

**UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA
COMPUTAÇÃO**

RANGEL GUSTAVO RAULINO

**UMA ABORDAGEM HÍBRIDA PARA SOLUCIONAR
PROBLEMAS DE OTIMIZAÇÃO ATRAVÉS DOS
ALGORITMOS: GENÉTICO E *SIMULATED ANNEALING***

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do grau de Mestre em Ciência da Computação

**Prof. José Mazzucco Junior, Dr.
Orientador**

Florianópolis, Maio de 2002

**UMA ABORDAGEM HÍBRIDA PARA SOLUCIONAR
PROBLEMAS DE OTIMIZAÇÃO ATRAVÉS DOS
ALGORITMOS: GENÉTICO E SIMULATED ANNEALING**

RANGEL GUSTAVO RAULINO

Esta dissertação foi julgada adequada para a obtenção do Título de


Mestre em Ciência da Computação

Área de Concentração: Sistemas de Computação, e aprovada em sua forma final pelo Curso de Pós-Graduação em Ciência da Computação – CPGCC




Prof. Fernando A. Osthuni Gauthier, Dr. – Coordenador

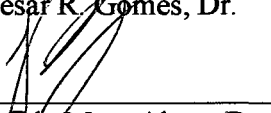
Banca Examinadora



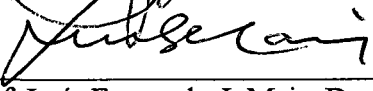
Prof. José Mazzucco Junior, Dr. – Orientador



Prof. Paulo César R. Gomes, Dr.



Prof. João Bosco da Mota Alves, Dr.



Prof. Luís Fernando J. Maia, Dr.

Dedico este trabalho à minha família, pelo apoio recebido,
não apenas neste momento, mas durante
toda minha vida acadêmica.

Agradecimentos

Primeiramente a Deus, pois ele é o responsável por tudo de bom que me acontece, inclusive a conclusão deste trabalho.

Ao professor José Mazzucco Junior pela orientação, mas acima de tudo pela amizade que ele demonstrou em todos os momentos.

A meus pais, Sirio José Raulino e Isalete Raulino, por toda a força e incentivo que me deram ao longo de minha vida.

A minha namorada, presença importante em minha vida, ela contribuiu em muito, me apoiando e incentivando durante todo o tempo.

A todos os colegas e amigos, principalmente ao meu amigo Danton Cavalcante Franco Junior. Com certeza foi através do apoio destes que encontrei motivação para seguir em frente.

A todos aqueles que direta ou indiretamente contribuíram para a realização deste trabalho.

Resumo

Este trabalho tem como objetivo principal o desenvolvimento de uma abordagem híbrida para a solução de problemas de otimização, em especial os combinatórios.

Esta nova abordagem tem como base dois dos mais importantes modelos computacionais inteligentes utilizados na otimização de problemas, os algoritmos: genético e *simulated annealing*. O primeiro baseia-se na evolução natural e cromossômica das espécies vivas e o segundo no recozimento (*annealing*) de sólidos. Ambos são algoritmos de otimização (algoritmos que buscam por uma solução aceitável, o que não garante que a mesma seja a melhor).

Nesta abordagem, o algoritmo genético é utilizado como algoritmo principal e o algoritmo *simulated annealing* é introduzido no processo do algoritmo genético como sendo um operador genético.

Para avaliar o desempenho desta nova abordagem, foram realizados testes utilizando um dos mais conhecidos *benchmarks* na área de otimização, o problema do caixeiro viajante, e os resultados obtidos estão demonstrados neste trabalho.

Abstract

This work has as main objective the development of a hybrid approaching for the solution of optimization problems, specially the combinatorious.

This new approaching has as base two of the most important and intelligent computational models used in the optimization of problems, the algorithms: genetic and simulated annealing. The first one is based on the natural and chromosomic evolution of live species and the second on the annealing of solids. Both are optimization algorithms (algorithms that search for an acceptable solution, what does not guarantee it will be the best solution).

In this approaching the genetic algorithm is used as the main algorithm and the simulated annealing algorithm is introduced in the process of the genetic algorithm as being a genetic operator.

Understanding the execution this approaching, tests were made using on of the most well-known benchmarks in the optimization field, the travelling salesman problem, and the results are demonstrated in this work.

Sumário

1	Introdução	1
1.1	Motivação.....	3
1.2	Objetivo do trabalho	3
1.3	Organização do texto.....	3
2	Otimização	5
2.1	Conceituação básica.....	5
2.2	Especificação formal	7
2.3	Critérios para otimização.....	8
2.4	Métodos de otimização.....	8
2.4.1	Programação linear	9
2.4.2	Branch and bound	9
2.4.3	Busca local	10
2.5	Problemas NP-Completo	12
2.6	Ótimo local e global.....	13
2.7	Problemas conhecidos	13
2.7.1	Corte e empacotamento	14
2.7.2	Planejamento de tarefas	14
2.7.3	Caixeiro viajante	14
3	Algoritmo Genético.....	17
3.1	Introdução.....	17
3.2	O que são algoritmos genéticos	21
3.3	Representação cromossômica	23
3.4	Fluxo do algoritmo genético	25

3.4.1	Processo de inicialização	26
3.4.2	Processo de avaliação	26
3.4.3	Processo de seleção	27
3.4.4	Operadores genéticos	32
3.4.4.1	Cruzamento (<i>Crossover</i>)	32
3.4.4.2	Mutação (<i>Mutation</i>)	38
3.4.4.3	Inversão	41
3.4.5	Condições de término	42
3.5	Teorema dos esquemas	43
3.6	Blocos de construção	43
3.7	Variações de algoritmos genéticos	44
3.7.1	Algoritmos genéticos elitistas	44
3.7.2	Algoritmos genéticos paralelos	45
3.7.3	Algoritmos genéticos com população de tamanho variável	45
3.7.4	Algoritmos genéticos híbridos	46
3.8	Problemas práticos de algoritmos genéticos	46
3.8.1	Determinação de parâmetros	47
3.8.2	Erros de amostragem	47
3.8.3	Convergência prematura	48
3.8.4	Balanco exploração-exploração	49
3.8.5	Tempo de processamento	49
4	Algoritmo Simulated Annealing	50
4.1	Introdução	50
4.2	<i>Simulated annealing</i> no processo de otimização	52
4.3	Desempenho do algoritmo	54
4.3.1	Temperatura inicial	54
4.3.2	Decremento da temperatura	55
4.3.3	Tamanho da busca	55
4.3.4	Critério de parada	56
4.4	<i>Simulated annealing</i> paralelo	56

4.5 Considerações teóricas	57
5 Método desenvolvido	58
5.1 Introdução.....	58
5.2 Definição formal do problema	59
5.3 Modelagem.....	60
5.4 Método proposto	61
5.4.1 Descrição do método	61
5.4.2 Fluxo do algoritmo genético AS	62
5.4.3 Representação cromossômica	65
5.4.4 Avaliação da população.....	65
5.4.5 Seleção da população.....	66
5.4.6 Operadores genéticos.....	66
5.4.7 Operador de mutação SA.....	67
5.5 Implementação.....	67
5.6 Estrutura de dados.....	67
5.6.1 Procedimentos.....	69
6 Análise dos resultados.....	76
6.1 Metodologia utilizada.....	76
6.1.1 Origem das instâncias.....	77
6.1.2 Avaliação.....	78
6.1.3 População inicial.....	79
6.2 Ajuste dos parâmetros	79
6.2.1 Ajuste dos parâmetros relacionados ao algoritmo genético	80
6.2.2 Ajuste dos parâmetros relacionados ao algoritmo <i>simulated annealing</i>	81
6.3 Resultados obtidos	82
6.4 Comentários finais	86
7 Considerações finais.....	87
7.1 Conclusões.....	87
7.2 Proposta para novas pesquisas	88

Lista de figuras

Figura 1: Busca local presa em um mínimo local.	11
Figura 2: Instância de um problema do caixeiro viajante.	15
Figura 3: Figura 3: Termos mais utilizados em algoritmos genéticos	24
Figura 4: Conversão do valor da aptidão para o valor da expectativa MAZZUCCO (1999)...	29
Figura 5: Método de seleção denominado de “roleta”	31
Figura 6: Exemplo de cruzamento de um ponto.....	33
Figura 7: Exemplo de cruzamento multiponto.	34
Figura 8: Exemplo de cruzamento inválido.....	35
Figura 9: Exemplo de cruzamento com o operador OX.	36
Figura 10: Exemplo de cruzamento com o operador PMX.	36
Figura 11: Exemplo de um cruzamento com o operador CX.....	37
Figura 12: Exemplo de mutação.....	39
Figura 13: Exemplo de mutação baseada em posição.....	40
Figura 14: Exemplo de mutação baseada em ordem.....	40
Figura 15 Exemplo de mutação mista.....	41
Figura 16: Exemplo de inversão.....	42
Figura 17: Exemplo de seleção.....	48
Figura 18: Estratégia do algoritmo <i>simulated annealing</i>	54
Figura 19. Exemplo de instância em grade 4X4.....	78
Figura 20: Resultados obtidos com as instâncias de 76 e 152 cidades.....	83
Figura 21: Resultados obtidos com as instâncias de 299 e 1004 cidades.....	83
Figura 22: Resultados obtidos com as instâncias de 100 e 144 cidades.....	85
Figura 23: Resultados obtidos com as instâncias de 324 e 1024 cidades.....	85

Lista de quadros

Quadro 1: Algoritmo de busca local.....	11
Quadro 2: Fluxo básico de um algoritmo genético.	25
Quadro 3: Fluxo básico do algoritmo <i>simulated annealing</i>	53
Quadro 4: Fluxo do algoritmo genético original.	63
Quadro 5: Fluxo do algoritmo genético SA.....	63
Quadro 6: Fluxo do operador de mutaçãoSA	64
Quadro 7: Estrutura de uma cidade.	68
Quadro 8: Estrutura de uma rota.....	69
Quadro 9: Estrutura da população.	69
Quadro 10: Função de cálculo da distância entre duas cidades.....	70
Quadro 11: Função que calcula o tamanho de um percurso.	70
Quadro 12: Procedimento algoritmo genético.	71
Quadro 13: procedimento de cálculo da aptidão.	72
Quadro 14: Procedimento de seleção.	73
Quadro 15: Procedimento que aplica os operadores genéticos.	74
Quadro 16: Procedimento que aplica os operadores genéticos	75

CAPÍTULO 1

Introdução

A pesquisa sobre modelos computacionais inteligentes tem, nos últimos anos, se caracterizado pela tendência em buscar a inspiração na natureza (lugar onde se pode dizer que existem inúmeros processos considerados “inteligentes”). Para cientistas de computação, matemáticos e engenheiros, muitas das soluções que a mãe natureza encontrou, para problemas complexos de adaptação, fornecem modelos práticos interessantíssimos. Com base nestas soluções tem-se buscado inspiração para a criação de modelos computacionais que auxiliam na resolução de problemas das mais diversas áreas TANOMARU (1995).

Inspirado na natureza, a área de otimização teve um considerável aumento em estudos baseados em tais modelos. Um dos principais motivos que levam ao estudo destes modelos é o fato de que a maioria dos problemas de otimização de larga escala não possuem uma solução ótima em um tempo viável, podendo apenas ser resolvido de forma aproximada. No contexto de otimização, grande parte dos problemas são ditos NP-Completos (um problema não polinomial, indicando que o espaço de busca por soluções cresce exponencialmente com as dimensões do problema) LAARHOVEN & AARTS (1987).

Para a resolução destes problemas existem inúmeras pesquisas com abordagens baseadas em técnicas de otimização de busca local. Estas técnicas de otimização normalmente utilizam como base os algoritmos: genético, *tabu search* e *simulated annealing* MAZZUCCO (1999).

Este trabalho concentra-se no estudo da utilização de dois destes modelos para a solução de problemas de otimização, os algoritmos: genético e *simulated annealing*. O enfoque principal é o algoritmo genético, que terá um novo operador chamado de “mutação SA” (baseado no algoritmo *simulated annealing*).

Algoritmos genéticos (AGs) são métodos computacionais de busca baseados nos mecanismos da evolução natural e na genética. Em AGs, uma população de possíveis soluções para o problema em questão evolui de acordo com operadores probabilísticos concebidos a partir de metáforas biológicas, de modo que há uma tendência de que, na média, os indivíduos representem soluções cada vez melhores à medida que o processo evolutivo continua TANOMARU (1995).

Segundo BENDER (1985) e AARTS & KORST (1989), *simulated annealing* (SA) é um método computacional para resolução de problemas de otimização baseado no recozimento de sólidos que foi inicialmente introduzido pelos trabalhos de KIRKPATRIK et al. (1983) e CERNY (1985).

Este trabalho pretende utilizar a potencialidade destes dois métodos, em conjunto, para obter melhores resultados, quando aplicados a problemas de otimização.

Para demonstrar os resultados obtidos com esta nova abordagem é utilizado um dos mais conhecidos e estudados problemas de otimização, denominado de “problema do caixeiro viajante”. Este é um problema aparentemente bem simples, onde um caixeiro viajante possui um determinado número de cidades a percorrer e o deve fazer passando uma e somente uma vez por cada cidade e retornar então a cidade inicial, visando minimizar o custo da viagem LIN (1965). A escolha do problema do caixeiro viajante foi baseada no fato deste problema ser um dos *benchmarks* mais utilizados para avaliar a eficiência dos métodos de otimização. Os resultados obtidos com esta nova abordagem estão descritos neste trabalho.

1.1 Motivação

Este trabalho foi motivado no crescente estudo de modelos computacionais para a solução de problemas de otimização, demonstrando ser uma área de estudos muito interessante e com um futuro promissor.

1.2 Objetivo do trabalho

O objetivo principal deste trabalho está centrado no desenvolvimento de uma nova abordagem, com a utilização dos algoritmos: genético e *simulated annealing*, combinados, para a otimização de problemas, em especial o problema do caixeiro viajante.

1.3 Organização do texto

A estrutura básica deste trabalho esta dividida em 7 capítulos organizados da seguinte forma:

No capítulo 2, é introduzido uma fundamentação básica conceitual a respeito de problemas de otimização, problemas NP-Completo e alguns dos métodos utilizados na resolução dos mesmos.

O capítulo 3 aborda inicialmente o processo da evolução biológica natural que constitui a inspiração inicial do algoritmo genético. Então, o algoritmo é introduzido através da apresentação de seu processo e seus principais operadores. Finaliza com a apresentação formal de sua potencialidade.

No capítulo 4, é introduzida uma fundamentação básica a respeito do algoritmo: *simulated annealing*, onde os principais elementos básicos deste método são revistos.

No capítulo 5, estão definidos os métodos utilizados para a utilização em conjunto dos algoritmos estudados.

No capítulo 6 é apresentada a metodologia utilizada para a obtenção dos resultados, bem como as argumentações sobre decisões e ajustes de parâmetros. Também faz parte desse capítulo uma análise comparativa entre os resultados computacionais obtidos com o método proposto.

O trabalho termina com o capítulo 7, onde são apresentadas as considerações finais e também propostas para novos estudos no assunto.

CAPÍTULO 2

Otimização

O objetivo deste capítulo é introduzir uma visão geral do contexto no qual este trabalho se insere. Inicialmente é apresentada uma visão geral a respeito dos problemas de otimização e alguns dos métodos utilizados para a solução dos mesmos, incluindo o método de busca local (método que serve como base para os algoritmos estudados). Em seguida são abordados alguns dos problemas de otimização mais conhecidos, incluindo o problema do caixeiro viajante (problema utilizado para avaliar os resultados obtidos).

2.1 Conceituação básica

Entendemos por-otimização, não um processo de busca do melhor absoluto, mas a procura sistemática do melhor prático. Algumas vezes, conhecer o mecanismo de um certo problema, determinando as relações entre suas variáveis, já constitui um avanço considerável no caminho de sua solução NOVAES (1978).

Segundo YONEYAMA & NASCIMENTO (2000), o processo de otimização pode ser descrito da seguinte forma: a determinação de uma ação que proporciona um máximo de benefício, medido por um critério de desempenho pré-estabelecido.

Um problema de otimização pode ser tanto um problema de minimização ou maximização, dependendo do problema, e é especificado por um conjunto de instâncias (prováveis soluções) do problema em questão AARTS & KORST (1989).

Estes problemas estão naturalmente divididos em duas categorias: os problemas com variáveis contínuas e os problemas com variáveis discretas, chamados combinatórios. Nos problemas contínuos o que se procura é um conjunto de números reais ou mesmo uma função, nos problemas combinatórios o que se procura é um objeto dentre um finito ou contabilmente infinito conjunto de prováveis soluções (tipicamente um inteiro, permutação, conjunto ou gráfico) PAPADIMITRIOU (1982).

Durante as últimas décadas, uma grande coleção de problemas desta natureza tem surgido (nas mais diversas áreas como a ciência, computação, engenharia, etc), junto com uma coleção correspondente de técnicas para a solução dos mesmos. Estas técnicas são quase sempre essencialmente iterativas e sua convergência é estudada utilizando a matemática da análise real PAPADIMITRIOU (1982).

Uma importante descoberta no campo de otimização, obtida no final da década de 1960, é a suposição, ainda não verificada, de que existe uma classe de problemas de otimização de tal complexidade que qualquer algoritmo, resolvendo cada instância do problema para um resultado ótimo, requer um esforço combinatório que cresce super-polinomialmente com o tamanho do problema AARTS & KORST (1989). Isto resultou na distinção entre problemas fáceis e problemas difíceis, denominados NP-Completo.

Entretanto, estes problemas ainda precisam ser resolvidos e na construção de algoritmos para solução dos mesmos pode-se optar por dois caminhos. Um caminho que ainda busca por soluções ótimas sobre o risco de tempos de execução muito longos ou possivelmente impraticáveis. Outro caminho que busca por soluções rápidas sobre o risco de soluções sub-ótimas (soluções próximas às ótimas) em um tempo de processamento aceitável TANOMARU (1995).

Adicionalmente os dois casos ainda podem ser distinguidos entre algoritmos gerais e algoritmos costurados (*tailored*). Os algoritmos gerais podem ser aplicados a uma grande variedade de problemas podendo ser chamados de independentes do problema, já os algoritmos costurados usam informações específicas acerca do problema tornando sua aplicação limitada a um número restrito de problemas TANOMARU (1995).

2.2 Especificação formal

Segundo AARTS & KORST (1989), NOVAES (1978) e PAPADIMITRIOU (1982), uma instância de um problema de otimização pode ser formalizada como sendo um par (S, f) , onde o espaço de soluções S indica o conjunto finito de todas as possíveis soluções e a função de custo f é um mapeamento definido como:

$$f: S \rightarrow \mathbb{R} \quad (2.1)$$

No caso de minimização, o problema é achar uma solução $I_{opt} \in S$ que satisfaça:

$$f(I_{opt}) \leq f(I), \text{ para todo } I \in S \quad (2.2)$$

No caso de maximização, achar I_{opt} que satisfaça:

$$f(I_{opt}) \geq f(I), \text{ para todo } I \in S \quad (2.3)$$

Resumindo, encontrar uma solução com a menor (ou maior) função de custo dentre todas as possíveis soluções. Em muitos exemplos a função de custo receberá somente valores inteiros não negativos.

Tal solução I_{opt} é denominada de “ótimo-global”, mínima ou máxima, ou simplesmente ótima, mínima ou máxima; $f_{opt} = f(I_{opt})$ indica o custo ótimo e S_{opt} o conjunto de soluções ótimas NOVAES (1978).

É importante se fazer uma distinção entre problema e instância de problema. Informalmente, em uma instância temos os “dados de entrada” e temos informação suficiente para se obter uma solução; um problema é uma coleção de instâncias, geralmente todas geradas da mesma forma PAPADIMITRIOU (1982).

2.3 Critérios para otimização

Segundo NOVAES (1978), as técnicas de otimização são baseadas em duas premissas implícitas de conseqüências importantes na aplicação prática. Em primeiro lugar admite-se que possa ser definida uma "função objetivo" (função que exprime através de uma escala única a medida de mérito do sistema analisado, ou seja, o quanto a solução analisada é boa dentro do problema em questão). A segunda premissa refere-se ao caráter determinístico da avaliação: admite-se que as relações entre as variáveis dependentes e os parâmetros independentes ocorram deterministicamente, em outras palavras, um determinado conjunto de valores das variáveis independentes deve produzir apenas um resultado na função objetivo.

Na prática estas premissas freqüentemente não são satisfeitas. Em muitos casos observa-se uma pluralidade de objetivo a satisfazer. Este problema pode ser resolvido buscando uma medida de custo aproximada (mas única), efetuando-se uma análise de sensibilidade dos resultados para se levar em conta as eventuais características aleatórias das variáveis envolvidas NOVAES (1978).

2.4 Métodos de otimização

Segundo TANOMARU (1995), dentro de uma classificação grosseira, existem essencialmente três métodos gerais de otimização: métodos numéricos, enumerativos e probabilísticos.

Métodos numéricos: os métodos numéricos podem ser divididos em analíticos ou baseados em cálculos numéricos. Os métodos analíticos de otimização são aplicáveis basicamente quando a função de custo f é explicitamente conhecida e derivável, ou pode ser aproximada por alguma função derivável até o grau desejado de precisão. Neste caso, basta resolver as equações que resultam de igualar as derivadas da função a zero dentro do espaço de busca. Nos métodos baseados em cálculo numérico, quando o espaço de busca é linear e, portanto, convexo, técnicas de pesquisa operacional, como o método *simplex*, são

suficientes. Já em ambientes não-lineares, técnicas de gradientes ou de estatísticas de alta ordem são geralmente empregadas.

Métodos enumerativos: estes métodos examinam cada ponto do espaço de busca, um a um, em busca dos pontos ótimos. A idéia pode ser intuitiva mas é obviamente impraticável quando o número de pontos a examinar torna-se extremamente grande ou, como em muitos casos, possivelmente infinito.

