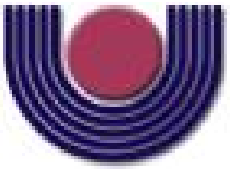


Unioeste - Universidade Estadual do Oeste do Paraná
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
Colegiado de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Um Estudo Sobre Algoritmos Meméticos e sua Eficiência em Relação aos Algoritmos Genéticos

Ana Paula Fredrich

CASCAVEL
2010

ANA PAULA FREDRICH

UM ESTUDO SOBRE ALGORITMOS MEMÉTICOS E SUA EFICIÊNCIA EM RELAÇÃO AOS ALGORITMOS GENÉTICOS

Monografia apresentada como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação, do Centro de Ciências Exatas e Tecnológicas da Universidade Estadual do Oeste do Paraná - Campus de Cascavel

Orientadora: Adriana Postal.

CASCAVEL
2010

ANA PAULA FREDRICH

UM ESTUDO SOBRE ALGORITMOS MAMÉTICOS E SUA EFICIÊNCIA EM RELAÇÃO AOS ALGORITMOS GENÉTICOS

Monografia apresentada como requisito parcial para obtenção do Título de Bacharel em Ciência da Computação, pela Universidade Estadual do Oeste do Paraná, Campus de Cascavel, aprovada pela Comissão formada pelos professores:

Prof. Adriana Postal(Orientadora)
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Josué Pereira de Castro
Colegiado de Ciência da Computação,
UNIOESTE

Prof. Suzan Kelly Borges Piovesan
Colegiado de Engenharia com Ênfase em
Controle e Automação, FAG
Colegiado de Sistema de Informação, UNIPAR

Cascavel, 29 de novembro de 2010

DEDICATÓRIA

à Deus,
ao meu Marido,
a minha Família.

AGRADECIMENTOS

Agradeço inicialmente a Deus por ter me dado forças e capacidade para chegar até aqui, caminhada longa, mas cheia de amizades, conhecimentos novos, alegrias...

Quero agradecer, especialmente, ao meu marido Cezar pelos finais de semanas que tive estudar, fazer trabalhos, não podendo lhe dar a atenção que merecia, também pelos *stress's* dos dias tumultuados. Mas principalmente pelo apoio, carinho e compreensão que fez com que eu não desanima-se nesta caminha de cinco anos. Muito obrigada, Amo Você...

A minha família, principalmente a minha mãe Marly, que também sempre estiveram presentes, ao longo desses anos, me dando forças e mostrando que o melhor caminho era continuar nesta jornada, sem eles não conseguiria chegar até aqui.

A todos os meus amigos e amigas, formandos e não formandos, que sempre estiveram do meu lado, dando força nas horas difíceis, fazendo trabalhos, estudando para as provas, ou mesmo trocando idéias de planos futuros. Desejo a todos muita sorte, que possamos ter uma carreira de sucesso, mas que principalmente sejamos felizes. Obrigado por tudo colegas e Tudo de Bom...

A todos os funcionários: Nelson, Carin, Andri... Muito obrigado, além de sempre nos ajudarem no que lhes cabia, são queridos amigos...

Por fim, agradeço aos professores do curso, que me proporcionaram adquirir o conhecimento conquistado até agora e também pela amizade. Um agradecimento especial para a professora Adriana, que além de acompanhar a nossa turma durante os cinco anos, ser responsável pelo Projeto de Pesquisa em Algoritmos ao qual fiz parte, ainda aceitou ser minha orientadora neste trabalho, Muito Obrigada...

Desculpe se esqueci de alguém. Que Deus ilumine os caminhos de todos...

Todos foram muito importantes nesta conquista...

Muito Obrigada !!!

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

AG	Algoritmos Genéticos
AM	Algoritmos Meméticos
DM	Displacement Mutation
PCV	Problema do Caixeiro Viajante
SEC	Strategic Edge Crossover
VMP	Vizinho Mais Próximo

Lista de Símbolos

P	um problema computacional de otimização
x	uma instância de P
$sol(x)$	conjunto de soluções factíveis para o problema P
$res(x)$	conjunto de respostas dado por um algoritmo que tenta resolver o problema P
$S(x)$	conjunto espaço de busca
s	elemento do conjunto $S(x)$
g	função de crescimento
N	função da relação de vizinhança
$N(x, s)$	conjunto vizinhança de s
s'	elemento vizinho de s
Fg	função guia
F	conjunto cujo elementos são valores de "*fitness*"
s_0	elemento 0 do $S(x)$
$Spais$	conjunto de "pais"
$Sdesc$	conjunto de "descendentes"

Sumário

Resumo

Este trabalho apresenta um estudo sobre os Algoritmos Meméticos (AMs), que são uma variação dos já consolidados Algoritmos Genéticos (AGs). O trabalho apresenta inicialmente os conceitos básicos de AGs, passando aos conceitos de AMs de forma mais aprofundada. Por fim, realiza uma comparação entre estas duas técnicas e a implementação tradicional para a resolução do Problema do Caixeiro Viajante (PCV), bem como os testes e as análises realizadas sobre a eficiência das mesmas.

Palavras-chave: Algoritmos Genéticos, Algoritmos Meméticos, Vizinho mais Próximo, Problema do Caixeiro Viajante.

Capítulo 1

Introdução

Os consolidados Algoritmos Genéticos (AGs) são utilizados em diversas áreas, tanto das ciências, como das engenharias. Foram inventados em 1960 por John Holland e apresentados a comunidade em 1975, através do lançamento de seu livro *"Adaptation in Natural and Artificial Systems"* [1].

No final da década de 1980, segundo Moscato e Cotta [2], depois de aproximadamente vinte anos da criação dos AGs, os Algoritmos Meméticos (AMs) foram introduzidos, apesar de alguns outros trabalhos nos anos anteriores já possuírem características semelhantes aos meméticos, mas que foram classificados como AGs *Híbridos*. O termo "*Algoritmo Memético*", foi realmente introduzido no ano de 1989 por Moscato, em seu trabalho: "*On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts*" [3], baseado no conceito de *meme*.

Moscato [3] comenta que a comunidade de AGs gosta de afirmar que os AMs são uma variação dos genéticos com um *hill-climbing*[1], e que Goldberg [4] em seu livro "*Genetic Algorithms in Search, Optimization e Machine Learning*", chamou algumas variações dos AGs similares aos meméticos, de "Algoritmos Genéticos Híbridos".

O presente trabalho tem por objetivo entender e descrever estes dois algoritmos, os AGs e os AMs, e compará-los através da implementação de um problema utilizando estas técnicas e também a implementação clássica do mesmo.

No capítulo 2 apresentamos uma revisão teórica sobre os AGs: o surgimento, partes que o compõem e algumas de suas características. O capítulo 3 apresenta conceitos sobre os AMs: quando e como surgiu, algumas características, partes que o compõem e o projeto de um AM. No capítulo 4 descrevemos os métodos, estratégias e operadores utilizados nas implementações.

[1] Subindo a encosta.

A descrição dos testes e análises das implementações estão no capítulo 5, e por fim, no capítulo 6 encontram-se algumas considerações finais sobre o trabalho como um todo e sugestões de trabalhos futuros.

Capítulo 2

Algoritmos Genéticos

Na década de 1960 John Holland inventou os Algoritmos Genéticos (AGs) e os desenvolveu nos anos de 1960 e 1970 juntamente com seus alunos e colegas da Universidade de Michigan. E foi em 1975 através do lançamento de seu livro *"Adaptation in Natural and Artificial Systems"* [1], que Holland apresentou os AGs à comunidade [5]. Para Melanie [5], o objetivo original de Holland não era projetar algoritmos para resolver problemas específicos, mas sim estudar o fenômeno da evolução como ocorre na natureza, e desenvolver maneiras para que os mecanismos de adaptação natural pudessem ser importados para os sistemas computacionais.

Segundo Moscato [6], os algoritmos genéticos desenvolvidos inicialmente por Holland eram simples, mas conseguiam resultados satisfatórios para problemas considerados difíceis naquela época. Foi a partir de 1980 que os AGs começaram a evoluir, através da introdução de novas inovações e mecanismos cada vez mais elaborados, que tinham a intenção e necessidade de resolver, mesmo que aproximadamente, problemas práticos.

Os AGs utilizam muitas terminologias análogas às biológicas, porém suas entidades são muito mais simples do que as reais da biologia [5]. No trabalho de Kondageski ([7] *apud* [8]), é feita uma pequena apresentação e comparação entre essas terminologias em relação a biologia e a dos AGs, como podemos observar a seguir:

- **Cromossomo:** é a estrutura de dados que codifica uma solução para um problema, podendo ser uma cadeia de bits, ou um vetor com as variáveis de decisão do problema;
- **Indivíduo:** é um membro da população, formado pelo cromossomo e sua aptidão;
- **População:** é o conjunto de indivíduos de uma mesma espécie, e representa um conjunto de pontos candidatos a solução do problema;

- **Gene:** para a biologia é a entidade de hereditariedade que é transmitida pelo cromossomo e que contém as características do organismo. Analogamente para os AGs é o parâmetro codificado no cromossomo;

- **Alelo:** para a biologia representa uma das características alternativas de um gene. Para os AGs, representa os valores que um gene pode assumir;

- **Genótipo:** para a biologia representa a composição genética, ou seja, o conjunto de genes de um indivíduo. Assim, nos AGs são representados pelas informações contidas no cromossomo;

- **Fenótipo:** é o cromossomo decodificado, ou seja, é o organismo que pode ser construído a partir das informações do genótipo.

Segundo Moscato [6], todos os AGs desenvolvidos para um determinado problema devem ter os seguintes elementos em comum:

1. uma representação, em termo de cromossomo, para as soluções candidatas a resolução do problema;

2. uma maneira de se criar a população inicial;

3. uma função de avaliação, que permite ordenar ou classificar os cromossomos de acordo com a função objetivo;

4. operadores genéticos, que permitam alterar a composição dos novos cromossomos gerados pelos pais, durante a reprodução;

5. e valores para os parâmetros que os AGs usam: tamanho da população, taxa de cruzamento e taxa de mutação.

Em geral os AGs também devem seguir um conjunto de passos, indicados na Figura 2.1 adaptado de [8], onde cada ciclo completo, desde a avaliação de aptidão ("*fitness*") dos indivíduos, até a aplicação do operador de mutação, é considerado uma nova população. No decorrer do capítulo descreveremos com mais detalhes cada passo.

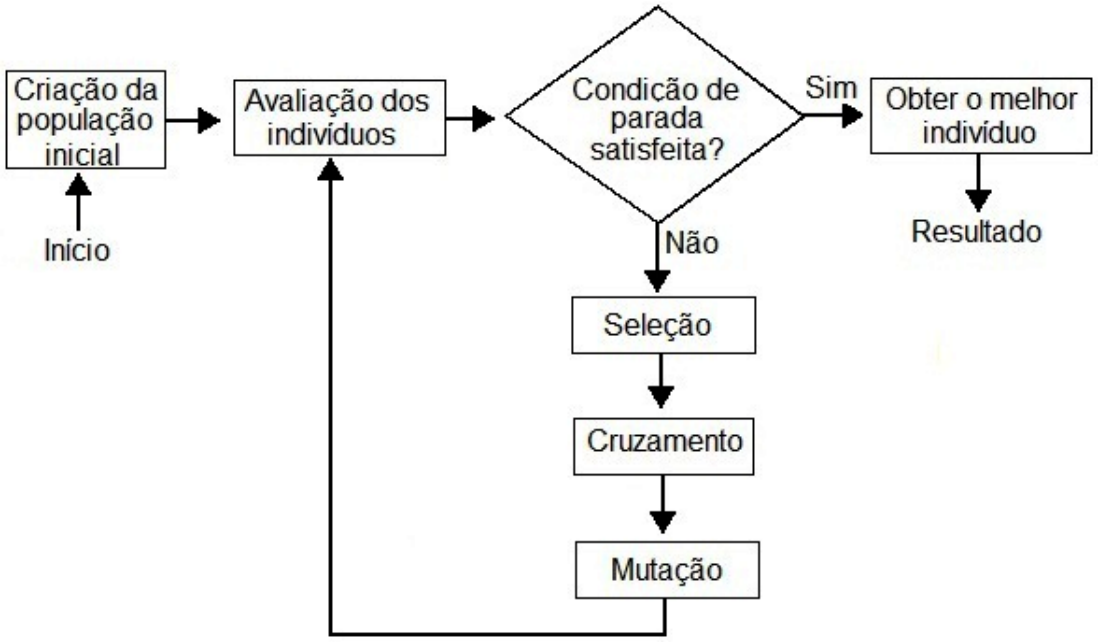


Figura 2.1: Passos da execução de um AG

2.1 Representação e Codificação

Conforme vimos anteriormente todos os AGs devem possuir uma representação para as soluções candidatas a resolução do problema, chamada de cromossomo. Sua estrutura na maioria das vezes é um vetor, porém outras estruturas também podem ser utilizadas como árvores e matrizes. As codificações, ou representações mais utilizadas são a binária e a real.

Segundo Moscato [6], os primeiros cromossomos desenvolvidos por Holland em seus AGs utilizavam a codificação binária para representar os problemas. Apesar de esta codificação ser de fácil utilização e compreensão, em alguns casos obter a representação binária a partir do problema não é um processo muito óbvio, porém para um grande número de problemas de otimização esta representação é um processo natural. Devido a sua ineficiência em alguns problemas práticos com aplicações industriais, outras codificações surgiram como, por exemplo, a codificação com caracteres ou números reais.

2.1.1 Representação Binária

Segundo Mognon [9] o código binário, que utiliza os símbolos 0 e 1 para representar as variáveis, é muito explorado para a codificação dos cromossomos. Na codificação binária os

cromossomos são compostos por strings de zeros e uns, e os parâmetros (genes) são representados por conjuntos de bits.

Porém, para Catarina [10], quando a codificação binária é utilizada em problemas com variáveis contínuas e que se espera uma boa precisão na resolução, os cromossomos se tornam longos. Consequentemente este aumento dos cromossomos resultará também em um aumento de tempo para seu processamento.

2.1.2 Representação Real

Pelos motivos apresentados na Seção 2.1.1 com relação a variáveis contínuas e resultados precisos, outras formas de codificação dos cromossomos foram desenvolvidas. Dentre estas uma das mais utilizadas é a codificação real.

Para Catarina [10] *"essa forma de codificação consiste em representar, em um gene ou cromossomo, uma variável numérica contínua através de seu próprio valor real"*. Assim, esta codificação faz com que os métodos de troca de informações genéticas, como cruzamento e mutação se tornem mais complexos, em consequência os operadores genéticos precisam de implementação específica [9].

2.2 Criação da População Inicial

Após a definição da codificação do cromossomo, o próximo passo é escolher como a população inicial será gerada, sendo sua criação realizada no primeiro passo da execução de um AG.

