NELSON HEIN

[illegible]

TÉCNICA DO COMPARTILHAMENTO SUCESSIVO

Um Algoritmo Memético na Otimização de Funções Multimodais

Tese apresentada como requisito parcial à obtenção do título de Doutor.
Curso de Engenharia de Produção e Sistemas
Universidade Federal de Santa Catarina
Orientador: Edgar Augusto Lanzer, Ph.D.

0.295.709-2
UFSC-BU
BU

FLORIANÓPOLIS
1998

NELSON HEIN

TÉCNICA DO COMPARTILHAMENTO SUCESSIVO
Um algoritmo Memético na Otimização de Funções Multimodais

Tese Aprovada como requisito parcial para a obtenção de Doutor no Curso de Pós-Graduação em Engenharia de Produção e Sistemas da Universidade Federal de Santa Catarina, pela comissão formada pelos professores:

Orientador: _____

Prof. Edgar Augusto Lanzer, Ph.D.

_____

Prof. Cláudio Loesch, Dr. Eng.

_____

Prof. Rogério Cid Bastos, Dr. Eng.

_____

Prof. Sérgio Fernando Mayerle, Dr. Eng.

_____

Prof. Geraldo Moretto, Dr.

Florianópolis, 05 de Outubro de 1998

"Das Leben ist gleich wie ein Traum
ein nichtger, leerer Wasseschaum;
es gleicht dem Gras, das heute steht
und schnell vergeht,
sobald der Wind darüber weht"

Joachim Neader

Tese dedicada ao amigo Plinio Stange, Dr. Ing. (*in memoriam*), que não teve tempo de vê-la pronta.

AGRADECIMENTOS

Ao professor Edgar Augusto Lanzer, PhD pela orientação, pela paciência, pela segurança transmitida e por tantos outros detalhes que me fizeram crescer. Obrigado **Lanzer**!

À minha esposa Salett, pelo sempre incentivo, pela sempre preocupação, pela sempre doação, por sempre ouvir, calar, falar, amar... . Obrigado **Salett**!

Aos meus pais **Ingo** e **Renata**, que entenderam a minha ausência, minhas falhas, meu humor. Com humildade me acompanharem neste período, sem entender claramente todo processo, porém compreendendo a importância disso tudo para mim.

Pai, acredito ter devolvido parte do seu braço com esta tese. Mãe, sua preocupação me acompanhou todo este tempo, e suas recomendações me guardaram em tantas idas e vindas pela 101.

Obrigado Pai! Obrigado Mãe!

Ao meu grande amigo, professor Cláudio Loesch, Dr. Eng. cúmplice deste trabalho. Meu incentivador, crítico e têrmometro de qualidade dos meus feitos. Obrigado **Loesch**!

Ao meu amigo João Carlos Redin, que em outro momento tive a honra de orientar, e que neste trabalho meu deu a honra de me auxiliar. Obrigado **Redin**!

A "minha" Universidade Regional de Blumenau, que me proporcionou este aprimoramento. Minha retribuição será em trabalho. Obrigado **FURB**!

Aos meus colegas do Departamento de Matemática-CCEN/FURB. Me orgulho de poder compartilhar meu trabalho e minha amizade com pessoas tão especiais. Obrigado **Amigos**!

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

É descrita uma técnica que permite com que métodos de otimização de funções unimodais serem estendidas para a localização eficiente de todos os ótimos de problemas multimodais. O trabalho descreve um algoritmo baseado em um algoritmo genético tradicional. Esta técnica consiste em "repetir" o algoritmo genético usando o conhecimento obtido durante uma iteração para evitar re-pesquisa, em iterações subseqüentes nas regiões do espaço do problema onde soluções já tenham sido encontradas. Este ganho é alcançado multiplicando a função original por uma função auxiliar de modificação, fazendo com que os valores dos *fitness* sejam desvalorizados, nas regiões do espaço de busca do problema, onde as soluções já tenham sido encontradas. Conseqüentemente, a probabilidade de descobrir uma "nova" solução em cada iteração é bastante incrementada. A técnica pode ser usada com vários estilos de algoritmos genéticos ou com outros métodos de otimização, tal como a têmpera simulada. A efeciência do algoritmo está demonstrada através de testes com funções multimodais, mostrando que a técnica é pelo menos tão rápida quanto o *fitness sharing*, fornecendo uma aceleração de 1 a 10**p**, dependendo da quantidade de **p** ótimos a serem localizados, e da complexidade do tempo na convergência.