Métodos probabilísticos: métodos probabilísticos são métodos que empregam a idéia de busca probabilística, o que não quer dizer que são métodos totalmente baseados na sorte, como é o caso dos chamados métodos aleatórios.

2.4.1 Programação linear¹

Programação linear é uma técnica utilizada para resolver uma determinada classe de problemas em que se procura alocar recursos limitados a atividades ou decisões diversas, de maneira ótima. Este tipo de problema aparece freqüentemente nos setores de planejamento e operações de indústrias, empresas de transporte, órgãos governamentais, etc. Dentre as diversas técnicas de pesquisa operacional, a programação linear é talvez a mais conhecida e utilizada em suas diversas formas NOVAES (1978).

2.4.2 *Branch and bound*

Métodos *branch and bound* fazem parte dos métodos de otimização enumerativos. São métodos que buscam por uma solução ótima através da verificação exaustiva de todas as possibilidades.

¹ O termo linear significa que todas as funções definidas do modelo matemático que descreve o problema devem ser lineares.

O princípio do *branch and bound* é a enumeração de todas as soluções viáveis de um problema de otimização, diga-se um problema de minimização, tal que propriedades ou atributos não compartilhados por qualquer solução ótima são detectados tão cedo quanto possível. Um atributo (ou ramo da árvore de enumeração) define um subconjunto do conjunto de todas as soluções viáveis do problema original onde cada elemento do subconjunto satisfaz este atributo CURY (1999).

2.4.3 Busca local

Os algoritmos de busca local constituem uma interessante classe dos métodos de otimização. São algoritmos baseados em melhoramento gradativo de uma solução para a outra pela exploração de sua vizinhança de acordo com algumas regras bem definidas AARTS & KORST (1989) e PIRLOT (1992).

Uma estratégia de busca local inicia de uma solução arbitrária $I \in S$ e a cada passo n uma nova solução I_{n+1} é escolhida a partir da vizinhança $V(I_n)$ da solução atual I_n . Isto leva a definição de uma estrutura de vizinhança em S ; para cada $I \in S$ é associado um subconjunto $V(I) \subseteq S$ chamado de vizinhança de I . A vizinhança de I é obtida a partir de I através de um movimento elementar (pequena modificação na solução corrente que resulta em uma nova solução viável do problema) PIRLOT (1992).

A evolução da solução atual I_n , $n = 1, 2, \dots$ desenha uma trajetória no espaço de busca S . O critério mais comum para a escolha da próxima solução I_{n+1} é escolhendo a melhor solução dentre a vizinhança de I_n , ou seja, uma solução $I_{n+1} \in V(I_n)$ que satisfaça:

$$f(I_{n+1}) \leq f(I) \quad \forall I \in V(I_n) \quad (2.4)$$

No caso de minimização, e para maximização que satisfaça:

$$f(I_{n+1}) \geq f(I) \quad \forall I \in V(I_n) \quad (2.5)$$

Então I_{n+1} torna-se a nova solução corrente caso seja melhor que a solução atual I_n , caso contrário, a busca acaba. O quadro 1 traz um exemplo de algoritmo de busca local.

```

Procedure Busca_Local
Inicio
  Inicializar( $i_{\text{inicio}}$ )
   $i := i_{\text{inicio}}$ 
Repita
  Gerar( $j$  de  $V(i)$ )
  Se  $f(j) < f(i)$  entao  $i := j$ 
Ate que  $f(j) \geq f(i)$ , para todo  $j \in V(i)$ 
Final

```

Quadro 1: Algoritmo de busca local.

Nota-se que a escolha de uma boa estrutura de vizinhança é extremamente importante para a eficiência do processo que tem como ponto fraco a inabilidade de escapar de um mínimo-local, como no exemplo da figura 1, onde um mínimo-local foi escolhido como sendo a solução para o problema. Neste caso, todas as possíveis soluções em sua vizinhança $V(I_n)$ são piores do que a solução atual I_n , embora mais adiante exista uma solução melhor que não pode ser alcançada.

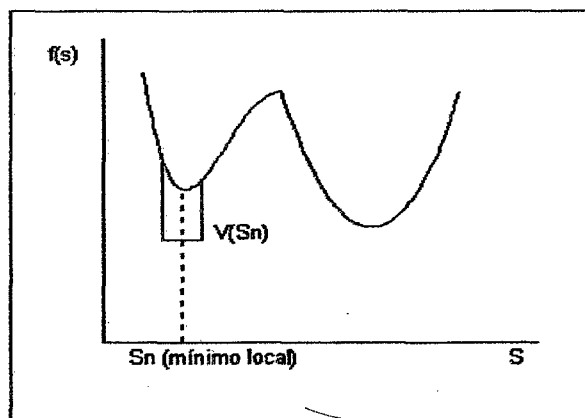


Figura 1: Busca local presa em um mínimo local.

AARTS & KORST (1989) sugerem algumas técnicas alternativas para evitar as desvantagens de um algoritmo de busca local, tais como:

- executar o algoritmo várias vezes com diferentes instâncias iniciais (com o custo de um aumento no tempo de processamento). Escolhem-se os melhores resultados obtidos (o fato de se executar o algoritmo com diferentes estados iniciais, embora pareça bastante intuitivo, não traz nenhuma garantia de que uma solução ótima será encontrada, além de criar um novo problema a respeito de quantos diferentes estados iniciais seria necessário para se obter um resultado aceitável);
- usar as informações obtidas das execuções anteriores do algoritmo para melhorar a escolha da instância inicial para a próxima execução;
- introduzir um mecanismo de geração do próximo estado mais complexo, na tentativa de saltar os mínimos locais (quanto mais complexo o mecanismo mais conhecimento a respeito do problema é requerido), e
- aceitar transições que correspondem a um incremento na função de custo de um modo limitado.

2.5 Problemas NP-Completo

Problemas NP-Completo são problemas não-polinomiais, indicando que o espaço de busca por soluções cresce exponencialmente com o tamanho do problema. Como consequência direta, estes problemas não podem ser resolvidos, para uma solução ótima, em uma quantidade razoável de tempo. Intuitivamente, um problema NP-Completo é um problema computacional que é tão difícil quanto qualquer outro problema PAPANITRIU (1982).

Segundo BENDER (1987), CORMEN et al. (1990) e PAPANITRIU (1982) esta classe de problemas tem as seguintes propriedades:

- nenhum problema NP-Completo pode ser resolvido por qualquer algoritmo polinomial conhecido (mesmo com os esforços persistentes de muitos pesquisadores brilhantes durante décadas), e
- se existir um algoritmo polinomial para qualquer problema NP-Completo, então existem algoritmos polinomiais para todos os problemas NP-Completo.

Com base nestes fatos muitas pessoas presumiram que não pode existir nenhum algoritmo polinomial para qualquer problema NP-Completo, entretanto, ninguém conseguiu provar isto. De fato se acredita que esta prova nunca virá sem o desenvolvimento de técnicas matemáticas totalmente novas PAPADIMITRIOU (1982).

Os problemas NP-Completo aparecem nas mais diversas áreas, tais como: lógica booleana, gráficos, aritmética, modelagem de redes, problemas de particionamento, problemas de armazenamento e recuperação, sequenciamento e escalonamento, programação matemática, álgebra, jogos, otimização de programas, etc CORMEN & LEISERSON (1990).

2.6 Ótimo local e global

Achar uma solução dita “ótimo global” para uma instância de alguns problemas pode ser muito difícil ou mesmo impossível, mas ainda é possível achar uma solução I_n que é a melhor no sentido de que não existe nenhuma solução melhor em sua vizinhança $V(I_n)$. A esta solução damos o nome de “ótimo local” PAPADIMITRIOU (1982).

2.7 Problemas conhecidos

Esta seção descreve alguns dos problemas de otimização mais conhecidos, como: problemas de corte e empacotamento, planejamento de tarefas e o problema do caixeiro viajante (problema utilizado para avaliar a abordagem desenvolvida neste trabalho).

2.7.1 Corte e empacotamento

Problemas de corte e empacotamento apresentam problemas relacionados à determinação de padrões de corte, de unidades de material, de modo a produzirem um conjunto de unidades menores, satisfazendo a determinadas restrições. Dependendo do tipo de objeto (barra, placas, caixas, e outros) temos os chamados problemas unidimensionais, bidimensionais, tridimensionais e outros. Os problemas específicos da área são propícios ao uso de heurísticas² LORENA (2001).

2.7.2 Planejamento de tarefas

Este é um campo interessante e importante na prática. O problema mais representativo é o chamado “*job-shop scheduling*”. Informalmente, o problema da programação da produção do tipo *job-shop* pode ser descrito da seguinte forma. São dados um conjunto de *jobs* e um conjunto de máquinas. Cada *job* consiste de uma cadeia de operações em que cada uma das quais deve ser processada durante um período de tempo ininterrupto, de um dado tamanho, em uma determinada máquina. Cada máquina pode processar no máximo uma operação por vez. Um programa consiste da alocação das operações em cada máquina. O problema é determinar um programa que realize todas as operações, no menor tempo possível MAZZUCCO (1999).

2.7.3 Caixeiro viajante

O problema do caixeiro viajante é, provavelmente, um dos mais conhecidos na área de otimização e pode ser definido da seguinte forma: um caixeiro viajante é requisitado a

² Qualquer técnica usada para melhorar a busca que depende de informações especiais acerca do problema em questão WINSTON (1998).

visitar cada uma das n cidades uma e somente uma vez. O viajante deve iniciar de qualquer cidade e retornar a cidade inicial. Que rota, ou excursão ele deve escolher visando minimizar o total da distância a percorrer ? LIN (1965).

Na prática, um problema com n cidades tem seu espaço de busca definido pela permutação:

$$S = (n - 1)! / 2 \quad (2.6)$$

Onde S é o número de rotas distintas que podem ser percorridas. Isto dá um número formidável mesmo para poucas cidades. Para $n = 30$, por exemplo, há um total de $4,42 \times 10^{30}$ rotas distintas. Com um computador capaz de analisar um milhão de rotas por segundo, a busca exaustiva levaria o equivalente a 10^7 vezes a idade do universo TANOMARU (1995).

Na figura 2 temos uma instância de um problema do caixeiro viajante com oito cidades e uma possível solução definida $I = (1, 2, 3, 4, 5, 6, 7, 8)$. Aqui o espaço de busca S é definido, através da equação 2.6, pela permutação das 8 cidades somando um total de 2.520 rotas distintas. A função f a ser minimizada busca encontrar o menor custo para percorrer todas as cidades entre as possíveis rotas. Pode-se notar que, mesmo para uma quantidade muito pequena de cidades, o espaço de busca torna-se assustador.

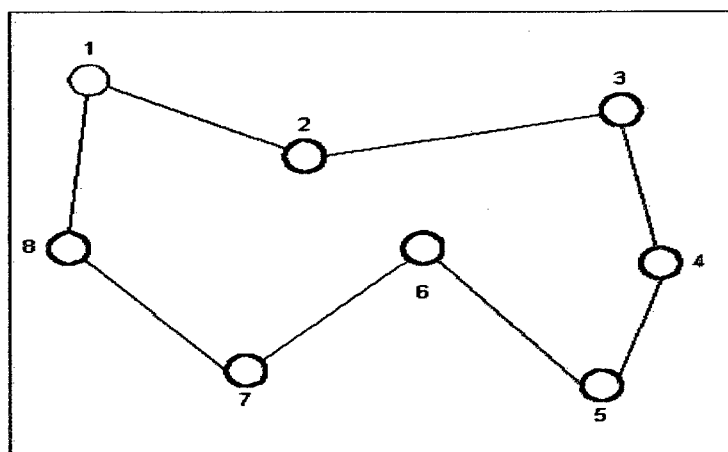


Figura 2: Instância de um problema do caixeiro viajante.

Segundo BRASSARD & BRATLEY (1996), a solução do problema do caixeiro viajante, utilizando-se qualquer algoritmo conhecido, torna-se impraticável para largas instâncias do problema (atualmente problemas com até 4461 cidades já podem ser resolvidos).

Uma boa amostra de problemas do caixeiro viajante pode ser encontrada em REINELT (2002).

CAPÍTULO 3

Algoritmo Genético

Este capítulo aborda, inicialmente, o processo da evolução biológica natural que constitui a inspiração inicial do algoritmo genético. Então o algoritmo é introduzido através da apresentação de seu processo e seus principais operadores. Finaliza com a apresentação formal de sua potencialidade.

3.1 Introdução

A teoria da evolução através da seleção natural¹ foi inicialmente apresentada por Charles Darwin em 1853. No ano seguinte, Darwin publica seu livro “*On the Origin of Species by Means of Natural Selection*” DARWIN (1953), com sua teoria completa sustentada por muitas evidências colhidas durante suas viagens a bordo do *Beagle* CARVALHO (2002).

O processo evolutivo se caracteriza pela produção de descendentes modificados de seus geradores e pela sobrevivência seletiva de alguns dos descendentes que produzirão novos descendentes. Estes dois aspectos: mudanças durante a reprodução e sobrevivência seletiva são suficientes para produzirem gerações de indivíduos cada vez melhores (para

¹ Princípio da natureza de incorporar nos descendentes de um ser, modificações que condicionam vantagens para a sobrevivência de sua espécie RICIERI (1994).

realizarem aquelas tarefas que contribuem para sua habilidade de reprodução) NILSSON (1998).

Segundo MAZZUCCO (1999), muitas experiências comprovam que a seleção natural é um fato incontestável, podendo não ser o único mecanismo evolutivo, porém sem dúvida um dos fatores principais do processo. Uma das principais constatações dessa evolução foi alcançada através de uma experiência realizada com bactérias. As bactérias foram expostas a doses de penicilina cada vez maiores e somente as bactérias mais fortes sobreviviam e se reproduziam. Este processo se repetiu por cinco gerações e no final apresentou-se uma linhagem de bactérias resistente a uma dose de penicilina duas mil e quinhentas vezes maior do que a inicial. O importante aqui é o fato de que a resistência à penicilina não foi uma característica desenvolvida individualmente e sim herdada das gerações anteriores.

Outra constatação interessante foi observada em uma população de pombos. Os pombos viviam perto de uma mina de carvão e tinham como cor predominante, a cor preta (isto porque, com a mina de carvão, o ambiente onde os pombos viviam era sujo e com a cor preta os pombos conseguiam mais facilmente se camuflar e escapar de seus predadores naturais). Quando a mina de carvão fechou, com o passar dos tempos observou-se uma gradativa alteração na cor predominante daquela população. Os pombos passaram a ter cores mais claras. (como a mina havia fechado, o ambiente tomara-se mais limpo e com isso os pombos mais claros se escondiam mais facilmente de seus predadores).

Vista de uma forma global, a evolução natural implementa mecanismos adaptativos de otimização que embora estejam longe de serem uma forma de busca aleatória, com certeza evoluem aleatoriamente. Na evolução natural, o problema que cada espécie enfrenta é a busca por adaptações benéficas para um ambiente complicado e de constantes mudanças.

Embora os mecanismos que conduzem à evolução natural não estejam ainda plenamente compreendidos, algumas de suas importantes características são conhecidas. A

evolução dos seres vivos se processa nos cromossomos² e parte da criação de um ser vivo é realizada através de um processo de decodificação de cromossomos.

Segundo GOLDBERG (1989) e MAZZUCCO (1999), os processos específicos de codificação e decodificação dos cromossomos ainda não estão totalmente esclarecidos, mas existem algumas características gerais na teoria desse assunto que estão plenamente consolidadas:

- a evolução é um processo que se realiza nos cromossomos e não nos seres que os mesmos codificam;
- a seleção natural é a ligação entre os cromossomos e o desempenho de suas estruturas decodificadas. Os processos da seleção natural determinam que aqueles cromossomos bem sucedidos devem ser reproduzidos mais freqüentemente do que os mal sucedidos;
- é no processo de reprodução que a evolução se realiza. Através da mutação (*mutation*) o cromossomo de um ser descendente pode ser diferente do cromossomo de seu gerador. Através do processo de cruzamento (*crossover*) dos cromossomos de dois seres geradores é também possível que os cromossomos do ser descendente se tornem muito diferentes daqueles dos seus geradores, e
- a evolução biológica não possui memória, a produção de um novo indivíduo depende apenas de uma combinação de genes da geração que o produz.

O professor John Holland (da universidade de Michigan), em suas explorações dos processos adaptativos de sistemas naturais e suas possíveis aplicabilidades em projetos de softwares de sistemas artificiais, no final da década de 1970, conseguiu incorporar importantes características da evolução natural em um algoritmo que denominado de “algoritmo genético”. Seus estudos foram formalmente introduzidos em seu livro “*Adaptation in Natural and Artifical Systems*” HOLLAND (1993), MAZZUCCO (1999) e VACA (1995).

² Dispositivos orgânicos onde as estruturas destes seres são codificadas.

HOLLAND (1993) ficou convencido de que as características da evolução biológica poderiam ser incorporadas em um algoritmo para construir um método extremamente simples para a resolução de complexos problemas, imitando o processo natural da evolução. Desta forma, ele começou a trabalhar em algoritmos que manipulavam cadeias de dígitos binários, as quais chamou de “cromossomos”. Tal como no processo biológico, seus algoritmos realizavam evoluções simuladas em populações de cromossomos sem nada conhecer a respeito do problema a ser resolvido. A única informação que era concedida, era a função de avaliação de cada cromossomo. Desta forma, os cromossomos que melhor evoluíam apresentavam uma tendência maior de reproduzirem, com mais frequência, do que os cromossomos de evolução ruim.

Segundo GOLDBERG (1989), KOZA & RICE (1994) e VACA (1995), a utilização do algoritmo genético na resolução de um determinado problema depende fortemente da realização dos seguintes passos:

- encontrar uma forma adequada de se representar soluções possíveis do problema em forma de cromossomos;
- determinar uma função de avaliação que forneça uma medida do valor (importância) de cada cromossomo gerado, no contexto do problema;
- determinação do tamanho da população e número de gerações, e
- determinação dos operadores genéticos e suas probabilidades associadas.

Uma das vantagens do algoritmo genético é a simplificação que eles permitem na formulação e solução de problemas de otimização. Apesar de serem aleatórios, os algoritmos genéticos exploram informações históricas para encontrar novos pontos de busca onde são esperados melhores desempenhos. Isto ocorre através de processos iterativos, onde cada iteração é denominada de geração CHAMBERS (1995).

Segundo MICHALEWICS (1999), os algoritmos genéticos diferem das tradicionais técnicas de otimização em quatro características fundamentais:

- a propriedade de paralelismo implícito, que permite analisar e avaliar um conjunto de soluções simultaneamente e não somente, estimar e melhorar uma única solução (como a maioria dos algoritmos de busca);
- a utilização da codificação de um conjunto de parâmetros ao invés dos próprios parâmetros;
- a utilização de regras de utilização probabilísticas e não determinísticas, e
- algoritmos genéticos utilizam informações de custo ou recompensa e não derivadas ou outro conhecimento auxiliar que tornam a implementação difícil e dependente do problema.

3.2 O que são algoritmos genéticos

Como mencionado no capítulo 1, algoritmos genéticos são métodos computacionais de busca baseados nos mecanismos de evolução natural e na genética. Em AGs, uma população de possíveis soluções para o problema em questão evolui de acordo com operadores probabilísticos concebidos a partir de metáforas biológicas, de modo que há uma tendência de que, na média, os indivíduos representem soluções cada vez melhores à medida que o processo evolutivo continua TANOMARU (1995).

Algoritmos genéticos combinam a sobrevivências dos melhores indivíduos entre estruturas com uma troca estruturada (ainda que randômica) formando um algoritmo de busca com um pouco do “instinto humano de busca”. Em cada geração um novo grupo de indivíduos (estruturas) é criado usando pedaços dos melhores indivíduos da geração anterior e ocasionalmente novas partes são geradas. Mesmo sendo randômicos, os algoritmos genéticos não são simples buscas randômicas e exploram informações históricas, de forma eficiente, para especular novos pontos de busca com uma melhora na performance GOLDBERG (1989).

O tema central na pesquisa de algoritmos genéticos tem sido sua força, o balanço entre eficiência e eficácia, necessários para a sobrevivência em muitos ambientes

diferentes. Se sistemas artificiais podem ser feitos mais robustos, o custo de sua reutilização pode ser minimizado ou talvez eliminado. Se altos níveis de adaptação podem ser alcançados, sistemas existentes podem realizar suas tarefas melhor e por mais tempo GOLDBERG (1989).

A busca de uma solução adequada em um algoritmo genético tem início com a criação aleatória de uma população inicial, que é composta por n elementos (número cromossômico), onde cada elemento é composto por uma cadeia de bits, que representa um único cromossomo. Essas estruturas são então avaliadas e associadas a uma probabilidade de reprodução de tal forma que as maiores probabilidades são associadas aos cromossomos que representam uma melhor solução para o problema. A aptidão da população é tipicamente definida com relação à população corrente e é calculada com base na função objetivo do problema em questão CHAMBERS (1995).

Uma das vantagens dos algoritmos genéticos é o fato de não ser necessário muito conhecimento a respeito do problema. Geralmente (com a utilização dos métodos tradicionais de otimização) são necessários, antes do início do processo de modelagem do problema, uma série de conhecimentos a respeito do problema, conhecimentos estes indispensáveis ao entendimento assim como a solução do problema. Isto consiste em um processo demorado, podendo levar meses ou até mesmo, anos. Já no caso do algoritmo genético basta definir um item genérico através de um conjunto de parâmetros, composto de respostas a características específicas pré-determinadas, criar alguns exemplares deste item e testá-los uns contra os outros até que gerem itens satisfatórios, o que pode levar algumas horas, ou mesmo, alguns dias PRICE (1994).