Segundo Moscato [6] a população inicial de um AG pode ser criada de maneiras diferentes, entre as quais podemos citar:

- **inicialização de forma aleatória:** como o próprio nome já sugere, a população é gerada de forma aleatória, sem seguir nenhum padrão ou algoritmo. Pode ser considerado um método bom para testar o funcionamento de um AG, já que as soluções obtidas são puramente consequência da evolução dessa população aleatória, e não possuem interferência das características dos métodos utilizados para gerar a população inicial;

- **inicialização heurística:** neste caso a população é gerada através de alguns métodos

mais diretos como, por exemplo, o algoritmo guloso, inicialização randômica ponderada, ou mesmo uma inicialização gerada por um especialista no problema, sendo está forma muito utilizada especialmente em algumas aplicações industriais.

2.3 Função de Avaliação

A função de avaliação, mais conhecida como função *fitness*, é utilizada no segundo passo da execução de um AG, onde é realizada a avaliação dos indivíduos. Esta função é utilizada para avaliar cada indivíduo da população de uma geração, de acordo com a aptidão do indivíduo em relação ao problema que se está resolvendo [9], e a partir desta avaliação decidir se a condição de parada foi satisfeita ou não.

Mognon [9] considera que a função de avaliação é calculada a partir das informações que estão contidas no cromossomo. Esta função é elaborada levando-se em consideração as restrições do problema. Já em relação ao valor obtido pela função, quanto maior for o valor da função *fitness* do indivíduo, maior será seu grau de aptidão, tendo mais chances de se reproduzir e de sobreviver nas próximas gerações.

2.4 Operadores Genéticos

Após a realização da avaliação dos indivíduos e a não satisfação do critério de parada, inicia-se o processo de criação de uma nova geração, com a intenção de melhorar a aptidão dos indivíduos. Este processo de criação está fundamentado na aplicação dos operadores básicos de seleção, cruzamento e mutação, porém outros operadores podem ser utilizados e também variações destes considerados básicos. Nesta seção explicaremos melhor cada um destes operadores e seus principais métodos.

2.4.1 Operador de Seleção

Para que o processo de criação de uma nova geração inicie é necessário a seleção dos indivíduos que passarão para a nova geração, ou os chamados *pais* que irão gerar, através do cruzamento (reprodução), a nova geração. O operador responsável por essa classificação é o operador de seleção.

O operador de seleção assemelha-se ao processo de seleção natural, onde os indivíduos mais aptos, no caso de acordo com a função *fitness*, possuem uma probabilidade maior de serem selecionados. Contudo, a seleção não deve se basear apenas nos melhores indivíduos, pois estes podem não estar perto da solução ótima global, devendo também ser considerado os indivíduos com aptidão mais baixa, dando-lhes alguma chance de participar da reprodução [9].

Dois dos métodos de seleção mais utilizados serão brevemente apresentados, o método da roleta e o do torneio, mas existem muitos outros, como o *rank* e elitismo.

Método da Roleta

É um dos mais utilizados para a seleção dos indivíduos. Neste método, cada indivíduo recebe uma porção da roleta de acordo com sua aptidão. Assim, consequentemente, os indivíduos com uma aptidão maior receberão também uma porção maior da roleta, o que aumentará sua probabilidade de ser selecionado. Após as atribuições, a roleta é girada e a porção sorteada corresponderá ao indivíduo selecionado, podendo ser girada quantas vezes forem necessárias de acordo com o número de indivíduos que se deseja para a próxima fase de cruzamento.

Para Mognon [9], este método tem a vantagem de todos os indivíduos terem a chance de serem selecionados. Entretanto, pode sofrer o efeito de dominância se algum indivíduo possuir uma aptidão alta em relação à média dos demais. A Figura 2.2 mostra um exemplo de configuração da roleta para uma população.

Indivíduo	Fitness	Porção
11110	2,47	0,16
10101	3,56	0,23
11010	1,03	0,07
10101	6,67	0,44
11001	1,50	0,10

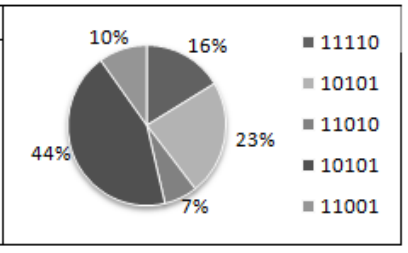


Figura 2.2: Exemplo de seleção com o método da roleta

Método de Torneio

O primeiro passo desse método é a escolha de um subconjunto de indivíduos, que podem ser escolhidos aleatoriamente ou através de outro método, como por exemplo, o método da roleta. Esse subconjunto participará então do torneio, que consiste na competição dos indivíduos entre

si considerando o seu valor de aptidão, ou seja, o vencedor será o indivíduo cujo valor de aptidão seja maior. Por fim, todos os membros do subconjunto são colocados novamente na população e o processo se repete, até que o número de indivíduos selecionados seja igual ao dos desejados [9]. A Figura 2.3, adaptada de Mognon [9], mostra um exemplo fictício do método de torneio para uma população.

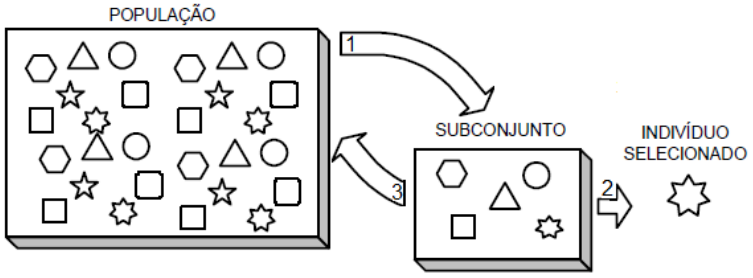


Figura 2.3: Exemplo de seleção com o método de torneio

2.4.2 Operador de Cruzamento

Após a seleção dos pais, um ou mais pares de indivíduos, o próximo passo para a geração dos novos indivíduos, os filhos, é a reprodução desses pares de indivíduos. O cruzamento é o operador responsável pela permutação do material genético dos pais durante o processo reprodutivo, permitindo assim que as próximas gerações, os filhos, herdem essas características. Este operador é aplicado seguindo-se uma taxa de cruzamento, descrita com mais detalhes na subseção 2.5.2. O operador de cruzamento pode ser implementado de diversas maneiras, dentre elas as mais conhecidas são: cruzamento de um-ponto, cruzamento multiponto e cruzamento uniforme ou pontos aleatórios.

Cruzamento de Um-ponto

É o cruzamento mais simples de ser implementado, já que o processo consiste da escolha aleatória de um ponto no cromossomo dos pais, dividindo-os em duas partes. Os filhos são gerados pela combinação destas partes dos pais, devendo possuir partes (material genético) dos dois pais. Assim, se a primeira parte do cromossomo for adquirida do primeiro pai, a segunda deverá ser adquirida do segundo. A Figura 2.4, adaptada de Mognon [9], mostra um exemplo deste cruzamento.

Figura 2.4: Exemplo de cruzamento de um-ponto

Cruzamento Multiponto

Para o cruzamento multiponto, o processo consiste na escolha aleatória de dois ou mais pontos no cromossomo dos pais, dividindo-os em várias partes. Os filhos são gerados pela combinação destas partes, devendo possuir pelo menos uma parte de cada um dos pais. A Figura 2.5, adaptada de Mognon [9], mostra um exemplo deste cruzamento com dois pontos, que divide os pais em três partes.

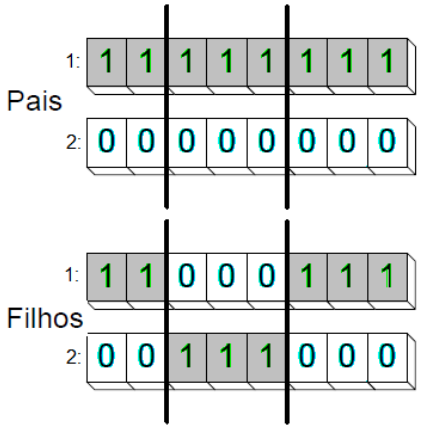


Figura 2.5: Exemplo de cruzamento de dois-ponto

Cruzamento Uniforme ou de Pontos Aleatórios

No cruzamento uniforme, o processo é um pouco diferente. Para cada gene do cromossomo do filho é sorteado entre os pais, qual deles fornecerá aquela característica para o filho, ficando

para o outro filho à característica do pai não escolhido, para que os filhos não fiquem iguais. A Figura 2.6, adaptada de Mognon [9], mostra um exemplo deste cruzamento.

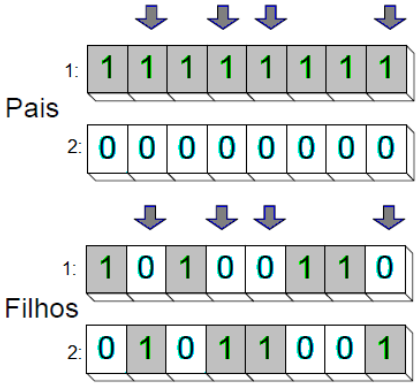


Figura 2.6: Exemplo de cruzamento uniforme

2.4.3 Operador de Mutação

Este operador é necessário para que na nova geração possam ser inseridos novos materiais genéticos, ou seja, diversificar o material genético, como ocorre na natureza através de mutações ou anormalidades dos indivíduos. Por isso, este operador é aplicado levando-se em consideração a taxa de mutação (descrita na subseção 2.5.3), que geralmente é bem pequena. É muito simples de ser implementado: cada gene é analisado, através da probabilidade de mutação, para saber se aquela posição irá alterar seu valor ou não [9]. A Figura 2.7, adaptada de Mognon [9], mostra um exemplo da aplicação deste operador, onde somente houve a alteração do valor na quarta posição.

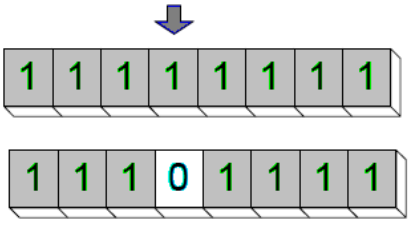


Figura 2.7: Exemplo de aplicação do operador de mutação.

2.5 Parâmetros Genéticos

Para que os algoritmos genéticos possam ser simulados se faz necessário a configuração de alguns parâmetros: tamanho da população, taxa de cruzamento e taxa de mutação. Segundo Catarina [10], são esses parâmetros que afetam o desempenho dos AGs, desde um aumento do tempo de convergência, até uma convergência prematura ou mesmo a não-convergência para uma solução aceitável. No decorrer desta seção esses parâmetros serão descritos com mais detalhes, bem como suas influências.

2.5.1 Tamanho da População

Este parâmetro determina o número de indivíduos que compõem cada população. Se a população for pequena, consequentemente a cobertura do espaço de busca do problema também será, o que poderá causar uma queda no desempenho do AGs. Assim, uma população grande resultará em uma cobertura maior do espaço de busca do problema, representando realmente o domínio do problema. Contudo, para se trabalhar com esta população maior, são necessários maiores recursos computacionas, ou/e um tempo maior de processamento do algoritmo [10]. Logo, este parâmetro deve ser ajustado de forma equilibrada, considerando os efeitos citados.

2.5.2 Taxa de Cruzamento

Representa a probabilidade com que ocorrerá o cruzamento dos indivíduos selecionados da população. Se a taxa for baixa, o algoritmo poderá se tornar muito lento, devido à inexistência de novos indivíduos. Quanto maior for esta taxa, mais rapidamente novas estruturas serão inseridas na população. Porém, se for muito alta poderá gerar efeitos indesejados, como a substituição da maior parte da população, o que poderá resultar na perda dos indivíduos de alta aptidão [10].

2.5.3 Taxa de Mutação

Indica a probabilidade de ocorrer a mutação na população. Segundo Catarina [10], se a taxa for baixa, ela previne que a solução fique estagnada em um valor, ou seja, a convergência prematura, e possibilita que se possa chegar em qualquer ponto do espaço de busca do problema. Se for uma taxa muito alta, tornará a busca essencialmente aleatória.

Capítulo 3

Algoritmos Meméticos

Os algoritmos meméticos (AMs), segundo Moscato e Cotta [2], originaram-se no final de 1980, apesar de alguns outros trabalhos nos anos anteriores já possuírem características semelhantes aos meméticos, mas que foram classificados como AGs *Híbridos*. O termo "*Algoritmo Memético*", foi realmente introduzido no ano de 1989 por Moscato, em seu trabalho: "*On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts*" [3], baseado no conceito de *meme*, detalhado na seção 3.1.

Inicialmente Moscato comparou os AMs, analogicamente, às artes marciais, mais explicitamente com o *Kung-fu* chinês. Estudos mostraram que os seres humanos tendem a lutar usando uma sequência de movimentos desordenados. Já os mestres de *Kung-fu*, usam movimentos que combinam simplicidade e efetividade. Também se sabe que as artes marciais exploram a habilidade do cérebro humano de associar eventos sequenciais. Assim, o conhecimento é repassado através de um conjunto de sequências de movimentos, denominadas **formas**. A forma, como um cromossomo, não é uma entidade indivisível, mas é composta por uma sequência de subunidades defensivas e agressivas que podem ser divididas, assemelhando-se aos genes e alelos. Porém os movimentos mais importantes são indivisíveis, e estes devem ser considerados *memes* [3].

Uma das maneiras que os indivíduos lutadores de *Kung-fu* podem utilizar para avaliar sua função *fitness* (seção 2.3), em relação a execução das formas, é através da participação em competições e torneios. Devemos observar que as informações repassadas a cada nova geração são melhoradas, e que nem todos podem ensinar, somente os que tiverem a faixa preta. Isto pode ser equiparado ao processo de cruzamento nos AGs (maiores detalhes na seção 2.4.2), onde seleciona-se os indivíduos com maior *fitness* para gerarem os novos indivíduos [3].

Os AMs são definidos por Moscato como uma tentativa de imitar a evolução cultural, enquanto os AGs são inspirados na tentativa de emular a evolução biológica. São também considerados por ele como um casamento entre uma busca global na população base e buscas locais heurísticas realizadas em cada um dos indivíduos.

3.1 *Memes*

A palavra ***meme***, que deu origem à denominação memético, foi introduzida pela primeira vez por R. Dawkins [11] para ser análoga ao gene, porém no contexto da evolução cultural. Dawkins a definiu como:

"Exemplos de memes são melodias, ideias, frases de efeito, modas de roupa, modos de fabricação de panelas ou de arcos de edifício. Da mesma maneira que genes se propagam na piscina de genes saltando de corpo em corpo por espermas ou ovos, assim memes se propagam na piscina de memes saltando de cérebro em cérebro mediante um processo que, no sentido amplo da palavra, pode ser chamado de imitação" [11].

Concilio [12] mostra que a diferença entre gene e *meme* está no processo de transmissão aos descendente. Quando o gene é transmitido, os descendentes herdam as habilidades e características dos seus progenitores. Já os *memes* são adaptados pelo indivíduo que o recebe com base no seu conhecimento e para atender suas necessidades. Assim os AGs emulam computacionalmente a evolução biológica e os AMs, fazem o mesmo em relação à evolução cultural.

Para Moscato [13], os processos dos AMs quando incorporam heurísticas, algoritmos de aproximação, técnicas de busca local, etc, estão incorporando a propriedade descrita anteriormente em relação à transmissão dos *memes*, ou seja, alterando, processando e aumentando as características meméticas.