ABSTRACT

A technique is described that allows unimodal function optimization methods to be extended to lacate all optima of multimodal problems efficiently. This work described an algorithm based on a traditional genetic algorithm. This technique involves iterating the genetic algorithm but uses knowlwdge gained during one iteration to avoid re-searching, on subsequent iterations, regions of problem space where solutions have already been found. This gain is achieved by applying a fitness derating function to the raw fitness fuction, so that fitness values are depressed in the regions of the problem space where solutions have already been found. Consequently, the likelihood of discovering a new solution on each iteration is dramatically increased. The technique may be used with various styles os genetic algorithms or with other optimization methods, such as simulated annealing. The effectiveness of the algorithm is demonstrated on a number of multimodal test functions. The technique is at least as fast as fitness sharing methods. It provides an accelerations of between 1 ans 10**p** on a problem with p optima, depending on the value of **p** and the convergence time complexity.

CAPÍTULO 1

1. INTRODUÇÃO

Este capítulo divide-se em quatro seções. Inicia-se com uma contextualização do tema, que serve de descrição da origem e abrangência do trabalho. Na segunda seção é apresentado o objetivo da tese, munido de algumas hipóteses centrais. A terceira seção apresenta os recursos utilizados na implementação da proposta de solução e na última seção é feita uma descrição organizacional da tese.

1.1. O Contexto

A Programação Matemática trabalha com a criação e fundamentação matemática de métodos eficientes que localizem o ótimo de uma função, podendo ser este ótimo: máximo ou mínimo, e ainda: local ou global; entretanto, em vários problemas, a função pode possuir vários ótimos que devem ser localizados. Para simplificar e sem perda de generalidade esta tese assume a maximização como objetivo.

Ao se avaliar projetos mecânicos, por exemplo, a função objetivo pode ser criada levando-se em consideração a adaptação de cada *design*. *Designs* totalmente diferentes podem possuir performances igualmente boas, porém, algum destes *designs* podem ser melhores que outros em termos de facilidade de produção, facilidade de manutenção, segurança, entre outros fatores. Usualmente, determinar um modo que combine todos esses fatores em uma simples pontuação de performance é difícil, tanto que nestes casos recorre-se a

otimização multiobjetivo, na qual a função de adaptação retorna com vários escores, cada um relatando um atributo a ser otimizado. Pode-se ainda avaliar as vizinhanças dos ótimos localizados, apostando na possibilidade de pontos sub-ótimos estarem melhores adaptados que algum ótimo local. Esta maneira de abordar o problema aumenta muito a complexidade do processo de pesquisa. Um esquema alternativo é usar somente os critérios mais importantes na construção da função de adaptação e fazer com que um algoritmo conhecido forneça várias soluções alternativas. Entre as respostas o decisor pode então escolher o projeto que mais lhe convier.

No campo dos algoritmos genéticos, somente uma modesta quantia de pesquisa vem sendo dedicada sobre o tópico da localização de todos os máximos de funções multimodais. Neste sentido o trabalho apresenta uma técnica iterativa baseada na resolução de um problema multimodal, determinando um máximo a cada iteração. Esta técnica será demonstrada no contexto dos Algoritmos Genéticos tradicionais (doravante tratados como AGs). Entretanto, é uma técnica que pode ser extendida a outros métodos de otimização baseada puramente na função de adaptação (doravante denominada de *fitness*).

1.2. Objetivo

O trabalho objetiva: descrever, implementar e avaliar um algoritmo aqui denominado de **TÉCNICA DO COMPARTILHAMENTO SUCESSIVO**. O termo "COMPARTILHAMENTO" justifica-se pelo uso da idéia de nichos biológicos, que de forma forma semelhante, ao encontrado na natureza, são utilizados aqui juntamente aos AGs. Para evitar a criação de um termo técnico novo: "nicheamento", optou-se em utilizar "compartilhamento", que aliás descreve muito bem o que ocorre algoritmicamente. O termo SUCESSIVO advém das repetições que a técnica executa nos AGs e que usando o conhecimento ganho,

durante as iterações, garantidamente, evitará a "re-pesquisa" em iterações subseqüentes, em regiões do espaço do problema onde soluções já foram obtidas.

A pesquisa é efetuada aplicando em cada rodada do AG uma função de adaptação (*fitness*) transformada, em relação a original, fazendo com que ótimos (máximos) já visitados não sejam alvo de nova pesquisa.

Tomou-se como hipótese central do trabalho que a probabilidade de descobrir uma "nova" solução em cada iteração fosse sensivelmente incrementada, e que sua eficiência pudesse ser avaliada e comprovada, em testes com funções de comportamento "patogênico", ou seja, em funções cuja configuração espacial dificulta a localização dos máximos através de métodos tradicionais da programação matemática.

1.3. Recursos Empregados

Os AGs, que são o tema central desta tese, tem despertado a curiosidade de um grande número de pesquisadores que se tem dedicado a uma imensa gama de variações do esquema básico fazendo variar o tamanho da população, o processo de seleção dos pais, modificações no processo de criação de novas direções de busca, realizando pequenas perturbações aleatórias (mutações), usando outras representações cromossômicas, etc.