Segundo GOLDBARG (2000), algoritmos genéticos são métodos de busca probabilística inteligente com as seguintes características básicas:

- os indivíduos são representados por cromossomos e competem por recursos e possibilidades de reprodução. Os cromossomos normalmente estão associados a soluções do problema modelado;

- os indivíduos que tiverem mais sucesso nas competições terão maior probabilidade de reprodução do que aqueles de menor performance. O mecanismo de avaliação é constituído por uma função de avaliação que classifica os indivíduos por sua performance e por regras que permitirão que estes melhores indivíduos se perpetuem e reproduzam, e
- os genes dos indivíduos avaliados como bons se propagam através das populações, de modo que possam ser aperfeiçoados e gerem proles cada vez mais adequadas. Os indivíduos selecionados serão transformados por operadores genéticos em novos indivíduos.

Atualmente são encontradas aplicações que utilizam algoritmos genéticos em problemas de planejamento de produção, problemas de transporte, otimização de movimentos de robôs, na economia, em *layout* de condutores, em sistemas de classificação, na otimização de funções matemáticas, na medicina, etc HOFFMEISTER et al. (1993) e BARROS (2001).

3.3 Representação cromossômica

O primeiro passo para a aplicação de um algoritmo genético em um problema qualquer é encontrar uma maneira de se representar cada possível solução S no espaço de busca como uma seqüência de símbolos s gerados a partir de um alfabeto finito A . Nos casos mais simples, usa-se o alfabeto binário $A = \{0, 1\}$ (muitos consideram estes como algoritmo genético puro, embora na prática a utilização de uma representação binária nem sempre seja possível). No caso geral, tanto o método de avaliação quanto o alfabeto genético dependem de cada problema. Porém, uma vez definida a estrutura mais apropriada para representar a solução do problema, esta deve conter todos os nós de busca e ser única (não é permitido a alteração da forma de representação no decorrer do processo) MICHALEWICS (1997).

Segundo GOLDBARG (2000), usando algumas das metáforas empregadas pelos teóricos e praticantes de AGs, os termos mais utilizados, que podem ser vistos na figura 3 são:

- população: conjunto de indivíduos (conjunto de soluções do problema);
- cromossomo: representa um indivíduo na população (uma configuração ou solução);
- gene: representa um componente do cromossomo (uma variável do problema);
- *allele* ou alelo: descreve os possíveis estados de um atributo do indivíduo (possíveis valores de uma variável do problema);
- *locus*: representa a posição do atributo no cromossomo;
- fenótipo: denota o cromossomo decodificado, e
- genótipo: representa a estrutura do cromossomo codificado;

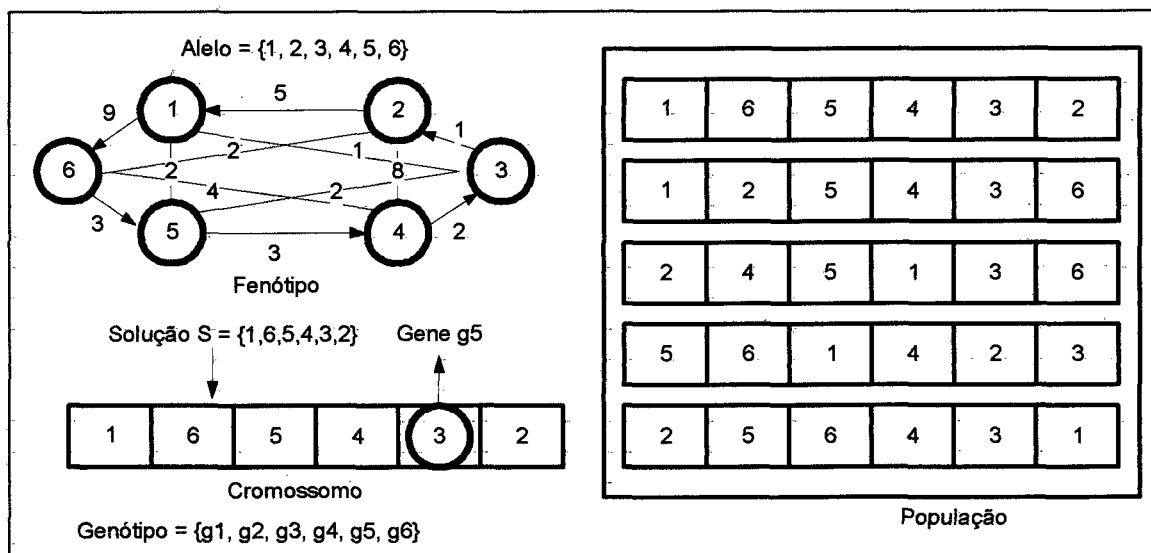


Figura 3: Termos mais utilizados em algoritmos genéticos.

Além disso, na maioria dos algoritmos genéticos assume-se que cada indivíduo seja constituído por um único cromossomo (fato que não ocorre na genética natural), razão pela qual é comum utilizar os termos cromossomos e indivíduos indistintamente. A maior parte

dos algoritmos genéticos, propostos na literatura, utilizam uma população de tamanho fixo com cromossomos também de tamanhos constantes TANOMARU (1995).

3.4 Fluxo do algoritmo genético

Uma vez definida a representação cromossômica para o problema, gera-se um conjunto de possíveis soluções chamadas de “soluções candidatas”. Este conjunto de soluções candidatas, de acordo com a representação selecionada, corresponde à população de indivíduos $P(0)$ em questão. Algoritmos genéticos são iterativos e a cada iteração a população é modificada. Cada iteração de um algoritmo genético é denominada de “geração”, embora nem todos os indivíduos de uma população sejam necessariamente “filhos” de indivíduos da geração anterior. Designando cada geração por um índice t , o fluxo básico de um algoritmo genético pode ser visto no quadro 2.

```
Procedure Algoritmo Genético
Início
  t = 0
  Inicializar P(t)
Enquanto (Condição <> Fim) Faca
  Início
    Avaliar P(t)
    t = t + 1
    Selecionar P(t) a partir de P(t - 1)
    Cruzamento em (Pt)
    Mutação em P(t)
  Final
Final
```

Quadro 2: Fluxo básico de um algoritmo genético.

O algoritmo é bem simples. Após a criação da população inicial (geralmente de forma aleatória) esta é avaliada de acordo com sua aptidão e os melhores indivíduos são pré-selecionados a formarem a próxima geração. Sobre esta nova geração são aplicados os operadores genéticos (com a intenção de melhorar a aptidão da população, através da exploração do material genético da população anterior e também gerar novos pontos de busca). Este processo se repete até que uma condição de término seja satisfeita, geralmente um determinado número de gerações. Além das etapas mencionadas acima, o que se costuma fazer ainda, durante o processo, é guardar a melhor solução (melhor cromossomo) encontrada durante a execução do algoritmo. Cada etapa do fluxo do algoritmo genético está explicada em seguida.

3.4.1 Processo de inicialização

Na maioria das aplicações, a população inicial de N indivíduos de um algoritmo genético é escolhida aleatoriamente ou através de um processo heurístico TANOMARU (1995).

Como no caso biológico, não há evolução sem variedade. Ou seja, a teoria da evolução natural ou “lei do mais forte” necessita que os indivíduos tenham diferentes graus de adaptação ao ambiente em que vivem. Por isso é muito importante que a população inicial cubra a maior área possível do espaço de busca CHAMBERS (1995).

3.4.2 Processo de avaliação

O processo de avaliação consiste na aplicação de uma função de avaliação em cada um dos indivíduos da população corrente. Esta função deve expressar a qualidade de cada indivíduo da população no contexto do problema considerado e é extremamente importante, uma vez que constitui o único elo entre o algoritmo genético e o problema em questão. A função de avaliação depende fortemente da forma como as soluções foram representadas

(nos casos mais simples usa-se justamente o valor da função que se quer otimizar) MAZZUCCO (1999).

A função de avaliação dá para cada indivíduo uma medida de quão adaptado ao ambiente ele está, ou seja, quanto maior o resultado da avaliação maior a chance do indivíduo sobreviver ao ambiente e reproduzir-se, passando parte de seu material genético para as futuras gerações. A avaliação de cada indivíduo resulta em um determinado valor de aptidão (que em inglês é denominado de “*fitness*”) TANOMARU (1995).

A maior parte das aplicações, baseadas em algoritmos genéticos, utilizam representações indiretas das soluções, o que significa que o algoritmo trabalha sobre uma população de soluções codificadas. Desta forma, antes que seja realizada a avaliação sobre os cromossomos uma tradução da solução codificada para a solução real deve ser realizada. Por outro lado em uma aplicação que empregue representação direta da solução, o valor que é passado para a função de avaliação é o próprio cromossomo. Neste caso toda a informação relevante ao problema deve ser incluída na representação da solução MAZZUCCO (1999).

3.4.3 Processo de seleção

Os mecanismos de seleção em algoritmos genéticos emulam os processos de reprodução assexuada e seleção natural. A seleção efetua com base nas aptidões individuais (calculadas através da função de avaliação), a seleção dos indivíduos que procriarão para formarem a próxima geração. Esta seleção é probabilística, assim, um indivíduo com aptidão alta tem maior probabilidade de se tornar um gerador do que um indivíduo com aptidão baixa. No caso de um algoritmo genético com a população fixa, são escolhidos tantos indivíduos quanto for o tamanho da população MAZZUCCO (1999).

O processo inicia-se com a conversão de cada aptidão individual em uma expectativa que é o número esperado de descendentes que cada indivíduo poderá gerar. No algoritmo original proposto por HOLLAND (1993), essa expectativa individual era

calculada através da divisão da aptidão individual pela média das aptidões de toda a população da geração corrente. Desta forma, a expectativa e de um indivíduo i , pode ser calculada por:

$$e_i = apt_i / map_t \quad (3.1)$$

Onde apt_i é a aptidão do indivíduo i e map_t é a aptidão média da população na geração t .

As expectativas assim produzidas têm as seguintes características: um indivíduo com aptidão acima da média terá expectativa maior do que I , enquanto que um indivíduo com aptidão abaixo da média terá uma expectativa menor do que I ; a soma dos valores de todas as expectativas individuais será igual ao tamanho da população.

No entanto, pelo menos dois problemas podem ser apontados na utilização desse método de cálculo das expectativas. Em primeiro lugar, o método não deve manipular valores de aptidão negativos, uma vez que poderiam ser convertidos em valores de expectativa negativos. Em segundo lugar, esse método pode apresentar efeitos indesejáveis logo no início, quando a população é aleatoriamente calculada e a média das aptidões é baixa, onde indivíduos aptos podem receber um número desordenado de descendentes, conduzindo a uma convergência prematura. Também no decorrer do processamento, onde esse método tem uma forte tendência em alocar a todos os indivíduos um descendente, mesmo para aqueles indivíduos com expectativa muito baixa, resultando em uma exploração ineficiente do espaço de busca e conseqüentemente em um retardo acentuado na convergência do método MAZZUCCO (1999).

Segundo MAZZUCCO (1999), um segundo método de conversão de valores de aptidão em números esperados de descendentes, referenciado como o “método do truncamento sigma” considera a aptidão média da população e o desvio padrão das aptidões sobre a população. Sua idéia básica é muito simples: um indivíduo, cuja aptidão seja igual à média das aptidões da população terá uma expectativa de produzir um descendente; um indivíduo, cuja aptidão seja um desvio padrão acima da média, terá uma expectativa de produzir um descendente e meio; um indivíduo, cuja aptidão seja dois desvios padrões acima da média, terá uma expectativa de produzir dois descendentes, e assim por diante. Este método pode ser observado através da figura 4.

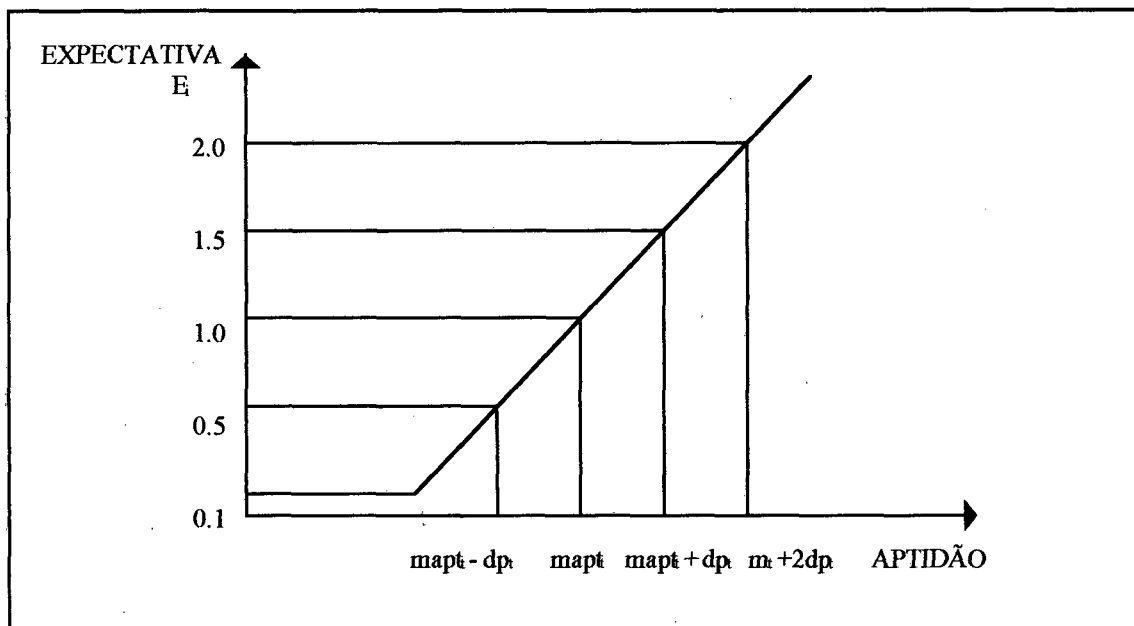


Figura 4: Conversão do valor da aptidão para o valor da expectativa MAZZUCCO (1999).

Baseado no método do truncamento sigma, a expectativa e de um indivíduo i pode ser calculada por:

$$e_i = (apt_i - m_{apt_i}) / 2d_{p_i} + 1 \quad (3.2)$$

No caso onde $d_{p_i} \neq 0$ ou:

$$e_i = 1 \quad (3.3)$$

No caso onde $dp_t = 0$, onde apt_i é a aptidão do indivíduo i e $mapt_t$, e dp_t são, respectivamente, a média e o desvio padrão das aptidões na geração t .

Caso o desvio padrão seja zero, qualquer indivíduo na população terá o mesmo valor de aptidão. Neste caso, automaticamente esse método irá atribuir um valor de expectativa igual a 1 para todos os indivíduos da população. Se o valor da expectativa tornar-se negativo, ele será alterado para um valor pequeno como 0.1, por exemplo, de forma que aqueles indivíduos com aptidões muito baixas terão pequenas probabilidades de se reproduzirem. Nota-se, que com esse segundo método, a soma dos números esperados de descendentes para todos os indivíduos na população não necessariamente será igual ao tamanho da população.

Depois de calculada a expectativa de cada indivíduo, resta ainda um último cálculo, onde o valor da expectativa é transformado em número definitivo de descendentes que cada indivíduo irá gerar. Esta última operação se faz necessária, uma vez que o valor da expectativa é um número real e não pode ser utilizado como valor real de descendentes, pois suponha que um indivíduo receba a expectativa 1,5, um indivíduo não pode gerar um descendente e meio.

Segundo GOLDBERG (1989) e MIRANDA (2002), o número inteiro e definitivo de descendentes que cada indivíduo receberá, chamado de “valor de descendente individual”, pode ser obtido através de uma técnica denominada de “roleta”. Para visualizar este método considere um círculo dividido em n regiões (com n sendo o tamanho da população). A área de cada região é proporcional à expectativa de cada indivíduo. Coloca-se sobre este círculo uma “roleta” com n cursores, igualmente espaçados. Após um giro da roleta, a posição dos cursores indica os indivíduos selecionados. Evidentemente, os indivíduos cujas regiões possuem maior área terão maior probabilidade de serem selecionados mais vezes. Como consequência, a seleção de indivíduos pode conter várias cópias de um mesmo indivíduo enquanto outros podem desaparecer. Aqui os indivíduos mais bem adaptados (com maiores

aptidões) têm mais chances de serem selecionados do que os indivíduos mais fracos (com aptidões menores).

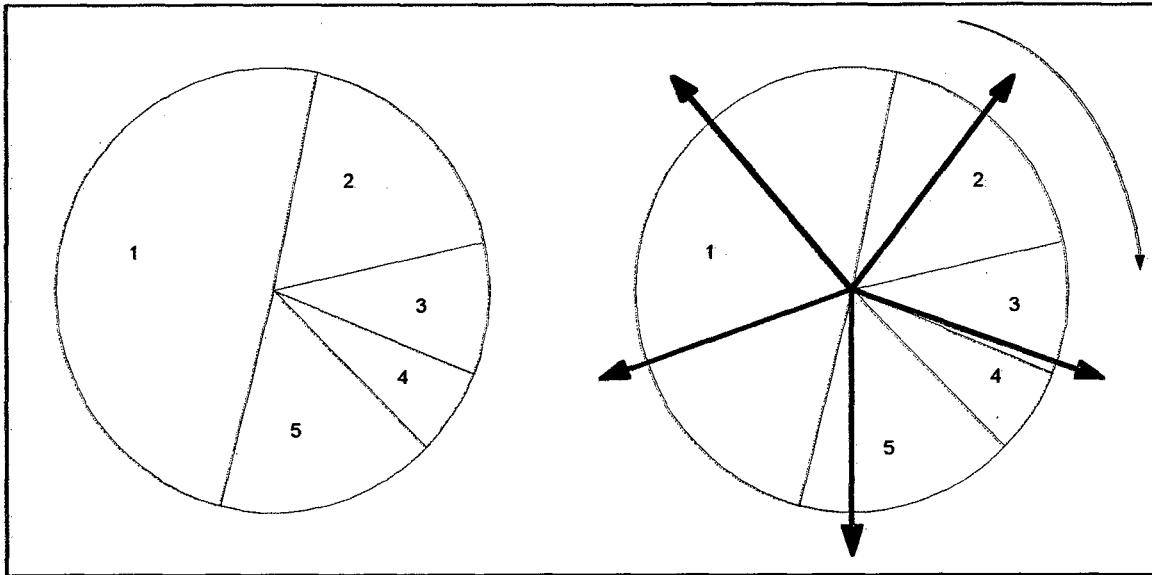


Figura 5: Método de seleção denominado de “roleta”.

A figura 5 demonstra o método da roleta aplicado a uma população com 5 cromossomos. Nesta figura, o cromossomo 1 tem sua aptidão muito maior que os demais cromossomos e, após o giro da roleta, o mesmo recebe dois descendentes. Já o cromossomo 4 tem sua aptidão muito inferior aos demais e após o giro da roleta ele não teve nenhum cursor apontando para sua área. Ele não irá gerar nenhum descendente e, conseqüentemente, será extinto.

Encerrando o segundo passo do algoritmo genético, o qual descreve o processo de seleção dos cromossomos que produzirão a próxima geração, é importante salientar que apesar de sua aparência confusa, este processo, de fácil implementação, vem tornar o algoritmo genético ainda mais robusto, uma vez que generaliza a função de avaliação, permitindo que a mesma retorne valores bem definidos MAZZUCCO (1999).

KOZA & RICE (1994) ressaltam que o processo de seleção utilizado pelo algoritmo genético é probabilístico e é um dos pontos essenciais do algoritmo. Como já aludido, o

processo aloca a cada indivíduo uma chance de ser selecionado (não importando o quanto a aptidão desse indivíduo seja pobre) e participar no processo dos operadores genéticos.

3.4.4 Operadores genéticos

Os operadores genéticos são operadores de variação dos algoritmos genéticos, possuindo a habilidade de transformar a população através de sucessivas gerações. Analisando sob o prisma da matemática, são operadores cuja função é criar novos pontos no espaço de busca com base nos elementos da população atual. Os operadores genéticos são necessários para que a população se diversifique e mantenha as características de adaptação adquiridas pelas gerações anteriores MICHALEWICS (1999).

Existem inúmeros operadores para a manipulação dos indivíduos, sendo os mais comuns, considerados clássicos, o cruzamento (*crossover*) e a mutação (*mutation*). Embora estes operadores sejam considerados os mais importantes, existe outro operador que tem obtido bons resultados e merece algum destaque, o operador de inversão (*inversion*).

3.4.4.1 Cruzamento (*Crossover*)

O cruzamento é o operador responsável pelo cruzamento de características de dois pais durante a reprodução, permitindo que as próximas gerações herdem estas características (muitos autores consideram este operador como a principal particularidade dos algoritmos genéticos frente às outras heurísticas) SPILLMAN (1993).

O cruzamento consiste no operador de acasalamento que permite a produção de novos indivíduos através da troca de informações parciais entre pares de cromossomos.

O processo de cruzamento é geralmente controlado por um parâmetro fixo que indica a probabilidade de um cromossomo sofrer cruzamento (normalmente se utiliza uma taxa de cruzamento moderada). Este operador possui o efeito de explorar as informações

contidas em seus geradores, ou seja, tendem a gerar melhores indivíduos com o passar das gerações baseados nas gerações anteriores TANOMARU (1995).

O operador de cruzamento pode variar conforme a representação utilizada. Segundo GOLDBERG (1989), quando a representação é binária, os dois tipos mais utilizados são: o cruzamento de um ponto e o cruzamento multiponto.

Cruzamento de um ponto: é realizado através da escolha aleatória de um só ponto de corte. Cada par de cromossomos escolhidos como pais gera dois descendentes através da permuta de suas partes finais, depois do ponto de corte. A figura 6 traz um exemplo de cruzamento de um ponto que é realizado depois gene 5 contando da esquerda para a direita.