Outra característica que diferencia os genes dos *memes* segundo Buriol [14], está relacionada com o momento da transmissão das informações: para os genes, o momento da transmissão, seria através da criação de uma nova geração, enquanto para os *memes*, a transmissão poderia acontecer sem que houvesse a necessidade da criação de uma nova geração. Afirma também que a informação cultural poderia ser transmitida de um indivíduo para vários outros, ou mesmo, para toda a população sem a necessidade da criação de uma geração.

3.2 Passos Gerais dos AMs

A partir de uma representação de um problema de otimização, um AM seguirá, em linhas gerais, os seguintes passos adaptados de Moscato [3]:

- **1º Passo:** certo número de indivíduos é criado, aleatoriamente ou escolhidos de acordo com algum procedimento de inicialização, podendo aplicar heurísticas para inicializar a população;
- **2º Passo:** cada indivíduo faz uma busca local. O objetivo da busca local pode ser o de alcançar um ótimo local ou melhorar o indivíduo (em relação à função *fitness*), até um determinado nível;
- **3º Passo:** quando o indivíduo atingiu certo desenvolvimento, ele interage com os outros membros da população. A interação pode ser:
  - **Competitiva:** ocorre entre indivíduos com base em um desafio, estando os indivíduos em locais distintos, e em determinados momentos podendo tanto desafiar como ser desafiado, estando assim, envolvidos em duas interações com seus vizinhos; a competição pode ser semelhante ao processo de seleção dos AGs (seção 2.4.1);
  - **Cooperativa:** pode ser entendida como os mecanismos de cruzamento nos AGs e recombinação para os AMs (seção 3.4), ou ainda de outros tipos de reprodução, que resultam na criação de um novo indivíduo; em geral, a cooperação serve como um intercâmbio de informações.
- **Critério de Parada:** a busca local (**passo 2**), e cooperação/competição (**passo 3**), são repetidos até que um critério de parada seja satisfeito. Normalmente este critério envolve uma medida de diversidade da população.

Logo, a ideia central dos AMs consiste em fazer melhorias individuais nas soluções, em cada um dos indivíduos, junto com processos de cooperação e competição populacional [2]. Na seção 3.5, será dada uma descrição mais detalhada dos passos em forma de algoritmo.

3.3 Operador de Busca Local

Buriol [14] afirma que, através da introdução de um ou mais operadores de busca local, a transmissão de informações meméticas é realizada. Assim a introdução de uma busca local em um AG o torna um AM. Na presença de mais de um operador de busca local nos AMs, eles podem ser executados com diferentes probabilidades, de forma a não executarem de maneira igual em todas as gerações, ou ainda serem executados igualmente em todas as gerações.

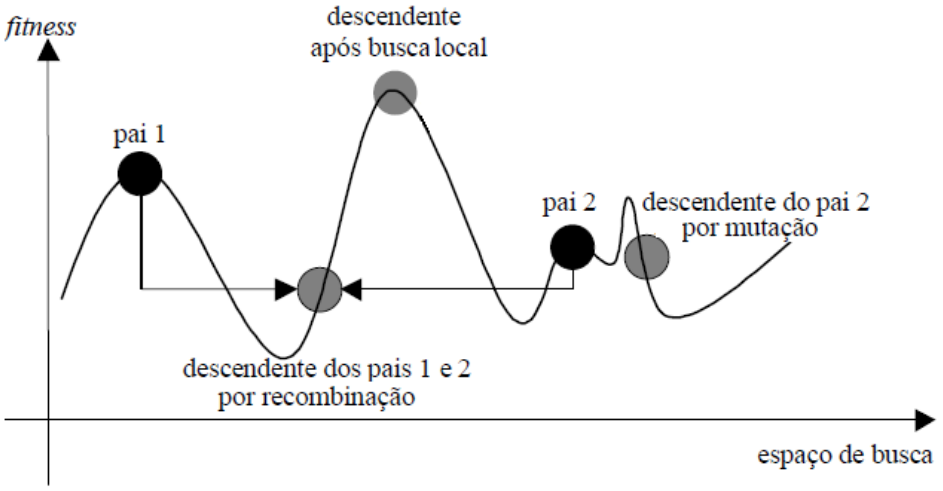


Figura 3.1: Exemplo do efeito da aplicação dos operadores de recombinação, busca local e mutação

Conforme Merz e Freisleben ([15] *apud* [12]), os indivíduos podem estar inseridos em uma região do espaço de busca contendo um ótimo local, chamada **base de atração do ótimo local**. Através da utilização da informação contida na população, novos pontos de partida podem ser descobertos após a busca local. Os operadores de mutação (seção 2.4.3) e recombinação (detalhados na seção 3.4) podem gerar indivíduos que estejam inseridos em bases de atração de ótimos locais ainda não identificados, de modo que um pico novo possa ser alcançado (maximização) ou um vale ser explorado (minimização). Este evento está ilustrado na Figura 3.1 [12], para o caso da maximização, onde podemos observar que após a recombinação, o filho gerado pode possuir um valor de *fitness* baixo, porém com um grande potencial para crescimento, de modo que uma busca local levaria o descendente a assumir um valor de *fitness* alto, já a mutação poderia levar a um pequeno aumento ou decréscimo do valor de *fitness*, pois representa

uma perturbação local junto à representação do indivíduo.

Para Moscato [13] antes de discutir o algoritmo de busca, outras três entidades devem ser definidas: o **espaço de busca**, a **relação de vizinhança**, e a **função guia**. A junção destas três entidades com uma instância de um problema gera a chamada **paisagem de aptidão**, que será explicada na seção 3.3.4.

Alguns conceitos básicos devem ser conhecidos para que possamos explicar cada uma das entidades citadas anteriormente:

- P é um problema computacional de otimização;
- x é uma instância de P, que contém um conjunto de possíveis dados de entrada, e uma sequência de tarefas algorítmicas;
- $sol(x)$ são as soluções factíveis para o problema P, dada a instância x, conhecido também como o conjunto de soluções válidas ou aceitas;
- $res(x)$ é o conjunto de respostas dado por um algoritmo que tenta resolver o problema P, dada a instância x.

3.3.1 Espaço de Busca

Segundo Moscato [13], sua identificação é feita pelo conjunto $S(x)$, onde cada elemento $s \in S(x)$ é definido como uma **configuração**, estando relacionado a uma resposta do conjunto $res(x)$, por uma função de crescimento chamada de g, assim g: $S(x) \rightarrow res(x)$. Contudo, algumas dessas configurações podem corresponder a soluções não-factíveis, já que a relação é com o conjunto de respostas e não com o conjunto de soluções, assim o algoritmo de busca deve estar preparado para lidar com esta situação.

Uma propriedade do espaço de busca em relação a problemas de otimização, consiste em haver a necessidade de pelo menos um dos elementos de $S(x)$, ser considerado um elemento ótimo do conjunto de $sol(x)$, ou seja, ser a melhor solução que maximize ou minimize a função do problema a ser otimizado. Assim o espaço de busca é considerado a base sobre a qual o algoritmo de busca agirá, ou seja, o algoritmo de busca se movimentará no conjunto imagem de $res(x)$ [13].

3.3.2 Relação de Vizinhança

Moscato [13] afirma que é através da relação de vizinhança que ocorre a relação de acessibilidade entre as configurações do espaço de busca, sendo denotada pela função N, onde N: $S(x) \rightarrow 2^{S(x)}$, definindo que para cada $s \in S(x)$, haverá um conjunto $N(x, s) \subseteq S(x)$ de configurações vizinhas de s. Assim $N(x, s)$ é chamado de **vizinhança** e cada elemento $s' \in N(x, s)$, é chamado de **vizinho** de s.

Na maioria das vezes a vizinhança é referenciada como um conjunto de possíveis **movimentos**, definidos como mudanças de alguma parte de s, que definiriam as possíveis **transições** entre as configurações do espaço de busca. A escolha de qual tipo de movimento usar, para gerar as mudanças em s, depende das características do problema e da representação escolhida, um exemplo de operador que pode ser utilizado para gerar estas mudanças, é o operador de mutação [13].

Uma característica, mostrada por Moscato [13], que deve ser atendida em relação às transições é a "*ergodicidade*", ou seja, para qualquer $s \in S(x)$ através de sucessivas transições deve-se alcançar qualquer outra configuração $s' \in S(x)$. Com isso, garante-se que pelo menos uma solução ótima poderá ser alcançada a partir de qualquer outra inicial [13].

3.3.3 Função Guia

Definida através da função Fg: $S(x) \rightarrow F$, onde os elementos do conjunto F são os valores de "*fitness*". Assim, para cada configuração $s \in S(x)$ tem-se um valor de Fg associado para avaliar a qualidade da solução [13].

O algoritmo de busca então é guiado por estes valores de aptidão. Segundo Moscato [13], no caso de problemas de otimização os valores de Fg e de "*fitness*" podem ser considerados até equivalentes, porém isso não é necessariamente obrigatório e nem mesmo desejável para alguns outros tipos de problemas, podendo F ser associado a outros valores e não à função *fitness*.

3.3.4 Paisagem de Aptidão

Após a definição destas três entidades anteriores, podemos explicar o que seria a paisagem de aptidão, que como já mencionado anteriormente consiste da junção dessas entidades com uma instância do problema e também a definição de busca local.

Moscato [13] definiu a paisagem como sendo um digrafo ponderado, considerando que os vértices seriam as configurações do espaço de busca $S(x)$, as arestas então conectariam as configurações vizinhas. Assim, os pesos das arestas representariam a diferença entre as funções guias dos dois vértices conectados pela aresta e o sentido da aresta se daria do vértice com maior valor de Fg, para o vértice com menor valor de Fg.

Com esta definição, a busca foi definida por Moscato como: o processo de percorrer este grafo, através dos pesos, baseados nas funções guias, até se chegar a um vértice cujo valor da função guia seja melhor do que todos os outros valores dos seus vizinhos, podendo ser considerado um ótimo local.

Portanto, o algoritmo de busca local começaria a partir de uma configuração atual $s_0 \in S(x)$, e utilizaria um processo iterativo, onde a cada passo verificaria se a transição, baseada na vizinhança da configuração atual, conduziria a outra configuração melhor: se for melhor a configuração atual, para o próximo passo, seria esta nova configuração gerada; caso contrário a configuração atual é mantida. Este processo iterativo deve ser repetido até que um critério de parada seja alcançado, como por exemplo, realizar um número pré-determinado de iterações ou parar depois de não ter achado melhora nas últimas n iterações [13]. No Algoritmo 3 podemos observar este processo com mais clareza, onde a configuração é o agente e a transição a Fg.

Moscato ainda definiu a busca local em relação aos algoritmos populacionais, mostrando que ela consistiria de visitas em um hipergrafo[1] e não mais em um grafo, onde cada vértice representaria um conjunto de configurações em $S(x)$, ou seja, a população. Os próximos vértices a serem visitados (novas populações) seriam estabelecidos de acordo com a composição das vizinhanças e dos mecanismos de movimentos, podendo, além do operador de mutação já indicado anteriormente, usar o operador de recombinação.

A fase de busca local, segundo Krasnogor e Smith [16], pode acontecer antes ou depois da recombinação, mutação, seleção, ou em qualquer combinação possível, com as buscas locais pertencendo a uma grande variedade de heurísticas, com algoritmos exatos ou aproximados.

[1] É uma generalização do conceito de grafos, podendo ter mais de duas arestas para cada vértice.

3.4 Recombinação

Conforme Merz e Freisleben ([15] *apud* [12]), os operadores de recombinação e mutação agem como estratégias de diversificação nos AMs.

Na definição de relação de vizinhança (seção 3.3.2) vimos que a busca local pode ser baseada no operador de mutação. Para Moscato [13] a utilização de mutação, apesar de simples, já se revelou um operador poderoso, pois obteve soluções de boa qualidade para muitos problemas NP-difíceis, mas que pesquisas estão sendo realizadas para encontrar novos operadores para serem utilizados de maneira individual, ou mesmo, para complementar o operador de mutação.

Segundo Moscato [13], a utilização de algoritmos de busca baseados em populações, já permitiu a definição de operadores generalizados para a realização dos movimentos, os chamados operadores de **recombinação**. O processo de recombinação pode ser definido como o conjunto de "pais": $Spais$, de n configurações, o qual é manipulado de maneira a criar outro conjunto de "descendentes" $Sdesc \in sol(x)$ de m configurações novas, sendo os descendentes criados a partir da combinação das características dos pais.

Algumas propriedades podem ser encontradas na maioria dos operadores de recombinação [13]:

- **Respeito:** responsável por mostrar o caráter explorador ou intensificador de um operador de recombinação. Um operador é respeitoso, se e somente se, gerar descendentes que levem todas as características básicas que são comuns aos pais;
- **Sortimento:** responsável por representar a diversificação do operador. Um operador é adequadamente sortido, se e somente se, gerar descendentes que possuam qualquer combinação de características compatíveis presentes nos pais;
- **Transmissão:** um operador está transmitindo quando as características presentes nos descendentes pertencem a pelo menos um dos pais.

Assim, podemos dizer que os operadores de recombinação somente transmitem as características dos pais para os descendentes, sem inserir novas características, sendo o operador de mutação responsável pela inserção de características novas.

Para Moscato [13] esses operadores de recombinação podem ainda ser classificados de acordo com a utilização ou não de informações da instância do problema (x):

- **Cegos:** são os operadores que não recebem nenhum outro dado de entrada além do conjunto de $Spais$, assim não utilizam informações da instância do problema;

- **Heurísticos ou Híbridos:** são os operadores que usam conhecimentos sobre o problema, assim utilizam as informações do problema para guiarem a criação dos descendentes, podendo ser feito de várias formas para cada problema, sem uma taxonomia fixa. Porém, o conhecimento do problema pode ser introduzido em dois momentos: na seleção das características dos pais que serão repassadas aos descendentes, ou na seleção das características não vindas dos pais que serão adicionadas aos descendentes.

Moscato conclui que os operadores de recombinação heurísticos geram soluções melhores do que os operadores classificados como cegos. Porém, os heurísticos requerem um maior custo computacional para serem executados.

3.5 Projeto de um AM

Um AM mantém a todo o momento, assim como um AG, uma população com diversas soluções para o problema. Cada elemento desta população é chamado de **agente**, equivalente aos indivíduos nos AGs. Estes agentes se relacionam através de competição e cooperação, já mencionados na seção 3.2. A cada novo relacionamento, aplicação de competições (seleção e busca local) e cooperações (recombinação), entre os agentes modificando a população, dizemos que temos uma nova **geração**.

Algorithm 1 Algoritmo da Função AM.

```
Função AM(tamanhoPopulacao: ℕ, opr: Operador[])
variáveis
pop: Agente[];
início
pop ← iniciarPopulacao(tamanhoPopulacao);
enquanto terminaCriterioAM() faça
    pop ← criarGeracao(pop, opr);
    se convergePrematura(pop) então
        pop ← reiniciarPopulacao(pop);
    fim se
fim enquanto
retorne melhor(pop);
fim
```

No Algoritmo 1, adaptado de [2], começaremos a descrever os passos genéricos para a construção de um AM, baseado na definição mostrada por Moscato e Cotta [2].

Podemos observar que o primeiro passo de AM consiste em **iniciar a população**, podendo ser de diferentes formas: de modo aleatório, através da utilização de heurísticas, ou ainda aplicando-se uma busca local. Esta última forma pode ser exemplificada no Algoritmo 2, adaptado de [2], onde inicialmente é gerado um agente de maneira aleatória e depois aplicado a busca local sobre o mesmo.