Por outro lado, certo número de pesquisadores tem reconhecido que para alguns problemas existem heurísticas rápidas de busca iterativa (pesquisa local). Assim, optaram por adaptar para esses problemas um método de busca local, baseado em metaheurísticas como a Pesquisa Tabu e/ou Têmpera Simulada, com métodos baseados no uso de populações e a utilização de processos de recombinação como a dos AGs. Este tipo de metaheurística tem sido classificada

pela denominação de Algoritmos Meméticos (AMs), como se lê em correspondência eletrônica recibida pelo autor da tese do Prof. Pablo Moscato:

> *"...você poderia dizer que os algoritmos meméticos são híbridos, que usam uma busca com recombinações de soluções de outros algoritmos (heurísticas, métodos exatos truncados, etc) para a busca local."*

Desta forma, o trabalho que se apresenta é classificado como um AM, por representar uma extensão das técnicas unimodais já existentes.

Fora sua classificação, a técnica descrita na seção 5.1 servirá de complemento, que pode ser usado com qualquer tipo de AG, ou em outras técnicas de otimização, como por exemplo a Têmpera Simulada. Finalizando, a técnica expõe o comportamento topográfico da função de adaptação, cuja configuração espacial se modificará após cada processo iterativo realizado.

1.4. Estrutura do Trabalho

O capítulo 1 - **Introdução** - apresenta o trabalho, sua origem, seu contexto, seu objetivo, sua importância e os recursos empregados para chegar aos resultados a qual se propõe.

O Capítulo 2 - **As Técnicas Heurísticas de Otimização** - apresenta um relato sobre o estado atual das pesquisas em técnicas heurísticas de otimização, sobre a qual se fundamenta o trabalho. São discutidas, ainda, novas metaheurísticas, a saber: têmpera simulada, pesquisa tabu, GRASP e algoritmos genéticos.

No capítulo 3 - **Algoritmos Genéticos** - são apresentados os fundamentos dos AGs, englobando tópicos pelos quais o trabalho de pesquisa da tese passou. Discutem-se os operadores e técnicas avançadas em pesquisa genética

atualmente conhecidas e usadas; e que serviram de ponto de referência, para a elaboração da técnica de resolução para o problema.

O capítulo 4 - **Colocação do Problema** - discute os trabalhos de otimização de funções multimodais desenvolvidos por Deb, Goldberg, Richardson, Grosso, Davidor e Jolley, os quais antecederam o autor deste trabalho com tratamentos similares ao dado nesta obra, e que serviram de inspiração direta na criação, desenvolvimento e avaliação da Técnica do Compartilhamento Sucessivo.

O capítulo 5 - **Técnica do Compartilhamento Sucessivo** - é o capítulo central da tese, apresentando o algoritmo que é tema da tese, justificando seu uso em parceria com os AGs. Se discute o uso de funções auxiliares que promovem a destruição de nichos ecológicos numa biocenose (comunidade).

Se destacarão neste momento os passos de resolução, que serão dados pela escolha de uma função de adaptação e uma métrica. Se descreverá a inicialização do algoritmo, que tratará em equiparar a função de adaptação modificada, em cada iteração, com a função de adequação original; a execução, que utilizará um AG sobre as funções de adequação modificadas, mantendo os registros dos melhores resultados no percurso; a atualização da função de adequação criando depressões na região próxima do melhor indivíduo; controle, que mede soluções obtidas com iterações anteriores, verifica valores limítrofes e apresenta soluções; e o critério de parada, que verificará se o número de soluções procuradas já foram encontradas.

O Capítulo 6 - **O Campo de Provas** - relata as funções que serão usadas para testes, descrevendo sua configuração espacial, justificando a sua escolha.

No Capítulo 7 - **Os Resultados** - é feita uma discussão sobre a performance da Técnica do Compartilhamento Sucessivo, apresentando seus resultados.

O Capítulo 8 - **Discussão e Conclusão** - faz uma avaliação geral do trabalho, relata suas vantagens e desvantagens no uso da técnica, descrevendo seu tempo computacional e apresentando novos desafios para novos trabalhos na área.