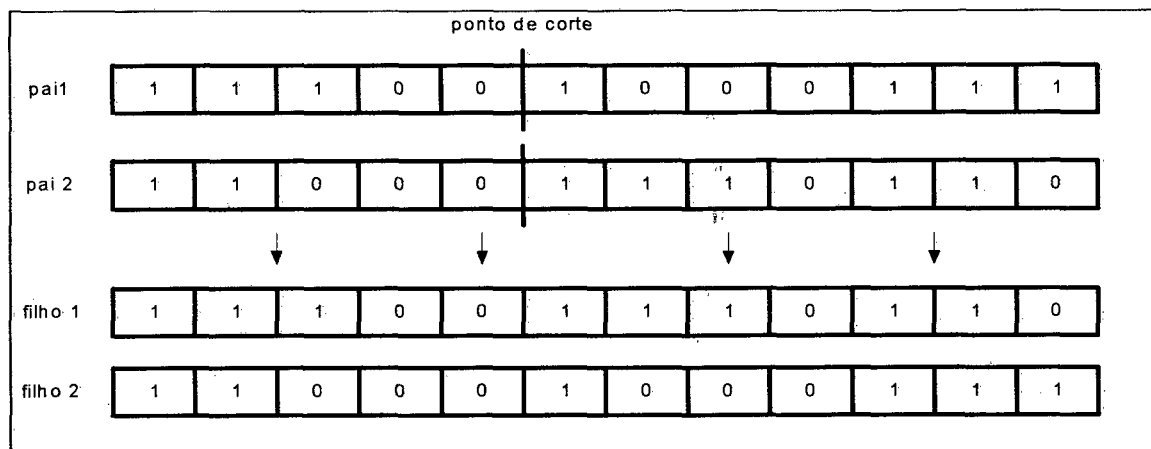


Figura 6: Exemplo de cruzamento de um ponto.

Um dos problemas deste tipo de cruzamento é que os descendentes sempre irão conter as partes finais dos seus pais, não podendo fazer outra combinação possível, caracterizando uma tendência de sobrevivência das partes finais SPILLMAN (1993).

Cruzamento multiponto: este operador surge como uma possibilidade de reduzir a carência do operador anterior. Aqui são escolhidos mais pontos de corte para troca de material genético entre o par de geradores. A figura 7 traz um exemplo de cruzamento multiponto com dois pontos de cortes. Neste exemplo, o primeiro ponto de corte é após o

gene 4 e o segundo após o gene 9 e o material genético trocado entre o par de geradores são os genes entre 4 e 9.

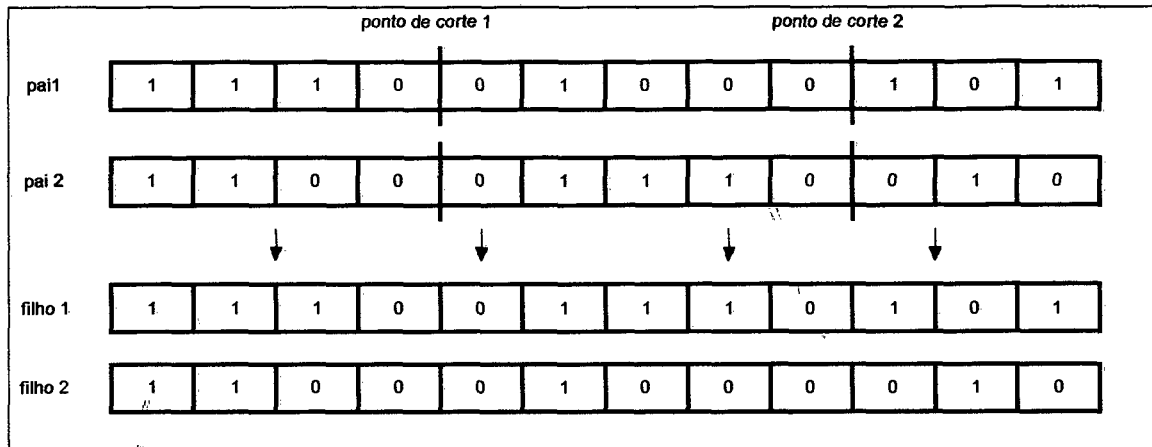


Figura 7: Exemplo de cruzamento multiponto.

Contudo, existem representações onde os operadores de cruzamentos mencionados não podem ser utilizados, como no caso da representação em forma de permutação. Uma das maiores dificuldades destas representações é que o operador de cruzamento pode produzir ciclos inviáveis, como no exemplo da figura 8. Esta figura traz um exemplo de cruzamento aplicado a cromossomos com representação através de números inteiros, onde, após aplicarmos o cruzamento de um ponto de corte o operador gerou dois cromossomos inválidos, pois os mesmos possuem alelos repetidos, o que não é permitido (no caso do problema do caixeiro viajante seria o mesmo que dizer que o viajante não percorreu todas as cidades uma e somente uma vez como deveria).

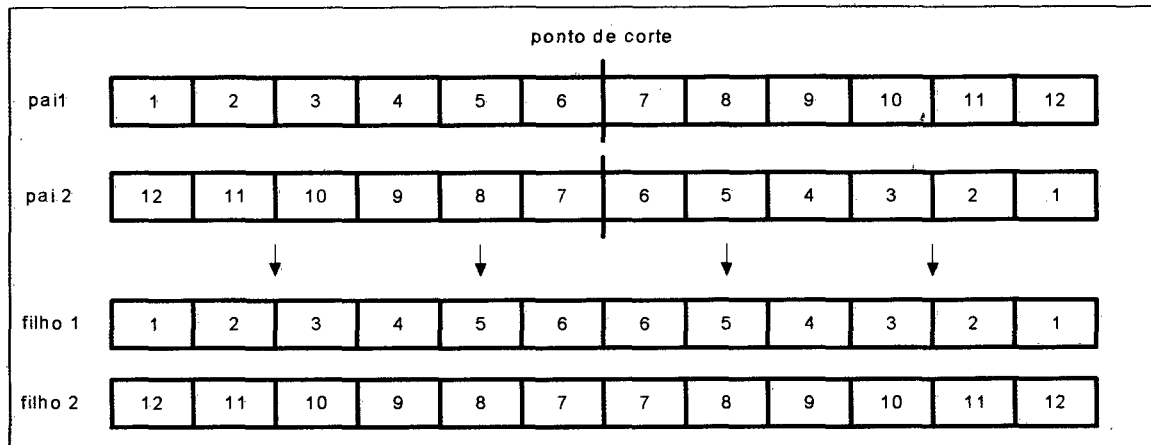


Figura 8: Exemplo de cruzamento inválido.

Visando solucionar estes problemas, foram desenvolvidos operadores especiais que evitam a produção de filhos inviáveis (geneticamente abortivos), dado que os pais sejam representações de soluções viáveis. Segundo GOLDBARG (2000), GOLDBERG (1989), MICHALEWICS (1999) e VACA (1995), dentre estes operadores, os que mais se destacam (devido à quantidade de aplicações em que são empregados) são: o operador OX, o operador PMX e o operador CX.

Operador OX (Order Crossover): este operador começa pela escolha aleatória de dois pontos de corte em cada um dos elementos selecionados. A seção definida entre estes dois pontos é copiada integralmente no descendente. Os lugares restantes são preenchidos usando as informações não repetidas na seção de cruzamento, começando do segundo ponto de corte. A figura 9 traz um exemplo de cruzamento OX, onde a seqüência j,k,c (volta ao início) d, e, f, g, h, i, é o gene contido no segundo pai, começando no segundo ponto de corte. De maneira similar, obtém-se a seqüência j, k, c, e, f, d, b, i, a, do segundo filho.

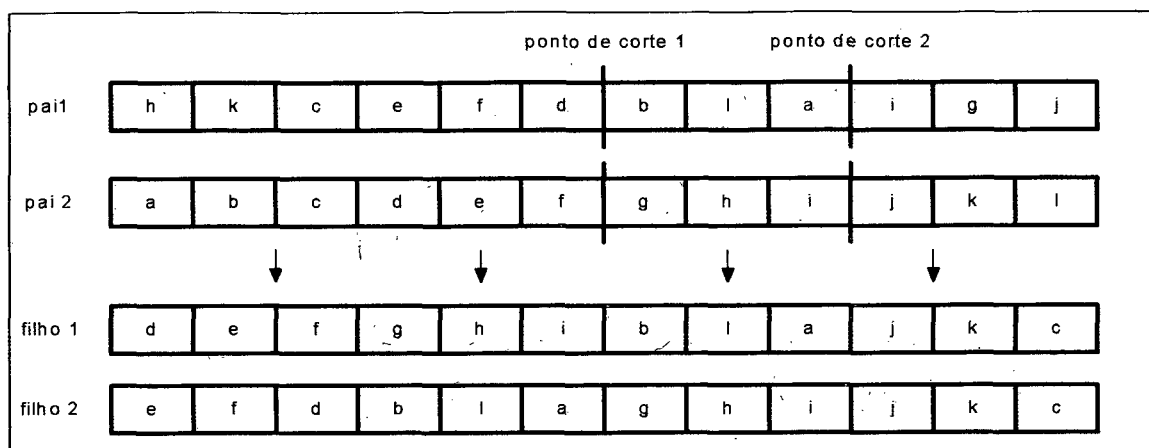


Figura 9: Exemplo de cruzamento com o operador OX.

Operador PMX (Partially Mapped Crossover): este operador também é executado escolhendo-se aleatoriamente dois pontos de corte. Este processo é ilustrado na figura 10 onde as informações contidas entre os dois pontos de corte, (b, i, a) e (g, h, i) são intercambiadas, obtendo-se as representações intermediárias. Porém, estas representações não são válidas, pois possuem informações repetidas e algumas faltando. Assim, o passo final é substituir estes genes repetidos mapeando (g, h, i) para (b, i, a) e vice-versa, obtendo assim as estruturas finais.

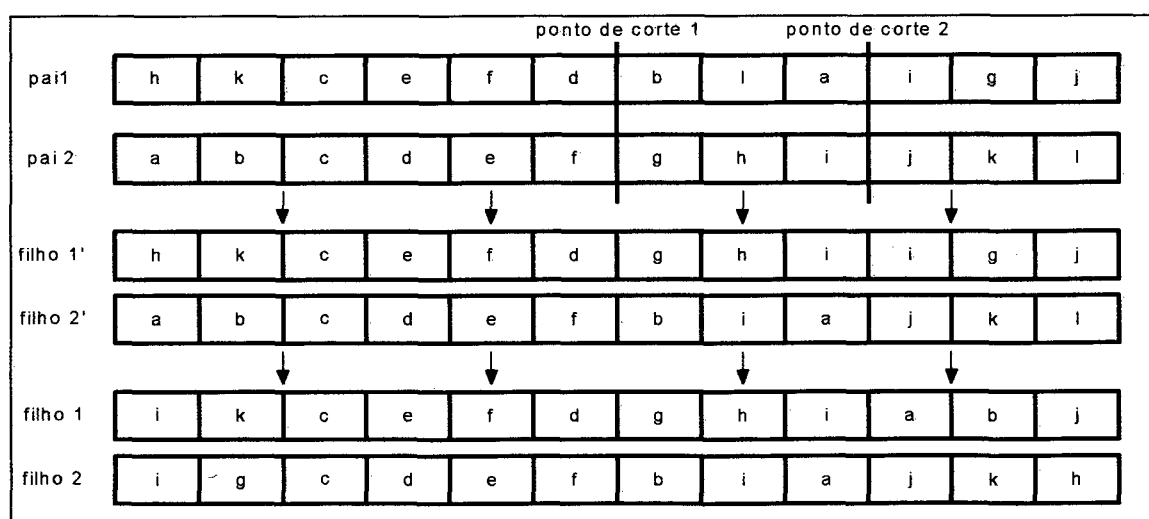


Figura 10: Exemplo de cruzamento com o operador PMX.

Operador CX (Cycle Crossover): o esquema do operador CX é muito diferente dos demais mostrados anteriormente. O operador CX executa recombinações de forma que cada um dos genes de seus descendentes venham da posição correspondente de qualquer um dos pais (neste tipo de cruzamento não há necessidade de escolher pontos de cruzamento, como os usados no operador PMX ou no operador de ordem OX).

Originalmente, este procedimento é iniciado da primeira posição mais à esquerda. Na figura 11 temos um exemplo de cruzamento CX onde o procedimento é iniciado a partir da primeira posição mais à esquerda, escolhendo um gene do primeiro pai. Já que por este operador cada gene vem da mesma localização de um dos pais, a escolha da atividade 2 do pai 1 significa que a atividade 5 da segunda posição deve ser escolhida, porque 2 ocupa a segunda posição do segundo pai. Esta seleção, por sua vez, implica que a atividade 4 seja escolhida do pai 1, que por sua vez leva a escolher a atividade 3 do pai 1, porque 3 está na sétima posição do pai 2. Neste ponto, se continuar com o processo, a escolha da atividade 3 significa ter que selecionar a atividade 2 do pai um. Entretanto, isto não é possível porque 2 já foi selecionado como o segundo gene da solução. A posição destes genes é dita de formar um ciclo que, eventualmente, retorna ao primeiro gene selecionado. Então o procedimento pára e os espaços vazios são preenchidos com os genes do segundo pai. O segundo filho é obtido realizando o cruzamento complementar.

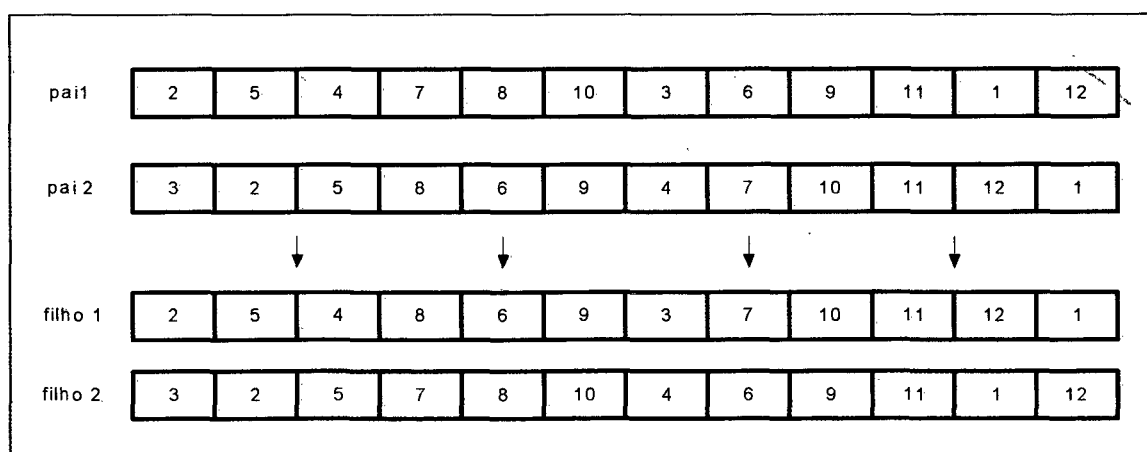


Figura 11: Exemplo de um cruzamento com o operador CX.

3.4.4.2 Mutação (*Mutation*)

Uma das principais dificuldades dos algoritmos genéticos (da mesma forma que a maioria dos algoritmos de busca) é a convergência prematura para uma solução de pouca qualidade. Foi observado que este problema está diretamente ligado a perda de diversidade da população durante o processo evolutivo, sendo, portanto, necessário um mecanismo que introduza novos materiais genéticos na população CHAMBERS (1995).

O operador de mutação é necessário para introdução e manutenção da diversidade genética na população, alterando arbitrariamente um ou mais componentes de uma estrutura (cromossomo) escolhida, fornecendo assim meios para a introdução de novos elementos na população. Desta forma, a mutação assegura que a probabilidade de alcançar qualquer ponto no espaço de busca nunca será completamente eliminada, além de contornar o problema de se cair em mínimos locais, pois a direção de busca do algoritmo genético é alterada sutilmente com a ocorrência da mutação CHAMBERS (1995).

O operador de mutação é bastante simples, basicamente, seleciona-se uma posição num cromossomo e muda-se o valor do gene correspondente aleatoriamente para outro alelo possível. A mutação é aplicada à população de cromossomos descendentes gerados após o cruzamento.

O processo é geralmente controlado por um parâmetro fixo que indica a probabilidade de um gene sofrer mutação (normalmente se utiliza uma taxa de mutação pequena). Este operador possui o efeito de aumentar a diversidade da população evitando que as estruturas tornem-se muito homogêneas. O aumento na diversidade das estruturas permite reduzir a possibilidade de convergência prematura, isto é, a obtenção de soluções sub-ótimas TANOMARU (1995).

Apesar das vantagens da utilização do operador de mutação, ele deve ser aplicado em pequena escala no processo do AG, semelhante à forma como a mutação ocorre na genética natural, pois a ocorrência excessiva de mutações pode gerar pontos muito dispersos na população, ao invés de convergir para pontos ótimos KOZA (1992)

A figura 12 traz um exemplo de mutação onde se percorre todos os alelos do cromossomo e se faz um sorteio, caso o sorteio resulte em um valor menor que a probabilidade de ocorrência de mutação, o alelo sofrerá mutação. Neste exemplo, um mesmo cromossomo sofre duas mutações, tendo seus genes 4 e 7 alterados de 0 para 1.

A exemplo do operador de cruzamento, a mutação também pode gerar cromossomos inválidos em certos tipos de representações, como a representação de caminho mencionada anteriormente. BUCKLES & PETRY (1992) sugerem 3 tipos de operadores de mutação para problemas desta natureza: a mutação baseada em posição, a mutação baseada na ordem e a mutação mista.

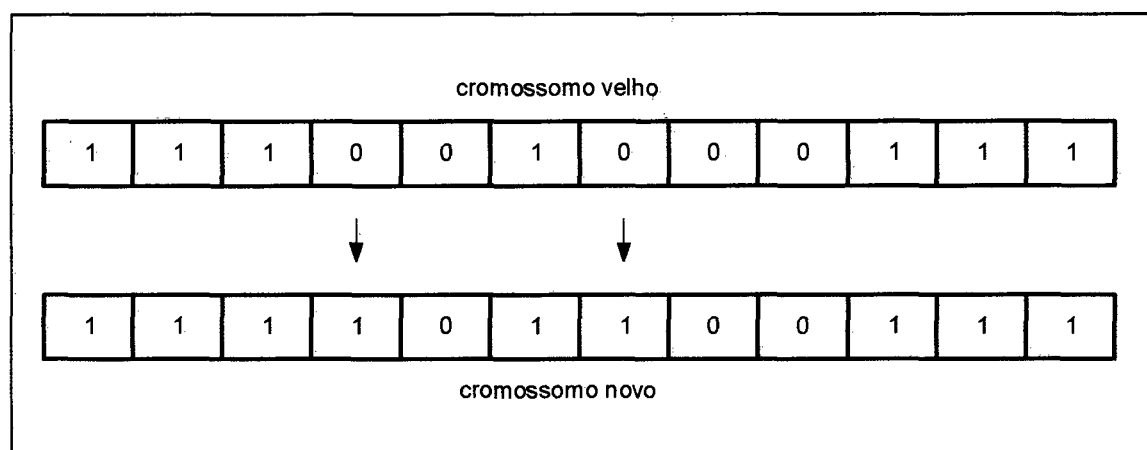


Figura 12: Exemplo de mutação.

Mutação baseada na posição (*Position-Based Mutation*): neste operador, duas posições são selecionadas randomicamente e o conteúdo da segunda posição é colocado antes da primeira. Na figura 13 temos um exemplo de mutação baseada na posição, neste caso, os alelos das posições 4 e 10 são selecionados, então o alelo da posição 10 é introduzido antes do alelo da posição 4.



Figura 13: Exemplo de mutação baseada em posição.

Mutação baseada em ordem (*Order-Based Mutation*): neste caso, duas posições são selecionadas também randomicamente e o conteúdo das mesmas é intercambiado. A figura 14 traz um exemplo de mutação baseada em ordem, aqui os alelos das posições 1 e 9 são intercambiados.

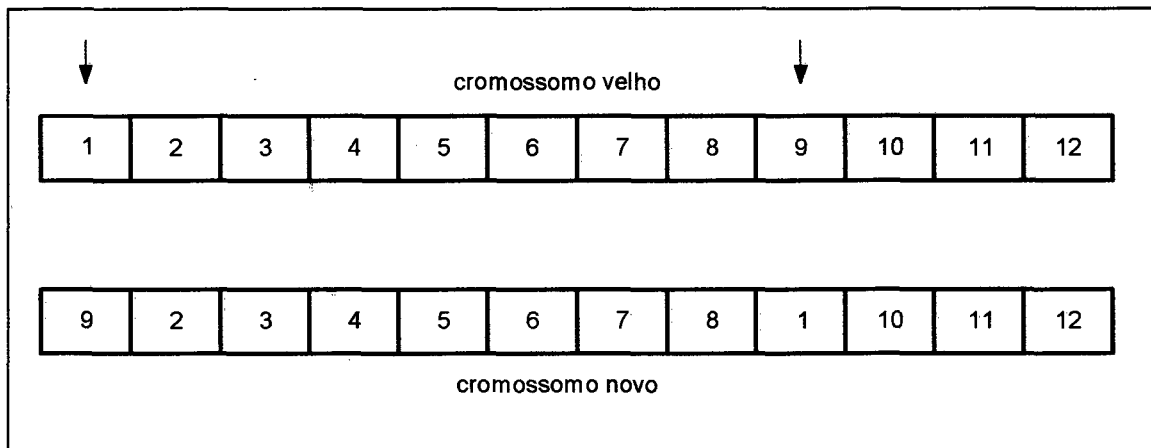


Figura 14: Exemplo de mutação baseada em ordem.

Mutação mista (*Scramble Mutation*): a mutação mista considera a vizinhança das posições importantes, neste caso, escolhe-se uma sub-lista randomicamente e permuta-se a ordem dos conteúdos. Na figura 15 os alelos das posições 1 a 4 são permutados resultando em um novo cromossomo.

