad \forall i = E + 1, \dots, k \\
total_ratio &= \sum_{i=E+1}^k ratio_v_i \\
aptidao_v_i &= \frac{ratio_v_i * (1-p)}{total_ratio} \quad ; \quad \forall i = E + 1, \dots, k
\end{aligned}$$

fim-se

fim-se

Figura 8.3: Algoritmo para Cálculo da aptidão para a escolha dos pais para a próxima geração.

8.1.2.3 Escolha dos pais para a próxima geração

Uma vez calculada a aptidão para a reprodução, de cada cromossomo na atual população, serão escolhidos de cada partição uma mesma quantidade predefinida de indivíduos para serem pais da próxima geração. Esta quantidade é determinada pelo parâmetro *pais_a_escolher_por_partição*. Como este número é fixo, o número total de indivíduos escolhidos para participar do processo de recombinação de genes é:

$$número_de_Partições * pais_a_escolher_por_partição$$

Como ilustrado no algoritmo da figura 8.4, o processo de escolha dos pais consiste, basicamente, em percorrer as partições e escolher de cada uma delas *pais_a_escolher_por_partição* indivíduos. Este percurso consta de duas etapas: na primeira, o percurso é feito de forma ascendente², desde a última partição até a primeira. Ao fim desta etapa, se ainda não houverem sido escolhidos todos os pais (uma ou mais partições podem ter menos que *pais_a_escolher_por_partição* indivíduos), prossegue-se com a segunda etapa, onde o processo de escolha recomeça na segunda partição, em forma descendente³ e vai até a última partição se for preciso. O porquê desta estratégia será explicado a seguir:

² Isto é da pior partição à melhor partição.

³ Ou seja, da melhor partição à pior partição.

Como indicado anteriormente, na etapa ascendente, pode acontecer que:

$$\text{tamanho_da_partição}_i < \text{pais_a_escolher_por_partição}$$

onde $1 \leq i \leq \text{numero_de_partições}$.

Neste caso, a diferença entre estes dois valores será incrementada ao número de pais a escolher na partição imediatamente superior. Como este processo é acumulativo, pode acontecer que cheguemos à primeira partição e tenhamos mais pais a escolher do que indivíduos disponíveis. Quando isto acontece, é necessária a segunda etapa. Começando na segunda partição, o processo de escolha dos pais que faltam escolher será sequencial; isto é, se faltarem escolher três pais e na segunda partição houver somente dois indivíduos ainda disponíveis, eles serão escolhidos e o pai restante será o melhor indivíduo disponível na terceira partição (veja o algoritmo da figura 8.4). Note que a primeira etapa é feita de forma ascendente pois se uma determinada partição não contiver indivíduos suficientes para serem escolhidos, é melhor que as próximas partições a serem examinadas tenham níveis de aptidão superiores e não inferiores, pois é delas que deverão ser escolhidos os pais que faltarem.

ALGORITMO:

```

i = numero_Partições                ( ... Começamos pela última partição )
Enquanto (i), fazer
    Se ( tamanho_da_partição_i = 0 ), então
        ( ... se a partição esta vazia )
        pais_a_escolher = pais_a_escolher + pais_a_escolher_por_partição
    senão
        Se ( tamanho_da_partição_i ≤ pais_a_escolher_por_partição ), então
            ( ... se a partição é menor que o número de pais a escolher por partição )
            Escolher todos os indivíduos desta partição para serem pais da próxima geração

        Se ( i )
            pais_a_escolher = pais_a_escolher_por_partição +
            ( pais_a_escolher - tamanho_da_partição_i );
        senão                                     //está na última partição ...

            pais_a_escolher = pais_a_escolher - tamanho_da_partição_i
        fim-se
        tamanho_da_partição_i = 0;
    senão
        ( ... A partição possui mais elementos que "pais_a_escolher_por_partição" )
        Escolher desta partição "pais_a_escolher_por_partição" indivíduos para serem pais da
        seguinte geração. Para isto, cada indivíduo tem uma probabilidade de ser escolhido igual
        a sua aptidão para seleção.

        tamanho_da_partição_i = tamanho_da_partição_i - pais_a_escolher
        Se (i>0), então
            pais_a_escolher = pais_a_escolher_por_partição
        senão
            pais_a_escolher = 0
        fim-se
    fim-se
fim-enquanto

i = 1
Enquanto ( i < numero_Partições ) e ( p < numero_de_Pais ), fazer
    Se ( tamanho_da_partição_i > 0 ), então
        pais_ainda_nao_escolhidos = menor_entre( tamanho_da_partição_i, pais_a_escolher )

        Escolher desta partição 'i' os "pais_ainda_nao_escolhidos" maiores indivíduos ainda não
        escolhidos

        pais_a_escolher = pais_a_escolher - pais_ainda_nao_escolhidos
    fim-se
    i = i + 1
fim-enquanto

```

Figura 8.4: Algoritmo para escolher os pais da próxima geração.

8.1.2.4 Aplicação do operador de recombinação e geração dos novos filhos

Existem muitos operadores genéticos para a recombinação de genes e muitos deles podem ser aplicados para um mesmo problema. Por isso, esta etapa do algoritmo genético proposto é dependente do problema que está sendo tratado. Porém, o número de filhos que serão gerados depende dos parâmetros do algoritmo: *filhos_a_ser_gerados* e *filhos_a_ser_considerados_para_inserção*; onde:

$$filhos_a_ser_gerados \geq filhos_a_ser_considerados_para_inserção$$

e

$$filhos_a_ser_considerados_para_inserção \leq tamanho_da_população$$

O primeiro parâmetro indica o número total de filhos que serão gerados como resultado do processo de recombinação de genes. Depois de uma ordenação, somente os *filhos_a_ser_considerados_para_inserção* melhores filhos serão considerados para serem inseridos na população em substituição aos indivíduos que serão eliminados da população atual.

8.1.2.5 Cálculo da aptidão para determinar os indivíduos a serem substituídos

Para calcular a aptidão de substituição dos indivíduos, procedemos de forma análoga ao cálculo da aptidão para reprodução; porém, desta vez os indivíduos com pior (maior) valor objetivo terão maior probabilidade de serem selecionados. Os parâmetros *tamanho_Elite_inferior* e *peso_Elite_inferior* indicarão o tamanho da Elite e a respectiva ponderação para este caso. Para cada partição, os *tamanho_Elite_inferior* piores cromossomos, receberão uma ponderação de *peso_Elite_inferior* que será distribuída entre eles, segundo seus valores objetivo. Para os cromossomos restantes da partição, aplica-se o mesmo processo, porém a ponderação é de $1 - peso_Elite_inferior$. O algoritmo para o cálculo da aptidão para substituição é apresentado na figura 8.5.

ALGORITMO:

Se ($k \geq 0$), então (... se houver elementos na partição)

Se ($E=0$) ou ($E \geq k$), então

(... se o tamanho da Elite inferior for igual a 0, ou)

(... se o tamanho da Elite for maior que o numero de elementos na partição)

(... então, distribuição sem ponderação para todos os indivíduos na partição)

$$total_objetivo = \sum_{i=1}^k v_i$$

$$aptidao_v_i = \frac{v_i}{total_objetivo} ; \forall i = 1, \dots, k$$

senão

Se (peso_Elite_inferior = 1), então

(... se o peso da Elite inferior for igual a 1)

(... então, distribuição somente para os elementos da Elite inferior)

$$total_objetivo = \sum_{i=E}^k v_i$$

$$aptidao_v_i = \frac{v_i}{total_objetivo} ; \forall i = E, \dots, k$$

$$aptidao_v_i = 0 ; \forall i = 1, \dots, E$$

senão

(... distribuição com ponderação para todos na partição)

(... primeiro a Elite inferior com ponderação p)

$$total_objetivo = \sum_{i=E}^k v_i$$

$$aptidao_v_i = \frac{v_i * p}{total_objetivo} ; \forall i = E, \dots, k$$

(... agora os indivíduos que não são da Elite, com ponderação $1-p$)

$$total_objetivo = \sum_{i=1}^E v_i$$

$$aptidao_v_i = \frac{v_i * (1-p)}{total_ratio} ; \forall i = 1, \dots, E$$

fim-se

fim-se

fim-se

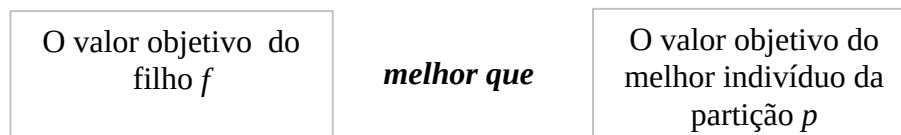
Figura 8.5: Algoritmo para o cálculo da aptidão de substituição dos cromossomos de cada partição.

8.1.2.6 Escolha dos indivíduos a serem substituídos

Depois de terem sido gerados os *filhos_a_ser_gerados* filhos e ordenados em forma decrescente (do melhor ao pior), serão considerados somente os *filhos_a_ser_considerados_para_inserção* primeiros filhos como candidatos a substituir um indivíduo da população atual. Determina-se para cada novo filho considerado, a partição na qual será inserido e dentro dessa partição o cromossomo a ser substituído por ele, da seguinte forma:

Seleção da partição a ter um cromossomo substituído

Para determinar a partição p ($1 \leq p \leq \text{numero_Partições}$) na qual um novo filho f ($1 \leq f \leq \text{filhos_a_ser_considerados_para_inserção}$) substituirá um indivíduo, analisamos seu valor objetivo. Caso o valor objetivo do novo filho f seja melhor que o melhor valor objetivo na população, o filho f será inserido no início da população como o novo melhor indivíduo, substituindo o pior indivíduo. Se o valor objetivo for pior que o valor objetivo do pior indivíduo da população, o filho f não será inserido. Em qualquer outro caso, f será inserido na primeira partição p onde o melhor indivíduo for pior do que ele, sendo as partições percorridas em ordem decrescente; ou seja:



onde: $1 \leq p \leq \text{numero_Partições}$.

Seleção do cromossomo a substituir

Uma vez determinada a partição p , será escolhido desta, um cromossomo a ser substituído. Para isto, cada indivíduo tem uma probabilidade de ser escolhido igual a sua aptidão para substituição. O algoritmo para a escolha dos cromossomos a serem substituídos é apresentado na figura 8.6.

ALGORITMO:

```

Enquanto ( houveram_substituições ) e ( houver filhos para avaliar ), fazer
    houveram_substituições = FALSE
    Se ( valor_do_filho_f < valor_do_pior_indivíduo_na_População ), então
        (... se o filho merece ser inserido na População)
        Se ( valor_do_filho_f < valor_do_melhor_indivíduo_na_População ), então
            (... tiramos o pior cromossomo na População)
            População = População - { pior_indivíduo_na_População }
            (... registramos que houve uma substituição)
            houveram_substituições = TRUE
        senão
            Procurar uma partição i tal que:
            - O número de elementos ainda não substituídos na partição i > 0; e
            - O valor do melhor (menor) indivíduo na partição i ≥ valor_do_filho_f
            Se ( existe a partição i ), então
                Escolher na partição i o indivíduo a ser substituído. Para isto, cada indivíduo
                tem uma probabilidade de ser escolhido igual a sua aptidão para substituição.
                (... registramos que houve uma substituição)
                houveram_substituições = TRUE
            fim-se
        fim_se
    fim-se
    filho_f = proximo_filho
fim-enquanto

```

Figura 8.6: Algoritmo para a seleção dos cromossomos a serem substituídos.**8.1.2.7 Substituição dos cromossomos selecionados pelos novos filhos**

Quando a etapa de seleção de cromossomos a ser substituídos tiver terminado, terão sido escolhidos uma quantidade de indivíduos igual ao número de filhos escolhidos para ser inseridos na população. A substituição de cromossomos é feita da seguinte maneira: primeiramente os cromossomos selecionados para serem substituídos são eliminados da população. Em seguida os filhos que os substituirão são inseridos de modo tal que a população seja mantida ordenada por valor objetivo, ou seja, o primeiro indivíduo da população será sempre aquele com melhor valor objetivo e o último aquele com pior valor.

8.1.3 Verificação do Critério de Convergência da População

Após muitas gerações, alguns cromossomos começam a dominar o processo de reprodução fazendo com que os seus descendentes sejam idênticos a eles. Neste instante, dizemos que a população ficou homogênea ou que ela convergiu para um ponto extremo.

O critério de convergência da população é dado por dois parâmetros do algoritmo: A tolerância ao número de gerações que se pode realizar sem que se melhore a melhor solução até agora encontrada (*tolerancia_gerações_sem_melhora*), e, a tolerância ao número de gerações sem que se realizem substituições na população (*tolerancia_gerações_sem_substituições*). Se uma destas duas tolerâncias é atingida, considera-se que a população convergiu.

8.1.4 Avaliação do Critério de Parada do Algoritmo

O algoritmo pára quando acontece uma das seguintes situações:

1. Quando é executado um dado número de processos de diversificação, indicado pelo parâmetro *máximo_número_diversificações*.
2. Quando é executado um determinado número de processos de diversificação sem que a solução seja melhorada. Este número é indicado pelo parâmetro *tolerancia_diversificações*.