CAPÍTULO 2

2. AS TÉCNICAS HEURÍSTICAS DE OTIMIZAÇÃO

Em problemas de decisão, que normalmente surgem em uma unidade empresarial, geralmente existem uma série de recursos escassos, como: mão-de-obra, matéria-prima, tempo ou ainda requisitos mínimos que devem ser cumpridos, nesse caso: produção necessária, horas de descanso, etc., que condicionam a eleição da estratégia mais adequada. Como em geral o objetivo ao tomar uma decisão consiste em levar a cabo o plano proposto de uma maneira ótima, incorrendo nos mínimos custos e/ou maximizando lucros, estes problemas podem ser colocados na seguinte forma matemática:

Otimizar f(x)

Sujeito as restrições: $h_i(\mathbf{x}) \leq b_i$, i=1,...,l

$h_i(\mathbf{x}) \geq b_i$, i=l+1,...,m

$h_i(\mathbf{x}) = b_i$, i=m+1,...,n

Não causa espanto o fato de que a resolução destes tipos de problemas tenham atraído a atenção de numerosos pesquisadores, principalmente depois da II Guerra Mundial com o surgimento da Pesquisa Operacional [LEN82]. No caso particular em que todas as funções, ou seja, a função objetivo e as restrições são lineares na forma $h_i(\mathbf{x}) \leq a_{ij}x_j$; com a_{ij}, $x_j \in \Re$ fez surgir uma nova disciplina, a programação linear, e um algoritmo eficiente para a sua resolução, o algoritmo Simplex [LIE78]. Da mesma maneira, inclusive neste caso, quando são muitas variáveis que intervém no modelo, como ocorre em numerosas ocasiões, e também do tipo de problema, o algoritmo Simplex pode apresentar problemas de tempo de

execução, fazendo com que o tempo computacional seja excessivamente grande, mesmo para computadores de grande porte.

Existe um tipo concreto de problemas de otimização que se mostram especialmente interessantes, denominados de problemas de otimização combinatória. Neles as variáveis de decisão são inteiras e, geralmente, o espaço de soluções está formado por ordenações ou subconjuntos dos números naturais. Os problemas combinatoriais mais famosos são o problema da mochila (*Knapsack Problem*) e do caixeiro viajante (**T***ravelling* **S***alesman* **P***roblem* - TSP) [CAM94]. O primeiro problema consiste em selecionar dentre um conjunto de n produtos, cada um com valor c_i e um volume v_i, aqueles que couberem em um recipiente de volume V, e que juntos tenham o maior valor possível, ou seja, determinar um subconjunto $I^* \subset \{1,...,n\}$ para o qual:

$$\sum_{I^*} c_i = \max_{I \subset \{1,...,n\}} \sum_{I} c_i$$

com a restrição

$$\sum_{I^*} v_i \leq V$$

O problema do caixeiro viajante, apesar de ser simples de construir, vem trazendo dificuldades aos pesquisadores, dado a complexidade em que resulta sua resolução. O problema trata em determinar, dado um mapa de vias (grafo), a ordem em que devem ser visitadas n cidades (nós), de forma que partindo de uma cidade (nó) qualquer, se percorra a menor distância e retorne à cidade (nó) de partida, depois de visitar uma única vez cada uma delas. Neste caso, em lugar de determinar um subconjunto de elementos, trata-se de eleger uma permutação das n cidades, de forma que as distâncias percorridas sejam mínimas.

Em muitas ocasiões tem-se tratado estes tipos de problemas combinatoriais formulando-os como modelos de Programação Linear Inteira (PLI), onde a

pertinência ou não de uma variável ao subconjunto pesquisado se representa através de variáveis binárias inteiras **0-1**.

No caso do problema da mochila, se a variável x_i toma o valor **1** então o item deve ser introduzido na mochila, e **0** em caso contrário. A formulação fica agora:

$$\max \sum_{i=1}^{n} c_i x_i$$

$$\text{sujeito as restrições: } \sum_{i=1}^{n} v_i x_i \leq V$$

$$x_i \in \{0,1\}, \forall i \in \{1,\ldots,n\}$$

2.1. Complexidade Computacional

Em problemas do tipo combinatório existe sempre um procedimento elementar para determinar a solução ótima procurada: realizar uma exploração exaustiva de conjuntos de soluções possíveis, ou seja, que satisfaçam as restrições, calcular para cada um o custo associado e eleger finalmente aquela rota que resulte na melhor resposta, no caso, o menor custo. De mesmo modo, ocorre algo similar ao que ocorria com o método simplex, sendo que este método, teoricamente, leva sempre para a solução ótima procurada, porém não eficientemente, pois o tempo computacional cresce exponencialmente com o número de itens do problema. Existem problemas combinatoriais para os quais não se conhecem algoritmos de resolução e quando uma solução é criada, ocorre uma explosão no tempo de cálculo. São problemas computacionalmente difíceis de tratar. Por outro lado, para alguns problemas combinatoriais existem algoritmos cujo tempo computacional é polinomial, por exemplo o problema da designação que consiste em empregar n trabalhadores com n postos de trabalho, sabendo o custo c_{ij} de empregar o trabalhador i no posto j. Na prática para o problema da

designação, o algoritmo Húngaro necessita um tempo de cálculo que cresce simplesmente segundo $n^{2.5}$ [CHR75].