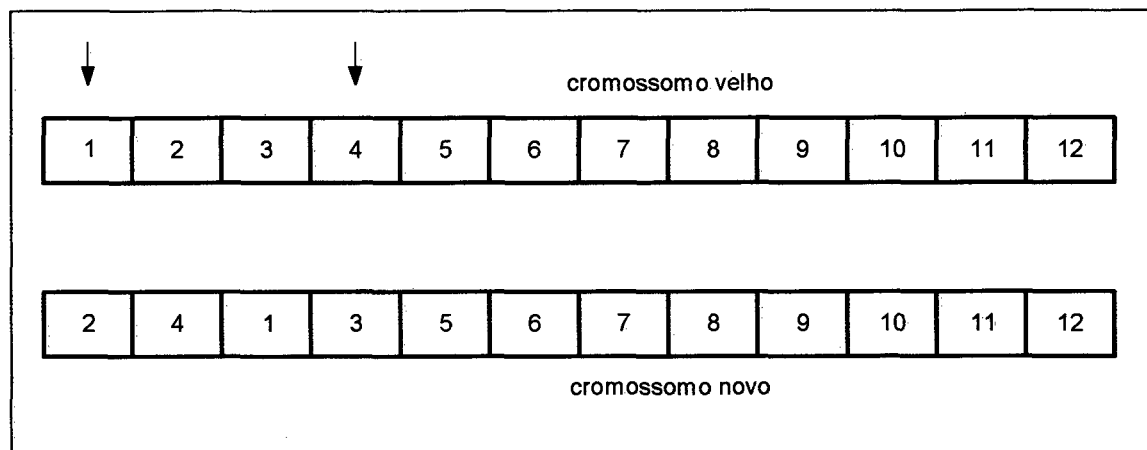


Figura15: Exemplo de mutação mista.

3.4.4.3 Inversão

A exemplo da mutação, a inversão é um operador secundário que somente suplementa o processo de cruzamento e pode ser definido como um operador unário (aplicado a um único cromossomo). Por ser um operador unário, na inversão não há a troca de informações entre os cromossomos MICHALEWICS (1997).

O operador de inversão é bastante simples, ele seleciona dois pontos de corte ao longo do cromossomo, corta o cromossomo entre estes dois pontos e inverte a ordem dos alelos MICHALEWICS (1997).

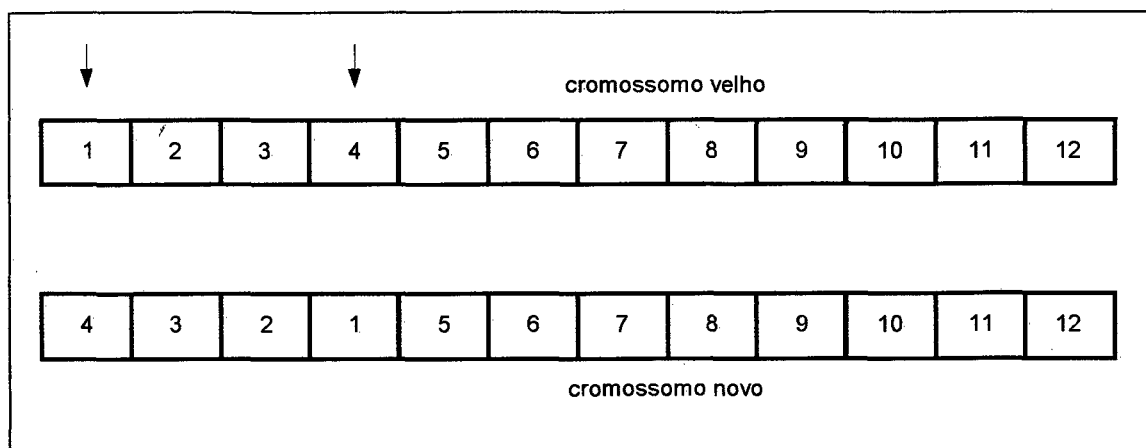


Figura 16: Exemplo de inversão.

Um exemplo de inversão pode ser visto na figura 16. Aqui foram escolhidos aleatoriamente as posições 1 e 4 como pontos de corte e os alelos entre estas posições são então invertidos, gerando um novo cromossomo.

3.4.5 Condições de término

Levando em conta os problemas de otimização, o ideal seria que o algoritmo terminasse quando o ponto ótimo fosse encontrado. Na prática, entretanto, na maioria das vezes não se pode afirmar com certeza que o ponto ótimo encontrado pelo algoritmo seja o ponto “ótimo-global”. Como consequência disso, geralmente o critério para o término utilizado é um número máximo de gerações ou um tempo limite de processamento (o que ocorrer primeiro) KOZA & RICE (1994) e TANOMARU (1995).

Outro critério interessante que também é empregado com frequência é o conceito de “estagnação”, onde o algoritmo genético termina quando nenhuma melhoria for encontrada na população depois de um certo número de gerações consecutivas MICHALEWICS (1999).

3.5 Teorema dos esquemas

O teorema dos esquemas baseia-se no fato de que quando se deseja um melhor entendimento do domínio do problema é essencial estudar tanto as similaridades entre os cromossomos como também sua aptidão. De alguma maneira o interesse se reorienta de cromossomos individuais a grupos de cromossomos de grande aptidão DAVIS (1987).

Um esquema é um conjunto (*template*) de similaridades, descrevendo um grupo de cromossomos com similaridades em certas posições e é construído introduzindo-se um símbolo (#) no alfabeto dos genes. Em um alfabeto binário, um esquema é uma cadeia de 1 *bit* tomado sobre {0,1,#} e o símbolo # pode representar qualquer *bit* 0 ou 1. É importante deixar claro que o símbolo # é somente um metasímbolo (um símbolo sobre outros símbolos), ele não deve ser processado pelo algoritmo genético.

Por exemplo, consideremos o esquema (# 1 1 1 1 0 0 1 0 0). Este esquema denota os dois cromossomos {(0 1 1 1 1 0 0 1 0 0), (1 1 1 1 1 0 0 1 0 0)}, e o esquema (# 1 # 1 1 0 0 1 0 0) denota os quatro cromossomos {(1 1 1 1 1 0 0 1 0 0), (0 1 0 1 1 0 0 1 0 0), (1 1 0 1 1 0 0 1 0 0), (0 1 1 1 1 0 0 1 0 0)}. Claro que o esquema (1 0 0 1 1 1 0 0 0 1) representa somente um cromossomo: {(1 0 0 1 1 1 0 0 0 1)}, e o esquema (#####) representa todas as combinações possíveis para 0 e 1.

3.6 Blocos de construção

É dado nome de blocos de construção a esquemas de grande aptidão e tamanho reduzido. Assim como uma criança constrói magníficas fortalezas através do arranjo de blocos de madeira, também o algoritmo genético procura por performance perto da ótima através da justaposição de blocos de construção GOLDBERG (1989). De certa forma, ao se trabalhar com blocos de construção, a complexidade do problema é reduzida; ao invés de construir cromossomos de alta-performance, tentando cada combinação possível, são construídos melhores e melhores cromossomos com as melhores partes das amostras anteriores.

É importante lembrar que o algoritmo genético depende da recombinação de blocos de construção na procura por melhores pontos. Se os blocos de construção estiverem desencaminhados devido à má codificação utilizada ou a própria função, o problema pode requerer longos tempos de espera para alcançar soluções quase ótimas GOLDBERG (1989).

3.7 Variações de algoritmos genéticos

Esta seção tem como objetivo demonstrar algumas variações que podem ser usadas na implementação de um algoritmo genético. Estas variações visam melhorar o desempenho e os resultados obtidos pelo algoritmo.

3.7.1 Algoritmos genéticos elitistas

Neste modelo, amplamente utilizado, garante-se que os melhores indivíduos de uma geração sempre aparecerão na geração seguinte. Ou seja, se a elite da população corrente não estiver na geração seguinte em decorrência de algum operador genético, então os elementos ausentes são inseridos artificialmente no lugar dos piores indivíduos. O método mais comum no elitismo supervisiona apenas o melhor indivíduo da população, mas para grandes populações pode ser interessante garantir que mais indivíduos sejam preservados. A quantidade de indivíduos que será preservado depende muito do problema a ser tratado TANOMARU (1995).

VARHOL (1994) ressalta que o processo de elitismo é bastante útil, porque muitas vezes o processo de convergência para uma solução boa pode ser muito difícil e a preservação dos melhores indivíduos pode ajudar neste processo. No entanto, quando a porcentagem de indivíduos a serem preservados for muito alta, o efeito pode ser o contrário do desejado, isso porque, como já mencionado, uma importante característica do algoritmo

genético é a diversidade genética de sua população e a medida que esta é eliminada, o processo tende a convergir para soluções de pouca qualidade.

3.7.2 Algoritmos genéticos paralelos

Um modelo de algoritmo genético paralelo mencionado por MICHALEWICS (1999) é o denominado de “insular”. Neste modelo, a população é particionada em sub-populações e a cada sub-população de indivíduos é atribuída a um processador de um computador paralelo com memória distribuída. Cada processador executa um AG convencional em sua sub-população e, periodicamente, envia cópias de seus melhores indivíduos para um processador vizinho, recebendo, em contrapartida, cópias dos melhores indivíduos do mesmo. As cópias recebidas são, em geral, usadas para substituir os piores elementos da população. O processo continua até que um critério de término seja satisfeito. Este processo de troca de indivíduos entre subpopulações é denominado de “migração”.

3.7.3 Algoritmos genéticos com população de tamanho variável

O tamanho da população é uma das escolhas mais importantes encontradas por qualquer usuário de algoritmo genético e pode ser crítica em muitas aplicações. Se a população for muito pequena, o algoritmo pode convergir muito rápido a mínimos locais. Se for uma população muito grande, o algoritmo pode necessitar de muito processamento e, por conseqüência, a espera por uma melhora na solução pode ser muito longa MICHALEWICS (1999).

Um modelo de algoritmo genético de tamanho variável de população não tem nenhuma alteração nos mecanismos de reprodução, ele simplesmente adiciona o conceito de idade (*age*) ao cromossomo. A idade de um cromossomo corresponde à quantidade de gerações que ele permanece vivo, substituindo o mecanismo de seleção da população e

influenciando diretamente no tamanho da população, uma vez que a idade do cromossomo depende somente de sua aptidão.

O tempo de vida de um cromossomo é atribuído a ele durante o processo de avaliação do mesmo e permanece constante durante o processo evolutivo, isto é, desde o nascimento de um cromossomo até sua morte. Este tempo de vida determina quantas gerações o cromossomo vai existir na população MICHALEWICS (1999).

3.7.4 Algoritmos genéticos híbridos

Muitos autores consideram que os algoritmos genéticos sozinhos nem sempre são a melhor solução para problemas de otimização específicos. Desta forma, os algoritmos híbridos utilizam os métodos de otimização tradicionais como ponto de partida para os algoritmos genéticos. A desvantagem destes algoritmos é a introdução de um *overhead* computacional devido à busca baseada em populações, característica dos AGs. A mistura de técnicas tradicionais com os algoritmos genéticos adiciona uma espécie de aprendizado no AG, pois os cromossomos utilizados podem resultar de várias técnicas, denominadas de *hill-climbing*, utilizada nos métodos de otimização tradicionais TANOMARU (1995).

3.8 Problemas práticos de algoritmos genéticos

Esta seção define os principais problemas encontrados na implementação de um algoritmo genético, levando em consideração a determinação de parâmetros, os erros de amostragem, a convergência prematura, o balanço exploração-exploração e o tempo de processamento.

3.8.1 Determinação de parâmetros

Num AG básico, o usuário deve definir o tamanho da população N , além das probabilidades de cruzamento e mutação, respectivamente.

Quanto ao parâmetro N , a intuição indica que quanto maior melhor, uma vez que em última análise, com a população cobrindo todo o espaço de busca, a solução ótima seria encontrada na primeira geração. Na prática, é óbvio que temos que lidar com tamanhos finitos e uma população com tamanho entre 50 e 200 cromossomos resolve a maior parte dos problemas. Existem ainda casos onde populações menores se fazem necessárias, isto porque o processo de avaliação de um cromossomo é excessivamente lento TANOMARU (1995).

Em relação à probabilidade de cruzamento e mutação, estudos têm mostrado que bons resultados geralmente são obtidos com alto valor de cruzamento (maior ou igual a 25%) e baixo valor de mutação (geralmente menor ou igual a 2%) TANOMARU (1995).

3.8.2 Erros de amostragem

Erros de amostragem estão relacionados ao processo de seleção, como o método da roleta explicado anteriormente. Neste método, é perfeitamente aceitável que em uma população de n indivíduos, a maior parte dos indivíduos selecionados possuam aptidão baixa, ou mesmo que indivíduos com aptidão muito alta recebam um número desordenado de descendentes. Estes erros de amostragem podem acarretar na perda de material genético importante, levando o algoritmo a obter resultados de baixa qualidade.

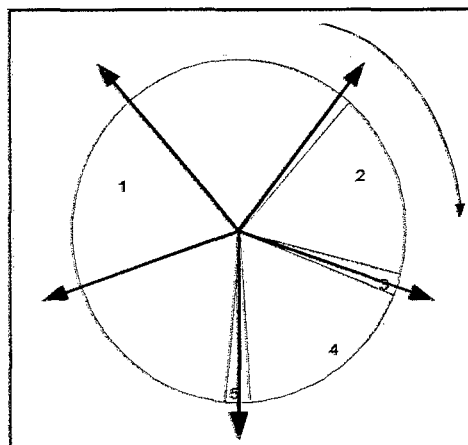


Figura 17: Exemplo de seleção.

Na figura 17 temos um exemplo de seleção, baseado no método da roleta, aplicado a uma população de 5 cromossomos. Neste exemplo, temos um super cromossomo (1) que recebe 3 cópias após o giro da roleta, enquanto que os cromossomos 3 e 5 que possuem aptidão muito baixa recebem uma cópia e os cromossomos 2 e 4 que possuem aptidão muito maiores irão desaparecer.

3.8.3 Convergência prematura

Em modelos de algoritmos simples, um fenômeno que se observa é que o algoritmo converge rapidamente (em umas poucas dezenas de gerações) para um ponto de alta qualidade, mas não ótimo global, num fenômeno denominado de convergência prematura.

Por exemplo, suponhamos que na população inicial, um indivíduo próximo de um ótimo local, mas não global, tenha o seu valor de adequabilidade muito maior do que o restante da população. Devido ao processo de seleção, tal superindivíduo terá vários descendentes na próxima geração. Em casos extremos, indivíduos próximos a um ótimo global, mas com baixa adequabilidade relativa, podem ser completamente extintos, o que

geralmente implica na convergência do algoritmo a um ótimo não-global TANOMARU (1995).

3.8.4 Balanço exploração-exploração

Balanço exploração-exploração significa o balanço entre “explorar” no sentido de investigar regiões desconhecidas do espaço de busca e “explorar” no sentido de usufruir o conhecimento genético de indivíduos da população.

Basicamente, o cruzamento usa a bagagem genética de indivíduos, ao passo que mutação faz uma busca aleatória no espaço de busca. Deste modo, o cruzamento “explora” indivíduos realizando uma busca local, enquanto que mutação “explora” novas regiões, realizando, deste modo, uma busca global TANOMARU (1995).

Com base na afirmação acima pode-se dizer que muito cruzamento e pouca mutação podem tornar a busca demasiadamente localizada, enquanto que muita mutação e pouca exploração podem tornar a busca desordenada e assemelhar-se a uma busca aleatória TANOMARU (1995).

3.8.5 Tempo de processamento

Um obstáculo na grande maioria das aplicações práticas dos algoritmos genéticos é o excessivo tempo de processamento. Em geral, a culpa não é dos AGs em si, mas do problema em questão. Geralmente AGs conseguem obter soluções no mínimo subótimas investigando apenas uma fração do espaço de busca. Quando o espaço de busca é demasiadamente amplo, a busca genética pode ser exaustivamente lenta CELKO (1993).

CAPÍTULO 4

Algoritmo Simulated Annealing

Este capítulo é composto por uma visão inicial a respeito do algoritmo *simulated annealing*, na sua versão original, utilizada na termodinâmica (simulando o processo de recozimento de sólidos). Em seguida, é apresentado o algoritmo como uma ferramenta utilizada na otimização de problemas.

4.1 Introdução

O nome recozimento (*annealing*) é dado ao processo de aquecimento de um sólido até o seu ponto de fusão, seguido de um resfriamento gradual e vagaroso, até que se alcance novamente o seu enrijecimento. Neste processo, o resfriamento vagaroso é essencial para se manter um equilíbrio térmico no qual os átomos encontram tempo suficiente para se organizarem em uma estrutura uniforme com energia mínima a cada temperatura. Se o sólido é resfriado bruscamente, seus átomos formam uma estrutura irregular e fraca com alta energia, em consequência do esforço interno gasto MAZZUCCO (1999).

Computacionalmente, o recozimento pode ser visto como um processo estocástico de determinação de uma organização dos átomos de um sólido que apresente energia mínima. Em temperatura alta, os átomos se movem livremente e com grande probabilidade podem mover-se para posições que incrementem a energia total do sistema. Quando se baixa a temperatura, os átomos gradualmente se movem em direção a uma estrutura regular e, somente com pequena probabilidade, incrementam suas energias MAZZUCCO (1999).

No ano de 1953, METROPOLIS et al. (1953) propôs um algoritmo para a eficiente simulação do processo de evolução de um sólido até seu equilíbrio térmico. Levou quase 30 anos para os estudiosos KIRKPATRICK et al. (1983) e posteriormente CERNY (1985) perceberem que existe uma profunda analogia entre o processo de minimizar o custo de uma função em um problema de otimização e o resfriamento vagaroso de um sólido até que ele atinja seu equilíbrio térmico LAARHOVEN & AARTS (1987).

Segundo AARTS & KORST (1989), o algoritmo introduzido por esses autores é baseado em técnicas de “monte carlo” e gera uma seqüência de estados do sólido como descrito a seguir. Dado um estado corrente i do sólido com energia E_i , então o estado subsequente j é gerado aplicando um mecanismo de perturbação que transforma o estado corrente no próximo estado por uma pequena distorção. A energia do próximo estado é E_j . Se a diferença de energia ($\Delta E = E_j - E_i$) for menor ou igual a 0, o estado j é aceito como estado corrente. Se a diferença de energia for maior que 0, o estado j é aceito com uma certa probabilidade que é dada por:

$$\exp(-\Delta E / k_b T) \tag{4.1}$$

Onde T denota a temperatura do banho quente e k_b uma constante da física conhecida como constante de *Boltzmann*. A regra de aceite descrita acima é conhecida como critério de *Metropolis* e o algoritmo que o usa é conhecido como algoritmo de *Metropolis*.

Se a redução da temperatura é feita suficientemente lenta, o sólido pode encontrar o equilíbrio térmico em cada temperatura. No algoritmo de *Metropolis*, isto é obtido gerando um grande número de transições em uma determinada temperatura AARTS & KORST (1989).

O método computacional que imita esse processo de recozimento de um sólido é chamado de *simulated annealing*.

4.2 *Simulated annealing* no processo de otimização

Simulated annealing é considerado um tipo de algoritmo que tem seu princípio básico nos algoritmos de busca local. Ele se constitui em um método de obtenção de boas soluções para problemas de otimização de difíceis soluções. Desde a sua introdução como um método de otimização, esse método vem sendo vastamente utilizado em diversas áreas, tais como projeto de circuitos integrados auxiliados por computador, processamento de imagens, redes neurais, etc MAZZUCCO (1999).

Segundo AARTS & KORST (1989), a semelhança do algoritmo *simulated annealing* com o método original baseado no recozimento de sólidos é muito grande, sendo que no caso dos problemas de otimização, o algoritmo segue a seguinte analogia:

- soluções em um problema de otimização são equivalentes a estados em um sistema físico;
- o custo de uma solução é equivalente à energia de um estado;
- a seleção de uma solução vizinha em um problema de otimização é equivalente à perturbação de um estado físico;
- o ótimo global de um problema é equivalente ao estado fundamental de um sistema de partículas, e
- um ótimo local de um problema é equivalente a resfriamento rápido no sistema físico.

Uma das maiores, senão a maior, qualidade deste algoritmo é o fato de que ele, ao contrário de seu algoritmo de origem (algoritmo de busca local, também conhecido como algoritmo descendente), às vezes aceita uma nova solução gerada mesmo que esta solução resulte em um aumento na energia (função de custo f no caso dos problemas de otimização) YONEYAMA & NASCIMENTO (2000). Segundo MAZZUCCO (1999), o aceite ou rejeição de uma nova solução que causará um incremento de δ em f , em uma temperatura T , é determinado por um critério probabilístico, através de uma função de aceite denominada por:

$$G(\delta, T) = e^{-\delta T} \quad (4.2)$$

Caso $\delta = f(j) - f(i)$ for menor que zero, a solução j será aceita como a nova solução corrente. Caso contrário, a nova solução será aceita se:

$$G(\delta, T) > \text{random}(0,1) \quad (4.3)$$

A semelhança com o método original de simulação de recozimento na termodinâmica como já aludido, é muito grande, pois o parâmetro δ corresponde à variação da energia de um estado para o outro (ΔE) e o parâmetro de controle T , corresponde à temperatura. Assim, o algoritmo *simulated annealing* é iniciado com um valor de temperatura T relativamente alto para evitar que prematuramente fique preso a um mínimo local. O algoritmo então procede tentando um certo número de movimento na vizinhança em cada temperatura, enquanto o parâmetro temperatura é gradualmente reduzido. O quadro 3 ilustra o fluxo básico do algoritmo *simulated annealing*.

```

Procedure Simulated Annealing
Inicio
  Inicializar (i)
Repita
  Para l := 1 ate L
  Inicio
    Gerar(j de i)
    Se f(j) ≤ f(i) Entao
      i := j
    Senao
      Se exp((f(i) - f(j)) / Temperatura) > random[0,1] Entao i:=j
  Final
  Calcular(Temperatura)
Ate que (CritérioParada)
Final

```

Quadro 3: Fluxo básico do algoritmo *simulated annealing*.