8.1.5 Inserção de Genes na População

Esta característica do algoritmo geral proposto é dependente do problema que está sendo tratado e da estratégia de evolução adotada. Para maiores detalhes sobre a inserção de genes na população, veja as seções 7.4.3 e 7.4.4.

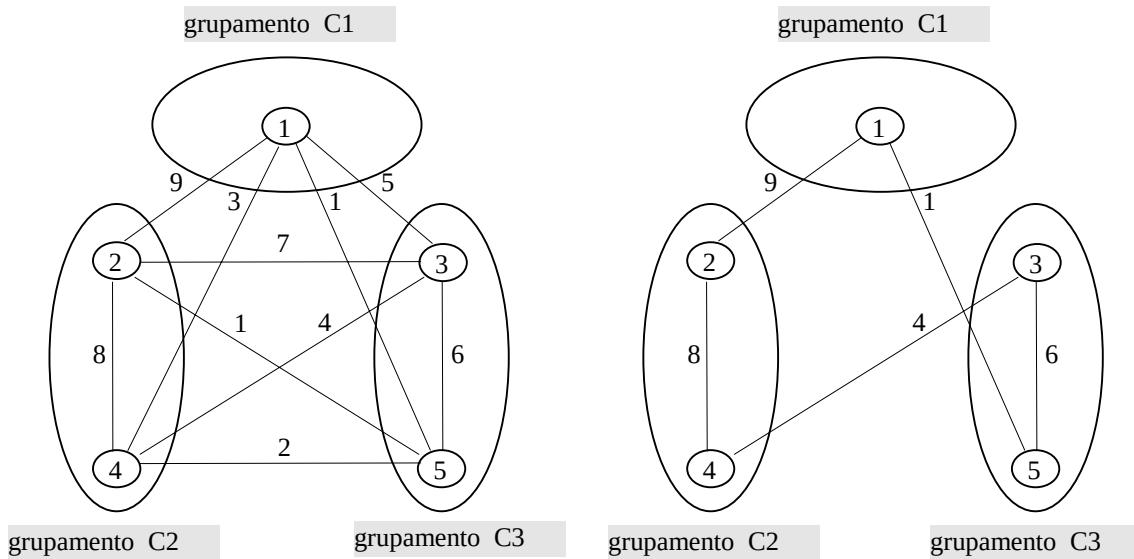
8.2 Aplicação do Algoritmo Genético Proposto ao CTSP

O algoritmo genético proposto tem um propósito geral e, assim, a maioria de suas características são genéricas o suficiente para adequarem-se a qualquer problema. Porém, algumas delas são sensíveis ao problema tratado. Por isto, nesta seção, nos limitaremos a descrever somente aquilo que for necessário para estabelecer uma conexão entre o algoritmo proposto e o *CTSP*. A seguir serão descritos: a representação de uma solução para o *CTSP*, como é criada a população inicial, o operador genético utilizado para o *CTSP* e como é feita a inserção de genes para prosseguir com a busca após a convergência da população.

8.2.1 Representação de uma solução

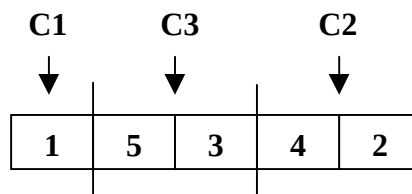
Por aplicarmos um operador baseado no operador de recombinação ERX^4 na fase de reprodução, é necessário utilizarmos a representação por caminhos, ou seja, por vetores de inteiros, nos quais, o n -ésimo elemento de um cromossomo representa a n -ésima cidade a ser visitada na solução do *CTSP*. Para exemplificar, considere a figura 8.7: na figura 8.7a está ilustrada uma instância do *CTSP*, a figura 8.7b mostra uma solução viável para esta instância e a figura 8.7c mostra como esta solução é representada em nossa aplicação.

⁴ Este operador será descrito na seção 8.2.3.1.



(a). Instância do CTSP com $n=5$ cidades e três grupamentos C1, C2, C3. ($k=3$).

(b). Uma Solução viável para a instância em (a).



(c). A sequência representa o tour $1 \rightarrow 5 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 1$ mostrada na solução em (b).

Figura 8.7: A representação de uma solução para o CTSP na aplicação do algoritmo.

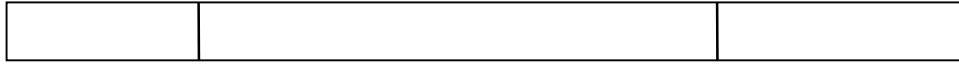
8.2.2 Inicialização da População de Indivíduos

Para a geração da população inicial, obtém-se para cada cromossomo uma sequência aleatória de visitas entre os grupamentos. A sequência de visitas no interior de cada grupamento é gerada da seguinte forma: a escolha dos vértices de entrada e saída de cada grupamento é feita através do algoritmo do vizinho mais próximo⁵ e, em seguida, é gerada uma sequência aleatória de visitas aos vértices restantes. Sejam: $S = \{1, 2, \dots, N\}$ as cidades do problema e $C = \{1, 2, \dots, K\}$ os grupamentos de cidades, a geração de um cromossomo para a população inicial está ilustrada na figura 8.8.

⁵ Vide seção 2.2.1.

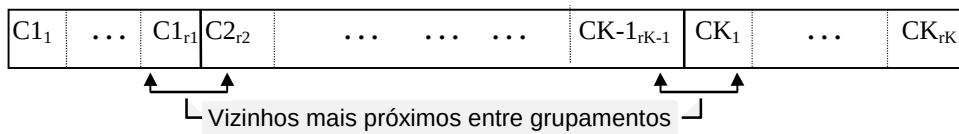
Geração de uma sequência aleatória de visita entre grupamentos

Grupamento C1 ... Grupamento CK



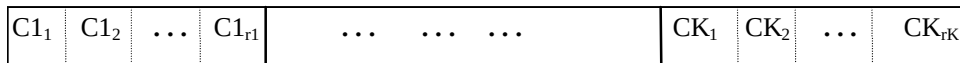
Determinação das cidades mais próximas entre os grupamentos, segundo a sequência aleatória gerada.

Grupamento C1 ... Grupamento CK



Geração aleatória da ordem de visitas dos vértices no interior de cada grupamento

Grupamento C1 ... Grupamento CK



onde:

$C_1, C_2, \dots, C_K \in \mathbf{C}$, C_{J_i} é a i -ésima cidade visitada no grupamento C_J , $1 \leq J \leq K$ e $r_i = |C_i|$; $i = 1, 2, \dots, K$.

Figura 8.8: Geração de um cromossomo para a população inicial.

8.2.3 Recombinação dos genes

Para a recombinação dos genes nos baseamos no conhecido operador de recombinação *ERX* (*Edge Recombination Crossover*) [WHITLEY, 89], bastante utilizado em algoritmos genéticos construídos para solucionar o *TSP*. Ele é considerado um dos melhores operadores para a resolução deste problema. Este operador será descrito na próxima seção e, a seguir, descreveremos as modificações propostas por Rabelo, Ochi e Maculan em [RABELO, 96] para adequá-lo ao *CTSP*, as quais foram utilizadas neste trabalho.

8.2.3.1 O operador de recombinação *ERX* (*Edge Recombination Crossover*)

O funcionamento do operador *ERX* será descrito através de um exemplo. Considere a instância do *TSP* representada na figura 8.9.

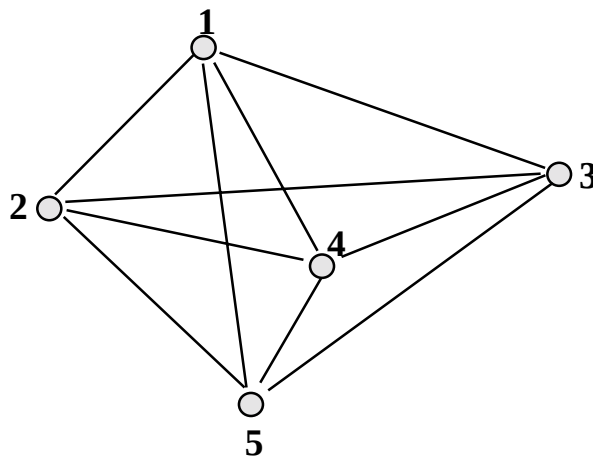


Figura 8.9: Uma instância do TSP.

O operador *ERX* constrói cromossomos filhos usando somente as arestas presentes nos pais.

Primeiro, a partir de ambos os pais, uma lista de cidades adjacentes a cada nó é construída. Sejam os pais, as seguintes soluções viáveis para a instância da figura 8.9:

P1: 1 2 5 4 3

P2: 3 5 4 2 1

as listas de cidades adjacentes resultantes são mostradas na figura 8.10.

Cidade	Cidades Adjacentes		
1	2	3	---
2	1	5	4
3	4	1	5
4	5	3	2
5	2	4	3

Figura 8.10: Listas de cidades adjacentes construídas a partir de P1 e P2.

Para gerar os novos filhos, as cidades são selecionadas, uma de cada vez, sendo escolhida a próxima cidade com o menor número de cidades adjacentes (uma escolha aleatória é feita em caso de empate). No exemplo, a primeira cidade a ser escolhida é a cidade 1, pois tem o menor número de cidades adjacentes. Então, a representação parcial do novo filho F é:

F: 1 _ _ _

Uma vez que uma cidade é escolhida, ela é removida da tabela e as cidades conectadas a ela são consideradas candidatas para a próxima escolha. No exemplo, estas cidades são as cidades 2 e 3, como ilustra a figura 8.11.

Cidade	Conectada a cidade		
X	2	3	---
2	1/	5	4
3	4	1/	5
4	5	3	2
5	2	4	3

⇒

Cidade	Conectada a cidade		
2	5	4	---
3	4	5	---
4	5	3	2
5	2	4	3

Figura 8.11: Cidades candidatas para a próxima escolha.

Este processo é repetido mais algumas vezes, até que todas as cidades tenham sido escolhidas, conforme ilustra a figura 8.12⁶.

⁶ Em caso de empate a escolha é feita aleatoriamente.

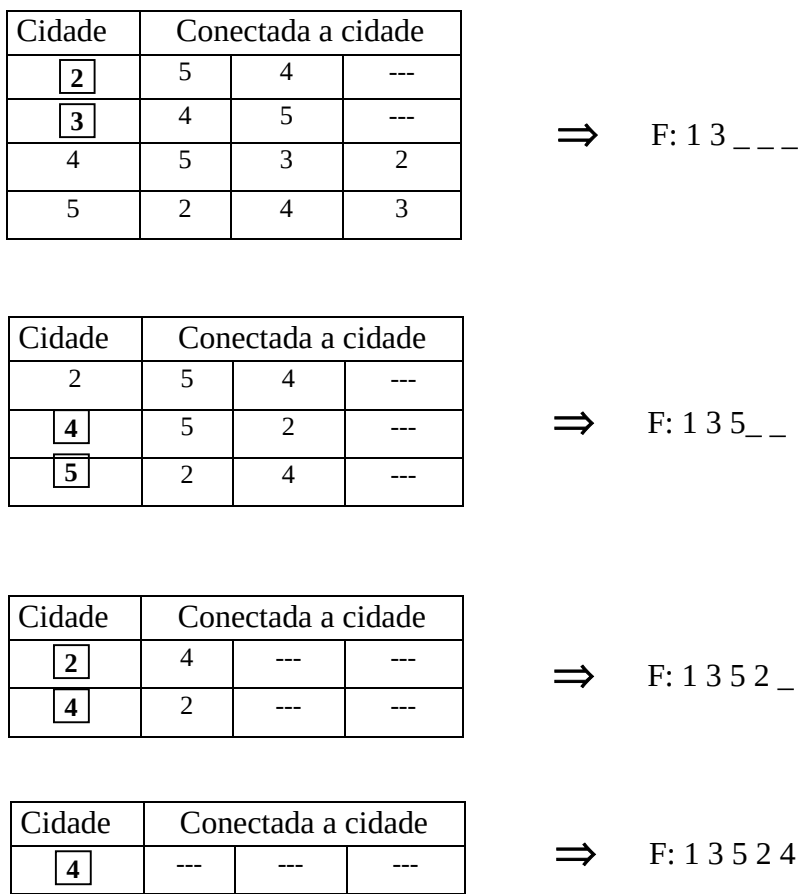


Figura 8.12: Continuação do processo de geração do filho F.

Os ciclos hamiltonianos representados pelo novo filho F e por seus pais são mostrados na figura 8.13.