Matematicamente se tratou de caracterizar um tipo e outro de problemas. Para aqueles que se conhece algoritmos e que necessitam tempo computacional polinomial para oferecer a solução ótima[1] se diz que pertencem a classe **P** (polinomial), e consideram-se que são de resolução eficiente. Por outro lado, a maioria dos principais problemas de otimização que aparecem na gestão empresarial e engenharia pertencem a outra, a denominada **NP** (*non determininistic polinomial algorithm*), na qual estão incluídos aqueles problemas para os quais não se conhece um algoritmo polinomial de resolução, ainda que seja possível, dada uma solução, comprovar em tempo polinomial se seu custo é melhor que um determinado valor.

Somente os problemas da classe P são algoritmicamente resolúveis eficientemente, e que $\mathbf{P} \subset \mathbf{NP}$. Se o caso contrário também ocoresse, $\mathbf{P} \supset \mathbf{NP}$, poder-se-ia dizer que para a maioria dos problemas de interesse existem algoritmos eficientes de resolução. Por outro lado, o fato é que até hoje nada se demonstrou: nem que $\mathbf{P}=\mathbf{NP}$ seja certa, nem mesmo que haja problemas da classe **NP** que estão em **P**. Esta é uma das mais importantes questões que atualmente encontra-se aberta na matemática.

Em 1971, Cook demonstrou que há problemas **NP** que são "especialmente difíceis" [COOK71]. São denominados de **NP**-completos. Estes problemas, dentre os quais incluem a maioria dos problemas de interesse da gestão empresarial [LEN82], são aqueles que ademais são **NP**-árduos, ou seja tem a pecularíedade de que todos os problemas da classe **NP** podem ser reduzidos polinomialmente a

---

[1] Rigorosamente as classes P e NP estão compostas não por problemas de otimização, mas sim por seus correspondentes "problemas de decisão", que consistem em determinar se para o problema de otimização existe, ou não, uma solução cujo custo melhora um certo valor.

eles, ou que, caso se possa dar uma solução em tempo polinomial para um deles se poderia dar para todos os da classe **NP** e portanto **P=NP**.

Este fato de que nunca ninguém conseguiu encontrar algoritmos eficientes para os problemas **NP**-completos leva os pesquisadores a pensar que uma vez se demonstre que um problema pertence a essa classe, já não "vale a pena" tratar de buscar algoritmos eficientes para ele [LEW78]. Como se comentou, lamentavelmente, muitos problemas reais que aparecem na pesquisa operacional são **NP**-completos.

2.2. Heurísticas

Dada a dificuldade prática para resolver de forma exata (simplex, *branch-and-bound*, teoria dos grafos, etc.) toda uma série de importantes problemas combinatoriais, para os quais é necessário oferecer alguma solução dado seu interesse prático, começaram a surgir algoritmos que proporcionam soluções viáveis. Estas soluções satisfazem as restrições do problema, os quais, ainda que não otimizem a função objetivo, se supõe que ao menos se aproximam do ótimo em um tempo de cálculo razoável. Pode-se chama-las de "satisfatórias", pois são suficientemente boas.

A este tipo de algoritmo se denomina de heurística, palavra que vem do grego: *heuriskein*. Uma tradução à palavra poderia ser: encontrar, pois como aponta Reeves [REE93]], são mais exatos, em princípio a que fazem os algoritmos que "buscam". Ainda que as heurísticas, em um primeiro momento, não foram bem vistas nos círculos acadêmicos acusados de escasso rigor matemático [EIL77], seu interesse prático como ferramenta útil, que dão soluções a problemas reais, foram abrindo espaço pouco-a-pouco, sobretudo a partir da

metade dos anos setenta, com a proliferação de resultados no campo da complexidade computacional.

Uma maneira possível de definir estes métodos é: "procedimentos simples, embasados no sentido comum, que se supõe que ofereçam uma boa solução (ainda que não necessariamente ótima) a problemas difíceis, de um modo fácil e rápido"[ZAN80].