O funcionamento do algoritmo é bem simples e, como já mencionado, seu ponto forte está centrado no fato de que às vezes, mesmo quando a função de custo é incrementada, o processo aceita novas soluções, visando encontrar melhores soluções mais adiante. A figura 18 ilustra este processo onde o algoritmo tenta fugir de mínimos locais.

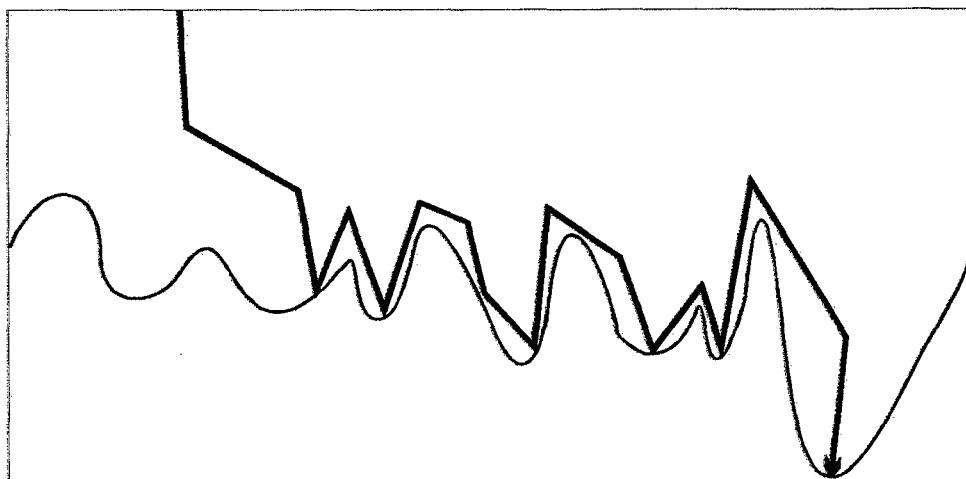


Figura 18: Estratégia do algoritmo *simulated annealing*.

4.3 Desempenho do algoritmo

O sucesso, assim como a falha, do algoritmo *simulated annealing* está diretamente relacionado à determinação de alguns parâmetros relacionados a temperatura inicial, decremento da temperatura, tamanho da busca e critério de parada.

4.3.1 Temperatura inicial

A temperatura inicial do processo é de extrema importância para a obtenção de bons resultados com o algoritmo. O processo deve iniciar relativamente livre, isto significa que a

temperatura inicial T_0 deve ser bastante alta a ponto de que qualquer nova solução gerada possa ser aceita AARTS & KORST (1989).

4.3.2 Decremento da temperatura

O processo inicia com uma temperatura inicial T_0 e esta temperatura é decrementada lentamente (vale ressaltar o fato já mencionado referente ao resfriamento muito rápido que pode resultar em uma solução de baixa qualidade).

Segundo AARTS & KORST (1989), a função que decrementa a temperatura mais conhecida é dada por:

$$T_{k+1} = \alpha * T_k, k = 1, 2, 3 \quad (4.4)$$

Onde α é uma constante menor, porém muito perto de 1. O primeiro valor proposto para esta constante pode ser encontrado em KIRKPATRICK et al. (1983), com $\alpha = 0,95$. Na prática, são usados valores entre 0,8 e 0,99 para este parâmetro.

4.3.3 Tamanho da busca

Para que o estado de equilíbrio seja alcançado em uma determinada temperatura, a cada estado do sistema (mudança de temperatura), um determinado número de possíveis soluções deve ser examinada. Técnicas mais elaboradas baseiam-se no argumento de que, para cada valor de temperatura, um número mínimo de transições deve ser aceitas. Entretanto, à medida que o valor da temperatura se aproxima de 0, as transições são aceitas a uma probabilidade decrescente e este número de transições pode não ser alcançado. Para solucionar este problema limita-se o número de tentativas em um valor máximo AARTS & KORST (1989).

4.3.4 Critério de parada

Segundo AARTS & KORST (1989), são 3 os critérios de parada mais utilizados. No primeiro, o algoritmo termina quando a variável de controle atinge um valor mínimo previamente determinado; no segundo, o sistema termina quando a temperatura tenha sido alterada um determinado número de vezes, e no terceiro, o sistema termina quando a solução aceita pelo sistema permanece inalterada por um determinado número de mudanças de temperatura.

4.4 *Simulated annealing* paralelo

O ponto chave no desenvolvimento de um algoritmo *simulated annealing* paralelo é a distribuição da execução das várias partes do algoritmo entre um determinado número de processos paralelos comunicantes. Esta variação é promissora no que diz respeito a acelerar o algoritmo, mas não é de forma alguma uma tarefa fácil. Esta dificuldade se deve à natureza seqüencial do algoritmo AARTS & KORST (1989).

A seguir, estão descritas duas estratégias para se desenvolver um algoritmo paralelo mencionadas por AARTS & KORST (1989): o algoritmo de única trilha e o algoritmo de múltiplas trilhas.

Algoritmo paralelo de única-trilha (*single-trial*): nesta abordagem, a tarefa de avaliar uma única trilha é distribuída entre um determinado número de processos paralelos. Aqui, o aumento na velocidade obtido está fortemente relacionado com o problema em questão. Em alguns problemas de otimização, a tarefa de avaliação de uma instância do problema pode ser subdividida entre processos concorrentes sem maiores dificuldades como no caso do problema de corte e empacotamento. Já em outros, como no caso do problema do caixeiro viajante, a tarefa de dividir a avaliação de uma instância do problema entre processos paralelos torna-se muito difícil. Infelizmente, este segundo caso é o da maioria dos problemas de otimização.

Algoritmo paralelo de múltiplas-trilhas (*multiple-trial*): nesta abordagem, as trilhas são avaliadas em paralelo. Aqui, os processos geram suas próprias instâncias a partir de uma mesma solução. Este processo se repete até que algum dos processos aceite uma nova solução. Então, todos os processos param e a nova solução aceita é transmitida para todos estes. O processo reinicia então a partir da nova solução gerada. Caso mais processos tentarem aceitar uma nova solução ao mesmo tempo, um dos processos é escolhido e a solução do mesmo é aceita, as demais são descartadas. Com esta abordagem, a aceleração do algoritmo é pequena para valores altos do parâmetro de controle, entretanto, enquanto o valor do parâmetro de controle vai caindo, a velocidade aumenta e, eventualmente, se torna proporcional ao número de processos.

4.5 Considerações teóricas

Segundo MAZZUCCO (1999), matematicamente o algoritmo *simulated annealing* pode ser modelado através da teoria de cadeias de Markov. Utilizando esse modelo, vários resultados importantes têm surgido na literatura. Estes resultados dizem respeito a condições suficientes para convergência. Entretanto, grande parte desses trabalhos não leva em consideração o número de iterações necessárias para se atingir esta convergência. Uma vez que o espaço de soluções S cresce exponencialmente com o tamanho do problema, o tempo de execução de um algoritmo desse tipo pode alcançar níveis inviáveis. Um resultado muito importante é fornecido pelo trabalho publicado por HAJAK (1988), no qual não só as condições necessárias e suficientes para a convergência assintótica do algoritmo para um conjunto de soluções ótimas globais são fornecidas, mas também o número de iterações necessárias para que esta convergência possa ocorrer.

CAPÍTULO 5

Método desenvolvido

Neste capítulo, encontra-se descrita a abordagem proposta, que combina os algoritmos genéticos e *simulated annealing* para a solução de problemas de otimização, mais especificamente o conhecido problema do caixeiro viajante.

Inicialmente, o problema do caixeiro viajante é formalmente definido assim como sua modelagem. Por fim, o método proposto é apresentado e suas principais características são descritas.

5.1 Introdução

O problema do caixeiro viajante é um dos mais importantes e estudados problemas de otimização. A importância de encontrar um algoritmo aproximado para este problema reside no fato de muitos problemas que ocorrem na ciência da computação, e em muitas outras áreas, poderem ser modelados a partir do mesmo. Outra importante característica é a grande dificuldade de se achar uma solução exata para instâncias maiores, visto que é um dos problemas de otimização mais difíceis, dentre os conhecidos. Nas últimas décadas, vem crescendo o desenvolvimento de algoritmos aproximados, geralmente utilizando métodos heurísticos que forneçam soluções aceitáveis em tempo de processamento compatível com as necessidades LIN (1965) e GOLDBARB (2000).

Atualmente, observa-se um aumento considerado no interesse por abordagem baseadas em técnicas de otimização de busca local para o tratamento de problemas NP-Completo. São métodos aproximados que normalmente utilizam como base os algoritmos: genético, *simulated annealing*, *tabu search* e redes neurais ARAUJO (2001).

Esses algoritmos têm sido intensivamente pesquisados em soluções de problemas gerais de otimização e, especialmente, nos combinatórios. O grande sucesso alcançado por esses algoritmos, principalmente nos últimos anos, é consequência de diversas características que os mesmos têm, dentre eles: a utilização de mecanismos modelados diretamente da natureza; aplicabilidade geral; flexibilidade nas adaptações às características específicas em casos reais; e a excelente relação qualidade e facilidade de implementar ARAUJO (2001).

5.2 Definição formal do problema

O problema do caixeiro viajante pode ser definido da seguinte forma: um viajante é requisitado a visitar cada uma das n cidades uma e somente uma vez. O viajante deve iniciar de qualquer cidade e retornar a cidade inicial, percorrendo o menor caminho existente.

Neste trabalho, este problema é tratado como sendo um problema de otimização e pode ser definido como um par (S, f) onde S é o conjunto finito de todas as possíveis rotas e f é a função de custo a ser minimizada. Uma possível solução I para o problema é definida por uma permutação $\pi (i_1, i_2, i_3, \dots, i_n)$ de inteiros de 1 até n , com n sendo o número de cidades.

Em geral, algumas restrições são colocadas quando da obtenção de alguma solução para o problema. A mais comum é dizer que o problema é euclidiano, ou seja, as distâncias estão distribuídas num plano e obedecem à desigualdade triangular e à simetria ARAUJO (2001).

5.3 Modelagem

Para qualquer aplicação que envolva os algoritmos: genéticos e *simulated annealing* na otimização de problemas, é fundamental que se defina primeiramente a forma de representação das soluções do problema. Uma representação natural do espaço de soluções para o problema em questão é, portanto, aquela que contém o conjunto de todas as permutações possíveis. Define-se, assim, uma solução I para o problema, como sendo um elemento do conjunto de todas as permutações, que é uma possível rota.

$$I = (i_1, i_2, \dots, i_n) \quad (5.1)$$

Onde cada elemento i do conjunto deve ser interpretado como as cidades naquela solução I e a ordem na qual as n cidades devem ser visitadas.

Dessa forma, utilizando esse modelo de representação, o problema passa a ter seu espaço de busca como sendo o conjunto de todas as permutações possíveis, isto é, determina-se, para cada permutação, o custo da rota correspondente. A melhor solução será aquela que apresentar a rota de menor custo.

O processo de avaliação da função de custo para uma rota do problema pode considerar inúmeras variáveis, como, por exemplo, a distância, o tempo, o custo, entre outros. Neste trabalho, a função de custo considera apenas o tamanho do percurso e pode ser definida da seguinte forma.

$$f = c_{i_1, i_2} + c_{i_2, i_3} + \dots + c_{i_n, i_1} \quad (5.2)$$

Onde c significa o custo para ir de uma cidade a outra e i_1, i_2, \dots, i_n são as cidades na ordem que as mesmas devem ser visitadas.

5.4 Método proposto

O método proposto neste trabalho tem a intenção de criar uma nova abordagem para a obtenção de soluções de boa qualidade para problemas de otimização, mais especificamente o problema do caixeiro viajante. Não existe a intenção de se obter resultados recordes em relação a tempo de processamento ou a quantidade de cidades envolvidas.

5.4.1 Descrição do método

Em um projeto que envolva a aplicação do algoritmo genético, um ponto importante a ser tratado são os operadores genéticos utilizados. A maioria dos projetos adota o esquema original onde são utilizados dois operadores genéticos: os operadores de cruzamento e a mutação.

O operador de cruzamento tem a função de combinar as melhores estruturas de seus geradores para formarem novos indivíduos que na teoria são melhores que seus geradores.

O operador de mutação cria novos pontos no espaço de busca através de um processo aleatório (desta forma, a nova estrutura pode, perfeitamente, ser de qualidade inferior).

O princípio do algoritmo é bastante simples e de fácil implementação, mas na prática muitas vezes o algoritmo genético não é capaz de obter bons resultados, quando aplicado a problemas de otimização. Um dos fatores responsáveis pela má qualidade dos resultados obtidos são os operadores genéticos utilizados. Com base em seus operadores pode-se dizer que muito cruzamento e pouca mutação podem tornar a busca demasiadamente localizada, enquanto que muita mutação e pouco cruzamento podem tornar a busca desordenada e semelhante à busca aleatória TANOMARU (1995).

O método proposto, como já mencionado, se caracteriza pela introdução de um novo operador no processo do algoritmo genético. Este operador é baseado no algoritmo

simulated annealing e deve ser aplicado à população de forma semelhante aos demais operadores. A taxa a ser utilizada deve se aproximar às taxas utilizadas pelo operador de mutação e deve ser definida de acordo com o problema.

O objetivo deste operador é tornar a busca mais agressiva uma vez que além de criar novos pontos no espaço de busca, estes pontos serão, em média, melhores (isto porque, como mencionado, este novo operador é baseado no algoritmo *simulated annealing*). Teoricamente, este novo operador é semelhante a um operador de mutação, mais aprimorado, garantindo a diversidade da população e ainda um melhoramento na aptidão média da população (embora esta teoria não seja conclusiva).

Daqui por diante o algoritmo genético básico será tratado como “algoritmo genético original” e o método proposto será tratado como “algoritmo genético SA”.

5.4.2 Fluxo do algoritmo genético SA

O fluxo do algoritmo genético original como explicado no capítulo 3 é composto pelos operadores de cruzamento e mutação. Neste trabalho foram adicionados os operadores de inversão e elitismo, também mencionados no capítulo 3. Assim, o fluxo do algoritmo genético original pode ser visto no quadro 4

O processo do algoritmo genético é bem simples, basta criar uma população inicial, geralmente de forma aleatória, em seguida aplicar o processo de avaliação a esta população, então uma nova população é formada com base na aptidão da população anterior. Tendo gerado a nova população, são aplicados os operadores genéticos de cruzamento, mutação, inversão e elitismo. Este processo se repete até que uma condição de término seja satisfeita.

```

Procedure Algoritmo Genético;
Inicio
  Geracao = 0;
  Inicializar Populacao
Enquanto (Condição <> Fim) Faca
  Avaliar Populacao
  Geracao = Geracao + 1
  Selecionar Populacao(Geracao) a partir de Populacao(Geracao - 1)
  Cruzamento em Populacao
  Mutação em Populacao
  Inversão em Populacao
  Elitismo em Populacao
Final
Final

```

Quadro 4: Fluxo do algoritmo genético original.

Como mencionado acima, a modificação do método proposto é realizada pela introdução de um novo operador no processo do algoritmo genético, o operador de “mutacaoSA”, assim um novo fluxo é definido e pode ser visto no quadro 5.

```

Procedure Algoritmo Genético SA
Inicio
  Geracao = 0
  Inicializar Populacao
Enquanto (Condição <> Fim) Faca
  Avaliar Populacao
  Geracao = Geracao + 1
  Selecionar Populacao(Geracao) a partir de Populacao(Geracao - 1)
  Cruzamento em Populacao
  Mutação em Populacao
  MutacaoSA em Populacao
  Inversão em Populacao
  Elitismo em Populacao
Final
Final

```

Quadro 5: Fluxo do algoritmo genético SA.

O fluxo deste algoritmo é semelhante ao fluxo do algoritmo genético original, a única alteração é a inclusão de uma nova etapa no processo, onde a população sofre o efeito do operador “mutacaoSA”. O funcionamento deste operador pode ser visto no quadro 6.

```

Procedure MutacaoSA
Inicio
   $i := \text{CromossomoEscolhido}$ 
Repita
  Para  $l := 1$  ate  $L$ 
    Inicio
      Gerar( $j$  de  $i$ )
      Se  $f(j) \leq f(i)$  Entao
         $i := j$ 
      Senao
        Se  $\exp((f(i) - f(j)) / \text{Temperatura}) > \text{random}[0,1]$  Entao  $i := j$ 
    Final
  Calcular(Temperatura)
Ate que (CritérioParada)
Final

```

Quadro 6: Fluxo do operador de mutaçãoSA.

Como já mencionado, este operador é baseado no algoritmo *simulated annealing*, ou seja, quando um cromossomo for selecionado a sofrer o efeito do operador de mutacaoSA, este cromossomo é passado ao operador que irá executar o algoritmo *simulated annealing* convencional sobre este cromossomo e devolve o resultado que é inserido no lugar do cromossomo selecionado.

Na teoria, este novo operador deve tornar a busca mais agressiva, permitindo a obtenção de melhores resultados pela exploração de somente uma fração do espaço de busca.

Neste trabalho é feita uma avaliação comparativa do algoritmo genético SA aplicado ao problema do caixeiro viajante. São comparados os resultados obtidos com o algoritmo

genético original, que é a implementação básica do AG, com os resultados obtidos pelo método proposto. Os testes foram realizados sempre com as mesmas instâncias, quantidade de gerações e probabilidades associadas aos operadores.

5.4.3 Representação cromossômica

Existe uma concordância a respeito de que a representação binária não é aplicável ao problema do caixeiro viajante, o que não é difícil de se entender uma vez que o que se está procurando é a melhor permutação de cidades. Neste caso, a representação binária não traria nenhuma vantagem, pelo contrário, somente traria desvantagens, pois a representação requer algoritmos de reparo, uma vez que uma simples alteração em um alelo poderia resultar em uma rota ilegal MICHALEWICS (1999).

MICHALEWICS (1999) explica que durante os últimos anos 3 tipos de representação (baseada em vetor) vêm sendo estudadas para problemas de caixeiro viajante, são elas: a representação adjacente, a representação ordinal e representação de caminho; sendo a representação de caminho, provavelmente, a representação mais natural para uma rota.

Neste trabalho a notação adotada foi a baseada na representação do caminho. Esta escolha se deu pelo fato de que esta notação permite a utilização dos operadores genéticos mencionados no capítulo 3.

5.4.4 Avaliação da população

Como mencionado anteriormente, processo de avaliação consiste na aplicação de uma função de avaliação em cada um dos indivíduos da população corrente. Esta função deve expressar a qualidade de cada indivíduo da população, no contexto do problema considerado. Como a qualidade de um cromossomo está diretamente relacionada com o

tamanho do percurso, a função de avaliação é obtida calculando-se o tamanho do percurso através da equação 5.2.

5.4.5 Seleção da população

No processo de seleção, o primeiro passo é a conversão da aptidão individual em uma expectativa, que é o número esperado de descendentes, para cada indivíduo. O método utilizado para esta conversão é baseado no método universal estocástico explicado no capítulo 3.

No segundo passo, os valores de expectativas são transformados em valores reais de descendentes que cada indivíduo irá produzir. O método escolhido para esse cálculo foi o método da “roleta” também explicado no capítulo 3.

5.4.6 Operadores genéticos

A seguir, são apresentadas as características do algoritmo genético no que diz respeito aos operadores genéticos de mutação, inversão, cruzamento e elitismo utilizados.

Cruzamento: Os operadores de cruzamento testados foram os mencionados no capítulo 3: o operador CX, o operador PMX e o operador OX.

Mutação: da mesma forma que no cruzamento, os operadores de mutação testados foram os mencionados no capítulo 3: a mutação baseada na ordem, a mutação baseada na posição e a mutação mista.

Inversão: como mencionado anteriormente, o operador de inversão tem obtido bons resultados quando aplicado ao problema do caixeiro viajante. Tendo em vista esta afirmação resolveu-se incluir este operador no protótipo implementado e o método utilizado foi o descrito neste trabalho.

5.4.7 Operador de mutação SA

Este é o operador introduzido pelo método proposto, baseado no algoritmo *simulated annealing*. Neste processo, quando um cromossomo é selecionado a sofrer o efeito do operador de mutação SA, ele é retirado da população, é aplicado sobre este cromossomo, o algoritmo *simulated annealing* e o resultado do algoritmo é introduzido novamente na população.

O algoritmo *simulated annealing* foi implementado com base no método proposto por HANSEN (1995)

5.5 Implementação

A implementação do algoritmo foi realizada na linguagem de programação Pascal 7.0, em um micro computador PC compatível com processador K7, com *clock* de 800 Mhz, 256 Mb de memória RAM e sistema operacional Windows 98.

O protótipo utiliza como base a abordagem dos algoritmos: genético e *simulated annealing* para solucionar instâncias moderadas do problema do caixeiro viajante, dentro do conceito de soluções aproximadas.

5.6 Estrutura de dados

Utilizar a estrutura de dados adequada é uma questão crucial para o bom desempenho de qualquer *software* SAHNI & HOROWITZ (1994). A escolha da estrutura de dados adequada pode ter um grande impacto na eficiência do algoritmo e quando não for a mais adequada, pode levar o mesmo a obter resultados muito aquém dos desejados.

Um extenso comentário sobre o comportamento em termos de eficiência das principais estruturas de dados que podem ser utilizadas em algoritmos heurísticos para

resolver o problema do caixeiro viajante pode ser encontrado em FREDMAN et al. (1993), onde são examinados seus desempenhos tanto do ponto de vista teórico quanto experimental. Dentre essas estruturas de dados está a representação tradicional que é a baseada em vetor (*array*) ARAUJO (2001).

O algoritmo analisado neste trabalho utiliza uma estrutura de dados bastante simples. Os principais elementos desta estrutura, que merecem ser mencionadas, são a definição de uma cidade (alelo) em termos de coordenadas no plano, a representação de uma rota (cromossomo) e o conjunto de cromossomos (população), conforme relatado a seguir.