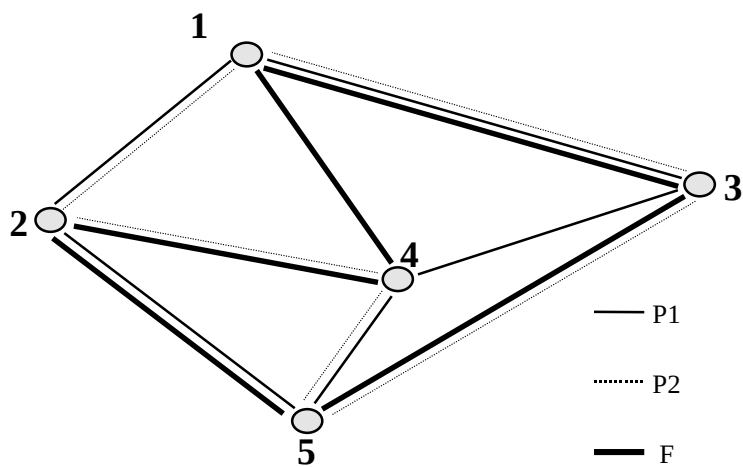


Figura 8.13: Ciclos Hamiltonianos representados por P1, P2 e F.

Uma otimização do *ERX* consiste em manter um registro das arestas presentes em ambos pais e forçar sua escolha. Por exemplo, sejam os pais:

P1: 1 2 5 4 3

P2: 3 5 4 2 1

As arestas comuns são: (1,2), (1,3) e (5,4).

A figura 8.14 ilustra os passos do operador *ERX* com esta otimização.

Cidade	Conectada a cidade		
1	2	3	---
2	1	5	4
3	4	1	5
4	5	3	2
5	2	4	3

⇒ F: 1 _ _ _ _

Cidade	Conectada a cidade		
2	5	4	---
3	4	5	---
4	5	3	2
5	2	4	3

⇒ F: 1 3 _ _ _

Cidade	Conectada a cidade		
2	5	4	---
4	5	2	---
5	2	4	---

⇒ F: 1 3 5 _ _

Cidade	Conectada a cidade		
2	---	4	---
4	2	---	---

⇒ F: 1 3 5 4 _

Cidade	Conectada a cidade		
2	---	---	---

⇒ F: 1 3 5 4 42

Figura 8.14: Geração do filho F com através do operador *ERX* otimizado.

Os ciclos hamiltonianos representados pelo novo filho F e por seus pais são mostrados na figura 8.15.

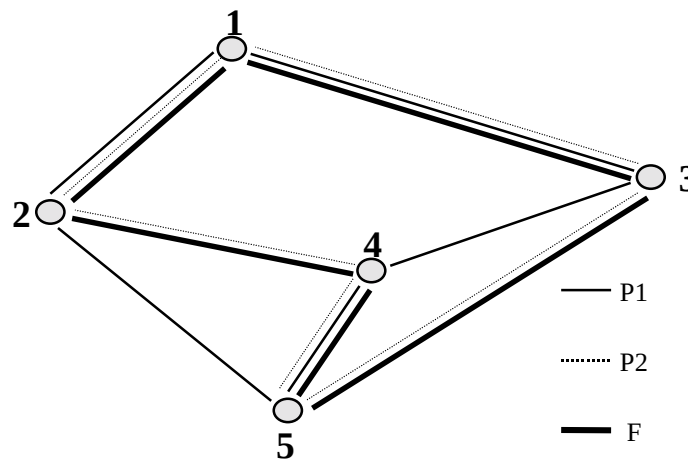


Figura 8.15: Ciclos Hamiltonianos representados por P1, P2 e F.

8.2.3.2 O operador de recombinação *ERX* adaptado ao *CTSP*

Como visto na seção anterior, no operador *ERX* constrói-se, a partir dos cromossomos pais, para cada cidade i do problema, uma lista de arestas contendo as cidades conectadas a i .

Para o problema do caixeiro viajante com grupamentos, existe na literatura uma adaptação do *ERX* proposta por Rabelo, Ochi e Maculan [RABELO, 96]. Nesta adaptação, são criadas dois tipos de listas ao invés de uma. Para cada cidade, é criada uma lista denominada *Lista_Grupamentos*, a qual contém apenas as cidades conectadas à cidade i , pertencentes ao mesmo grupamento de i . E para cada grupamento c é criada uma outra lista, denominada *Lista_Inter_Grupamentos*, a qual contém as cidades de outros grupamentos adjacentes a ele.

Nós nos inspiramos na de adaptação Rabelo, Ochi e Maculan para desenvolver nosso trabalho. Adicionalmente às duas listas, nós também criamos uma outra lista de adjacências, V , a qual contém, para cada cidade i , a cidade mais próxima de i em cada um dos grupamentos.

A geração de um filho, é feita da seguinte forma:

- Constrói-se as listas *Lista_Grupamentos* e *Lista_Inter_Grupamentos*, a partir dos pais escolhidos.
- Encontra-se a primeira cidade a ser inserida no cromossomo, a qual é aquela que aparece na *Lista_Grupamentos* e tem o menor número de adjacências. Ela indica qual o primeiro grupamento a ser preenchido. Em caso de empate, se faz uma escolha aleatória.
- Segundo o *ERX* tradicional, gera-se um caminho hamiltoniano dentro do grupamento, o qual é inserido no cromossomo.
- A determinação do próximo grupamento a ser inserido no cromossomo, é feita a partir da última cidade *c* inserida, da seguinte forma:

Avaliando cada uma das cidades adjacentes ao grupamento atual na lista *Lista_Inter_Grupamentos*, selecionamos entre estas cidades aquelas que na lista *V* tenham como cidade mais próxima ao grupamento atual a última cidade escolhida neste grupamento. Dentre as cidades selecionadas, escolhemos, aleatoriamente, uma para ser a próxima cidade a ser inserida no cromossomo. Porém se nenhuma cidade houver sido selecionada, a próxima cidade a ser escolhida é aquela cidade, ainda não escolhida, que aparece na lista *Lista_Inter_Grupamentos* como adjacente ao grupamento atual e que é a mais próxima dele. Em caso de haver mais de uma cidade satisfazendo esta condição, faz-se uma escolha aleatória dentre as cidades empatadas.

- Este processo é repetido para cada grupamento, até a geração completa do cromossomo.

8.2.4 Inserção de Genes na População

Através deste processo novos genes são espalhados para todos os indivíduos na população. Uma nova população será gerada. Ela conterà o melhor indivíduo da atual população e todos os demais serão obtidos através de uma mutação⁷, que será descrita a seguir.

⁷ Vide seção 7.3.2.2.

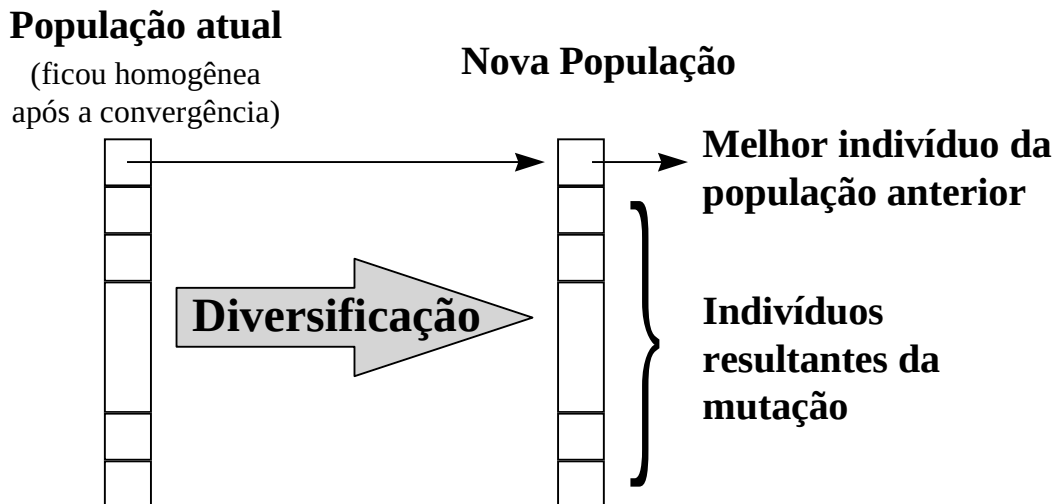


Figura 8.16: A diversificação da população.

8.2.4.2 A Mutação

Para determinar quais os novos genes a serem inseridos na população, vários procedimentos podem ser utilizados. O mais frequentemente empregado e também aqui utilizado é a Mutação, que consiste na geração de um novo indivíduo a partir de alterações nos genes de outro indivíduo. Em nosso caso, um indivíduo mutante é gerado a partir do cruzamento do melhor indivíduo da população com um ou mais indivíduos gerados aleatoriamente. De esta forma os genes do atual melhor indivíduo são alterados de forma a conter alguns dos genes do indivíduo com quem esta sendo combinado.

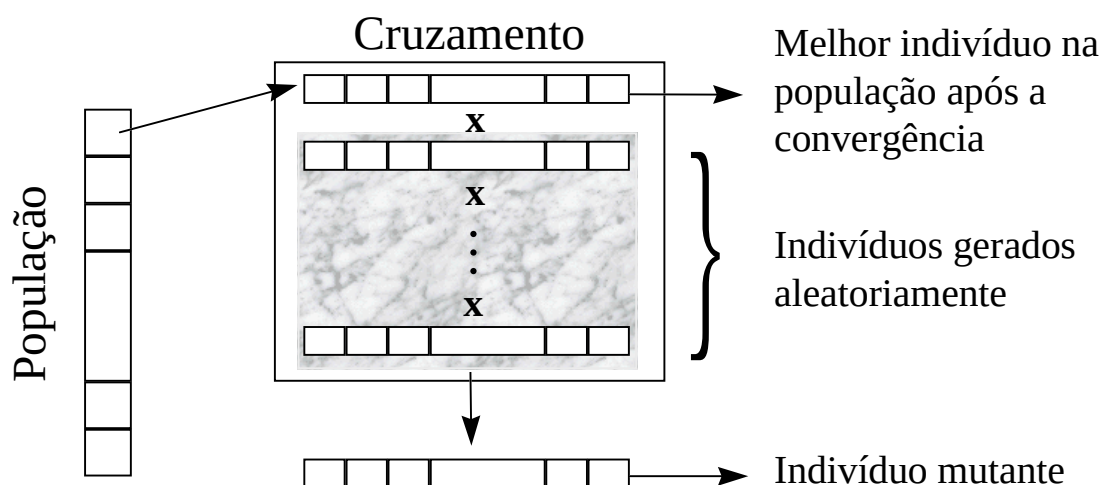


Figura 8.17: A mutação.

Capítulo 9

O Algoritmo Genético Proposto com *Ant Colony Systems* para solucionar o CTSP

Como, geralmente, os algoritmos genéticos são bons em determinar rapidamente áreas promissoras do espaço de busca, freqüentemente são utilizados em combinação com métodos de busca local [RABELO, 96] ou de refinamento da melhor solução encontrada.

O *Ant Colony System*, por outro lado, é um método com um sistema de busca local bastante aguçado que tem sido aplicado com sucesso a vários problemas [GAMBARDELLA, 97A] [MANIEZZO, 94]. Porém, devido ao fato de que a atualização da trilha de feromônios e a emulação de algumas características comportamentais consomem um tempo computacional significativo, este sucesso têm sido alcançado apenas para instâncias relativamente pequenas. Assim, tornou-se necessário o desenvolvimento de formas de superar esta desvantagem; tais como estrutura de dados sofisticadas [GAMBARDELLA, 97A].

O *Ant Colony System* é um sistema de aprendizado lento que se torna mais eficiente a medida que vai adquirindo conhecimento. Se unirmos a rapidez dos algoritmos genéticos em adquirir conhecimento e a facilidade do *Ant System* em aproveitar conhecimento prévio poderemos conseguir soluções boas em um tempo computacional relativamente pequeno.

Neste capítulo, é apresentada uma metaheurística que sugere uma forma de combinar um algoritmo genético e um *Ant Colony System* para resolver instâncias do CTSP.

9.1 Parâmetros do algoritmo

O algoritmo recebe como parâmetros tanto os parâmetros requeridos pelo algoritmo genético apresentado no capítulo 8 quanto os parâmetros do *Ant Colony System* adaptado ao *CTSP*, descrito na seção 6.4. Além destes, requer os parâmetros *max_iterações*, que determina o número de iterações da metaheurística proposta e *numero_de_soluções_iniciais*, que determina a quantidade de soluções obtidas pelo Algoritmo Genético que serão passadas ao *Ant Colony System*.