Algorithm 2 Algoritmo da Função $iniciarPopulacao$.

Função $iniciarPopulacao$($tamanhoPopulacao$:$\mathbb{N}$)
variáveis
pop: Agente[];
$agente$: Agente;
j: $\mathbb{N}$;
início
para $j \leftarrow 1$ **até** $tamanhoPopulacao$ **faça**
  $agente \leftarrow agenteAletorio()$;
  $pop[j] \leftarrow buscaLocal(agente)$;
fim para
retorne pop;
fim

Algorithm 3 Algoritmo da Função $buscaLocal$.

Função $buscaLocal$ ($atual$: Agente, op: Operador)
variáveis
$novo$: Agente;
início
enquanto $terminaCriterioLocal()$ **faça**
  $novo \leftarrow aplicar(atual, op)$;
  se $Fg(novo)$ **melhor** $Fg(atual)$ **então**
    $atual \leftarrow novo$;
  fim se
fim enquanto
retorne $atual$;
fim

A **busca local**, mostrada no Algoritmo 3, adaptado de [2], nada mais é do que a tentativa de encontrar um agente melhor a partir de um agente base, ou seja, melhorar o valor da função guia Fg do agente base, através da aplicação de um operador, até um certo critério de parada. O

operador geralmente aplicado é o operador de mutação, sendo que o agente atual só será substituído se o novo agente gerado pela mutação tiver um valor melhor (que pode ser a maximização ou minimização do valor, dependendo do problema) da função guia [2]. Os critérios de parada podem ser os mesmos vistos na definição de busca local (seção 3.3)

Os próximos passos do AM: **criar uma nova geração** e **verificar se houve a convergência** para então **reiniciar a população**, detalhados a seguir, são repetidas várias vezes até que um critério seja alcançado, *terminaCriterioAM*. Para Moscato [13] este critério pode ser definido de maneira semelhante ao critério de parada da busca local, visto na seção 3.3.

A **criação de uma nova geração**, representada na Algoritmo 4, adaptado de [2], tem início na seleção dos agentes chamados de "criadores", mais conhecidos como pais, dentre a população, para mediante a recombinação (função "reproduzir") de suas características criarem novos agentes, e por fim a população ser atualizada [2]. Para Moscato e Cotta [2] é através da seleção e atualização que ocorre a competição entre os agentes, sendo a seleção responsável por eleger, através da função guia Fg, os melhores agentes presentes na população. Já a atualização encarrega-se de limitar o tamanho da população, eliminando alguns agentes para permitir a entrada de outros novos, não se tem um valor fixo para essa eliminação o mesmo depende da estratégia escolhida, a função guia também pode ser utilizada para a seleção dos agentes a serem eliminados.

Algorithm 4 Algoritmo da Função $criarGeracao$.

Função $criarGeracao$ (*pop*: Agente[], opr: Operador[])
variáveis
$criadores$, $novaPop$: Agente[];
início
$criadores \leftarrow selecionar$(*pop*);
$novaPop \leftarrow reproduzir$($criadores$, opr);
pop $\leftarrow atualizar$(*pop*, $novaPop$);
retorne *pop*;
fim

Moscato [13] mostra que existem duas possibilidades para a atualização da população: a estratégia "*plus*" e a "*comma*", onde na primeira a população é gerada a partir dos melhores agentes resultantes da união da população atual com a nova população gerada pela reprodução, ou seja, os melhores dentre $pop \cup novaPop$, já na segunda estratégia seleciona-se somente os melhores agentes da nova população. Afirma ainda que há vários estudos para a escolha de

procedimentos eficientes para a atualização.

A **reprodução**, mostrada no Algoritmo 5, adaptado de [2], é responsável pela criação de novos agentes, e merece um enfoque especial, já que é nessa fase que a recombinação é aplicada, gerando a cooperação entre os agentes, podendo ser considerada o núcleo do AM, sendo o processo de reprodução definido como a aplicação de um número variado de operadores [2].

Algorithm 5 Algoritmo da Função $reproduzir$.

Função $reproduzir$ (pop: Agente[], opr: Operador[])
variáveis
buf: Agente[][];
j, $numOp$: $\mathbb{N}$;
início
$numOp \leftarrow |opr|$;
$buf[0] \leftarrow pop$;
para $j \leftarrow 1$ **até** $numOp$ **faça**
  $buf[j] \leftarrow aplicarOperador(opr[j], buf[j\text{-}1])$;
fim para
retorne $buf[numOp]$;
fim

Segundo Moscato [13], os operadores utilizados pelos AGs para a reprodução, se resume somente a dois operadores: o cruzamento, e logo após a mutação, porém, se um operador heurístico de cruzamento for utilizado pode ser mais conveniente a aplicação da mutação antes do cruzamento. Já os AMs geralmente utilizam quatro operadores, inserindo duas buscas locais no processo descrito anteriormente, ficando nesta ordem: recombinação, busca local, mutação e busca local, no entanto a ordem entre mutação e recombinação não é problemático para os AMs no caso de operadores de recombinação heurísticos serem utilizados, devido ao fato de incluírem buscas locais.

A **verificação se houve a convergência** é obtida através da verificação entre todos os agentes da população. Se houver grande similaridade entre eles pode-se dizer que o AM convergiu, ou seja, houve a convergência prematura para um local subótimo do espaço de busca, dificultando a exploração de outras regiões [2]. Segundo Moscato e Cotta [2], uma maneira de se quantificar a convergência é utilizando-se medidas de diversidade de informação na população, como a entropia de Shannon[2] [17].

[2]Para cada agente é associado uma probabilidade, de acordo com as informações que possui, ou de acordo com a função guia, e através dessas probabilidades é associado à população um escalar chamado entropia. Se esse valor

Moscato [13] mostra que existe outra possível estratégia para a verificação da convergência: a utilização de metodologias probabilísticas para determinar com certo grau de confiança se a população convergiu.

Uma vez verificado que houve a convergência prematura da população, a mesma deve ser **reiniciada**, já que todos os agentes são muito similares dificultando a exploração de outras regiões. Este processo de reiniciar a população pode ser realizado de várias maneiras, a mais utilizada, segundo Moscato e Cotta, é a estratégia de se manter uma percentagem da população atual, e completar o restante com novos agentes, conforme exemplificado no Algoritmo 6, adaptado de [2].

Algorithm 6 Algoritmo da Função $reiniciarPopulacao$.

Função $reiniciarPopulacao$ (pop: Agente[])
variáveis
$novaPop$: Agente[];
j, $cons$, $tamPop$, $porcentagemConservar$: $\mathbb{N}$;
início
$tamPop \leftarrow |pop|$;
$cons \leftarrow tamPop * porcentagemConservar$;
para $j \leftarrow 1$ **até** $cons$ **faça**
  $novaPop[j] \leftarrow melhor(pop)$;
fim para
para $j \leftarrow (cons + 1)$ **até** $tamPop$ **faça**
  $novaPop[j] \leftarrow agenteAletorio()$;
  $novaPop[j] \leftarrow buscaLocal(novaPop[j])$;
fim para
retorne $novaPop$;
fim

Outra estratégia evidenciada por Moscato [13] para a reinicialização é a utilização de um operador de mutação "forte"[3], para enviar a população para longe de sua posição atual do espaço de busca. Ele também evidencia alguns cuidados que devem ser observados em relação a essas duas estratégias de reinicialização: na primeira, tomar precauções para a população que foi mantida não "dominar" os novos agentes inseridos, por exemplo, diminuindo a chance dos agentes mantidos de serem selecionados para a recombinação; na segunda, criar um operador de mutação forte que pondere a relação entre ser forte ao ponto de distribuir qualquer informação

de entropia for menor que um limiar pré-definido, de acordo com a representação do problema, considera-se que houve a convergência da população.

[3]Que insere várias características novas no agente.

presente na população atual, mas não tão forte ao ponto de levar a população para uma nova convergência em poucas interações.

Assim, com as descrições apresentadas anteriormente finalizamos a definição do funcionamento dos AMs de maneira generalizada, ficando a escolha dos operadores de seleção, mutação, recombinação, e algumas outras estratégias a cabo do projetista do AM. Estas escolhas devem ser realizadas levando sempre em consideração a definição do problema, podendo optar-se por: utilizar operadores e estratégias já presentes na literatura; ou desenvolver novos operadores e estratégias específicos ou gerais para o problema.

Capítulo 4

Problema do Caixeiro Viajante

Para Goldbarg e Luna [18], o Problema do Caixeiro Viajante (PCV) é um dos mais tradicionais e conhecidos problemas de otimização combinatória. A primeira menção conhecida do problema é devida a Hassler Whitney em 1934 em um trabalho na Universidade de Princeton. O PCV é definido, por Goldbarg e Luna, como um problema de otimização associado ao problema da determinação dos Caminhos Hamiltonianos em um grafo qualquer, com o objetivo de encontrar, nesse grafo, o caminho Hamiltoniano de menor custo.

A denominação Hamiltoniano surgiu em 1857, em homenagem a Willian Rowan Hamilton, que propôs um jogo que denominou *Around the World*. Este jogo era feito sobre um dodecaedro em que cada vértice estava associado a uma cidade, cujo grafo do problema é mostrado na Figura 4.1. O desafio era encontrar uma rota através dos vértices que iniciasse e terminasse em uma mesma cidade sem nunca repetir vértices [18].

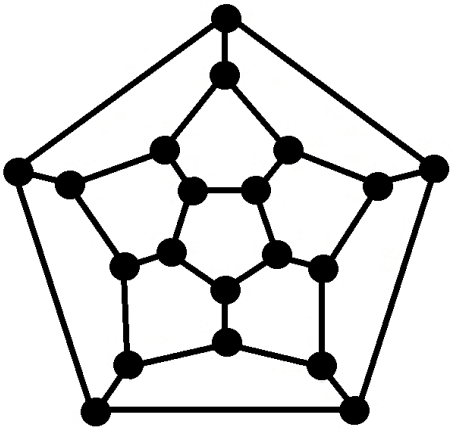


Figura 4.1: Grafo do jogo de Hamilton

O PCV pode então ser definido como um grafo $G = (N, E)$ onde: $N = 1, ..., n$ é o conjunto de vértices; $E = 1, ..., m$ é o conjunto de arestas de G; e c_{ij} é o custo associado a cada aresta que liga os vértices i e j deste grafo, devendo ser encontrado o menor ciclo Hamiltoniano de G, sendo o tamanho do ciclo a soma dos custos das arestas que formam o ciclo.

Podemos classificar o PCV como:

- **simétrico:** se para todos os pares de vértices i, j, os custos c_{ij} e c_{ji} forem iguais;
- **assimétrico:** oposto ao simétrico, ou seja, se para todos os pares de vértices o $c_{ij} \neq c_{ji}$.

No ano de 1990, segundo Prestes [19], Reinelt criou uma biblioteca contendo várias instâncias, que variam de 14 até 85.900 cidades, do PCV que vinham sendo e são até hoje testadas e discutidas na literatura. Essa biblioteca é conhecida como TSPLIB[1] e contém mais de 100 exemplos. Na fase de testes e análises a mesma será utilizada como entrada para as soluções implementadas.

4.1 Métodos para a Solução do PCV

Devido à grande importância do PCV, que pode ser aplicado em várias situações reais, como: controle de robôs, fabricação de placas de circuitos eletrônicos, sequenciamento de tarefas, entre outras [14], muitas abordagens têm sido desenvolvidas com o propósito de resolver um conjunto cada vez maior de instâncias desse problema. Estas abordagens podem ser dividas em, pelo menos, duas grandes categorias: Métodos Exatos e Métodos Heurísticos.

4.1.1 Métodos Exatos

A primeira intuição que temos para resolver o PCV é testar todas as possibilidades, para encontrar a com menor tamanho, porém o PCV é um problema de otimização combinatória e estes são considerados problemas NP-difícil, assim, esta abordagem é impraticável em função do crescimento exponencial das possibilidades em relação ao número de cidades [19].

No entanto, existem outras estratégias que podem garantir a obtenção de uma solução ótima, como: *branch-and-bound* e *branch-and-cut*, contudo estes métodos podem demorar milhões

[1]Esta biblioteca pode ser encontrada no site: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

ou até bilhões de anos para retornarem a solução exata, dependendo do número de cidades presentes na instância.

4.1.2 Métodos Heurísticos

Na maioria das vezes não geram soluções ótimas, mas aproximadas, contudo o tempo de processamento para grandes instâncias do PCV é menor do que os métodos exatos, e seus resultados são considerados satisfatórios.

Alguns dos métodos heurísticos mais utilizados, segundo Goldbarg e Luna [18], seriam: Algoritmos Heurísticos Construtivos; Heurística de Bellmore e Nemhauser; Heurísticas de Inserção; Heurísticas de K-Substituições ou K-Opt; Heurística Dynasearch e Heurística de *Saving* ou das Economias. São classificados ainda como heurísticos as soluções baseadas em Busca Tabu, AGs, AMs, "*Simulated Annealing*", Redes Neurais, Colônia de Formigas, entre outros.

Considerando que o interesse deste trabalho é comparar a eficiência dos AMs em relação aos AGs, vamos abordar, no decorrer deste capítulo, como esses dois métodos e o método clássico do Vizinho Mais Próximo (VMP), se comportam para a resolução do PCV.

4.2 Solução do PCV Através do VMP

Este método é considerado simples e de fácil implementação, já que consiste na idéia de visitar a cidade mais próxima que ainda não foi visitada. A seguir podemos ver os três passos do algoritmo do Vizinho Mais Próximo (VMP), adaptado de Levitin [20]:

- **Passo 1:** Escolher uma cidade de maneira aleatória para ser a cidade inicial;
- **Passo 2:** Visitar a cidade mais próxima a última visitada, até que todas as cidades sejam visitadas;
- **Passo 3:** Retornar para a cidade de partida (inicial).

Segundo Levitin este método tem um problema, que consiste na possibilidade de no último passo (passo 3) forçar o viajante a percorrer um caminho grande para retornar a cidade inicial.

4.3 Solução do PCV Através de AGs

Como vimos no Capítulo 2, a primeira decisão a ser tomada na elaboração de um AG, é em relação a escolha da representação das soluções candidatas a resolução do problema, no caso o PCV. Uma das representações mais naturais e utilizadas é a sequência de inteiros, onde o cromossomo é formado pela sequência dos vértices do percurso [18], conforme o exemplo da Figura 4.2, onde o cromossomo representa uma das possíveis soluções para a instância do PCV exemplificada no grafo.