Cidade: uma cidade é definida por duas coordenadas do tipo real que representam a localização da cidade em relação ao plano no qual as mesmas se encontram, que na nomenclatura da linguagem Pascal será representada por uma estrutura de dados do tipo registro. Esta notação pode ser vista no quadro 7.

```
Type Cidade = Record  
    X, y : Real;  
End;
```

Quadro 7: Estrutura de uma cidade.

Rota: uma rota de n cidades é definida como uma seqüência de cidades que devem ser visitadas pelo viajante. Na nomenclatura da linguagem Pascal uma rota será representada por uma estrutura de dados do tipo vetor de n elementos do tipo cidade como mostra o quadro 8.

```
Type Rota = Array[1..n] of Cidade;
```

Quadro 8: Estrutura de uma rota.

População: uma população com m elementos é definida como uma seqüência de cromossomos, que na nomenclatura da linguagem Pascal será representada por uma estrutura de dados do tipo vetor de m elementos do tipo rota como mostra o quadro 9.

```
Type Populacao = Array[1..m] of Rota;
```

Quadro 9: Estrutura da população.

Embora haja conveniências na escolha da forma de representação do problema, algumas formas de representação facilitam mais o entendimento do problema, enquanto outras favorecem mais a sua solução o que deixa o programador bastante à vontade nessa escolha ARAUJO (2001).

5.6.1 Procedimentos

A seguir serão apresentados os principais procedimentos que merecem destaque, pois são críticos dentro do contexto geral da implementação. Estes procedimentos são pequenos e bastante funcionais, o que os tornam fáceis de serem entendidos.

Procedimento distância: este procedimento calcula a distância entre duas cidades quaisquer através de suas coordenadas, utilizando a métrica euclidiana. Este procedimento está descrito no quadro 10.


```

Function distancia (p, q : Cidade) : Double;
Var
    dx, dy : real;
Begin
    dx := q.x - p.x;
    dy := q.y - p.y;
    distancia := sqrt(dx * dx + dy * dy)
end

```

Quadro 10: Função de cálculo da distância entre duas cidades.

Procedimento tamanho: o viajante visita as cidades em ordem numérica $1, 2, \dots, n$ antes de retornar para cidade 1. O custo da rota é a soma das distâncias entre as cidades sucessivas. O procedimento que calcula o tamanho do percurso de uma rota pode ser visto no quadro 11.

```

Function Tamanho (a : Rota) : Real;
Var
    i : Integer;
    sum : Real;
Begin
    Sum := Distancia (a[n], a[1]);
    For i := 1 to n-1 do
        Sum := sum + distancia( a[i], a[i + 1] );
    Custo := sum;
end

```

Quadro 11: Função que calcula o tamanho de um percurso.

Procedimento algoritmo genético: este é o procedimento principal do algoritmo, onde a população inicial é criada, em seguida a população é avaliada, selecionada e sofre os efeitos dos operadores genéticos. Este processo se repete um determinado número de gerações ou até que o final do processo seja solicitado. O quadro 12 traz o procedimento do algoritmo genético.

```
Procedure AlgoritmoGenetico;  
var  
    TotFitnes : Double;  
    GeracaoParcial : Integer;  
begin  
    IniciaPopulcao;  
    GeracaoAtual := 0;  
    Repeat  
        Inc(GeracaoAtual);  
        Aptidao(TotFitnes);  
        SeleccionaPopulacao(TotFitnes);  
        OperadoresGeneticos;  
        Inc(GeracaoParcial)  
    Until (GeracaoAtual = Geracao) or (Keypressed);  
end;
```

Quadro 12: Procedimento algoritmo genético.

Cálculo da aptidão: é neste procedimento que são aplicados os cálculos referentes à aptidão da população e número esperado de descendentes. Inicialmente, a aptidão é calculada através da função que calcula o tamanho do percurso, em seguida o número esperado de descendentes é calculado baseado no método do desvio padrão mencionado no capítulo 3. Este procedimento pode ser visto no quadro 13.

```

Procedure Aptidao(Var TotFitnes:Double);
Var
  Indice,Aux : Integer; Media,DesvioPadrao : Double;
begin
  for Aux := 1 to TamanhoPopulacao do
    begin
      Populacao[Aux].Custo:= Tamanho(QdeCidades,Populacao[Aux].Tour);
      Populacao[Aux].Fitnes := 1 / Populacao[Aux].Custo;
      TotFitnes := TotFitnes + Populacao[Aux].Fitnes;
    end;
  Media := TotFitnes / TamanhoPopulacao;
  For Aux := 1 to TamanhoPopulacao do
    begin
      DesvioPadrao := DesvioPadrao+Sqr(Populacao[Aux].Fitnes- Media);
    end;
  DesvioPadrao := Sqrt(DesvioPadrao / (TamanhoPopulacao - 1));
  TotFitnes := 0;
  if DesvioPadrao <> 0 then
    begin
      For Aux := 1 to TamanhoPopulacao do
        begin
          Populacao[Aux].Fitnes := ((Populacao[Aux].Fitnes - Media) /
            (2*DesvioPadrao)) + 1;
          TotFitnes := TotFitnes + Populacao[Aux].Fitnes;
        end;
      end
    else
      begin
        For Aux := 1 to TamanhoPopulacao do
          begin
            Populacao[Aux].Fitnes := 1;
            TotFitnes := TotFitnes + Populacao[Aux].Fitnes;
          end;
        end;
      end;
    end;
end;

```

Quadro 13: procedimento de cálculo da aptidão.

Procedimento de seleção: este procedimento faz a seleção dos indivíduos que irão compor a próxima geração. O método escolhido para o processo de seleção foi o método da roleta. Inicialmente o processo aloca a cada cromossomo uma fatia, correspondente à sua expectativa. Depois, após o giro da roleta, com cursores igualmente espaçados, os cromossomos que tiverem cursores apontando para sua área são selecionados. O quadro 14 traz o procedimento responsável pelo processo.

```

Procedure SeleccionaPopulacao(Var TotFitnes:Double)
var
    Espaco,Dardo, Parcial : Double; Aux,Indice : Integer;
    PopulacaoFilhos : TipoPopulacao;
Begin
    for Aux := 1 to TamanhoPopulacao do
        begin
            Populacao[Aux].Fitnes := (((100 * Populacao[Aux].Fitnes) /
            TotFitnes) / 100) + Parcial; Parcial := Populacao[Aux].Fitnes;
        end;
    Aux := 1; Espaco := 1 / TamanhoPopulacao; Dardo := Random;
    BuscaPai(Indice, Dardo); New(PopulacaoFilhos[Aux]);
    PopulacaoFilhos[Aux] := Populacao[Indice];
    for Aux := 2 to TamanhoPopulacao do
        begin
            Dardo := Dardo + Espaco;
            if Dardo > 1 then
                Dardo := Dardo - 1;
            BuscaPai(Indice, Dardo); New(PopulacaoFilhos[Aux]);
            PopulacaoFilhos[Aux] := Populacao[Indice]^;
        end;
    For Aux := 1 to TamanhoPopulacao do
        begin
            Populacao[Aux] := PopulacaoFilhos[Aux];
            Dispose(PopulacaoFilhos[Aux]);
        end;
    end;

```

Quadro 14: Procedimento de seleção.

Operadores genéticos: tendo selecionado os indivíduos que formarão a próxima geração são aplicados os operadores genéticos sobre esta nova população. O primeiro operador a ser aplicado é o operador de cruzamento, depois é a vez do novo operador, denominado de “mutaçãoSA”, em seguida são aplicados os operadores de mutação, inversão e elitismo. Este procedimento pode ser visto no quadro 15.

```
Procedure OperadoresGeneticos;  
begin  
  CruzaFilhos;  
  MutacaoSA;  
  Mutacao;  
  Inversao;  
  Elitismo;  
end;
```

Quadro 15: Procedimento que aplica os operadores genéticos.

Operador de mutacaoSA: este é o operador adicionado no processo do algoritmo genético. Quando um cromossomo é selecionado a sofrer o efeito do operador de mutacaoSA, ele é passado para a procedure que executa o algoritmo *simulated annealing*, utilizando como solução inicial o cromossomo passado como parâmetro e, em seguida, o resultado da execução é novamente inserido na população do algoritmo genético no lugar do cromossomo selecionado. O funcionamento deste operador pode ser visto no quadro 16.

```
procedure MutacaoSA (n : Longint; var t : TipoTour; Tmax, alfa :  
                    Double ; passos, tentativas, changes : Longint );  
  
var  
    temp : double; k : integer;  
  
begin  
    temp := Tmax; MaxTemperatura := Tmax; K := 0;  
    Repeat  
        Inc(K);  
        busca ( n, t, temp, tentativas, changes );  
        temp := (temp * alfa) ;  
    Until ( Temp <= Tmin) or (KeyPressed);  
end;
```

Quadro 16: Procedimento que aplica os operadores genéticos.

CAPÍTULO 6

Análise dos resultados

Este capítulo aborda, inicialmente, a metodologia de avaliação do método proposto, assim como as formas de se obter os resultados. Também faz referência às origens das instâncias e procedimentos de como construir a população inicial. Por último, os resultados computacionais obtidos com a aplicação do método proposto são comparados com resultados obtidos pelo algoritmo genético original.

6.1 Metodologia utilizada

Neste trabalho, o método proposto é avaliado através de um protótipo baseado no algoritmo genético original, onde, além dos operadores normais, o protótipo utiliza um novo operador baseado no algoritmo *simulated annealing*. Esta avaliação é realizada comparando o desempenho do algoritmo em relação ao algoritmo genético original, aplicando os dois algoritmos às mesmas instâncias de problemas.

As instâncias utilizadas para testes estão divididas em duas categorias. Na primeira categoria, serão utilizados *benchmarks* com dados obtidos de situações reais. A segunda categoria utilizada pertence às instâncias em grade e tem como principal característica a possibilidade de se calcular o custo ótimo, independente da quantidade de cidades envolvidas.

Os dados, para análise dos resultados, serão obtidos através da execução de ambos os algoritmos, 100 vezes para cada instância do problema que se pretende testar, escolhendo-se a melhor solução encontrada.

6.1.1 Origem das instâncias

As diversas instâncias do problema do caixeiro viajante que são encontradas têm suas origens baseadas nas mais diversas situações, que tanto podem representar situações do mundo real como imaginárias. Dentre estes problemas, os que mais têm causado interesse, principalmente em função de sua aplicabilidade, em situações práticas, é o que trata de instâncias euclidianas, ou seja, as cidades correspondem a pontos no plano e as distâncias são calculadas na métrica euclidiana ARAUJO (2001).

As instâncias utilizadas para testes neste trabalho, foram retiradas de duas fontes. A primeira pode ser encontrada em REINELT (2002) e a segunda categoria pertence às instâncias em grade.

A primeira fonte diz respeito a uma biblioteca de amostras de instâncias para o problema do caixeiro viajante, e problemas relacionados, onde podem ser encontrados os mais recentes resultados obtidos em pesquisas sobre o assunto, contendo várias instâncias com os mais variados números de cidades (variando entre 14 e 85.900 cidades).

A segunda fonte pertence às instâncias em grade. Este tipo de instância tem uma importante característica que motivou sua utilização, elas possibilitam a obtenção do custo ótimo. Nas instâncias em grades, as cidades são colocadas nos vértices de uma grade regular quadrada no espaço euclidiano HANSEN (1995).

A figura 19 mostra uma grade com 16 cidades. A distância entre duas cidades é determinada pelo comprimento do lado do quadrado ou pela distância da diagonal.

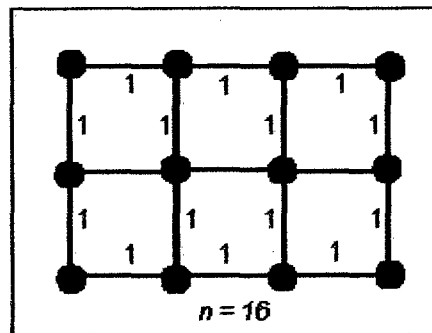


Figura 19. Exemplo de instância em grade 4X4.

Segundo ARAUJO (2001), com a utilização de instâncias em grade, a obtenção do custo ótimo se torna bastante simples, bastando calcular a soma das distâncias. O custo ótimo é obtido da seguinte forma:

$$f_{opt} = gn \quad (6.1)$$

No caso onde o número de cidades for par e:

$$f_{opt} = g(n-1+\sqrt{2}) \quad (6.2)$$

Quando o número de cidades for ímpar.

6.1.2 Avaliação

Será avaliada a melhor execução, dentre todas as execuções, para uma determinada instância. A qualidade da solução obtida será medida em termos de porcentagem em relação à melhor solução conhecida. ARAUJO (2001) sugere o cálculo da qualidade baseado na fórmula:

$$Q = ((f(.) - f_{opt}) / f_{opt}) \cdot 100 \quad (6.3)$$

Onde $f(.)$ é o custo encontrado e f_{opt} o custo da melhor solução conhecida para aquela instância.

6.1.3 População inicial

Várias técnicas de geração da população inicial podem ser utilizadas visando melhorar o desempenho do algoritmo genético. Como, neste trabalho, o objetivo é avaliar o desempenho do método proposto em relação ao algoritmo genético original, não se deu muita importância às técnicas de geração da população inicial e o método de geração da população inicial utilizado foi o método randômico. Um cromossomo (rota) inicial é gerado através de uma escolha aleatória da seqüência de cidades que deverão ser visitadas pelo caixeiro viajante. Inicialmente, a rota está vazia, então são escolhidas aleatoriamente uma a uma as cidades que farão parte na seqüência da rota, até que todas as cidades estejam presentes na rota. Este processo é repetido tantas vezes quanto for o tamanho da população.

6.2 Ajuste dos parâmetros

Como o objetivo deste capítulo é demonstrar a qualidade do método proposto através de sua aplicação em instâncias do problema do caixeiro viajante, uma importante questão a ser solucionada são os ajustes dos parâmetros do protótipo, assim como a escolha dos operadores mais adequados com suas probabilidades associadas.

Visando determinar estes parâmetros, o que se fez foi testar o protótipo em diversas instâncias do problema do caixeiro viajante (instâncias da internet e instâncias em grade) e os parâmetros, assim como os operadores e suas respectivas probabilidades, foram ajustados, na medida do possível, para que se comportem bem de uma forma geral (buscou-se encontrar valores que se comportassem bem de uma forma geral e não valores que obtivessem resultados de ótima qualidade para instâncias específicas).

6.2.1 Ajuste dos parâmetros relacionados ao algoritmo genético

Com relação ao algoritmo genético foram realizados testes referentes aos seguintes parâmetros: a condição de término; quais os operadores de cruzamento, mutação e inversão mais adequados (incluindo suas probabilidades); a quantidade de cromossomos que serão preservados através do processo de elitismo e o tamanho da população.

Condição de término: com relação à condição de término, testou-se duas alternativas, na primeira, o programa termina quando uma determinada quantidade de gerações for avaliada e na Segunda, o programa termina quando nenhuma melhora for encontrada no processo em um determinado número de gerações consecutivas.

Os testes realizados com a segunda alternativa (número consecutivo de gerações) não trouxe resultados muito animadores, além de criar um novo problema referente à quantidade de gerações consecutivas que se deve esperar antes de terminar o processo (existe uma variação muito grande entre as diferentes instâncias). Sendo assim, optou-se pela utilização da primeira condição de término mencionada (quantidade fixa de gerações) e com relação à quantidade de gerações, os testes realizados determinarão que o algoritmo consegue chegar a bons resultados nas primeiras 3.000 gerações, e a partir daí as melhorias encontradas são insignificantes.

Operador de cruzamento: apesar do operador de cruzamento ser considerado por muitos como o operador mais importante no processo do algoritmo genético, os operadores de cruzamento testados neste trabalho não apresentaram resultados muito animadores, mesmo assim a utilização do mesmo foi mantida. Os operadores testados foram os mencionados no capítulo 3 (CX, PMX e OX) e o operador que obteve melhor resultado foi o operador CX. Com relação à probabilidade de ocorrência de cruzamento, os melhores resultados foram obtidos com a taxa de 25 %.

Operador de mutação: com relação ao operador de mutação, os testes também foram realizados com os operadores mencionados no capítulo 3 (mutação baseada na posição, mutação baseada na ordem e mutação mista) e o operador que obteve o melhor

resultado foi a mutação baseada na ordem. Testes foram realizados com diferentes taxas e a taxa que obteve o melhor resultado foi a taxa de 0.5 %.

Operador de mutação SA: o operador de mutação SA é o operador adicionado no processo do algoritmo genético, baseado no algoritmo *simulated annealing*. Testes realizados com este operador revelaram que os melhores resultados foram obtidos quando o operador foi aplicado a taxas muito pequenas, sendo fixada em 0.5 %.

Operador de inversão: o operador de inversão utilizado é o mesmo que o apresentado neste trabalho e a taxa de ocorrência foi fixada em 5 %.

Elitismo: o operador de elitismo com a preservação de 5 indivíduos apresentou resultados de boa qualidade, sendo que, à medida que se aumenta este valor, a qualidade dos resultados diminui.

Tamanho da População: como mencionado anteriormente, na teoria, quanto maior o tamanho da população melhor a solução encontrada, uma vez que, em última análise, se o tamanho da população fosse igual ao tamanho do espaço de busca a solução ótima seria encontrada na primeira geração. Na prática, isso não é possível e deve-se lidar com populações menores. Nos testes realizados, uma população de 100 cromossomos conseguiu obter resultados dentro do esperado.

6.2.2 Ajuste dos parâmetros relacionados ao algoritmo *simulated annealing*

O protótipo no que diz respeito ao algoritmo *simulated annealing* foi baseado na implementação proposta por HANSEN (1995). Como o algoritmo é utilizado no protótipo como sendo um operador genético, resolveu-se fazer algumas alterações visando agilizar o processo (mesmo sobre o risco de obter soluções de pouca qualidade). As alterações realizadas dizem respeito à temperatura final e ao tamanho da busca.

Temperatura final: no método proposto por HANSEN (1995), o processo termina quando a temperatura for alterada um determinado número de vezes. No protótipo implementado, o processo termina quando a temperatura atingir um valor mínimo.

Tamanho da busca: no protótipo implementado, o tamanho da busca foi reduzido visando agilizar o processo. Enquanto que no método original a quantidade de tentativas e trocas era fixada em $100 * n$ e $10 * n$ respectivamente, com n sendo a quantidade de cidades. No protótipo implementado, as tentativas e trocas foram fixadas em $10 * n$ e n .

6.3 Resultados obtidos

Tendo sido ajustados os parâmetros, a qualidade do método proposto foi avaliada em instâncias do problema do caixeiro viajante, e os resultados obtidos foram comparados com o algoritmo genético original.

Os resultados estão demonstrados através de gráficos de linha, que comparam os resultados obtidos no decorrer do tempo (gerações). A escolha deste método se deu pelo fato de que, com sua utilização, torna-se mais simples verificar a busca melhor e mais agressiva que o método proposto realiza.

Uma importante característica a ser mencionada é o fato de que os gráficos demonstram os resultados obtidos em um intervalo de 100 gerações e como o algoritmo preserva o melhor indivíduo durante o processo, as informações contidas nos gráfico dizem respeito ao melhor resultado obtido até o intervalo em questão.

A primeira classe de instâncias testadas foi a dos problemas conhecidos (internet), sendo que os testes foram realizados com 4 instâncias crescentes em número de cidades. A primeira com 76, a segunda com 152, a terceira com 299 e a última com 1004 cidades. Os resultados obtidos podem ser vistos nas figuras 20 e 21.

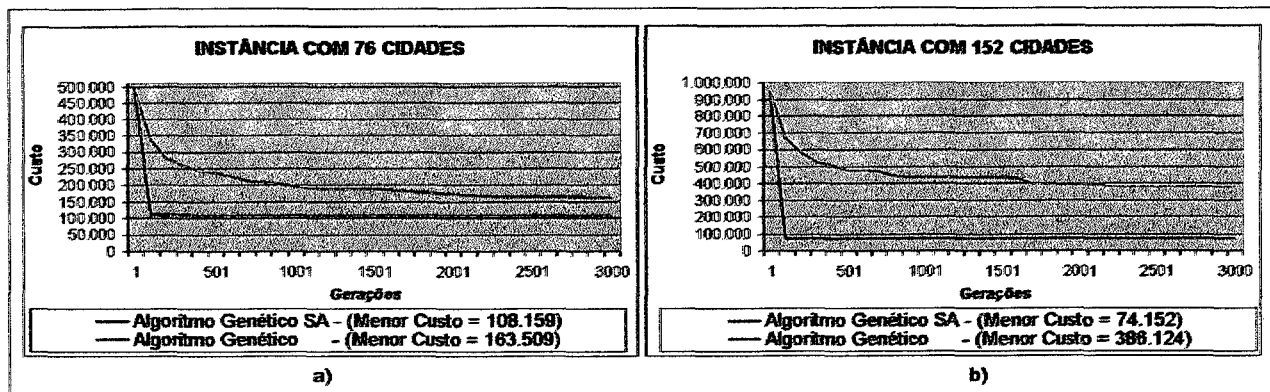


Figura 20: Resultados obtidos com as instâncias de 76 e 152 cidades.