9.2 A estrutura do algoritmo

A metaheurística proposta é composta por dois passos que são executados *max_iterações* vezes. No primeiro passo executa-se o algoritmo genético proposto um dado número de vezes, determinado pelo parâmetro *numero_de_soluções_iniciais*. A cada rodada do algoritmo genético, a melhor solução encontrada é armazenada. Este passo está orientado a varrer o espaço de busca na procura de regiões promissoras para as quais o algoritmo convergirá. O segundo passo está orientado a fazer uma busca em torno das regiões encontradas na primeira etapa, para isto, utilizamos o *Ant Colony System* adaptado ao *CTSP*, descrito na seção 6.4. Esta adaptação, requer um conjunto de soluções de qualidade como entrada, o qual é o próprio conjunto de *numero_de_soluções_iniciais* soluções armazenadas durante as rodadas do algoritmo genético. Quando a colônia de formigas terminar sua execução, passamos a solução obtida por ela ao algoritmo genético. Esta solução repassada ao algoritmo genético para ser um indivíduo da população inicial, a cada rodada. Os demais indivíduos serão gerados aleatoriamente. Note que, na primeira iteração desta metaheurística, não é passada nenhuma solução inicial para o algoritmo genético e assim, todos os indivíduos da população são gerados aleatoriamente, em todas as rodadas do AG nesta iteração da metaheurística. A figura 9.1 mostra o algoritmo para a metaheurística proposta e a figura 9.2 apresenta uma ilustração deste algoritmo.

ALGORITMO:

$solução_inicial = NULL$

Para i de 1 até $max_iterações$ faça

PASSO 1

Para j de 1 até $numero_de_soluções_iniciais$ faça

$melhores_soluções[j] = algoritmo_genético(solução_inicial)$

fim-para

PASSO 2

$solução_inicial = Ant_Colony_System(melhores_soluções)$

fim-para

Figura 9.1: O algoritmo para a metaheurística proposta.

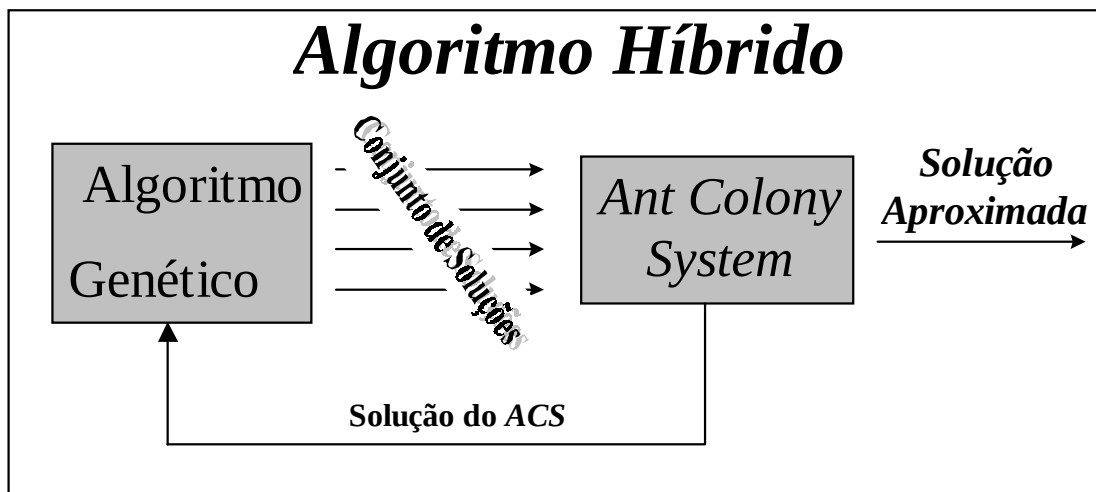


Figura 9.2: Uma representação do algoritmo da metaheurística proposta.

O algoritmo apresentado baseia-se no seguinte:

Seja P uma instância de um problema que consiste na escolha de um subconjunto de arestas de um grafo G , que satisfaça um conjunto de restrições e S o conjunto de todas as soluções viáveis para P . Seja $s^* \in S$ uma solução ótima para P e seja $\{s_1, s_2, \dots, s_n\} \subseteq S$ um conjunto qualquer de soluções tais que se $i \neq j$ então $s_i \neq s_j$, $1 \leq i \leq n$ e $1 \leq j \leq n$, então:

(1) Quando $n = |S|$; $s^* \subset \bigcup_{i=1}^n s_i$.

(2) Se $n \ll |S|$, pode acontecer que $s^* \subset \bigcup_{i=1}^n s_i$

(3) Mas, se os $s_1, s_2, \dots, s_n$ fossem um conjunto de soluções aproximadas, dado por um ou mais algoritmos; então, podemos supor que s^* está parcialmente contida em $\bigcup_{i=1}^n s_i$

Nossa abordagem consiste em utilizar o algoritmo genético para determinar o subconjunto de soluções $\{s_1, s_2, \dots, s_n\}$ e o ACS para procurar s^* .

A avaliação dos algoritmos apresentados até agora será abordada no próximo capítulo, assim como conclusões sobre este trabalho e sugestões para trabalhos futuros.

Capítulo 10

Resultados de Testes e Conclusões

10.1 Resultados de Testes

10.1.1 Avaliação dos resultados obtidos pela metaheurística proposta

Nesta seção se apresentam os testes computacionais da metaheurística descrita neste trabalho aplicada ao CTSP. Analisamos seu desempenho, comparando a qualidade dos resultados obtidos e do tempo computacional empregado com os resultados e tempos obtidos por dois bons métodos para solução do CTSP, encontrados na literatura: a metaheurística de Rabelo, Ochi e Maculan [RABELO, 96] e a heurística H2 de Gendreu, Laporte e Potvin [GENDREU, 94].

Para tornar viável a comparação, utilizamos as mesmas instâncias usadas por Rabelo, Ochi e Maculan para comparar seus próprios resultados com os de Gendreu, assim os resultados dos métodos de Rabelo, Ochi e Maculan e da heurística H2 de Gendreu que apresentaremos, são os mesmos que foram apresentados por Rabelo, Ochi e Maculan.

Nossos testes foram realizados em uma *workstation* Sun Ultra 1; enquanto que os testes feitos por Rabelo *et al.* foram feitos em uma *workstation* Sun 2000, assim como os de Gendreau.

Utilizamos os mesmos valores para os parâmetros de nossa metaheurística em todos os testes. Esses valores são mostrados na tabela da figura 10.1. A configuração dos parâmetros do algoritmo genético proposto não ofereceu grandes dificuldades, pois nossos testes não buscavam encontrar a solução ótima e sim boas soluções iniciais em um tempo muito rápido.

Parâmetros da Metaheurística Proposta		
Parâmetro	Valor	Significado do Parâmetro
<i>número_de_soluções_iniciais</i>	2	Número de soluções iniciais para o ACS, geradas pelo AG proposto.
Parâmetros do Algoritmo Genético Proposto		
Parâmetro	Valor	Significado do Parâmetro
<i>tamanho_da_População</i>	21	Tamanho da população de indivíduos
<i>número_Partições</i>	2	Número de partições na população
<i>pais_a_escolher_por_partição</i>	1	Número de pais a escolher de cada partição para realizar o processo de reprodução
<i>filhos_a_ser_gerados</i>	12	Numero de filhos a gerar em cada processo de reprodução
<i>filhos_a_ser_considerados_para_inserção</i>	6	Numero de filhos a considerar para inserir na seguinte população
<i>tamanho_Elite_superior</i>	2	Tamanho da elite superior
<i>peso_Elite_superior</i>	0.675	Peso da elite superior
<i>tamanho_Elite_inferior</i>	3	Tamanho da elite inferior
<i>peso_Elite_inferior</i>	0.8	Peso da elite inferior
<i>tolerância_gerações_sem_melhora</i>	200	Numero máximo de gerações permitido sem que se melhore a solução
<i>tolerância_gerações_sem_substituições</i>	30	Numero máximo de gerações permitido sem que haja inserções na população
Parâmetros da Aplicação do Algoritmo Genético Proposto ao CTSP		
Parâmetro	Valor	Significado do Parâmetro
<i>número_de_mutações</i>	1	Numero de Mutações
<i>máximo_número_diversificações</i>	15	Numero de Diversificações
<i>tolerância_diversificações</i>	3	Tolerância de Diversificações
Parâmetros da Colônia de Formigas		
Parâmetro	Valor	Significado do Parâmetro
<i>número_de_formigas_na_colônia</i>	10	Numero total de formigas na Colônia
ρ	0.1	Taxa de evaporação da trilha de feromônios
α	0.1	Sensibilidade à trilha de feromônios
β	2	Sensibilidade à distância (Importância relativa da função heurística)
q_0	0.9	Balanceamento entre exploração e exploração
<i>número_de_iterações_da_colônia</i>	7000	Número de iterações da colônia de formigas

Figura 10.1: Valores dos parâmetros de nossa metaheurística utilizados nos testes.

A comparação entre as soluções obtidas por nossa metaheurística e as obtidas pelos métodos de Rabelo et al. e de Gendreu está ilustrada na tabela da figura 10.2. Os valores apresentados por Rabelo et al. são os melhores valores obtidos em seus testes¹. Eles não apresentaram valores médios². A nossa metaheurística foi executada 10 vezes para

¹ A heurística H2 é uma heurística determinística, assim, os valores que ela obtém para cada instância serão sempre os mesmos.

² Eles também não indicaram como foram obtidos seus melhores valores; porém, supomos que estes sejam os melhores resultados de um conjunto pequeno de testes para cada instância, como é usual.

resolver cada instância. Para cada instância, o melhor valor dentre os 10 valores obtidos é mostrado na coluna denominada **Melhores Valores**. A coluna denominada **Valores Médios** apresenta a média destes 10 valores.

Note que os dados da tabela mostram que nossa metaheurística apresenta resultados competitivos, mesmo sendo comparada a dois bons métodos encontrados na literatura. Até mesmo as médias dos valores obtidos nas 10 rodadas de cada instância não estão distantes dos valores apresentados por Gendreu.

Instância	Número de Cidades	Número de Grupamentos	Metaheurística de Rabelo, Ochi e Maculan	Heurística H2 de Gendreu	Metaheurística Proposta	
					Melhores Valores	Valores Médios
1	100	4	4567,375**	4667,729	4574,193*	4705,638
2	100	4	3568,793*	3626,316	3524,966**	3598,198
3	100	4	3554,992**	3618,509	3557,535*	3682,964
4	200	9	5321,892*	5495,488	5269,330**	5419,727
5	200	9	6001,821**	6209,414	6051,791*	6213,620
6	200	9	6148,784	6110,287*	6033,467**	6150,529
7	300	9	6419,445*	6355,857**	6436,917	6532,585
8	300	9	6345,086*	6417,269	6313,229**	6443,440
9	300	9	7393,025**	7532,795	7479,671*	7709,988
10	500	25	9934,789*	-	9773,473**	10519,002
11	500	25	9937,689*	-	9852,567**	10720,173
12	500	25	10408,581*	-	10344,837**	10756,420
Número de melhores soluções			4	1	7	-
Número de segundas Melhores soluções			7	1	4	-

(**) Melhor solução

(*) Segunda melhor solução

Figura 10.2: Tabela comparativa dos custos das melhores soluções obtidas pela metaheurística de Rabelo *et. al*, pela heurística H2 de Gendreu e pela metaheurística proposta neste trabalho.

Os tempos de CPU consumidos para a obtenção dos resultados apresentados na tabela 10.2 são mostrados na tabela da figura 10.3. Cada linha desta tabela corresponde à mesma linha na tabela de custos apresentada na figura 10.2.

Para cada instância, o tempo apresentado por Rabelo *et al.* é o tempo total consumido pela rodada que obteve a melhor solução; os tempos apresentados por Gendreu também são os tempos totais consumidos pelo algoritmo; na coluna dos resultados de nossa metaheurística, a coluna denominada **Melhor Tempo** apresenta tempos relativos à

rodada que obteve a melhor solução: a subcoluna **melhor sol.** apresenta o tempo consumido até que fosse encontrada a melhor solução e a subcoluna **global** apresenta o tempo total consumido; a coluna denominada **Valores Médios** apresenta na subcoluna **melhor sol.**, a média dos 10 tempos de obtenção da melhor solução e na subcoluna **global** a média dos 10 tempos totais consumidos.

Como foi dito, nossos testes foram executados em uma máquina diferente daquela utilizada por Rabelo para executar o seu algoritmo e o de Gendreu.

Instância	Número de Cidades	Número de Grupamentos	Metaheurística de Rabelo, Ochi e Maculan	Heurística H2 de Gendreu	Metaheurística Proposta			
					Melhor Tempo		Tempo Médio	
					melhor sol.	global	melhor sol.	global
1	100	4	38	87	56,08	63,04	46,92	62,38
2	100	4	42	113	59,48	66,34	48,54	67,90
3	100	4	42	57	54,48	66,82	47,63	69,71
4	200	9	118	404	196,15	406,14	167,17	447,89
5	200	9	111	264	110,02	183,12	164,37	190,56
6	200	9	107	409	126,80	174,44	162,31	186,92
7	300	9	661	1342	539,29	577,19	461,09	517,93
8	300	9	653	994	424,11	445,27	381,45	432,89
9	300	9	597	1352	345,00	359,86	356,43	383,70
10	500	25	683	-	519,86	597,77	450,87	571,95
11	500	25	680	-	549,27	554,22	385,63	554,08
12	500	25	682	-	584,68	631,26	423,37	535,60

Figura 10.3: Tempos de CPU consumidos pela Metaheurística de Rabelo et. al, pela heurística H2 de Gendreu e pela metaheurística proposta neste trabalho. Estes tempos são correspondentes aos resultados apresentados na tabela da figura 10.1.