São vários os fatores que tornem interessante a utilização de algoritmos heurísticos para a resolução de um problema:

a) quando não existe um método exato de resolução, ou que este requer muito tempo computacional (ou memória). Oferecer então uma solução que somente seja aceitavelmente boa, resulta interessante frente a alternativa de não se obter nenhuma solução em absoluto;
b) quando não se necessita a solução ótima. Se os valores que adquire a função objetivo são pequenos, pode "não-valer-a-pena" esforçar-se (custo em tempo e dinheiro) em obter uma solução ótima que, por outro lado, não representará um benefício importante em relação a outra apenas sub-ótima. Neste sentido, caso se possa oferecer uma solução melhor do que a atualmente disponível, isto por ser já de interesse suficiente em muitos casos;
c) quando os dados são pouco confiáveis. Neste caso, quando o modelo é uma simplificação da realidade, pode não ser imperativo o interesse na busca de uma solução exata, dado que por si esta não será mais que uma nova aproximação da realidade, embasando-se em dados que não são reais;
d) quando limitações de tempo, espaço para armazenamento de dados, entre outros, obriguem o emprego de métodos de resposta rápida, ainda que cause dano à precisão;
e) como passo intermediário na aplicação de outro algoritmo. Às vezes são usadas soluções heurísticas como ponto de partida de algoritmos exatos de tipo iterativo (a regra de Vogel, por exemplo, dentro do método do

transporte, não é mais que um procedimento inteligente que aponta uma solução inicial).

Uma importante vantagem que apresentam as heurísticas em relação às técnicas que buscam soluções exatas é que, em geral, permitem uma maior flexibilidade para o manejo das características do problema. Ademais, geralmente oferecem mais de uma solução, o que permite ampliar as possibilidades de decisão daquele que decide, sobretudo quando existem fatores não quantificáveis que não podem ser anexados ao modelo, porém que também devem ser considerados.

Por outro lado, resulta ser mais fácil entender (por parte de diretores de empresas e pessoas não especialistas em formulação) a fundamentação das heurísticas do que os complexos métodos matemáticos, os quais utilizam a maioria das técnicas exatas.

Os métodos heurísticos também apresentam inconvenientes. Um deles é que, geralmente não é possível conhecer a qualidade da solução, ou seja, quão próxima está do ótimo $\mathbf{x}^*$ a solução $\mathbf{x}_h$ que é oferecida. Se por exemplo, o problema é de maximização, a única coisa que se sabe é que $\mathbf{x}_h \leq \mathbf{x}^*$.

Felizmente, existem métodos para realizar cortes que dão uma orientação a respeito da qualidade da solução obtida. Um procedimento consiste em relaxar o problema (por exemplo eliminando alguma das restrições, ou efetuando a relaxação Lagrangeana) de modo que assim o problema seja mais fácil de resolver. Se o ótimo do problema relaxado é $\mathbf{x}'$, sabendo-se que $\mathbf{x}_h \leq \mathbf{x}^*$, $\mathbf{x}^* \leq \mathbf{x}'$, já que eliminar restrições aumenta o conjunto de soluções, e pode então surgir um ótimo melhor que o original. Deste modo, valores $\mathbf{x}'$ próximos a $\mathbf{x}_h$ garantem que a heurística está dando uma boa aproximação.

Quando estes tipos de procedimentos de avaliação da heurística não são possíveis, cabe utilizar métodos simples que detectam simplesmente que a

heurística não é boa. Assim, poder-se-ia gerar aleatoriamente várias soluções e se forem similares a x_h, cabe colocar em dúvida a eficiência da heurística.

Não obstante, apesar de todas as vantagens, não há dúvida de que quando uma técnica exata está disponível ele deve ser preferida a qualquer tipo de heurística, sobretudo quando os montantes econômicos manejados são importantes e, portanto, pequenas variações em relação ao ótimo representam quantidades consideráveis de unidades monetárias.

2.2.1.Tipos de Heurísticas

Existem diferentes tipos de heurísticas (muitas vezes mistas), segundo o modo em que buscam e constroem suas soluções. Uma possível classificação segundo Silver, Vidal e Werrá [SIL80] é apresentada a seguir:

a) métodos construtivos: consistem em ir paulatinamente anexando componentes individuais para a solução até que se obtenha uma solução viável. O mais popular destes métodos constituem os algoritmos GRASP (*greedy randomized adaptive search procedure*) que se comportam de forma gulosa e devoradora. Estes algoritmos constroem passo a passo a solução, buscando o máximo benefício em cada passo;

b) métodos de decomposição: se trata de dividir o problema em subproblemas menores, sendo a saída de um a entrada de seu seguinte, de forma que ao se resolver todos, obtenha-se uma solução para o problema global. Um exemplo de aplicação deste método em um problema de programação linear, consiste em decidir de alguma forma (podendo ser mediante outra heurística) uma solução para as variáveis inteiras, e logo resolver o problema de programação linear obtido ao substituir essas variáveis por seu valor. Alguns autores [BAL81] diferenciam entre métodos de decomposição (quando os subproblemas se

resolvem em cascata) e métodos de partição (quando os subproblemas são independentes entre si);

c) métodos de redução: tratam de identificar alguma característica que presumivelmente deva possuir a solução ótima e desse modo simplificar o problema. Assim, pode detectar-se que alguma variável deva tomar sempre valor zero, quais outras estão correlacionadas, e assim por diante.