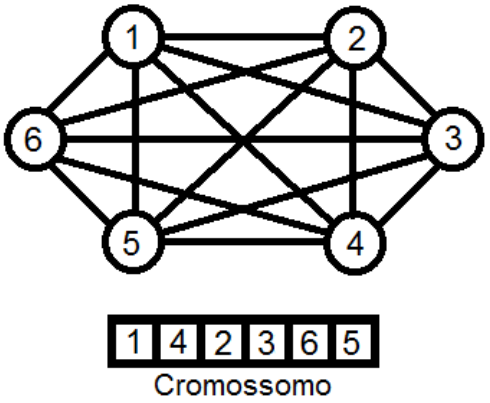


Figura 4.2: Exemplo da representação com sequência de inteiros

Devemos escolher também a estratégia a ser utilizada para a criação da população inicial e a função de avaliação. Optamos por inicializar a população com o método VMP (seção 4.2), por se tratar de um método simples, de rápida execução e por retornar um percurso perto da solução ótima, o que não aconteceu com a tentativa de inicializar a população de maneira aleatória, gerando uma população longe da melhor solução e que acarretaria na necessidade de um número maior de gerações para se alcançar uma solução satisfatória. Para avaliar os indivíduos (função *fitness*), usamos o custo total para se percorrer o percurso: $\sum c_{ij}$.

Definida a representação, função de avaliação e a estratégia de inicialização a serem utilizadas, passamos para a fase da escolha dos operadores. Para a seleção dos indivíduos que passarão para a fase de cruzamento, optamos pelo método da roleta, já exposto na seção 2.4.1.

Já para o operador de cruzamento podemos encontrar, na literatura, vários operadores desenvolvidos especialmente para a resolução do PCV [18], [21], [22]. Na Tabela 4.1, adaptada de Larrañaga et al [22] e complementada com Ozcan e Erenturk [21] e outros trabalhos, podem

Tabela 4.1: Alguns operadores de cruzamento para o PCV

Operador	Autores	Ano
Partially-mapped Crossover	Goldberg e Lingle	(1985)
Order Crossover	Davis	(1985)
Sorted Match Crossover	Brady	(1985)
Cycle Crossover	Oliver, Smith e Holland	(1987)
Heuristic Crossover	Grefenstette	(1987)
Maximal Preservative Crossover	Mühlenbein, Schleuter e Krämer	(1988)
Edge Recombination Crossover	Whitley, Timothy e Fuquay	(1989)
Voting Recombination Crossover	Mühlenbein	(1989)
Order Based Crossover	Syswerda	(1991)
Position Based Crossover	Syswerda	(1991)
Alternating-positions Crossover	Larrañaga, Kuijpers, Poza e Murga	(1996)
Distance Preserving Crossover	Freisbein e Merz	(1996)
Edge Assembly Crossover	Nagata e Kobayashi	(1997)
Inver-over Operator	Tao e Michalewicz	(1998)

ser visualizados alguns desses operadores.

Estes operadores buscam evitar a produção de filhos inviáveis, ou seja, ciclos inválidos. Um exemplo de filhos inviáveis é mostrado na Figura 4.3, onde podemos observar que com o cruzamento de um-ponto (seção 2.4.2) das características dos pais 1 e 2, os filhos gerados possuirão características repetidas, o que invalidaria o percurso do caixeiro viajante, já que o mesmo deve passar em cada cidade uma única vez e retornar para a cidade de origem.

Figura 4.3: Exemplo de filhos inviáveis

Dos operadores de cruzamento mostrados acima, levando-se em consideração a escolha de implementar o PCV simétrico completo, devido as entradas da biblioteca TSPLIB serem para este tipo de PCV, a representação escolhida e trabalhos que já compararam alguns destes operadores entre si, escolhemos o operador *Strategic Edge Crossover* (SEC) para ser implementado, já que este é um melhoramento do operador *Edge Recombination Crossover* que mostrou-se

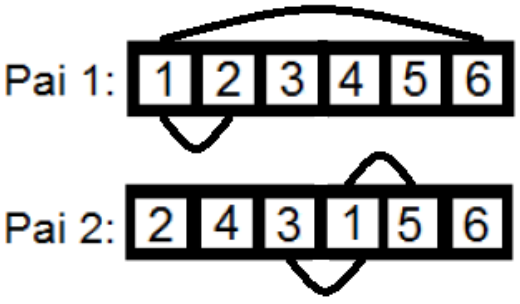
eficiente nos trabalhos de comparação [22], [23]. Por isso, vamos explicar mais detalhadamente como ele funciona.

4.3.1 *Strategic Edge Crossover*

Desenvolvido por Moscato e Norman no ano de 1992 [23], este operador de cruzamento busca construir os filhos maximizando a possibilidade de presença de arestas que sejam comuns a ambos os pais, e é apropriado para a resolução do PCV simétrico.

O operador *Strategic Edge Crossover* (SEC), utiliza um "*edge map*", que nada mais é do que uma tabela onde para cada cidade, pertencente ao percurso (coluna Cidade da Figura 4.4(b)), estão relacionadas as cidades, sem repetições, que se conectam a ela nos cromossomos dos pais (coluna Cidades Conectadas da Figura 4.4(b)). Por exemplo, se os pais selecionados para a fase de cruzamento tiverem os seguintes cromossomos: [1 2 3 4 5 6] e [2 4 3 1 5 6], podemos ver na Figura 4.4(a) as cidades conectadas a cidade 1 nos cromossomos dos pais.

Já na Figura 4.4(b) temos a tabela do "*edge map*" completa, onde observamos que na coluna de Cidades Conectadas algumas cidades estão com o sinal negativo (-), sendo assim marcadas para dizer que se repetem nas conexões, da cidade daquela linha da tabela, nos cromossomos dos pais. Por exemplo, na Figura 4.4(a) a cidade 4 no Pai1 está conectada as cidades 3 e 5 e no cromossomo do Pai2 está conectada as cidades 2 e novamente a cidade 3, o que faz com que a cidade 3 receba o sinal negativo no *edge map*.



(a) Cidades conectadas a cidade 1

Cidade	Cidades Conectadas
1	2, 6, 3, 5
2	1, 3, 4, 6
3	2,-4, 1
4	-3, 5, 2
5	4,-6, 1
6	1,-5, 2

(b) *Edge map* completo

Figura 4.4: Exemplo de "*edge map*" para os pais [1 2 3 4 5 6] e [2 4 3 1 5 6]

Após montado o "*edge map*" o SEC seguirá os seguintes passos, adaptado de Moscato e Norman [23], para a geração de um filho:

- **1º Passo:** Escolhe-se dentre as cidades uma para ser a cidade inicial, a escolha pode ser aleatória ou de acordo com o critério definido no 4º passo, e a atribui a cidade atual;

- **2º Passo:** Remover do "*edge map*" (coluna Cidades Conectadas) todas as ocorrências da cidade atual;

- **3º Passo:** Verificar se a cidade atual possui cidades conectadas a ela, se houver ir para o passo 4, caso contrário ir para o passo 5;

- **4º Passo:** Determinar dentre as cidades conectadas à cidade atual, qual delas será escolhida. Se tiver alguma cidade com sinal negativo esta deve ser escolhida. Senão, verificar qual delas tem o menor número de cidades conectadas a si, se duas ou mais cidades tiverem o mesmo número, escolher entre elas de maneira aleatória. A cidade escolhida passa a ser a cidade atual e retorna-se ao 2º passo;

- **5º Passo:** Se não houver mais cidades a serem visitadas, então o percurso já está completo e pode-se parar o cruzamento. Caso contrário, escolhe-se aleatoriamente uma cidade dentre as não visitadas para ser a cidade atual e retorna-se ao 2º passo.

Para o exemplo do "*edge map*" da Figura 4.4(b) e assumindo a cidade 2 como a escolhida inicialmente de maneira aleatória, um dos filhos que podemos obter é o cromossomo [2 4 3 1 6 5]. Como os passos descritos acima geram somente um filho, na implementação optamos por executá-los duas vezes para obtermos dois filhos, escolhendo as cidades iniciais de maneira aleatória.

Por fim, para realizar a mutação dos indivíduos, segundo Ozcan e Erenturk [21], também temos alguns operadores especialmente desenvolvidos na literatura para o PCV: *Simple Inversion Mutation*, Holland (1975); *Insertion Mutation*, Fogel (1988); *Exchange Mutation*, Banzhaf (1990); *Inversion Mutation*, Fogel (1990); *Scramble Mutation*, Syswerda (1991) e *Displacement Mutation*, Michalewizc (1992). Para a implementação utilizaremos o *Displacement Mutation* (DM), descrito na seção seguinte, já que no trabalho de Larrañaga et al [22] ele aponta o DM como um dos que obtiveram melhores resultados combinados com outros operadores de cruzamento, dentre eles o *Edge Recombination Crossover* que é a inspiração do SEC.

4.3.2 *Displacement Mutation*

Este operador de mutação seleciona um subpercurso de tamanho e posição randômica do cromossomo, removendo-o e inserindo-o em outra posição do cromossomo, também escolhida randomicamente [22].

Por exemplo, na Figura 4.5 o subpercurso [3 4 5] é selecionado e removido do cromossomo, restando o percurso [1 2 6 7 8], supondo que a cidade 7 seja escolhida randomicamente para receber o subpercurso, o resultado da mutação *Displacement Mutation* (DM) seria o cromossomo [1 2 6 7 3 4 5 8].

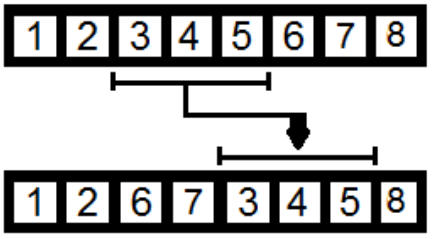


Figura 4.5: Exemplo de uma mutação DM

Por fim, após criada a nova geração fazemos a atualização desta com a população gerada anteriormente através da estratégia "*plus*", já descrita na seção 3.5.

4.4 Solução do PCV Através de AMs

Como já observado nos capítulos anteriores, o que diferencia os AMs dos AGs é a inserção de uma Busca Local, e também a utilização de informações da instância do problema para a fase de recombinação. Contudo, para uma melhor comparação entre eles optamos por implementar o mesmo operador, SEC, tanto para o AG como para o AM, apesar de este operador tentar melhorar os resultados do cruzamento, o que não seria feito se um operador de cruzamento dos AGs "puros" fosse escolhido, mas isso não desclassifica de AG o algoritmo implementado. Também, todas as outras estratégias e operadores foram mantidos na implementação do AM.

Para a fase de Busca Local, vários métodos podem ser utilizados, podemos citar: Monte Carlo, utilizado por Moscato e Norman [23]; *all-pairs* ou inserção, utilizado por Garcia [24]; Lin-Kernighan, utilizado por Ozcan e Erenturk [21] e muitos outros autores. Optamos pelo

método *2-Opt*, que deu origem ao Lin-Kernighan que utiliza *k-Opt*, devido a ser de fácil implementação e segundo Goldbarg e Luna [18] os resultados deste método são de boa qualidade. A seguir descrevemos como o mesmo funciona.

4.4.1 Busca *2-Opt*

Este algoritmo é baseado na remoção de duas arestas, o que resultaria na divisão do percurso final em dois caminhos distintos, e na reconecção desses dois caminhos de uma outra maneira alterando o percurso final.

Dado um percurso todas as possíveis trocas de pares de arestas *2-Opt*, devem ser realizadas, cada troca realizada é considerada um vizinho, por fim aquele vizinho que resultar no menor percurso será o escolhido para substituir a solução encontrada até o momento.

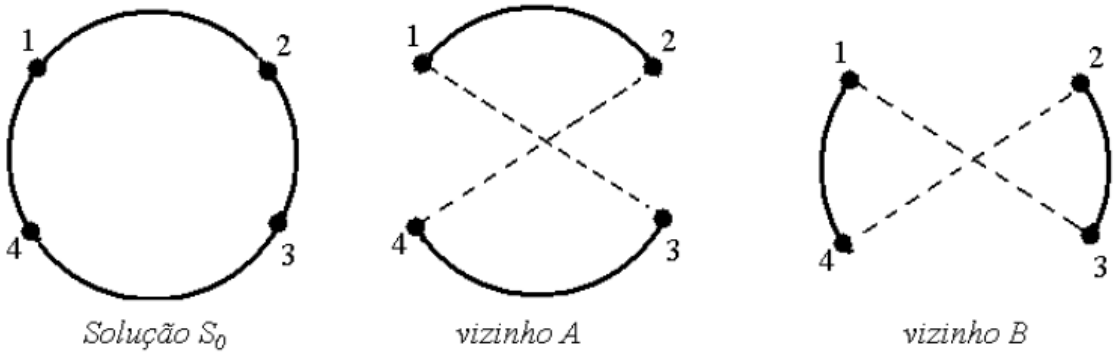


Figura 4.6: Exemplo de vizinhos obtidos com *2-Opt*

Na Figura 4.6, adaptada de Prestes [19], temos os vizinhos gerados a partir de uma busca *2-Opt* sobre a solução inicial S_0. Observamos que as soluções vizinhas da solução S_0 são geradas a partir da remoção dos pares de arestas:

- **Vizinho A:** remoção do par (1,4) e (2,3) que após a reconecção se tornam as arestas (1,3) e (2,4), gerando o percurso [1 2 4 3];
- **Vizinho B:** remoção do par (1,2) e (3,4) que após a reconecção se tornam as arestas (1,3) e (2,4), gerando o percurso [1 3 2 4].

Na implementação, o número de vizinhos buscados em cada busca *2-Opt*, foi igual a duas vezes o tamanho do percurso, ou seja, para um percurso de 50 cidades buscamos 100 vizi-

nhos. Esta decisão foi tomada para que o algoritmo executasse mais rápido, já que a intenção do presente trabalho não é o de encontrar os resultados ótimos para a solução do PCV, apenas comparar os algoritmos genéticos e meméticos e servir como base para trabalhos futuros. Contudo se melhores soluções desejam ser encontradas todas as possíveis trocas de pares de arestas podem ser implementadas, porém o tempo de execução será muito maior.

No capítulo 5 encontra-se a complementação deste, onde descrevemos os resultados encontrados através dos testes e analisamos os algoritmos, já que neste capítulo apenas descrevemos as estratégias, métodos e operadores utilizados.

Capítulo 5

Testes e Análises

Para a realização dos testes das implementações dos AGs, AMs e VMP, foram utilizados os arquivos encontrados na biblioteca TSPLIB, escolhendo-se 10 arquivos com números distintos de cidades, entre 51 e 1060. Na Tabela 5.1 podemos ver os nomes dos arquivos escolhidos, bem como seu número de cidades e também as melhores soluções encontradas até hoje segundo o site[1] da biblioteca.

Tabela 5.1: Arquivos de entrada utilizados para os testes dos AG, AM e VMP

Arquivo	Nº de Cidades	Melhor Solução
eil51	51	426
rat99	99	1211
bier127	127	118282
kroA200	200	29368
linhp318	318	41345
pr439	439	107217
rat575	575	6773
p654	654	34643
rat783	783	8806
u1060	1060	224094

Essas entradas são para a implementação de PCVs simétricos e completos, ou seja, as ligações não possuem distâncias diferentes entre ida e volta e as cidades estão ligadas todas entre si.

Como foi visto na seção 2.5, os AMs também utilizam os mesmos parâmetros: tamanho da população, taxa de cruzamento e taxa de mutação. Nas implementações os dois algoritmos foram testados com os mesmos parâmetros, sendo eles: *Tamanho da População* igual a 200

[1] Site: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/

indivíduos; *Taxa de Cruzamento* de 80%; e *Taxa de Mutação* de 5%.