10.1.2 Experiências

Foi implementada uma versão de nossa metaheurística, onde foi acoplada ao algoritmo da colônia de formigas uma lista de cidades candidatas para cada cidade, como indicado em [GAMBARDELLA, 97A]. Para cada cidade i sua lista de cidades candidatas é composta pelas *tamanho_lista_candidatas* cidades mais próximas da cidade i ; onde, *tamanho_lista_candidatas* é um parâmetro adicional incorporado à esta versão de nossa metaheurística. Esta é a forma padrão de construção de listas de cidades candidatas.

Na tabela da figura 10.4, apresentamos uma comparação entre os resultados obtidos pela versão original, sem listas de cidades candidatas com os obtidos pela versão com

listas de cidades candidatas. Para a execução das duas versões foram utilizados os mesmos valores para os parâmetros mostrados na tabela da figura 10.1. O valor empregado para o parâmetro *tamanho_lista_candidatas* da versão modificada foi 10. Cada uma das versões foi executada 10 vezes para cada uma das 12 instâncias. Para cada instância e versão da metaheurística, o valor apresentado na tabela é o melhor dentre os 10 valores obtidos.

Observe que nos testes realizados, raras vezes a versão acoplada das listas de cidades candidatas superou os resultados obtidos com a versão original. Como a utilização de listas de cidades candidatas é um recurso que costuma ser empregado para diminuir substancialmente o tempo requerido pelo algoritmo, os tempos consumidos por esta versão modificada apresentaram-se menores que os consumidos pela versão original, porém, note que ambas realizaram a mesma quantidade de iterações.

Instância	Número de Cidades	Número de Grupamentos	Nossa Metaheurística sem Listas de Cidades Candidatas	Nossa Metaheurística com Listas de Cidades Candidatas
1	100	4	4574.592	4583.477
2	100	4	3524.966	3531.628
3	100	4	3560.399	3628.201
4	200	9	5315.525	5304.175
5	200	9	6084.859	6128.009
6	200	9	6538.798	6663.413
7	300	9	6448.339	6491.813
8	300	9	6314.149	6336.124
9	300	9	6471.555	6427.233
10	500	25	9773.473	9906.732
11	500	25	9852.567	9974.439
12	500	25	10344.837	10291.993

Figura 10.4: Tabela comparativa dos custos das melhores soluções obtidas por nossa metaheurística com e sem listas de cidades candidatas acopladas à colônia de formigas.

Outros testes foram realizados para avaliar o efeito da inicialização da colônia de formigas com soluções obtidas pelo AG. Para efetuar a avaliação comparamos os resultados obtidos pela colônia de formigas pura e sem inicialização com os obtidos pela nossa metaheurística, que equivalem aos resultados obtidos pela colônia de formigas pura, inicializada com soluções geradas pelo AG. Realizamos os testes para duas versões da colônia, uma com lista de candidatas e outra sem. Da mesma forma, testamos

as duas versões da metaheurística. Os valores dos parâmetros utilizados para as versões da metaheurística foram os mesmos apresentados na tabela da figura 10.1, excetuando-se o do parâmetro *número_de_iterações_da_colônia* cujo valor, desta vez, foi 5000. Para a execução das versões da colônia de formigas pura, foi utilizado o subconjunto de valores de parâmetros da metaheurística correspondentes aos parâmetros da colônia de formigas, apresentados na tabela da figura 10.1, porém, também neste caso o valor do parâmetro *número_de_iterações_da_colônia* foi distinto: 2500 iterações. Executamos 10 vezes cada versão da metaheurística e cada versão da colônia de formigas pura, todas para uma mesma instância do CTSP gerada aleatoriamente, com 200 cidades e 9 agrupamentos. Os resultados médios destes testes são ilustrados nos gráficos das figuras 10.5 e 10.6, que apresentam respectivamente, comparações entre valores das soluções e tempos totais.

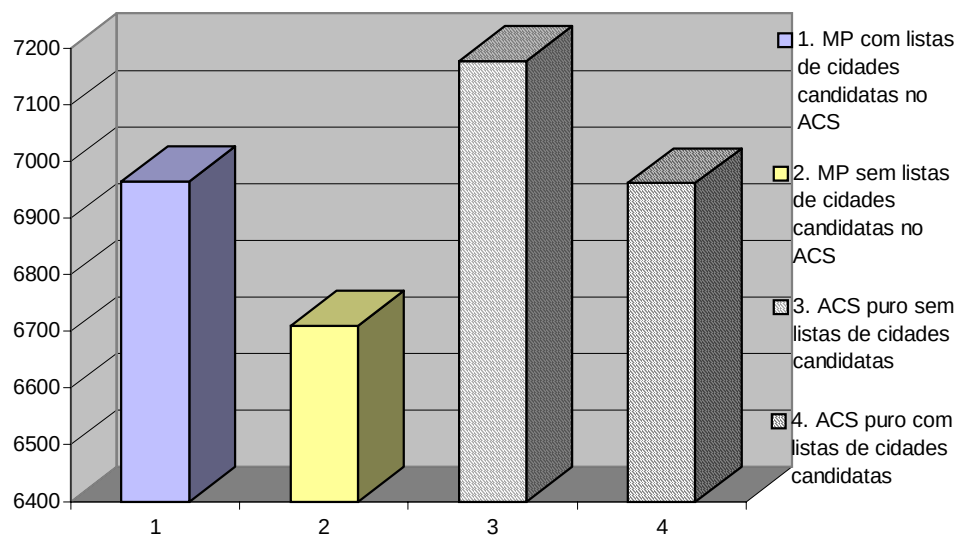


Figure 10.5: Gráfico mostrando os valores médios das soluções obtidas pelas duas versões da Metaheurística Proposta (MP) e da colônia de formigas pura (ACS puro), ambas com e sem listas de cidades candidatas.

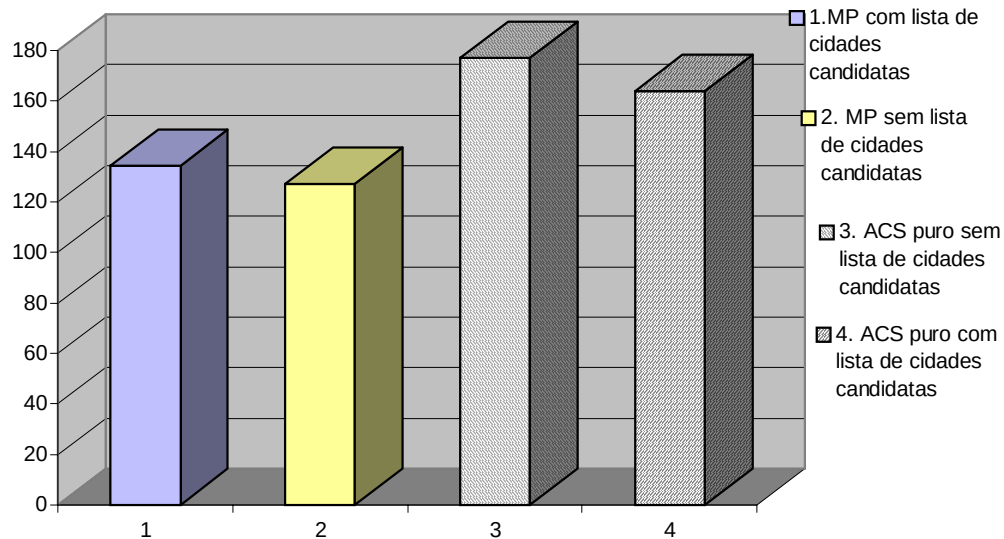


Figure 10.6: Gráfico mostrando os tempos totais médios consumidos pelas duas versões da Metaheurística Proposta (MP) e da colônia de formigas pura (ACS puro), ambas com e sem listas de cidades candidatas.

Os gráficos sugerem que inicializar o *Ant Colony System* com soluções de boa qualidade obtidas pelo AG, melhora seu desempenho tanto na qualidade da solução obtida como no tempo computacional consumido. Note que mesmo tendo o ACS da metaheurística sido executado a metade das iterações do ACS puro, a média dos valores obtidos por ela foi melhor que a média obtida pelo ACS puro, comparando-se as versões sem listas de cidades candidatas; e a média dos valores obtidos pela versão com listas de candidatas, apesar de ser pior que a obtida pelo ACS puro é muito próxima dela. A melhora no tempo era de se esperar, visto que ao inicializar a colônia de formigas o valor da primeira solução que ela constrói é muito próximo ao melhor valor das soluções com as quais foi inicializada.

10.2 Conclusões

A partir dos testes apresentados nas seções anteriores, pudemos formular alguns pontos interessantes.

Levando-se em consideração os gráficos das figuras 10.5 e 10.6 percebemos que inicializar o *Ant Colony System* com alguma informação adicional sobre o problema (por exemplo, soluções de boa qualidade obtidas pelo AG), melhora seu desempenho tanto na qualidade da solução obtida como no tempo computacional consumido. Por

outro lado, os dados da tabela da figura 10.4 mostram que a utilização das listas de cidades candidatas quando o ACS é inicializado com conhecimento prévio, não melhora seu desempenho. Talvez a informação contida nas soluções iniciais também possa ser utilizada para a construção das listas de cidades candidatas, visando melhorar a eficiência destas listas.

Outro ponto observado, é o fato de que a união entre a rapidez dos algoritmos genéticos em detectar regiões promissoras e a facilidade do *Ant Colony System* em aproveitar conhecimento prévio é uma forma alternativa para resolver problemas combinatórios e, nos testes iniciais, os resultados obtidos por esta metaheurística foram bastante encorajadores. Talvez seja possível generalizar e assim, reforçamos a suposição de que a união entre algoritmos com características complementares, é um enfoque que merece ser considerado. Isto abre uma ampla possibilidade de estudos para outras combinações de algoritmos e características. Mais uma vez um conceito de uma teoria baseada na natureza aparece como inspiração para o desenvolvimento de novas técnicas para a solução de problemas: a combinação de boas características de indivíduos (agora algoritmos) diferentes tem potencial de gerar indivíduos melhores que os originais.

10.3 Trabalhos Futuros

A seguir, apresentamos algumas sugestões para extensões de nosso trabalho:

- Desenvolvimento de operadores genéticos adequados para serem usados com o algoritmo genético proposto.
- Combinação do algoritmo genético proposto com outros mecanismos capazes de trabalhar com várias soluções iniciais, tais como *scatter search* e *path relinking*.
- Aplicação do algoritmo genético proposto a outros problemas.
- Acoplamento de listas de cidades candidatas construídas com base em soluções iniciais, ao ACS.
- Paralelização da metaheurística proposta.

APÊNDICE A

Uma Visão Histórica Simplificada do Aprendizado Mediante Reforços

Embora o termo específico "Aprendizado Mediante Reforços" nunca tenha sido usado por psicólogos, as raízes do aprendizado mediante reforços estão nas teorias sobre aprendizado desenvolvidas por psicólogos experimentais ao longo deste século. Nos concentraremos nas explorações iniciais somente do poder computacional do aprendizado mediante reforços, tentando não obscurecer o fato de que perspectivas computacionais e psicológicas são as vezes difíceis de distinguir.

Na década de 60, os termos “reforços” e "Aprendizado Mediante Reforços” aparecem na literatura de engenharia pela primeira vez: em [MINSKY, 61], [WALTZ, 65] e [MENDEL, 66]. Estes termos são usados para se referir a idéia geral do aprendizado através de recompensas e punições: aprendizado por tentativa e erro, onde ações seguidas de resultados bons ou ruins são, respectivamente, fortalecidas ou enfraquecidas.

Esta primeira noção do RL é um espelho exato de um princípio clássico de psicologia, a “Lei do Efeito” (“*Law of Effect*”) de Thorndike (1911):

“Das várias respostas possíveis para uma mesma situação, as que são acompanhadas ou seguidas de perto por satisfação para o animal vão, outras coisas sendo iguais, ser mais firmemente conectadas com a situação, de forma que, quando esta ocorrer periodicamente, elas serão mais prováveis de ocorrer; as respostas que são acompanhadas ou seguidas de perto por desconforto para o animal vão, outras coisas sendo iguais, ter suas conexões com a situação enfraquecidas, de forma que, quando esta ocorrer periodicamente, elas serão

menos prováveis de ocorrer. Quanto maior a satisfação ou desconforto, maior o fortalecimento ou o enfraquecimento do laço.”

Apesar deste princípio ter gerado considerável controvérsia em psicologia, assim como em outros campos, ao longo dos anos, ele permanece influente, pois sua idéia geral é suportada por muitas experiências, além do fato de ter muito sentido e ser intuitivo. É uma maneira elementar e óbvia de combinar busca e memória: busca na forma de tentar muitas ações e memória na forma de lembrar as ações que obtiveram melhores resultados.