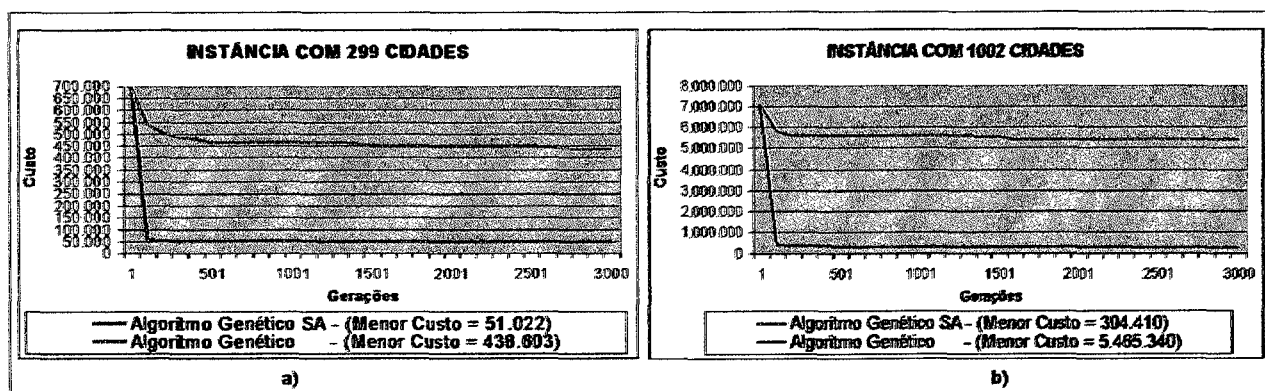


Figura 21: Resultados obtidos com as instâncias de 299 e 1004 cidades.

Baseado nos resultados obtidos pelo algoritmo proposto quando aplicado a instâncias obtidas na internet, algumas conclusões podem ser tiradas:

- os resultados obtidos pelo algoritmo genético SA foram surpreendentemente melhores quando comparados ao algoritmo genético original. Os dados da tabela 1 mostram claramente que algoritmo genético SA obteve resultados muito melhores para todas as instâncias utilizadas;
- o método proposto se comportou dentro do esperado, introduzindo uma busca mais agressiva no processo do algoritmo genético. Os gráficos mostram que o algoritmo genético SA obteve bons resultados em umas poucas dezenas de gerações, enquanto que o algoritmo

genético original somente obteve melhores resultados depois de algumas centenas de gerações;

- o método proposto obteve resultados muito próximos aos resultados ótimos para os 4 testes, inclusive chegando ao resultado ótimo para a instância com 76 cidades como mostra a tabela 1, enquanto que o algoritmo genético original não conseguiu se aproximar dos melhores resultados;
- com base na tabela 1, uma conclusão importante que pode ser tirada é que a diferença do algoritmo genético SA em relação ao algoritmo genético original se torna mais acentuada à medida que se aumenta a quantidade de cidades para a instância testada (apesar de não ser um fato conclusivo). Para os testes com a instância de 76 cidades, enquanto que o algoritmo genético original ficou 51,17 % acima da solução ótima, o algoritmo genético SA encontrou a solução ótima, somando uma diferença de 51,17 % entre os dois algoritmos. Esta diferença cresce à medida que a quantidade de cidades aumenta, chegando a uma diferença de 1.992,29 % nos testes realizados com a instância de 1002 cidades;
- o algoritmo genético SA obtém bons resultados em umas poucas dezenas de gerações, ficando preso a este resultado com poucas alterações no decorrer do processo. Analisando os gráficos, fica claro que o algoritmo genético SA obtém os melhores resultados nas primeiras 200 gerações, sendo que a partir daí a melhoria diminui, e
- a utilização do algoritmo genético SA resulta em um aumento considerável no tempo de processamento, como mostra a tabela 1, uma vez que o método proposto utiliza um novo operador, baseado no algoritmo *simulated annealing*.

Instância	Custo ótimo	Algoritmo Genético Original			Algoritmo Genético SA			% Diferença
		Menor Custo	% Diferença	Tempo (mm:ss)	Menor Custo	% Diferença	Tempo (mm:ss)	
76 cidades	108.159	163.509	51,17	1:20	108.159	-	6:46	51,17
152 cidades	73.682	386.124	424,04	2:20	74.152	0,64	10:45	423,40
299 cidades	48.191	438.603	810,13	4:20	51.022	5,87	27:27	804,26
1002 cidades	259.045	5.465.340	2.009,80	14:03	304.410	17,51	122:07	1.992,29

Tabela 1: Resultados obtidos com instâncias da internet.

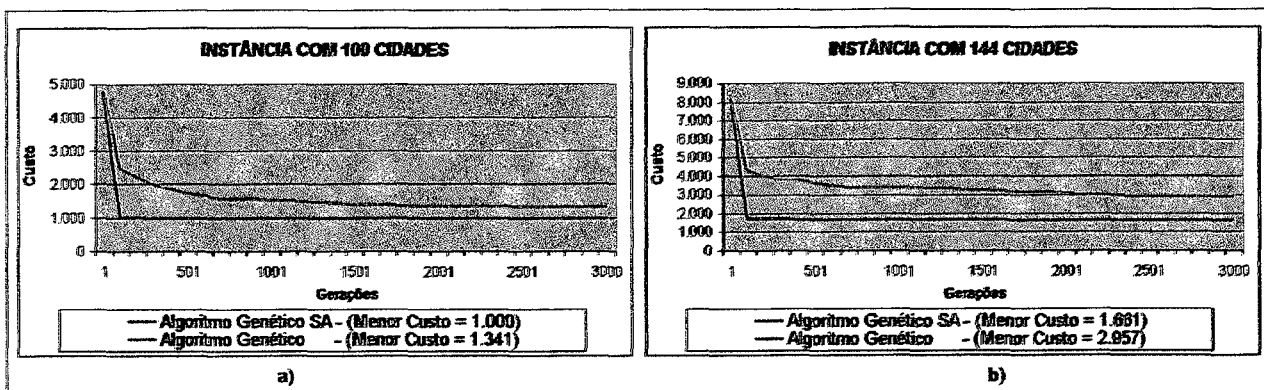


Figura 22: Resultados obtidos com as instâncias de 100 e 144 cidades.

A segunda classe de instâncias analisadas foi a dos problemas em grade, sendo que os testes foram realizados também com 4 instâncias crescentes em número de cidades. A primeira com 100, a segunda com 144, a terceira com 324 e a última com 1024 cidades. Os resultados obtidos podem ser vistos nas figuras 22 e 23.

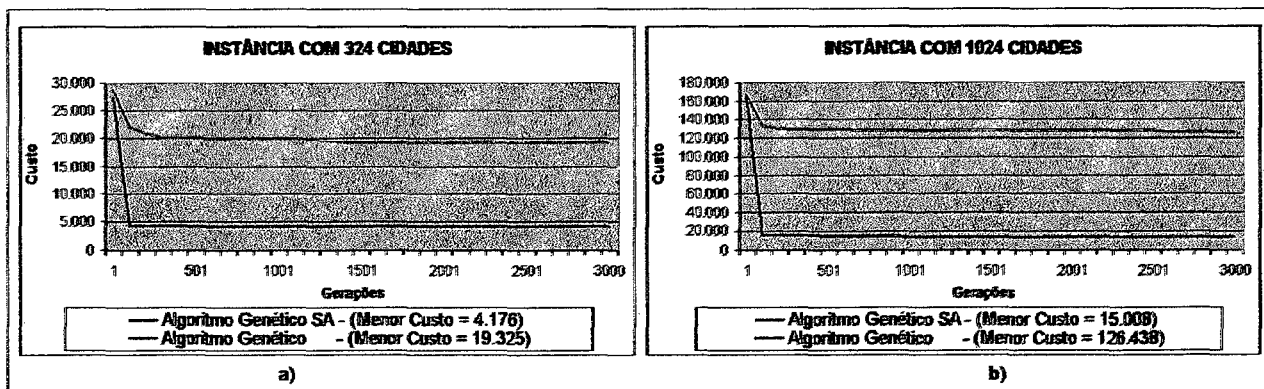


Figura 23: Resultados obtidos com as instâncias de 324 e 1024 cidades.

Os resultados obtidos com os testes realizados nas instâncias em grade se comportaram de maneira semelhante aos testes com as instâncias obtidas da internet. Novamente, o algoritmo genético SA superou as expectativas, obtendo resultados de excelente qualidade, sendo que nas instâncias menores o algoritmo chegou a resultados muito próximos aos resultados ótimos (chegando ao resultado ótimo nos testes com a instância de 100 cidades) e à medida que se aumenta a quantidade de cidades o algoritmo se afasta da solução ótima, como pode ser visto na tabela 2.

Instância	Custo ótimo	Algoritmo Genético Original			Algoritmo Genético SA			% Diferença
		Menor Custo	% Diferença	Tempo (mm:ss)	Menor Custo	% Diferença	Tempo (mm:ss)	
100 cidades	1.000	1.341	34,10	01:21	1.000	-	07:00	34,10
144 cidades	1.440	2.957	105,35	02:19	1.661	15,35	10:40	90,00
324 cidades	3.240	19.325	496,45	04:55	4.176	28,89	28:01	467,56
1024 cidades	10.240	126.438	1.134,75	16:15	15.008	46,56	125:25	1.088,18

Tabela 2: Resultados obtidos com as instâncias em grade.

6.4 Comentários finais

Finalizando os testes com o método proposto, restam ainda algumas considerações a serem tecidas:

- as análises realizadas neste trabalho não se preocuparam muito com o tempo de processamento, apesar de ser uma importante característica quando se trata de algoritmos de aproximação. Um dos motivos para se desconsiderar o tempo de processamento foi a dificuldade de se encontrar *benchmarks* que possibilitassem uma melhor análise em relação ao tempo de processamento. Na maior parte dos resultados encontrados, como as instâncias da internet, apenas os valores ótimos estão expostos, sendo que são desconsiderados os métodos de otimização utilizados, o tempo de processamento para se obter os resultados, assim como as máquinas utilizadas;
- foi observado que o método proposto, embora tenha se comportado como o esperado (tomando a busca mais agressiva e mostrando bom desempenho quando comparado ao algoritmo genético original), comporta-se muito bem no início do processo e, em seguida, o processo fica estagnado (justamente um dos problemas que se tenta evitar no processo de um algoritmo genético), e
- o método proposto obteve soluções ótimas quando aplicado a instâncias menores (até 100 cidades) e a qualidade das soluções diminui proporcionalmente ao aumento do número de cidades do problema.

CAPÍTULO 7

Considerações finais

Este capítulo tem como objetivo ressaltar a importância do método proposto e sua contribuição no que diz respeito à utilização de técnicas para melhorar a performance de métodos de otimização que utilizam como base os algoritmos: genético e *simulated annealing*. O capítulo termina com algumas sugestões para futuras pesquisas, ou mesmo possíveis continuações deste trabalho.

7.1 Conclusões

Este trabalho investiga a potencialidade de um novo método, baseado em abordagem híbrida, com a utilização dos algoritmos: genéticos e *simulated annealing* para solução de problemas de otimização, em especial o conhecido problema do caixeiro viajante. A principal característica do método proposto foi a adição de um operador genético novo no processo do algoritmo, este operador é baseado no algoritmo *simulated annealing*.

O problema do caixeiro viajante é um dos mais importantes problemas de otimização e tem sua importância atual baseada em sua grande aplicação prática; na enorme relação do problema com outros métodos; e na grande dificuldade de se achar uma solução exata para instâncias maiores, visto que consiste em um dos problemas de otimização mais difíceis, dentre os conhecidos GOLDGARB (2000).

Atualmente, observa-se uma forte tendência em se utilizarem métodos aproximados na resolução desta classe de problemas. Uma abordagem dessa natureza, apesar de não

garantir uma solução ótima para o problema, é capaz de oferecer uma solução de boa qualidade, em tempo de processamento aceitável, o que é perfeitamente viável em muitas situações reais.

Para demonstrar a viabilidade do novo método, foram realizados testes utilizando várias instâncias do problema do caixeiro viajante (instâncias da internet e instâncias em grade). Os testes realizados mostraram que o método proposto pode ser considerado como uma boa alternativa a ser empregada na solução de problemas de otimização baseados nos algoritmos: genéticos e *simulated annealing*, superando de forma surpreendente a implementação original do algoritmo genético.

Os resultados obtidos, embora ainda não conclusivos, produzem uma projeção muito otimista em relação ao potencial do método proposto, já que estes são os primeiros testes realizados e ainda sem nenhum estudo de controle de parâmetros, estudos estes que poderão alcançar uma melhora ainda maior.

7.2 Proposta para novas pesquisas

Como mencionado anteriormente, o método proposto por este trabalho não teve como preocupação considerar o tempo de processamento do algoritmo como critério de qualidade. Outro problema observado pelo novo método, como já aludido, é o fato de que apesar de apresentar resultados dentro do esperado, o método converge em umas poucas dezenas de gerações e não consegue mais sair deste estado. Um melhor estudo nos operadores genéticos, assim como no processo de seleção podem produzir melhores resultados. Baseado nos fatos mencionados, podem ser propostas novas pesquisas nas seguintes áreas:

Agilização do algoritmo: o método estudado incluiu um novo operador baseado no algoritmo *simulated annealing*, o fato de se utilizar este operador introduziu um aumento considerável no tempo de processamento. Visto que uma importante qualidade a ser considerada em um método de otimização é o tempo de processamento, estudos poderiam

ser realizados com a intenção de tornar o algoritmo mais ágil, talvez através da utilização de técnicas paralelas;

Arquiteturas distribuídas: ainda considerando o tempo de processamento do algoritmo, outra alternativa para melhorar seu desempenho seria através de sua utilização em arquiteturas distribuídas, o que poderia trazer contribuições muito interessantes, uma vez que o algoritmo genético possui a propriedade de paralelismo implícito, como mencionado no capítulo 3.

Estudo de novos operadores: observou-se que, durante o processo do algoritmo genético SA, os melhores resultados são alcançados em umas poucas dezenas de gerações e assim permanecem durante o restante da execução. Assim, novos operadores poderiam ser estudados com a intenção de evitar a estagnação do algoritmo;

Hibridização: o método estudado provou que técnicas de hibridização podem ser bastante úteis e de grande ajuda na otimização de problemas. Neste trabalho, foram utilizados dois métodos, os algoritmos: genético e *simulated annealing* para se obter bons resultados quando aplicados ao problema do caixeiro viajante. Assim, o estudo de novas técnicas de hibridização pode trazer contribuições importantes no que diz respeito a técnicas de otimização.

Referências Bibliográficas

AARTS, Emile; KORST, Jan. **Simulated annealing and boltzmann machines: a stochastic approach to combinatorial optimization and neural computing**. Grã-Bretanha: Courier, 1989.

ARAUJO, Haroldo Alexandre. **Algoritmo simulated annealing: uma nova abordagem**. 2001. Dissertação (Mestrado em Ciências da Computação) – Curso de Pós-Graduação em Ciências da Computação, Universidade Federal de Santa Catarina, Florianópolis.

BARROS, Neto J. F.. **Análise de desempenho dos operadores genéticos aplicados ao problema do caixeiro viajante**. Disponível em: <<http://po1.pep.uftj.br/~jfbarros/agcap#.html>>. Acesso em: 19 dezembro 2001.

BENDER, Edward A. **Mathematical methods in artificial intelligence**. Washington: IEEE Computer Society, 1987.

BRASSARD, Gilles; BRATLEY, Paul. **Fundamentals of algorithmics**. New Jersey: Prentice Hall, 1996.

BUCKLES, Bill P.; PETRY, Frederick E.. **Genetic algorithms**. Washington: KNI, 1992.

CARVALHO, André Ponce de Leon F. **Algoritmos genéticos**. Disponível em: <<http://www.icmsc.sc.usp.br/~prico/gene1.html>>. Acesso em: 12 fevereiro 2002.

CELKO, Joe. **Genetic algorithms and database indexing: finding the best set of indexes**, Dr Dobb's Journal, p.30 – 34, Abril 1993.

CERNY, V. **Thermodynamical approach to the traveling salesman problem : an efficient simulation algorithm**, Journal of Optimisation Theory and Applications, n. 45, p. 41-51, 1985.

CHAMBERS, Lance. **Practical handbook of genetic algorithms applications**. Rio de Janeiro: CRC Press LLC, 1995.

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.. **Introduction to algorithms**. Cambridge: McGraw-Hill, 1990.

CURY, Ricardo Martins. **Uma abordagem difusa para o problema de flow-shop scheduling**. 1999. Tese (Doutorado em Engenharia da Produção) - Curso de Pós-Graduação em Engenharia da Produção, Universidade Federal de Santa Catarina, Florianópolis.

DARWIN, Charles. **The origin of species by means of natural selection**. Chicago: Encyclopedia Britannica, 1953.

DAVIS, Lawrence. **Genetic algorithms and simulated annealing**. Cambridge Massachusetts: Morgan Kaufman, 1987.

FREDMAN, M. L.; JOHNSON, D. S.; MCGEOCH, L. A., OSTHEIMER. **Data structures for traveling salesman**. In proceedings of the 4th annual ACM-SIAM symposium on discrete algorithms, p. 145-154, 1993.

GOLDBARG, Marcos Cesar; LUNA, Henrique Pacca L. **Otimização combinatória e programação linear: modelos e algoritmos**. Rio de Janeiro: Editora Campus, 2000.

GOLDBERG, David E. **Genetic algorithms in search, optimization, and machine learning**. Massachusetts: Addison Wesley Publisher Company, 1989.

HAJAK, B. **Cooling schedules for optimal annealing**. São Paulo: Edgard Blucher, 1988.

HANSEN, P.B. **Studies in Computational Science: Parallel Programming Paradigms**. Prentice Hall, 1995.

HOFFMEISTER, Frank; BACK, Thoas; SCHWEFEL, Hans Paul. **Applications of evolutionary algorithms**. Technical Report SYS-2/92, Agosto 1993.

HOLLAND, J. H. **Adaptation in natural and artificial systems**. Cambridge: MIT Press, 1993.

KIRKPATRICK, S.; GELATT, C. D. JR; VECCHI, M. P. **Optimization by simulated annealing**. Science 220, p. 671 – 680, 1983,.

KOZA, John R.. **Genetic programming: on the programming of computer by means of natural selection**. Cambridge: MIT Press, 1992.

KOZA, John R.; RICE, James P.. **Genetic programming II: automatic discovery of reusable programs**. Cambridge: MIT Press, 1994.

KRISHNA, K.; GANESHAN, K.; JANAKI, Ram D. **Distributed simulated annealing algorithms for job shop scheduling**. IEEE Transactions on Systems, Man, and Cybernetics, v. 25, n. 7, July 1995.

LAARHOVEN, P. J. M.; AARTS, Emile. **Simulated annealing: theory and applications**. Holanda: Kluwer Academic Publishers, 1987.

LIN, S. **Computer solutions of the traveling salesman problem**. Bell System Technical Journal, n. 44, p. 2245-2269, 1965.

LORENA, Luiz Antonio Nogueira. **Otimização combinatória em sistemas de informação assistidos por computador**. Disponível em: <http://jupiter.lac.inpe.br/~lorena/1relatorio.html>. Acesso em: 12 novembro 2001.

MAZZUCCO, José Júnior. **Uma abordagem híbrida do problema da programação da produção através dos algoritmos genéticos e Simulated Annealing**. 1999. Tese (Doutorado em Engenharia da Produção) - Curso de Pós-Graduação em Engenharia da Produção, Universidade Federal de Santa Catarina, Florianópolis.

METROPOLIS, N.; ROSENBLUTH, A.; ROSENBLUTH, M.; TELLER, A.; TELLER, E. **Equation of state calculations by fast computing machines**, Journal of Chemical Physics 21, p. 1087-1092, 1953.

MICHALEWICS, Zbigniew. **Genetic algorithms + data structures = evolution programs**. Berlin: Springer, 1999.

MIRANDA, Marcio Nunes. **Algoritmos genéticos: fundamentos e aplicações**. Disponível em: <http://www.gta.ufrj.br/~marcio/genetic.html>. Acesso em: 23 janeiro 2002.

NILSSON, Nils L.. **Artificial intelligence: a new synthesis**. San Francisco: Morgan Kaufmann Publishers, 1998.

NOVAES, Antonio Galvão. **Métodos de otimização aplicações aos transportes**. São Paulo: Edgard Blucher, 1978.

OLIVER, Jim. **Finding decision rules with genetic algorithms**. AI Expert, p. 33 – 39, Outubro 1994.

PAPADIMITRIOU, Christos H.. **Combinatorial optimization: algorithms and complexity**. Massachusetts: Prentice-Hall, 1982.

PARDALOS, Panos M.; HORST, Reiner. **Handbook of global optimization**. Países Baixos: Klurter Academic, 1995.

PIRLOT, Marc. **General local search methods**, European journal of operational research, v. 92, p 493-511.

PRICE, Kenneth V. **Genetic annealing**. Dr Dobb's Journal, p.127 – 132, Outubro 1994.

PRICE, Kenneth; STRON, Rainer. **Differential evolution**. Dr Dobb's Journal, p.18 – 24, Abril 1997.

REINELT, Gerhard. **TPSLIB**. Disponível em: <http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/>>. Acesso em: 12 fevereiro 2002.

RICIERI, Aguinaldo Pradani. **Otimização natural: a genética dos máximos e mínimos**. Guarulhos: Editora Parma, 1994.

SAHNI, Sartaj; HOROWITZ, Ellis. **Fundamentals os data structures in pascal**. New York: Computer Science Press, 1994.

SPELLMAN, Richard. **Genetic algorithms**. Dr Dobb's Journal, p. 26 – 30, Fevereiro 1993.

TANOMARU, Julio. **Motivação, fundamentos e aplicações de algoritmos genéticos II**. Congresso Brasileiro de Redes Neurais, Curitiba, 1995.

VACA, Oscar Ciro López. **Um algoritmo evolutivo para a programação de projetos multi-modos com nivelamento de recursos limitados**. 1995. Tese (Doutorado em Engenharia da Produção) - Curso de Pós-Graduação em Engenharia da Produção, Universidade Federal de Santa Catarina, Florianópolis.

VARHOL, Peter D.. **Genetic programming**. Dr Dobb's Journal, p.30 – 34, Janeiro 1994.

WINSTON, Patrick Henry. **Inteligência artificial**. Rio de Janeiro: LTC –Livros Técnicos e Científicos, 1988.

YONEYAMA, Takashi; NASCIMENTO, Cairo L. Júnior. **Inteligência artificial em controle e automação**. São Paulo: Edgard Blucher LTDA, 2000.