Além desses parâmetros, foi fixado em 50 o número máximo de gerações que o algoritmo alcançará. Os testes foram executados em uma máquina com as seguintes configurações: Processador Intel (R) Core (TM) 2 Duo T6400 2.00Ghz; 3GB de Memória RAM; e Sistema Operacional Windows 7 32bits.

5.1 Resultados Obtidos e Análises

Começaremos pelos resultados encontrados para o algoritmo do VMP, já que este algoritmo também foi utilizado na inicialização da população dos algoritmos genéticos e meméticos, como já foi mostrado na seção 4.3.

5.1.1 Algoritmo do VMP

Na Tabela 5.2 listamos as distâncias encontradas, através do algoritmo VMP, para cada percurso dos arquivos de entrada, bem como o tempo de execução, em segundos, do algoritmo para que se obtivesse esses resultados.

Tabela 5.2: Resultados obtidos com o algoritmo do VMP

Arquivo	Melhor Solução	VMP	Tempo (Seg.)	Diferença (%)
eil51.tsp	426,0000	495,4429	0,0320	16,30
rat99.tsp	1211,0000	1383,7725	0,0310	14,27
bier127.tsp	118282,0000	136884,6655	0,0470	15,73
kroA200.tsp	29368,0000	37117,8222	0,1090	26,39
linhp318.tsp	41345,0000	50357,0873	0,2650	21,80
pr439.tsp	107217,0000	137385,7329	0,5140	28,14
rat575.tsp	6773,0000	8021,3903	0,8890	18,43
p654.tsp	34643,0000	42128,4465	1,1380	21,61
rat783.tsp	8806,0000	10571,0926	1,6380	19,99
u1060.tsp	224094,0000	285175,2213	2,9320	27,26

Também incluímos uma coluna denominada “Diferença”, que mostra a diferença, em porcentagem, conforme a Formula 5.1, entre o resultado encontrado e a melhor solução encontrada até o momento.

$$\frac{[(Resultado\,VMP) - (Melhor\ Solucao)] * 100}{Melhor\ Solucao} \tag{5.1}$$

Analisando os resultados, temos que em média as soluções obtidas pelo VMP foram $20,99\%$ mais altas que a melhor solução encontrada até o momento, com uma variação de $4,72\%$ para mais ou para menos. Já em relação ao tempo de execução, obtivemos uma média de $0,76$ segundos, com $0,89$ segundos de variação, o que comprova que este é mesmo um algoritmo de rápida execução e que retorna valores próximos da melhor solução.

5.1.2 Algoritmo Genético

A Tabela 5.3 segue o mesmo padrão apresentado e explicado na seção anterior, mas agora para os resultados obtidos com a implementação do AG, que já foi explicado de maneira geral no capítulo 2 e especificamente para o PCV na seção 4.3.

Tabela 5.3: Resultados obtidos com o AG

Arquivo	Melhor Solução	AG	Tempo (Seg.)	Diferença (%)
eil51.tsp	426,0000	446,8852	60,3250	4,9
rat99.tsp	1211,0000	1303,8663	92,6790	7,67
bier127.tsp	118282,0000	123748,1599	125,2680	4,62
kroA200.tsp	29368,0000	33852,8857	231,1920	15,27
linhp318.tsp	41345,0000	47653,0490	501,2290	15,26
pr439.tsp	107217,0000	122755,5275	893,0550	14,49
rat575.tsp	6773,0000	7921,8742	1514,5130	16,96
p654.tsp	34643,0000	40638,4917	1919,1310	17,31
rat783.tsp	8806,0000	10566,0426	2751,4960	20,04
u1060.tsp	224094,0000	270428,9520	4950,6690	20,68

Os resultados nos mostram que em média as soluções obtidas pelo AG foram $13,72\%$ mais altas que a melhor solução encontrada até o momento, com uma variação de $5,61\%$. Em relação ao tempo de execução, a média ficou em $1303,96$ segundos, ou seja, aproximadamente 21 minutos, com variação de 24 minutos para mais ou para menos.

Assim, enquanto o VMP chega à porcentagem de 16 a 25% próximo à melhor solução, o AG chega de 8 a 19%, melhorando os resultados encontrados, conforme pode ser analisado no gráfico da Figura 5.1, onde os resultados do AG se mantém sempre abaixo dos resultados encontrados pelo VMP.

Podemos considerar que em média os resultados do AG foram $6,52\%$ melhores que os resultados do VMP. Porém, o tempo de execução do AG é maior, enquanto no VMP é de segundos, no AG chega a horas.

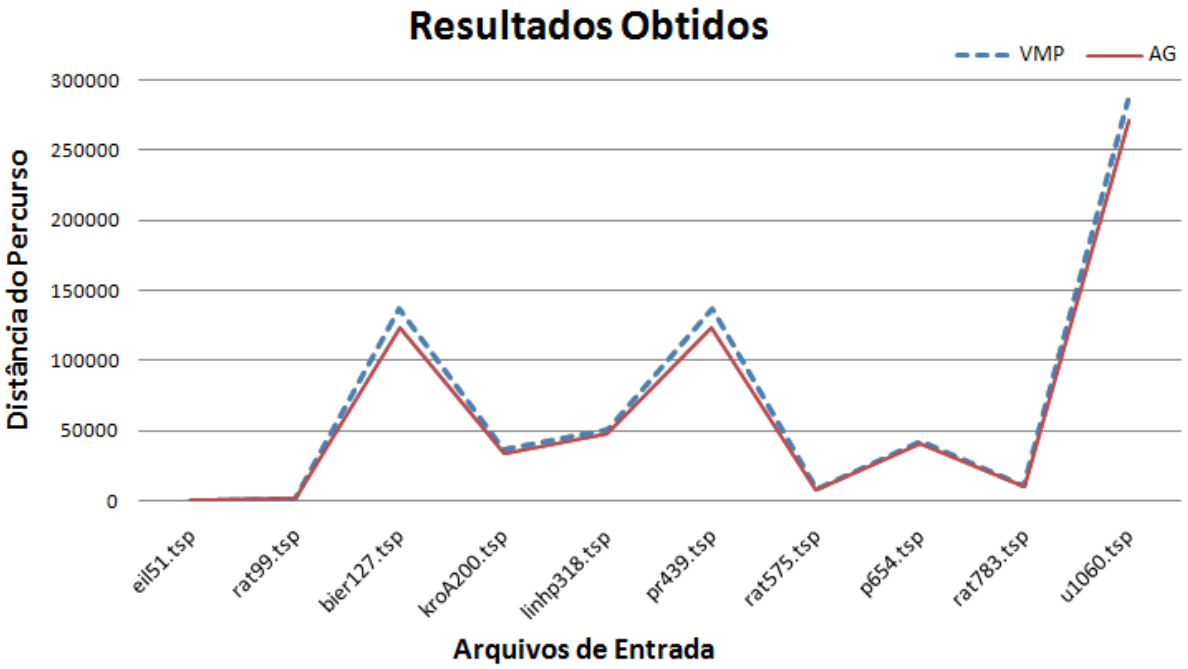


Figura 5.1: Comparação dos resultados obtidos com o algoritmo do VMP e o AG

5.1.3 Algoritmo Memético

A Tabela 5.4 segue o mesmo padrão apresentado e explicado na primeira seção (5.1.1), porém, para os resultados obtidos com a implementação do AM, explicado de maneira geral no capítulo 3 e na seção 4.4 especialmente para o PCV.

Tabela 5.4: Resultados obtidos com o AM

Arquivo	Melhor Solução	AM	Tempo (Seg.)	Diferença (%)
eil51.tsp	426,0000	433,5419	165,1420	1,77
rat99.tsp	1211,0000	1282,4055	292,3920	5,9
bier127.tsp	118282,0000	123748,1599	386,3340	4,62
kroA200.tsp	29368,0000	32903,0496	692,8600	12,04
linhp318.tsp	41345,0000	47653,0490	1369,2150	15,26
pr439.tsp	107217,0000	122555,5275	2343,9980	14,31
rat575.tsp	6773,0000	7722,6486	3693,7900	14,02
p654.tsp	34643,0000	40492,7835	4611,3840	16,89
rat783.tsp	8806,0000	10491,8709	6441,0650	19,14
u1060.tsp	224094,0000	269405,8038	20209,9200	20,22

A análise dos resultados mostra que em média as soluções obtidas pelo AM foram $12,42\%$ mais altas em relação à melhor solução encontrada até o momento, variando $5,98\%$ para mais

ou para menos. Para o tempo de execução, obtivemos a média de $4020,61$ segundos, ou seja, 1 hora e 7 minutos, com 1 hora e 6 minutos de variação.

Comparando o AM ao algoritmo do VMP, como foi feito na seção anterior para o AG, chegamos no seguinte resultado: enquanto o VMP chega na porcentagem de 16 a 25% próximo da melhor solução, o AM chega de 6 a 18%, assim, os resultados do AM são em média $7,79\%$ melhores que os resultados do VMP. No gráfico da Figura 5.2 podemos comprovar visualmente que o AM é realmente melhor que o VMP, como o AG.

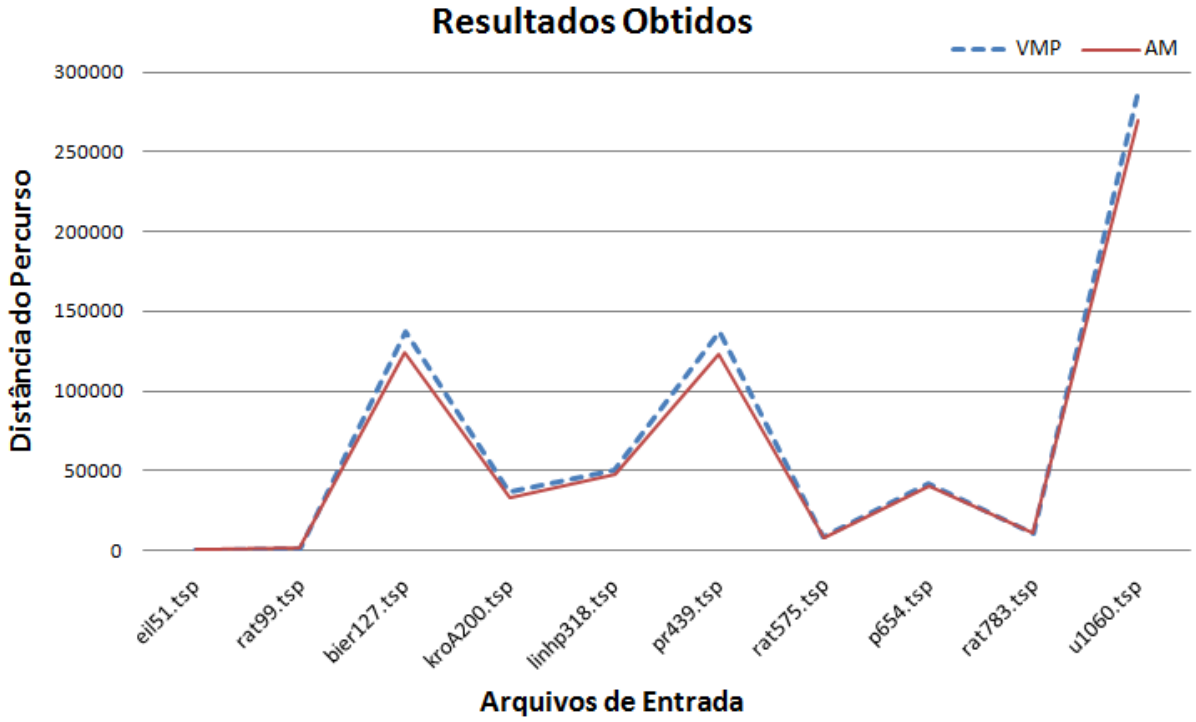
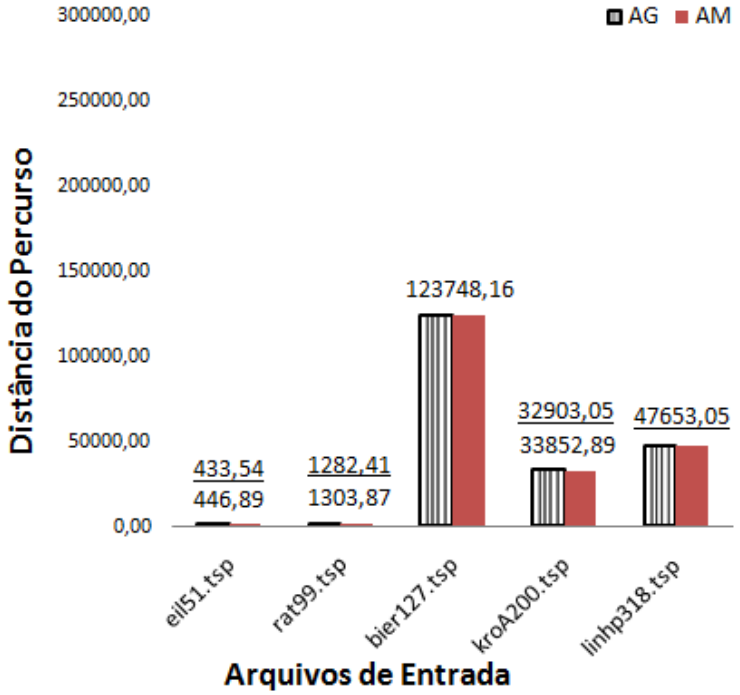


Figura 5.2: Comparação dos resultados obtidos com o algoritmo do VMP e o AM

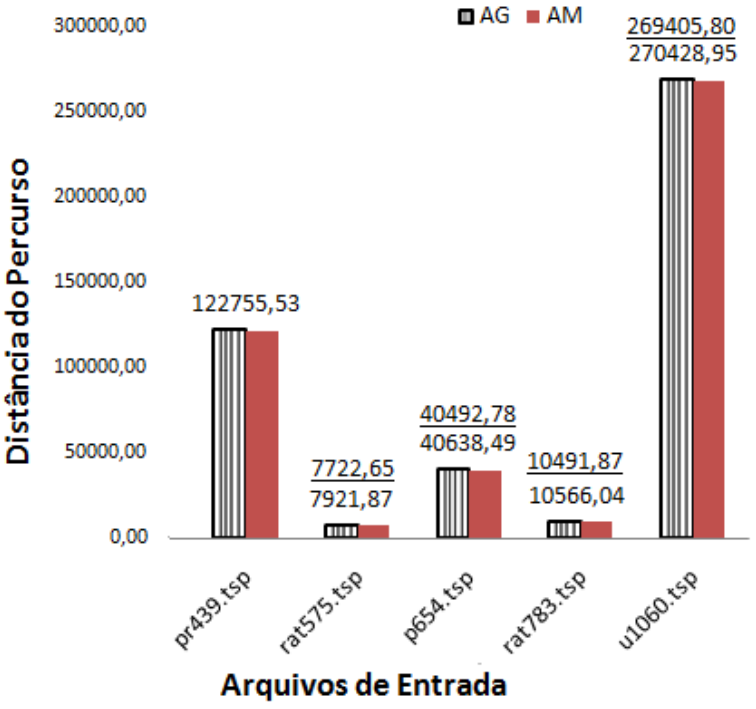
Na comparação com o VMP, os resultados do AM foram $1,27\%$ melhores que os resultados do AG na mesma comparação, contudo, o tempo de execução do AM é ainda maior que o do AG. Não mostraremos o gráfico visual da comparação entre os três algoritmos, por que, os resultados do AM e do AG são muito próximos ficando indistinguíveis no gráfico.