As primeiras investigações computacionais sobre a “Lei do Efeito” conhecidas, foram de Minsky e de Farley e Clark, ambas publicadas em 1954. Na sua dissertação de Doutorado, Minsky [MINSKY, 54] descreve de uma máquina analógica, o *SNARC* (*Stochastic Neural-Analog Reinforcement Calculator*) que foi projetado para aprender por tentativa e erro. Farley e Clark [FARLEY, 1954] descrevem outra máquina de aprendizado através de rede neural, porém, notando sua habilidade para “generalizar”, a discutem mais em termos de aprendizado supervisionado do que em termos de aprendizado mediante reforços. Isto gerou uma confusão sobre a relação entre estes tipos de aprendizado. Muitos pesquisadores passaram a acreditar que estavam estudando aprendizado mediante reforços quando na verdade estudavam aprendizado supervisionado. Esta confusão persiste até hoje. Até mesmo livros modernos sobre redes neurais freqüentemente descrevem redes que aprendem através de exemplos de treinamento como sistemas de aprendizado por tentativa e erro, pois elas utilizam informações de erro para atualizar pesos de conexões. Mesmo sendo esta, uma confusão compreensível, o caráter de aprendizado pela Lei do Efeito, que é o significado original do termo "aprendizado por tentativa e erro", perde-se substancialmente.

É perceptível que pioneiros em redes neurais como Rosenblat [ROSENBLAT, 61] e Widrow e Hoff [WIDROW, 60], assim como os psicólogos Bush e Mosteller [BUSH, 55], estavam pensando em aprendizado mediante reforços, pois eles usaram a linguagem de recompensas e punições. Entretanto os sistemas por eles estudados tornaram-se, mais

claramente, sistemas de aprendizado supervisionado, satisfatórios para reconhecimento de padrões porém não para interação direta com um ambiente.

Nas décadas de 60 e 70, o aprendizado mediante reforços foi gradualmente obscurecido e perdido como um tópico distinto, enquanto que o aprendizado supervisionado, particularmente na forma de reconhecimento de padrões, passou a ser amplamente estudado. Outras claras exceções para esta tendência merecem ser mencionadas: Em 1963, Andreae descreveu uma máquina de aprendizado mediante reforços chamada *STeLLA*, que incluía o que nós hoje chamamos de um modelo ambiental para facilitar o aprendizado. Andreae estava explicitamente preocupado sobre como uma máquina poderia aprender interagindo com seu ambiente. Desenvolvimentos posteriores incluíram um “monólogo interno” para lidar com o problema da observação de estado parcial, o que ainda é uma importante questão para o aprendizado mediante reforços. Embora o trabalho posterior de Andreae tenha continuado a enfatizar o aprendizado através de interação, este trabalho passou a dar mais importância ao papel do professor. A pesquisa pioneira de Andreae não é muito conhecida, mas ainda contém lições para a pesquisa moderna sobre aprendizado mediante reforços.

Também na década de 60, Donald Michie enfocou o aprendizado mediante reforços. Ele descreveu um sistema simples de aprendizado mediante reforços para se jogar *Tic-Tac-Toe* chamado *MENACE* (*Matchbox Educable Noughts and Crosses Engine*). Consistia de uma caixa de fósforo para cada possível posição do jogo contendo um número de contas coloridas, uma cor para cada movimentação possível a partir daquela posição. Para determinar um movimento de *MENACE*, bastava desenhar uma conta qualquer na caixa de fósforo correspondente a posição do jogo. Quando um jogo terminava, contas eram adicionadas ou removidas das caixas usadas durante o jogo para recompensar ou punir as decisões de *MENACE*. *MENACE* seria lembrado como uma coleção de autômatos aprendizes estocásticos simples. Em 1968, Michie e Chambers [MICHIE, 68] descreveram um aprendiz mediante reforços de *Tic-Tac-Toe* mais avançado, denominado *GLEE* (*Game Learning Expectimaxing Engine*) que estimava uma função de valor usando o que eles chamaram de “*Expectimaxing*”. Hoje, *GLEE* é considerado intimamente ligado à Programação Dinâmica. Michie e Chambers [MICHIE, 68] também

descreveram uma versão mais avançada do *MENACE*, implementada em um sistema chamado *BOXES*, que foi aplicado ao problema de aprender a equilibrar um poste preso a um carro em movimento, fornecendo sinais de fracasso somente quando o poste caía ou o carro alcançava o fim do caminho. Este trabalho foi parcialmente inspirado pelo sistema de equilíbrio de Widrow e Smith [WIDROW, 64], que aprendia através de aprendizado supervisionado por um professor já capaz de acompanhar a tarefa. (Comparar o sistema de equilíbrio do poste de Widrow e Smith com o de Michie e Chambers é uma ótima maneira para se começar a perceber as diferenças entre aprendizado supervisionado e aprendizado mediante reforços.) *BOXES*, que não estimava uma função do valor, inspirou o sistema de equilíbrio de poste de Barto, Sutton e Anderson [BARTO, 83] que estimava uma função do valor. (Estes sistemas foram seguidos por outros sistemas de aprendizado mediante reforços de equilíbrio de poste muito numerosos para serem mencionados.)

Embora Widrow e seus colegas enfatizassem claramente o aprendizado supervisionado, ele reconheceu como ele diferia do aprendizado mediante reforços. Em 1973, Widrow, Gupta e Maitra [WIDROW, 73] modificou a regra *Widrow-Hoff* de aprendizado supervisionado (frequentemente chamada de regra *LMS* (*Least-Mean-Square*)) para produzir uma regra de aprendizado mediante reforços que poderia aprender com sinais de sucesso e fracasso, ao invés de com exemplos para treinamento. Esta forma de aprendizado foi chamada de “*selective bootstrap learning*” e a frase “aprendendo com um crítico” passou a ser usada no lugar de “aprendendo com um professor”.

Outro pesquisador que resistiu à tendência do aprendizado supervisionado foi Harry Klopf [KLOPF, 72]; [KLOPF, 82], que apresentou uma teoria “hedônica” e “heterostática” sobre a função Neural e a Inteligência Artificial. Klopf reconheceu que aspectos essenciais do comportamento adaptativo estavam sendo perdidos conforme os estudantes de pesquisa passaram a focalizar quase que exclusivamente o aprendizado supervisionado. Estes são os aspectos hedônicos: a direção para alcançar um resultado do ambiente e controlá-lo para fins desejados, evitando fins indesejados.

Importantes contribuições para o aprendizado mediante reforços foram feitas também por John Holland [HOLLAND, 75]; [HOLLAND, 86]. Embora mais conhecido por seu desenvolvimento de algoritmos genéticos, Holland esboçou uma teoria bastante geral de sistemas adaptativos que acentua o aprendizado interativo baseado em princípios seletivos. De fato, seu sistema classificador [HOLLAND, 86] é um sistema de aprendizado mediante reforços que atualiza funções de valor usando o que ele chamou de “*bucket brigade algorithm*”, intimamente ligado aos métodos de estimativa de função de valor. Embora os Algoritmos Genéticos sejam candidatos naturais para a implementação do que chamamos de aproximação evolucionária para o aprendizado mediante reforços; em contraste com métodos de função de valor, Holland não sugeriu esta aproximação. Os sistemas classificadores usam tanto Algoritmos Genéticos, como funções de valor. Embora a idéia de aprendizado através da estimativa de funções de valor sobre a experiência tenha aparecido na dissertação de Minsky [MINSKY, 54], ela foi mais influentemente introduzida por Samuel [SAMUEL, 59] no seu programa de aprendizado para jogar xadrez usando o que hoje seria classificado como um método de diferença temporal. O trabalho de Samuel foi uma influência seminal para a aproximação de aprendizado mediante reforços apresentada aqui. Seus artigos [SAMUEL, 59]; [SAMUEL, 67] revelam extraordinária perspicácia para quase todas as questões que ainda desafiam pesquisadores atuais. Embora a dissertação de Minsky tenha sido um trabalho precoce sobre o aprendizado mediante reforços, mais influente foi seu artigo de 1961 “*Steps Toward Artificial Intelligence*” (Passos para Inteligência Artificial)[MINSKY, 61] que, como os artigos de Samuel contém irresistíveis discussões de questões que ainda são relevantes ao moderno aprendizado mediante reforços. Merece atenção especial, a discussão de Minsky sobre o que ele considera ser o principal problema que os sistemas de aprendizado mediante reforços deveriam resolver para se tornarem bem sucedidos. Ele o chamou de *credit-assignment problem* (problema da alocação de crédito):

“Aplicando-se estes métodos para programas complexos, encontramos uma séria dificuldade - distribuir créditos para o sucesso de uma complexa estratégia entre as várias decisões envolvidas.”

Se as várias decisões envolvidas no alcance do sucesso devem ser reforçadas, então suas relativas contribuições para este alcance devem ser avaliadas. Minsky considerou o método de estimativa de função de valor utilizado pelo jogador de xadrez de Samuel como uma importante aproximação a este problema, apontando que ele está intimamente relacionado com o fenômeno do reforço condicionado que ocorre no aprendizado animal.

O breve histórico apresentado nos mostra que as principais idéias sobre aprendizado mediante reforços estão presentes em Inteligência Artificial desde seus primórdios. Entretanto, apenas recentemente elas vem sendo difundidas e tem atraído a atenção. Uma das razões para que estas idéias tenham tido pouca influência inicialmente é sua associação com visões comportamentais de aprendizado e inteligência. A pesquisa em Inteligência Artificial na década de 60 seguiu as áreas aliadas de psicologia, trocando de aproximações baseadas no comportamento animal para aproximações mais cognitivas, deixando pouco espaço para as teorias de reforços; de fato, deixando pouco espaço para quaisquer teorias de aprendizado. Assim como não se pode entender ou gerar **todo** o comportamento inteligente, utilizando-se apenas aprendizado mediante reforços, também não se pode fazê-lo, somente através das teorias cognitivas e dos sistemas de IA baseados nos outros princípios básicos da IA. Realmente o clima tornou-se consideravelmente mais morno para os princípios clássicos de IA, incluindo aprendizado mediante reforços, pois os pesquisadores estão usando-os em sistemas baseados mais em perspectivas cognitivas do que nas primeiras teorias sobre o comportamento animal.

Um fator relacionado que limitou a influência dos princípios de aprendizado mediante reforços em IA é a reputação de que eles são computacionalmente muito fracos para ter qualquer utilidade. Entretanto, atualmente, existe uma ampla evidência de que o aprendizado mediante reforços pode ser muito poderoso. Alguns dos mais impressionantes resultados de aprendizado artificial, foram alcançados através de aprendizado mediante reforços.

Bibliografia

- [AHUJA, 93] Ravindra K. Ahuja, Thomas L. Magnanti and James B. Orlin (1993). "Network Flows: theory, algorithms, and applications", Prentice-Hall, New Jersey. - ISBN 0-13-617549-X.
- [ALOIMONOS, 93] Aloimonos, Yiannis, editor (1993), "Active Vision", Lawrence Erlbaum Associates, Inc.
- [ANILY,98] Shoshana Anily, Julien Bramel and Alain Hertz (1998), "A 5/3-Approximation Algorithm for the Clustered Traveling Salesman Tour and Path Problems", anily@post.tau.ac.il, jdb@columbia.edu, hertz@dma.epfl.ch
- [ARKIN, 97] Esther M. Arkin, Refael Hassin and Limor Klein (1997), "Restricted Delivery Problems on a Network", Networks 29, pages. 205-216.
- [BÄCK, 96] Bäck, T. (1996), "Evolutionary Algorithms in Theory and Practice: Evolutionary Strategies, Evolutionary Programming, Genetic Algorithms", Oxford University Press.
- [BARTO, 83] Barto, AG, Sutton, RS, & Anderson, CW (1983). Neuronlike elements that can solve difficult learning control problems. IEEE Transactions on Systems, Man, and Cybernetics, 13, 835--846. Reprinted in J. A. Anderson and E. Rosenfeld, Neurocomputing: Foundations of Research, MIT Press, Cambridge, MA, 1988.
- [BARTO, 91] Barto, A. G.; Sutton, R. S. and Watkins, C . J. C. H. (1991), "Learning and Sequêncial Decision Making". In M. Gabriel and J. W. Moore, editors, Learning and Computational Neuroscience. MIT Press, Cambridge.
- [BATES, 92] Bates. J., Bryan Loyall A. And Scott Reilly W., (1992). "Integrating reactivity, goals and, emotion in a broad agent". Technical Report CMU-CS-92-142, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.
- [BEASLEY, 93] David Beasley, David R. Bull & Ralph R. Martin (1993) – "An Overview of Genetic Algorithms Part 1.", Fundamentals, University Computing, Vol. 15(2), pgs 58-69.
- [BECKERS, 92] Beckers, R.; Deneubourg, J. L. and Goss, S. (1992), "Trails and U-turns in the Selection of the shortest path by the Ant *Lasius niger*". Journal of Theoretical Biology, 159, 397 - 415.
- [BELLMAN, 57] Bellman, R., (1957). "Dynamic Programming". Princeton University Press.
- [BLICKLE, 95] Blickle, T.B. and Thiele, L. (1995), "A Comparasion of Selection Schemes used in Genetic Algorithms", TIK-Report N° 11, Computer Engineering and Communications Networks Labs, Swiss Federal Institute of Technology, Zurich - Switzerland.
- [BONDY, 76] Bondy, J. A. and Murty, U. S. R., (1976). "Graph theory with applications", North Holland, New York - Oxford