d) manipulação do modelo: estas heurísticas modificam a estrutura do modelo com o objetivo de de torná-lo mais simples de resolver, deduzindo, a partir de sua solução, a solução do problema original. Podem consistir em reduzir o espaço de soluções, linearizar funções que são não-lineares, agrupar variáveis para reduzir seu número, impor novas restrições baseando-se no comportamento esperado que deveria ter a solução ótima), ou inclusive aumentá-la (eliminando restrições do problema). O presente trabalho se faz valer deste tipo de procedimento.

e) métodos de pesquisa (busca) em vizinhança: dentro desta última categoria e onde se encontra a maioria das metaheurísticas nas quais o trabalho está interessado e que posteriormente serão analisados com mais profundidade. Estes métodos partem de uma solução viável inicial (obtida, talvez até, por outra heurística) e mediante alterações dessa solução, vão passando de forma iterativa a outras soluções factíveis de sua vizinhança (até que se cumpra determinado critério de parada) apresentado como ótima a melhor das soluções visitadas.

Um conceito chave na categoria apresentada, é como realizar um passo de uma solução viável a outra. Para isto é interessante definir o espaço de vizinhança V(s) da solução s, ou seja, o conjunto de soluções "parecidas" a ela. O significado aqui "parecido" deve ser entendido como a possibilidade de se obter uma solução s e V(s) a partir de s, realizando somente uma operação elementar, chamada movimento sobre s, como por exemplo eliminar ou adicionar um elemento no conjunto de soluções, trocar elementos, etc.).

Estes métodos se baseiam portanto, em buscar dentre os elementos da vizinhança V(s) da solução atual, aquele que tenha um melhor valor de acordo com algum critério pré-definido, mover-se a ele e repetir a operação até que se considere que não é mais possível determinar uma situação melhor, talvez porque não haja nenhum elemento na vizinhança da solução atual, ou ainda porque algum critério de parada se verifique. Uma classe especial dentro deste método é o de "busca local ou de descida". Nele a cada iteração o movimento se produz desde a solução atual a uma de sua vizinhança que seja melhor que ela, finalizando a busca quando todas de sua vizinhança forem piores, ou seja, a solução será sempre um ótimo local.

Um dos maiores inconvenientes que estes métodos enfrentam é a existência de ótimos locais que não são globais. Uma solução s se diz que é ótima local se $\forall s' \in V(s), c(s) \leq c(s')$ (supondo que objetivo seja minimizar e que c(s) representa o custo da solução s). Se durante a busca o método cai em um ótimo local, em princípio a heurística não saberia continuar pois ficaria estacionada nesse ponto. Uma primeira possibilidade para fugir dessa dificuldade poderia ser reiniciar a busca a partir de outra solução inicial e confiar que, nesta tentativa, a exploração siga por outros caminhos. Duas das técnicas que serão estudadas (pesquisa tabu e têmpera simulada), seguem diferentes estratégias para evitar cair em ótimos locais: a primeira mantendo uma memória da rota seguida para identificá-los, e a segunda permitindo soluções piores com certa probabilidade de aceitação.

Outro problema destas estratégias heurísticas está em conseguir chegar a solução ótima independentemente da solução inicial da qual se parta. O ponto inicial tem uma grande influência sobre a possibilidade de cair em um ótimo local.

Outra possível classificação das técnicas heurísticas, baseada em Barr *et al* [BARR95], é apresentada na **Figura 01**, onde o critério fundamental é a regra de parada que se define para um algoritmo.