Por fim, comparando o AG ao AM, um dos objetivos iniciais deste trabalho, observamos que enquanto o AG chega de 8 a 19% próximo da melhor solução, o AM chega de 6 a 18%, melhorando ainda mais os resultados encontrados. Essa melhora é observada no gráfico da Figura 5.3, dividida em duas partes para mellhor visualização, onde os valores sublinhados

correspondem aos resultados do AM, os não sublinhados aos resultados do AG, e nos casos em que o resultado foi o mesmo o valor é colocado apenas uma vez sobre a barra.



(a) Parte A



(b) Parte B

Figura 5.3: Comparação dos resultados obtidos com o AG e o AM

Ao analisar os resultados, podemos considerar que em média os resultados do AM foram $1,19\%$ melhores que os resultados do AG, com uma variação de $1,18\%$ para mais ou para menos, ou seja, o AM foi de $0,01\%$ a $2,37\%$ melhor que o AG.

Esta porcentagem pequena de melhora pode ser atribuída a escolha da estratégia de busca local implementada, descrita na seção 4.4, em que o número de vizinhos buscados, dentre a vizinhança, foi de apenas $2 * (tamanho\ do\ percurso)$. Mas, somente novos testes com uma busca local mais abrangente poderiam confirmar ou não esta teoria, o que podemos perceber, é que com uma busca local mais abrangente, um tempo de execução maior seria necessário para se chegar aos resultados.

Comparando o tempo de execução do AM em relação ao AG, como já era esperado, devido a mais uma fase ser inserida neste algoritmo, o memético demorou mais tempo para executar. No gráfico da Figura 5.4 podemos visualizar melhor esta diferença de execução, onde para todos os arquivos de entrada, o AM demorou mais tempo para executar do que o genético.

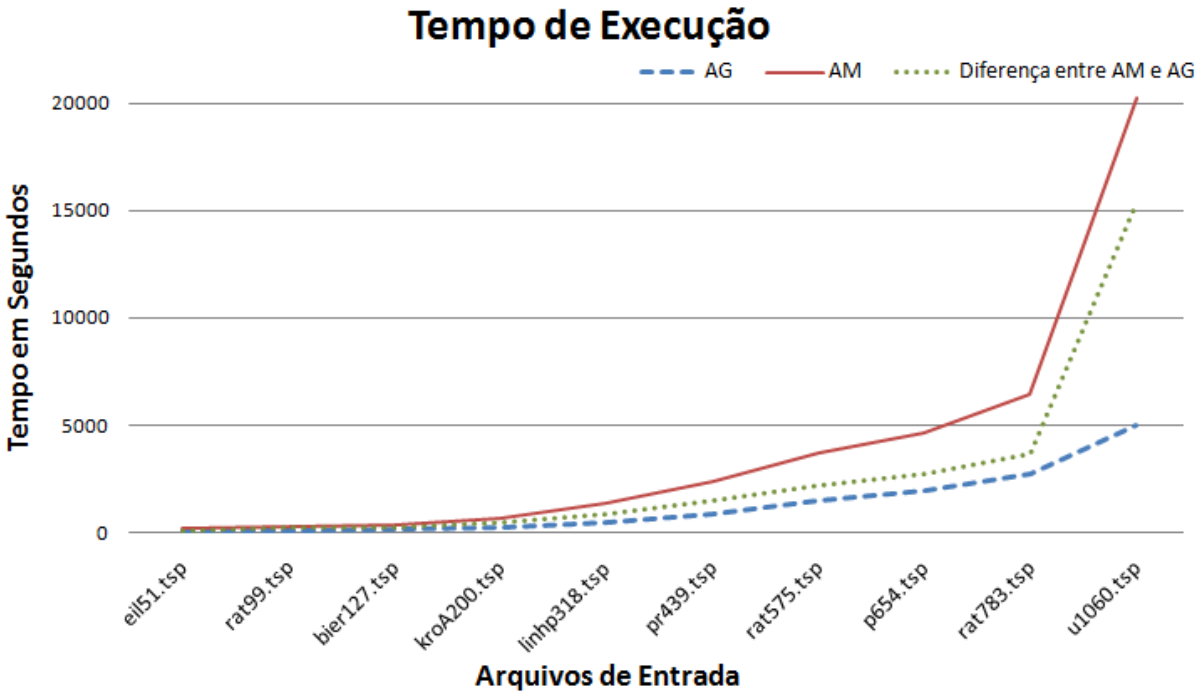


Figura 5.4: Comparação dos resultados obtidos com o AG e o AM

Na Figura 5.4, encontramos também a diferença entre o tempo de execução do AM em relação ao AG, mostrando que conforme o número de cidades do percurso aumenta, o tempo

de execução do AM consequentemente também aumenta devido a um número maior de buscas locais serem executadas, o que distancia ainda mais o tempo do AM em relação ao tempo de execução do AG. No apêndice A, são mostrados os percursos encontrados pelo AM para as entradas da biblioteca TSPLIB, listadas na Tabela 5.1 e no apêndice B para o AG.

Vale a pena ressaltar que se outras combinações, ou trocas de algumas escolhas de estratégias, métodos, operadores ou parâmetros, fossem implementadas, resultados e tempos menores poderiam ser encontrados, bem como, resultados piores e tempos mais demorados, tanto para o AG como para o AM.

Assim, uma ponderação deve ser feita entre o resultado desejado e o tempo que se deseja esperar para se alcançar este resultado, levando sempre em consideração para que fim a implementação será utilizada, pois em algumas situações o tempo é muito valioso, já em outras o resultado é que faz a diferença.

Mas, podemos concluir que o AM é realmente mais eficiente que o AG, em relação aos resultados, porém, em relação ao tempo de execução ainda deixa a desejar, devendo novos métodos de busca local serem pesquisados, para que haja um equilíbrio melhor entre tempo de execução e resultado. Contudo, se o tempo não tiver grande importância para a aplicação o AM pode ser utilizado com bons resultados.

Capítulo 6

Considerações Finais e Trabalhos Futuros

O estudo realizado sobre AMs e AGs, além do algoritmo do VMP, apresentado neste trabalho, ficará como material de pesquisa, para que novos trabalhos possam ser realizados nesta área, que ainda tem muito a crescer e desenvolver.

Além da definição teórica dos AMs e também dos já consolidados AGs, a utilização dos mesmos na resolução do PCV traz os passos, as estratégias, os métodos, os operadores e os parâmetros que devem ser escolhidos, quando se deseja resolver um problema com estes algoritmos, facilitando a implementação. Também podem ser encontradas as diferenças que distinguem os AMs dos AGs.

Neste trabalho demonstramos, principalmente, que os AMs conseguem encontrar resultados melhores que os AGs, porém, com tempos de execução maiores, sendo esta melhora e também o tempo de execução, uma consequência da combinação dos métodos e operadores escolhidos, principalmente do algoritmo de busca local, que pode fazer com que a aplicação tenha altos tempos de execução mas também resultar na melhora de seus resultados.

Assim, os interessados em utilizar os AMs em seus trabalhos, devem ter a consciência de que o AM é eficiente, contudo, possui altos tempos de execução, devendo tentar diminuir estes tempos.

Como trabalhos futuros, fica a sugestão de escolher outros métodos, estratégias, operadores ou parâmetros para a resolução do PCV utilizando o AM e compará-lo com os resultados obtidos neste trabalho, bem como, a utilização dos AMs na tentativa de resolução de outros problemas.

Apesar de os algoritmos implementados não terem obtido as melhores soluções encontradas até o momento para as entradas do PCV, os objetivos definidos no início deste trabalho foram alcançados.