- [BOOKER, 88] Booker, Lashon (1988). "Classifier Systems that Learn Internal World Models", Machine Learning Journal, Volume 1, Number 2,3.
- [BUSH, 55] Bush, RR & Mosteller, F (1955). Stochastic Models for Learning. New York: Wiley.
- [CAMPELLO, 94] Campello, Ruy E.; Maculan, N; "Algoritmos e Heurísticas". Editora da Universidade Federal Fluminense, Rio de Janeiro, Brasil, (1994).
- [COLORNI, 91A] Colorni A., Dorigo M., Maniezzo V.: "An Autocatalytic Optimizing Process", Technical Report n. 91-016 Politecnico di Milano, 1991.
- [COLORNI, 91B] Colorni A., Dorigo M., Maniezzo V.; "Distributed Optimization by Ant Colonies"; Proceedings of European Conference on Artificial Life (ECAL91), Elsevier Publishing, Paris-France, 1991; pag.134-142
- [CORMEN, 90] Cormen, Thomas H.; Leiserson, Charles E. and Rivest, Ronald L. (1990). "Introduction to Algorithms". MIT Press.
- [CHATTERJEE, 96] Sangit Chatterjee, Cecília Carrera & Lucy A. Lynch (1996) - "Genetic Algorithms and Travelling Salesman Problem", - European Journal of Operational Research, Vol. 93, pgs. 490-510.
- [CHISMAN, 75] Chisman, J.A. (1975), "The Clustered Traveling Salesman Problem", Computer & Operations Research 2, 115 - 119.
- [CHRISTÓFIDES, 79] N. Christofides, - "Travelling Salesman Problem", In Combinatorial Optimization (1979), John Wiley & Sons Inc., New York, pgs 131-149.
- [CHUNG, 83] Chung, Kai Lai. (1983). "Teoría elemental de la probabilidad y de los procesos estocásticos". Barcelona, Ed. Reverté, 3ed.
- [DANTZIG, 54] Dantzig, G.B.; Fulkerson, D.R.and Johnson, S.M.; "Solutions of large scale traveling salesman problem", Oper. Res. 2, 393-410. (1954)
- [DENEBOURG, 83] Denebourg J.L., Pasteels J.M., Verhaeghe J.C.: "Probabilistic Behaviour in Ants: a Strategy of Errors?", J. theor. Biol., 105, 1983, pag. 259-271.
- [DENEBOURG, 89] Denebourg J.L., Goss S.: "Collective patterns and decision-making", Ethology, Ecology & Evolution, 1, 1989, pag. 295-311.
- [DORIGO, 92] Dorigo, M., (1992). "Optimization, Learning and Natural Algorithms". PhD. Thesis, Politecnico di Milano, Italy, Eu.
- [DORIGO, 92] Dorigo, M. (1992), "Optimizations, Learning and Natural Algorithms". Ph.D. Thesis, Politecnico di Milano, Italy, EU. (In Italian)
- [DORIGO, 96A] Dorigo, M. and Gambardella, L.M. (1996), "Ant Colonies for the traveling Salesman Problem", Accepted for publications in BioSystems, 1997. In press.

- [DORIGO, 96B] Dorigo, M. and Gambardella, L. M. (1996), "A Study of some Properties of Ant-Q", Proceedings of PPSN IV-Fourth International Conference on Parallel Problem Solving From Nature, H.-M. Voigt, W. Ebeling, I. Rechenberg and H.-S. Schwefel (Eds.), Springer-Verlag, Berlin, 656 - 665.
- [DORIGO, 97] Dorigo, M. and Gambardella, L.M. (1997), "Ant Colonies for the traveling Salesman Problem", Accepted for publications in BioSystems, 1997. In press.
- [DRESCHER, 91] Drescher, G. (1991). "Made Up Minds: A Constructivist Approach to Artificial Intelligence". MIT Press.
- [EILON, 69] Eilon S., Watson-Gandy T.H., Christofides N.: "Distribution management: mathematical modeling and practical analysis", Operational Research Quarterly, 20, 1969.
- [EVANS, 87] Evans, James R; "Structural Analysis of local search heuristics in Combinatorial Optimizations", Computing & Operations Research vol. 14, No. 6, 465-477 (1987)
- [FARLEY, 54] Farley, BG & Clark, WA (1954). Simulation of self-organizing systems by digital computer. IRE Transactions on Information Theory, 4, 76--84.
- [FISCHETTI, 95] M. Fischetti, J. J. Salazar and P. Toth (1995), "The symmetric generalized traveling salesman polytope", Networks, 26, 113-123.
- [FITZGERALD, 88] Fitzgerald T.D., Peterson S.C.: "Cooperative foraging and communication in caterpillars", Bioscience, 38, 1988, pag. 20-25.
- [FONER, 94] Foner, Leonard N.; "Paying Attention To What's Important: Using Focus of Attention to Improve Unsupervised Learning", Mcs. Thesis in Media Arts and Sciences, Massachusetts Institute of Technology, June (1994).
- [GALLIERS, 88] Galliers, J. R. (1988). "A theoretical Framework for Computer Models of Cooperative Dialogue, Acknowledging Multi-Agent Conflict", PhD thesis, Open University, UK.
- [GAMBARDELLA, 97A] Gambardella, L.M.; Dorigo, M. (1997), "Solving Symmetric and Asymmetric TSP's by Ants Colonies", presented to the IEEE Conference on Evolutionary Computation (ICEC'96), May 20 - 22, 1996, Nagoya, Japan.
- [GAMBARDELLA, 97B] Gambardella, L.M.; Dorigo, M. (1997), "Ant-Q: A Reinforcement Learning approach to the Traveling Salesman Problem", Technical Report, IRIDIA, Université Libre de Bruxelles.
- [GAREY, 79] M. R. Garey & D. S. Johnson (1979), - "Computers and Intractability: A guide to the theory of NP-Completeness" – Ed. Freeman, San Francisco.
- [GARY, 79] Gary, M. R. and Johnson, D. S. (1979); "Computers and Intractability: A guide to the theory of NP-Completeness" – Ed. Freeman, San Francisco.
- [GATHERCODE, 98] Gathercode, Chris. (1998); "An Investigation of Supervised Learning in Genetic Programming", Ph.D. thesis, University of Edinburgh.

- [GENDREAU, 92] M. Gendreau, A. Hertz & G. Laporte (1992), "New insertion and post-Optimization Procedures for the Traveling Salesman Problem" – Operations Research, Vol. 40(6), pages 1086-1094.
- [GENDREAU, 94] Gendreau, M.; Laporte, G. and Potvin, J. Y. (1994), "Heuristics for the Clustered Traveling Salesman Problem", Technical Report CRT-94-54, Centre de recherche sur les transport, Université de Montréal.
- [GENDREAU, 96] M. Gendreau, G. Laporte and J-Y. Potvin (1996), "Heuristics for the Clustered Traveling Salesman Problem". Combinatorial Optimization 1, 41-56.
- [GENDREAU, 97] M. Gendreau, A. Hertz and G. Laporte (1997), "An Approximation Algorithm for the Traveling Salesman Problem with Backhauls". Operations Research 45, pages 639-641.
- [GLOVER, 89] Glover F.: "Tabu Search: Part I", ORSA Jou. on Computing, 1, 1989.
- [GLOVER, 90] Glover F.: "Tabu Search: Part II", ORSA Jou. on Computing, 2, 1990.
- [GOLDBERG, 89] Golberg, D. (1989), "Genetic Algorithms in search". Optimization and Machine Learning, Addison Wesley, Reading, MA.
- [GOLDEN, 84] B. Golden, L. Levy & R. Dahl (1981), - "Two Generalizations of the Travelling Salesman Problem", Omega, Vol. 19, N° 4,
- [GOLDEN, 85] Golden B., Stewart W.: "Empiric analysis of heuristics", in Lawler E.L., Lenstra J.K., Rinnooy-Kan A.H.G., Shmoys D.B.: "The Travelling Salesman Problem", Wiley, 1985, pag. 228.
- [GOODWIN, 93] Goodwin, R., (1993). "Formalizing properties of agents". Theoretical Report CMU-CS-93-159, School of Computers Science, Carnegie-Mellon University, Pittsburgh, PA.
- [GOSS, 89] Goss, S.; Aron, S., Deneubourg, J. L. and Pasteels, J. M. (1989), "Self-organized shortcuts in the Argentine Ant", Naturwissenschaften, 76, 579–581.
- [GOSS, 90] Goss S., Beckers R., Deneubourg J.L., Aron S., Pasteels J.M.: "How Trail Laying and Trail Following Can Solve Foraging Problems for Ant Colonies", in Hughes R.N. (ed.), NATO ASI Series, Vol. G 20, Behavioural Mechanisms of Food Selection, Springer-Verlag, Berlin, 1990.
- [GREFENSTETTE, 92] Grefenstette, J. J., K. A. De Jong, and William M. Spears (1992). Competition-Based Learning, chapter 6 in Foundations of Knowledge Acquisition: Machine Learning, 203-225. Alan Meyrowitz and Susan Chipman (editors), Kluwer Academic Publishers.
- [GRÖTSCHEL, 91] Grötschel, Martin and Holland, Olaf; "Solutions of large-scale Symetric Traveling Salesman Problem", Mathematical Programming 51, 141-202 (1991).
- [GUTTMAN-BECK, 97] N. R. Guttman-Beck, S. Khuller and B. Raghavarachi (1997), "Approximation Algorithms with Bounded Performance Guarantees for The Clustered Traveling Salesman Problem", Working paper, Department of Statistics and Operations Research, Tel-Aviv University.

- [HARMON, 97] Harmon, Mance E.; Harmon, Stephanie; (1997). "Reinforcement Learning: A Tutorial", Wright Laboratory, e-mail: mharmon@acm.org. <http://www-anw.cs.umass.edu/~mharmon/rltutorial/noframes.html#reinforcement>
- [HAYES-ROTH, 90] Hayes-Roth, B. (1990). "Architectural foundations for real-time performance in intelligent agents". The Journal of Real-Time Systems, 2:29-125.
- [HELD, 70] M. Held and R. M. Karp (1970), "The traveling salesman problem and minimum spanning trees", Operations Research 18 (1970), 1138-1162.
- [HELD, 71] M. Held and R. M. Karp (1971), "The traveling salesman problem and minimum spanning trees: Part II", Mathematical Programming 1 (1971), 6-26
- [HOLLAND, 75] Holland, John H. (1975), "Adaption in natural and artificial system" , MIT Press.
- [HOLLAND, 86] Holland, John H. (1986). "Escaping Brittleness: The Possibilities of General-Purpose Learning Algorithms Applied to Parallel Rule-Based Systems", Machine Learning: An Artificial Intelligence Approach, Volume 2, R S Michalski, J G Carbonell, and T M Mitchell (eds), Morgan Kauffman, Los Altos, CA..
- [HOLLAND, 86] Holland, JH (1986). Escaping brittleness: The possibility of general-purpose learning algorithms applied to rule-based systems. In: Machine Learning: An Artificial Intelligence Approach, Volume II, (RS Michalski, JG Carbonell, & TM Mitchell, eds) pp. 593--623. San Mateo, CA: Morgan Kaufmann.
- [HOLLAND, 98] Holland, John H.; "Emergence: From Chaos to Order". Addison-Wesley, Redwood City, California, USA, (1998).
- [HÖLDOBLER, 90] Hölldobler, B. and Wilson, E. O. (1990), "The Ants", Springer-Verlag, Berlin.
- [HOROWITZ, 96] HOROWITZ, Ellis; SAHNI, Sartaj; RAJASEKARAN, Sanguthevar, "Computer Algorithms/C++". Computer Science Press, New York, (1996)
- [JOHNSON, 74] Johnson, D.S.; "Approximation Algorithms for Combinatorial Problems", J. Compt. Systems Sci., v.9, pags. 256-278, (1974).
- [JOHNSON] Johnson, D. S.; and McGeoch, in press. The Travelling Salesman Problem: A Case Study in Local Optimizations. In Local Search in Combinatorial Optimization, E.H.L. Aarts and J.K. Lenstra (Eds). Wiley, New York
- [JONGENS, 85] K. Jongens and T. Volgenant (1985), "The Symmetric Clustered Traveling Salesman Problem", European Journal Of Operations Research 19, 68-75.
- [JONKER, 84] R. Jonker and A. Volgenant (1984), "Non-optimal edges for the symetric traveling salesman problem", to appear in Operations Research (1984).
- [KAELBLING, 96] Kaelbling, Leslie P.; Littman, Michael L. and Moore, Andrew W.; (1996). "Reinforcement Learning: A Survey", Journal **¡Error!Marcador no definido.** of Artificial Intelligence Research, Volume 4, pags. 237-285.
- [KAELBLING, 93] Kaelbling, Leslie P. (1993). "Learning in Embebed Systems". MIT Press.

- [KARP, 72] R. M. Karp (1972), - "Reducibility Among Combinatorial Problems" – Complexity of Computations, R. E. Miller & J. W. Thatcher (Eds), Plenum. New York, pgs 85-104.
- [KLOPF, 72] Klopf, AH (1972). Brain function and adaptive systems - A heterostatic theory. Technical Report AFCRL-720164 Air Force Cambridge Research Laboratories Bedford, MA.
- [KLOPF, 82] Klopf, AH (1982). The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence. Washington, D.C.: Hemisphere.
- [LAGEVIN, 90] LAGEVIN, André; SOUMIS, François; DESROSIERS, Jacques, "Classification of Traveling Salesman Problem Formulations", Operations Research Letters 9, 127-132 (1990).
- [LAPORTE, 83] G. Laporte and Y. Norbert (1983), "Generalized traveling salesman through n sets of nodes: An integer programming approach". INFOR, 21, 62-75.
- [LAPORTE, 87] G. Laporte, H. Mercure and Y. Norbert (1987), "Generalized traveling salesman problem through n sets of nodes: The asymmetrical cases", Discrete Appl. Math. 18, 185-197.
- [LAWLER, 85] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnoy Kan and D. B. Shmoys (Eds). (1985), "The traveling salesman problem: A guided tour of combinatorial optimization", Wiley, New York
- [LEERINK, 96] Leerink, R. L.; Schultz, S. R. and Jabri, M. A. (1996), "A Reinforcement Learning Exploration Strategy based on Ant Foraging Mechanism", Technical Report, Australian Gilt Securities LTD, 37 Pitt Street, Sydney NSW 2000, Australia.
- [LIN, 73] S. Lin & W. Kerningham (1973) – "Na Effective Heuristic Algorithm for Traveling Salesman Problem", Operation Research, Vol 21, pgs 498-516.
- [LITTLE, 63] J. D. C. Little, K. G. Murty, D. W. Sweeney and C. Karel (1963), "Na algorithm for the traveling salesman problem", Operations Reserach 11 (1963), 972-989.
- [LOKIN, 78] F. C. J. Lokin (1978), "Procedures for the traveling salesman problems with additional constraint", European Journal of Operations Research 3 (1978), 135-141
- [MAES, 92] Maes, P., (1992). "Learning Behavior Networks from Experience", Toward a Practice of Autonomous Systems, Proceedings of the First European Conference on Artificial Life, edited by F J Varela and P Bourguine, MIT Press/Bradford Books.
- [MAES, 94A] Maes, P., 1994a. "Modeling Adaptative Autonomous Agents", Artificial Life Journal, Volume 1, Numbers 1/2 (Fall 1993/Winter 1994), Christopher G. Langton (ed), MIT Press, Cambridge, MA, 1994.

- [MAES, 94B] Maes, P., (1994) "Social interface agents: Acquiring competence by learning from users and other agents", in O. Etzioni(ed.) *Software Agents - Papers from the 1994 Spring Symposium* (Technical Report SS-94-03), pp.71-78, AAAI Press.
- [MANIEZZO, 94] Maniezzo, V.; Colomi, A.; Dorigo, M., "The Ant System Applied to the Quadratic Assignment Problem", Technical report, IRIDIA/94-28, Université Libre de Bruxelles, Belgium, EU. (1994)
- [MENDEL, 70] Mendel, JM & McLaren, RW (1970). Reinforcement Learning control and pattern recognition systems. In: *Adaptive, Learning and Pattern Recognition Systems: Theory and Applications*, (JM Mendel & KS Fu, eds), pages. 287-318. New York: Academic Press.
- [MICHALEWICZ, 92] Michalewicz, Z.; "Genetic Algorithms + Data Structures = Evolution Programs", Springer Verlag, (1992).
- [MICHIE, 68] Michie, D & Chambers, RA (1968). BOXES: An experiment in adaptive control. In: *Machine Intelligence 2*, (E Dale & D Michie, eds) pages. 137-152. Oliver and Boyd.
- [MINSKY, 54] Minsky, ML (1954). *Theory of Neural-Analog Reinforcement Systems and its Application to the Brain-Model Problem*. PhD thesis Princeton University.
- [MINSKY, 61] Minsky, ML (1961). Steps toward artificial Intelligence. *Proceedings of the Institute of Radio Engineers*, 49, 8--30. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*. McGrawHill, New York, 406-450, 1963.
- [NAKAMURA, 95] Nakamura, M. and Kurumatani, K., (1995). "A mathematical model for the foraging of an ant colony and pattern formation of pheromone trail". *Fundamental Theories of Deterministic and Stochastic Models in Mathematical Biology*, Institute of Statistical Mathematics, pp. 120-131.
- [NOON, 91] C. E. Noon and J. C. Bean (1991), "A lagrangean based approach for the asymmetric generalized traveling salesman problem", *Opns. Res.* 39, 623-632.
- [OCHI, 95A] Ochi, L. S.; Santos, E. M. and Montenegro, A. A. (1995), "A Genetic Algorithm for the Traveling Salesman Problem", *INFORM-95*, Los Angeles - USA.
- [OCHI, 95B] Ochi, L. S.; Maculan, N., Santos, E. M. and Montenegro, A. A. (1995), "A Genetic Algorithm for the Traveling Purchaser Problem", *Proceedings of the Metaheuristics Intern. Conf. (MIC95)*, USA. Editors: Osman, H & Kelly, J. P., Kluwer Academic Publ., pp. 52 - 57.
- [OCHI, 97] Luiz S. Ochi, P. G. Rabelo and Nelson Maculan (1995), "A New Genetic Metaheuristic for the Clustered Travelling Salesman Problem", *Proceedings of the III Metaheuristic International Conference (III MIC 97)*, Sophia Antipolis France.

- [POTVIN, 95] Potvin, J. Y. and Guertin, F. (1995), "A Genetic Algorithm for the Clustered Traveling Salesman Problem with a Priori Order on the Clusters", Technical Report CRT-95-05, Centre de recherche sur les transports, Université de Montréal.
- [POTVIN, 96] Jean-Yves Potvin (1996), "Genetic Algorithms for the Travelling Salesman Problem", *Annals of Operations Research*, Vol. 63, pgs. 339-370.
- [RABELO, 96] Rabelo, P. G.; Ochi, L. S., Maculan, N. (1996), "A New Genetic Algorithm for the Clustered Traveling Salesman Problem", VIII CLAIO, Rio de Janeiro - Brasil.
- [RECHENBERG, 65] Rechenberg, I.; "Cybernetic solution path of an experimental problem". Royal Aircraft Establishment, Library Translation No. 1122, Farnborough, Hants, UK. (1965)
- [REEVES, 93] Colin R. Reeves (1993), "Modern Heuristic Techniques for Combinatorial Problems", John Wiley & Sons, Inc., pags. 151-196.
- [ROSENBLAT, 61] Rosenblat, F (1961). *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. 6411 Chillum Place N.W., Washington, D.C.: Spartan Books.
- [ROSENSCHEIN, 85] Rosenschein, S. (1985), "Formal theories of knowledge in AI and Robotics", *New Generation Computing*, pages 345-357.
- [ROSIN, 97] Rosin, C. D. (1997), "Coevolutionary Search Among Adversaries", Ph.D. dissertation thesis, Department of Computer Science and Engineering, University of California, San Diego – USA
- [SAMUEL, 59] Samuel, AL (1959). Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development*, 210--229. Reprinted in E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, McGraw-Hill, New York, 1963.
- [SAMUEL, 67] Samuel, AL (1967). Some studies in machine learning using the game of checkers. II-Recent progress. *IBM Journal on Research and Development*, 601-617.
- [SCHWEFEL, 65] Schwefel, H-P.; "Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik". Diploma thesis, Technische Universität, Berlin. (1965)
- [SCHWEFEL, 99] Schwefel, Hans-Paul; "Strategies for the Real World", *Newsletter of EvoNet: EvoNews*, Issue 9, Winter 1999.
- [SEPEHRI, 91] M. M. Spehri (1991), "The symmetric generalized traveling salesman problem", Ph.D. Dissertation, University of Tennessee, Knoxville.
- [SHAPIRO, 88] Shapiro J.A.: "Bacteria as multicellular organisms", *Scientific American*, 1988, pag. 82-89.
- [SHOHAM, 93] Shoham, Y. (1993). "Agent-oriented programming". *Artificial Intelligence*, 60(1):51-92.

- [SPEARS, 92] Spears, William M. and Kenneth A. De Jong (1992). Using Genetic Algorithms for Supervised Concept Learning, a chapter in Artificial Intelligence Methods and Applications, Nikolaos G. Bourbakis (editor), World Scientific.
- [SPEARS, 97] Spears, William M. (1997). Recombination Parameters, to appear in The Handbook of Evolutionary Computation, T. Baeck, D. Fogel and Z. Michalewicz (editors), IOP Publishing and Oxford University Press.
- [SUTTON, 90] Sutton, Rich S., (1990). "Integrated Architectures for Learning, Planning and Reacting Based on Approximating Dynamic Programming", In Proceedings of the Seventh International Conference in Machine Learning, Austin, TX.
- [SUTTON, 98] Sutton, Richard S. and Barto, Andrew G., (1998); "Reinforcement Learning: An Introduction". MIT Press, Cambridge, MA. A Bradford Book. <http://envy.cs.umass.edu/~rich/book/the-book.html>
- [THRUN, 92] Thrun, B.T. (1992), "Efficient Exploration in Reinforcement Learning", Technical Report CMU-CS-92-102, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA.
- [VAN KEMENADA, 95] van Kemenada, C.H.M.; "A Two-Level evolution Strategy balancing global and local search", paper presented at the 5th Belgian-Dutch Conference on Machine Learning, Brussels. (1995)
- [VOLGENANT, 82] A. Volgenant and R. Jonker (1982), "A branch and bound algorithm for the symmetric traveling salesman problem based on the 1-tree relaxation", European Journal of Operations Research 9 (1982), 83-89.
- [VOLGENANT, 83] A. Volgenant and R. Jonker (1983), "The symmetric traveling salesman problem and edge exchanges in minimal 1-trees", European Journal of Operations Research 12 (1983), 394-403.
- [VOLGENANT, 87] Ton Volgenant & Roy Jonker (1987), - "On Solve Generalizations of the Travelling Salesman Problem", Journal Operation Research Society, Vol. 38, N^o 11, pgs. 1073-1079.
- [WALTZ, 65] Waltz, MD & Fu, KS (1965). A heuristic approach to reinforcement learning control systems. IEEE Transactions on Automatic Control, 10, 390-398.
- [WATKINS, 89] Watkins, C. J. C. H. (1989), "Learning with delayed rewards". Ph.D. dissertation, Psychology Department, University of Cambridge, England.
- [WHITLEY, 89] Whitley, D.; Starkweather, T. and Fuquay, D. (1989), "Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator", Proceedings of the Third Int. Conference on Genetic Algorithms, Fairfax, VA, 133-140.
- [WIDROW, 60] Widrow, B & Hoff, ME (1960). Adaptive switching circuits. In: 1960 WESCON Convention Record Part IV pp. 96-104.
- [WIDROW, 64] Widrow, B & Smith, FW (1964). Pattern-recognizing control systems. In: Computer and Information Sciences (COINS) Proceedings, Washington, D.C.: Spartan.

- [WIDROW, 73] Widrow, B, Gupta, NK, & Maitra, S (1973). Punish/reward: Learning with a critic in adaptive threshold systems. IEEE Transactions on Systems, Man, and Cybernetics, 5, 455--465.
- [WILSON, 85] Wilson, Stewart W., (1985). "Knowledge Growth in an Artificial Animal", Proceedings of the First International Conference on Genetic Algorithms and their Applications, edited by Greffenstette, Lawrence Erlbaum Associates.
- [WOOLDRIDGE, 94] M. Wooldridge and N. Jennings, (1994). "Formalizing the cooperative problem solving process". In Proceedings of the Thirteenth International Workshop on Distributed Artificial Intelligence (IWDAI-94), pages. 403-417, Lake Quinalt, WA.
- [WOOLDRIDGE, 95A] M. Wooldridge and N. Jennings, (1994). "Intelligent agents: theory and practice", in The Knowledge Engineering Review, Vol.10:2, 1995, 115-152
- [WOOLDRIDGE, 95B] M. Wooldridge (1995). "This is MyWorld: The logic of an agent-oriented testbed for DAL", In Wooldridge, M and Jennings, N. R., editors, Intelligent Agents: Theories, Architectures, and Languages (LNAI Volume 890), pages 160-178. Springer-Verlag: Heidelberg, Germany.
- [YURET, 96] Yuret, D. (1996), "From Genetic Algorithms to Efficient Optimizations", A.I. Technical Report N° 1569, Massachusetts Institute of Technology (MIT), USA.