Apêndice A

Percursos Encontrados pelo AM para as Entradas da Biblioteca TSPLIB

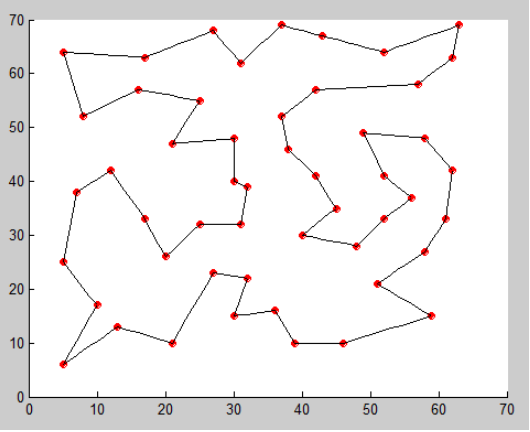


Figura A.1: Percurso obtido com o AM para a entrada *eil51.tsp*

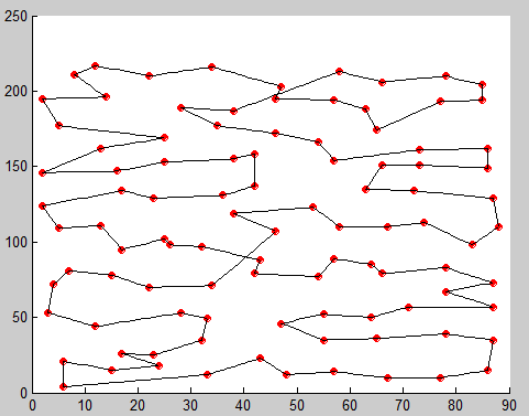


Figura A.2: Percurso obtido com o AM para a entrada *rat99.tsp*

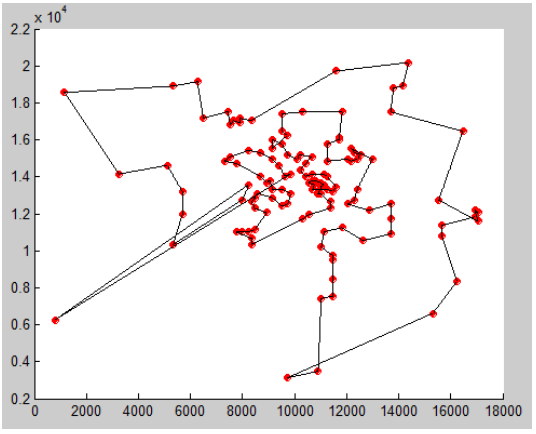


Figura A.3: Percurso obtido com o AM para a entrada *bier127.tsp*

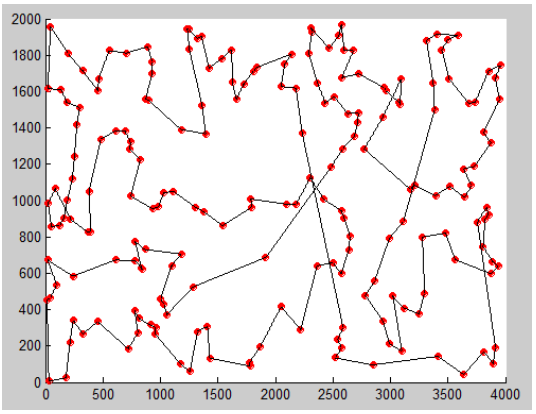


Figura A.4: Percurso obtido com o AM para a entrada *kroA200.tsp*

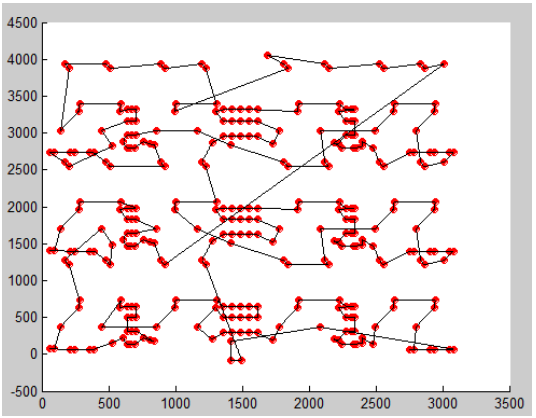


Figura A.5: Percurso obtido com o AM para a entrada *linhp318.tsp*

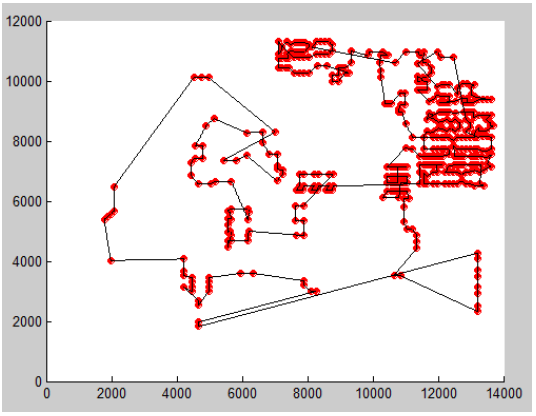


Figura A.6: Percurso obtido com o AM para a entrada *pr439.tsp*

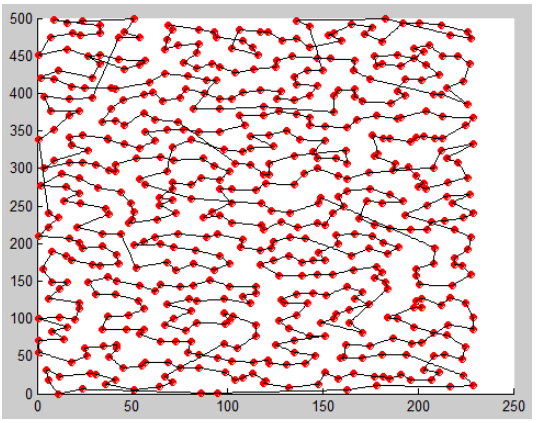


Figura A.7: Percurso obtido com o AM para a entrada *rat575.tsp*

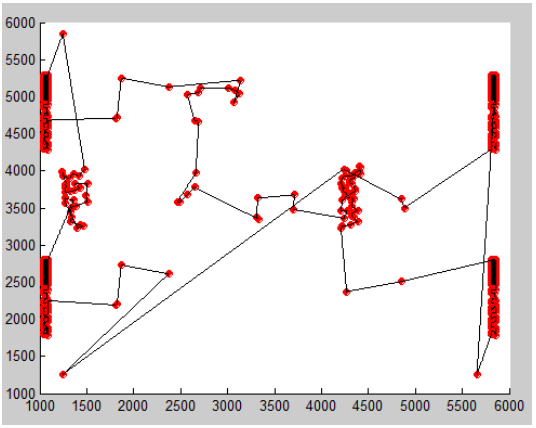


Figura A.8: Percurso obtido com o AM para a entrada *p654.tsp*

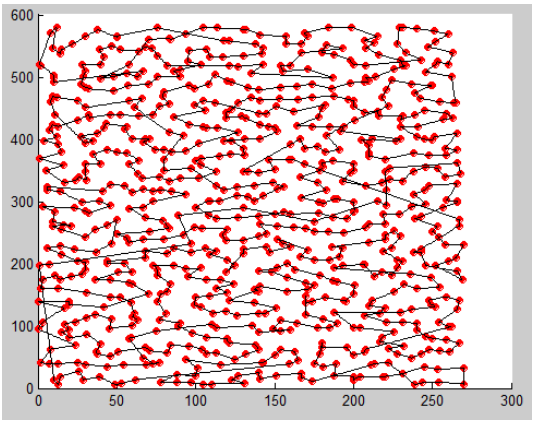


Figura A.9: Percurso obtido com o AM para a entrada *rat783.tsp*

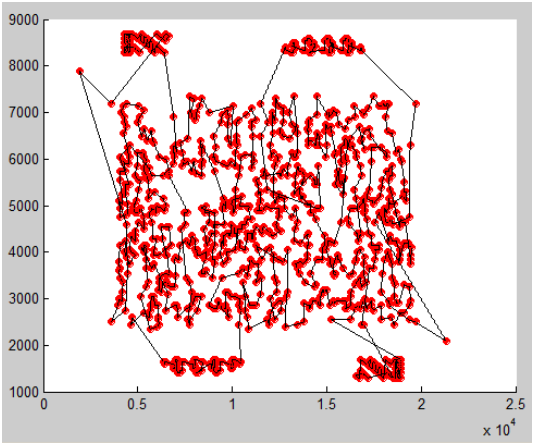


Figura A.10: Percurso obtido com o AM para a entrada *u1060.tsp*

Apêndice B

Percursos Encontrados pelo AG para as Entradas da Biblioteca TSPLIB

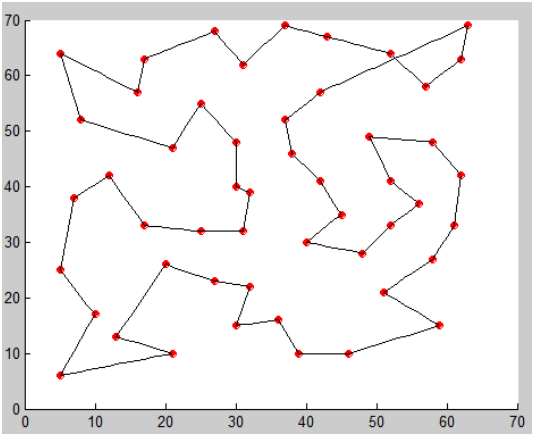


Figura B.1: Percurso obtido com o AG para a entrada *eil51.tsp*

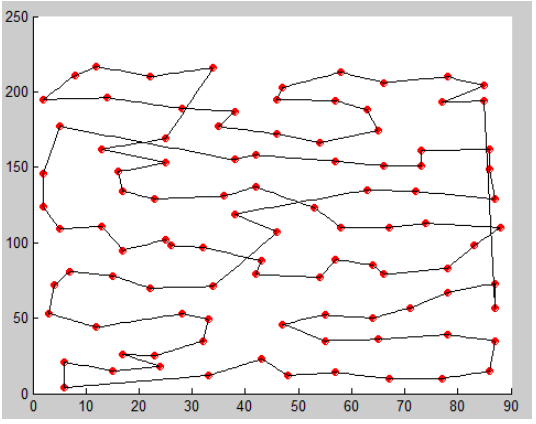


Figura B.2: Percurso obtido com o AG para a entrada *rat99.tsp*

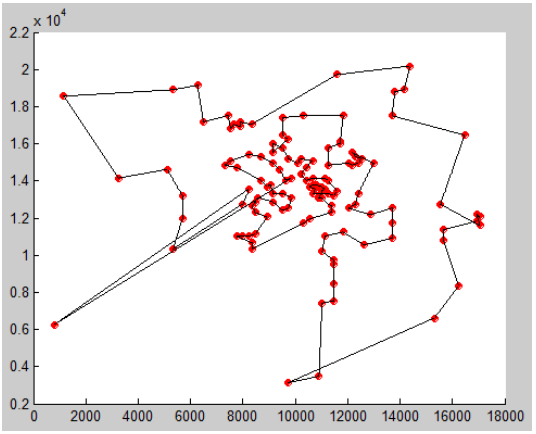


Figura B.3: Percurso obtido com o AG para a entrada *bier127.tsp*

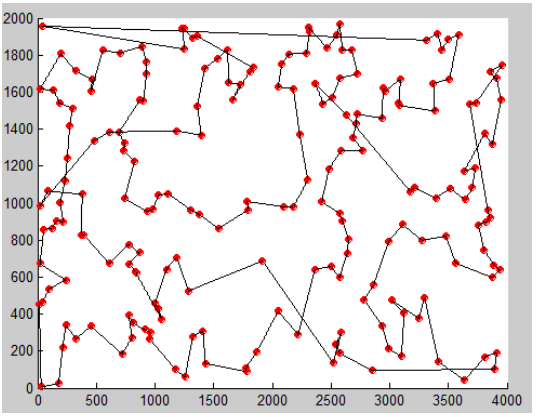


Figura B.4: Percurso obtido com o AG para a entrada *kroA200.tsp*

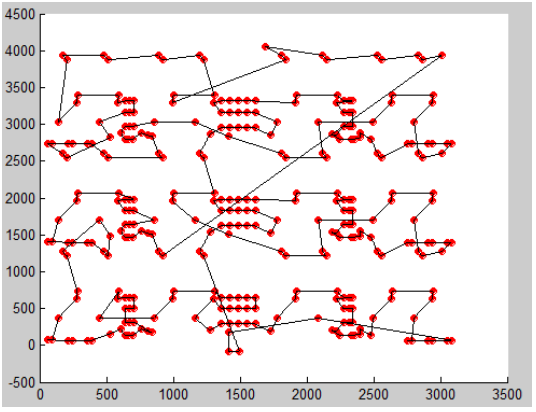


Figura B.5: Percurso obtido com o AG para a entrada *linhp318.tsp*

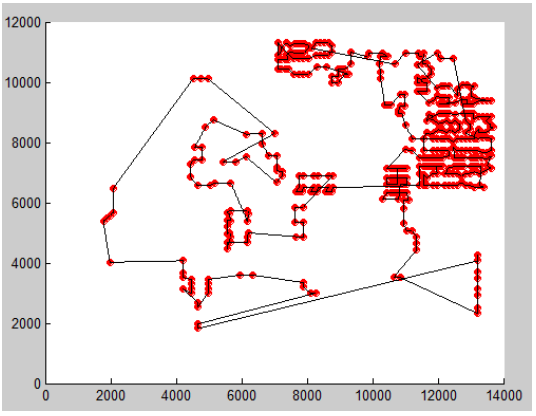


Figura B.6: Percurso obtido com o AG para a entrada *pr439.tsp*

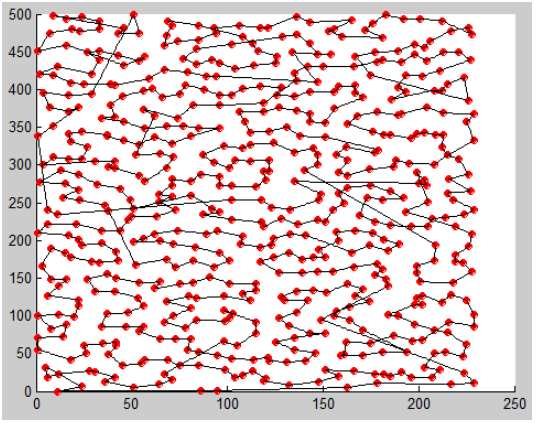


Figura B.7: Percurso obtido com o AG para a entrada *rat575.tsp*

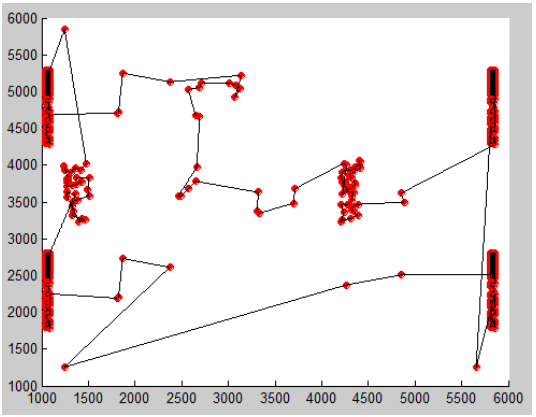


Figura B.8: Percurso obtido com o AG para a entrada *p654.tsp*

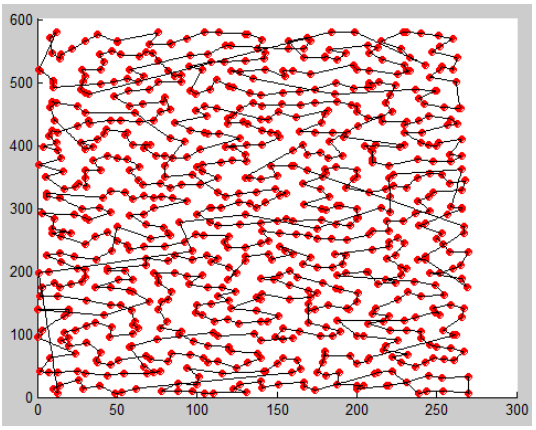


Figura B.9: Percurso obtido com o AG para a entrada *rat783.tsp*

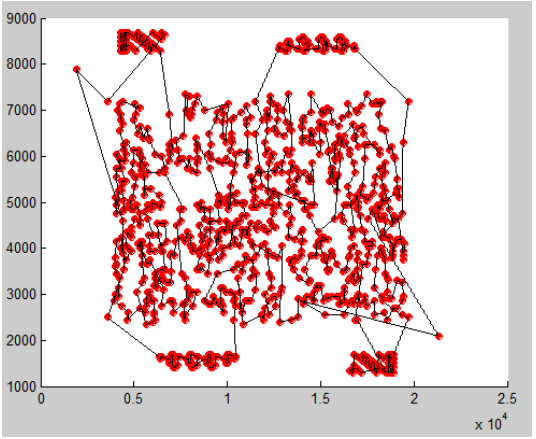


Figura B.10: Percurso obtido com o AG para a entrada *u1060.tsp*

Referências Bibliográficas

[1] HOLLAND, J. *Adaptation in Natural and Artificial Systems*. 1. ed. University of Michigan Press: Ann Arbor, 1975.

[2] MOSCATO, P.; COTTA, C. Una introducción a los algoritmos meméticos. *Revista Iberoamericana de Inteligência Artificial*, n. 19, p. 131 – 148, 2003.

[3] MOSCATO, P. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *C3P – Caltech Concurrent Computation Program*, n. 826, 1989.

[4] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization e Machine Learning*. 1. ed. Massachussets: Addison Wesley, 1989.

[5] MELANIE, M. *An Introduction to Genetic Algorithms*. 1. ed. Massachusetts, Londres: MIT Press, 1998.

[6] LAGUNA, M.; MOSCATO, P. Algoritmos genéticos. In: *Optimización Heurística y Redes Neuronales*. Madrid: Editorial Paraninfo, 1996. p. 67 – 103.

[7] LACERDA, E. G. M. de; CARVALHO, A. C. P. L. F. de. Introdução aos algoritmos genéticos. In: *Sistemas inteligentes: aplicações a recursos hídricos e ciências ambientais*. Porto Alegre - RS: Universidade/UFRGS, 1999. p. 99 – 150.

[8] KONDAGESKI, J. H. *Calibração de Modelos de Qualidade da Água para Rio Utilizando Algoritmo Genético*. Dissertação (Dissertação de Mestrado em Engenharia de Recursos Hídricos e Ambiental) — UFPR – Universidade Federal do Paraná, Curitiba - PR, 2008.

[9] MOGNON, V. R. *Algoritmos Genéticos Aplicados na Otimização de Antenas*. Dissertação (Dissertação de Mestrado em Engenharia Elétrica) — UFPR – Universidade Federal do Paraná, Curitiba - PR, 2004.

[10] CATARINA, A. S.; BACH, S. L. Estudo do efeito dos parâmetros genéticos na solução otimizada e no tempo de convergência em algoritmos genéticos com codificações binária e real. *Acta Scientiarum: Technology*, Maringá - PR, v. 25, n. 2, p. 147–152, 2003.

[11] DAWKINS, R. *The Selfish Gene*. 1. ed. Oxford: Oxford University Press, 1976.

[12] CONCILIO, R. *Contribuições à Solução de Problemas de Escalonamento pela Aplicação Conjunta de Computação Evolutiva e Otimização com Restrições*. Dissertação (Dissertação de Mestrado em Engenharia Elétrica) — UNICAMP – Faculdade de Engenharia Elétrica e de Computação, Campinas - SP, Dezembro 2000.

[13] MOSCATO, P. *Problemas de Otimização NP, Aproximabilidade e Computação Evolutiva: da Prática à Teoria*. Tese (Tese de Doutorado) — UNICAMP – Faculdade de Engenharia Elétrica e de Computação, Campinas - SP, Março 2001.

[14] BURIOL, L. S. *Algoritmo Memético para o Problema do Caixeiro Viajante Assimétrico como Parte de um Framework para Algoritmos Evolutivos*. Dissertação (Dissertação de Mestrado em Engenharia Elétrica) — UNICAMP – Faculdade de Engenharia Elétrica e de Computação, Campinas - SP, Fevereiro 2000.

[15] MERZ, P.; FREISLEBEN, B. A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem. In: *Proc. Congress on Evolutionary Computation, IEEE*. [S.l.]: Press, 1999. p. 2063 – 2070.

[16] KRASNOGOR, N.; SMITH, J. A tutorial for competent memetic algorithms: Model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, v. 9, n. 5, Outubro 2005.

[17] FIRKOWSKI, H. *Generalização Cartográfica de Grades Retangulares Regulares Baseada na Teoria Matemática da Comunicação*. Tese (Tese de Doutorado em Ciências Geodésicas) — UFPR – Universidade Federal do Paraná, Curitiba - PR, 2002.

[18] GOLDBARG, M. C.; LUNA, H. P. L. Problema do caixeiro viajante. In: *Otimização Combinatória e Programação Linear*. Rio de Janeiro: Editora Campus, 2000. p. 397 – 439.

[19] PRESTES Álvaro N. *Uma Análise Experimental de Abordagens Heurísticas Aplicadas ao Problema do Caixeiro Viajante*. Dissertação (Dissertação de Mestrado em Sistemas e Computação) — Universidade Federal do Rio Grande do Norte, Natal - RN, Julho 2006.

[20] LEVITIN, A. *Introdution to the Design e Analysis of Algorithms*. [S.l.]: Addison Wesley, 2003.

[21] OZCAN, E.; ERENTURK, M. A brief review of memetic algorithms for solving euclidean 2d traveling salesrep problem. In: *International Turkish Symposium on Artificial Intelligence and Neural Networks*. Izmir - Turquia: [s.n.], 2004.

[22] LARRAñAGA, P. et al. Genetic algorithms for the travelling salesman problem: A review of representations and operators. *Artificial Intelligence Review*, v. 13, p. 129–170, 1999.

[23] MOSCATO, P.; NORMAN, M. G. A "memetic"approach for the traveling salesman problem implementation of a computational ecology for combinatorial optimization on message-passing systems. In: *Proceedings of the International Conference on Parallel Computing and Transputer Applications*. [S.l.]: IOS Press, 1992. p. 177–186.

[24] GARCIA, V. J. *Algoritmo Memético Paralelo aplicado a Problemas de Otimização Combinatória*. Dissertação (Dissertação de Mestrado em Engenharia Elétrica) — UNICAMP – Faculdade de Engenharia Elétrica e de Computação, Campinas - SP, Março 2002.