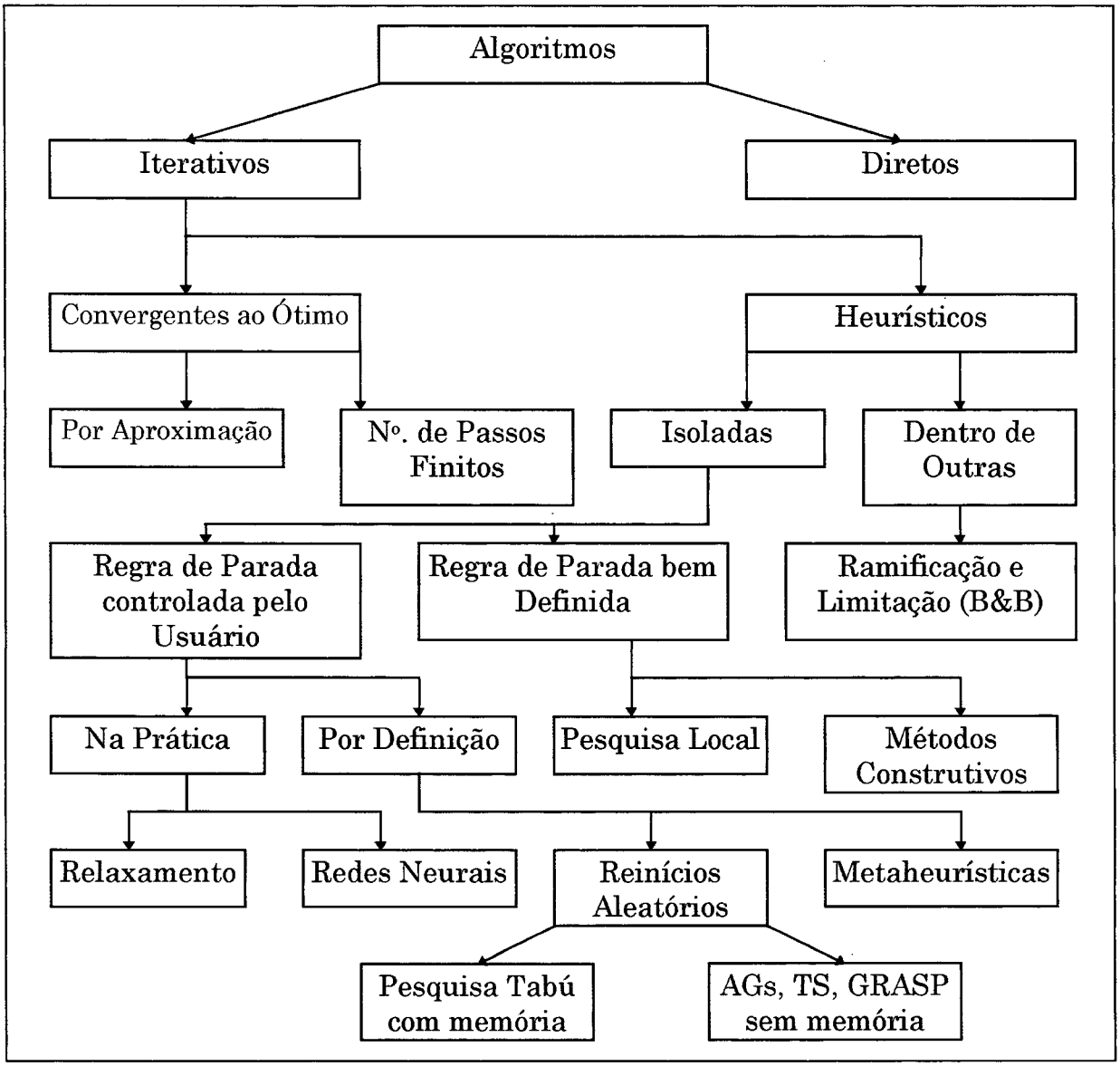


Figura 01- Classificação de técnicas heurísticas proposta em [BARR95]

Apesar de que os métodos heurísticos somente avaliam normalmente um pequeno número de alternativas do número possível, [HOM94] atribui seu êxito a aplicações da regra **80/20** (80% da riqueza está concentrada em 20% da população, neste caso com 20% de esforço se consegue 80% dos resultados). Uma

heurística bem projetada pode se aproveitar esta propriedade e explorar exclusivamente as soluções mais interessantes.

2.3. As Novas Metaheurísticas

2.3.1. Têmpera Simulada

A têmpera simulada faz uso de conceitos inicialmente descritos pela mecânica-estatística. Têmpera, segundo Aarts [AAR90], é "um processo térmico para obter estados de baixa energia em um sólido, mediante um banho térmico". O processo físico da têmpera primeiro eleva a temperatura do sólido à temperatura de fusão, e depois esfria-o lentamente até que as partículas se coloquem, por si mesmas, em seu "estado fundamental". Para cada temperatura durante o processo de têmpera, o sólido pode alcançar o equilíbrio térmico se o resfriamento for produzido muito lentamente. Caso o resfriamento se efetue demasiadamente rápido, o sólido pode chegar a estados metaestáveis em lugar do fundamental, no qual as partículas formam retículas perfeitas e seu sistema está em seu nível energético mais baixo. Enquanto que nas metaestáveis existem defeitos em forma de estruturas de alta energia.

2.3.2. Pesquisa Tabu

A pesquisa tabu é um tipo de busca por vizinhança que, conforme seu mentor: "guia um procedimento de busca local para explorar o espaço de soluções mais além do ótimo local" [GLO94]. Igualmente como na busca local a pesquisa

tabu seleciona de modo agressivo o melhor dos movimentos possíveis em cada passo. Ao contrário da busca local (que sempre se move a pontos melhores na sua vizinhança e finaliza com a chegada no ótimo local), a pesquisa tabu permite mover-se a uma solução de vizinhança mesmo que não seja tão boa como a atual, de modo que possa escapar de ótimos locais e continuar estrategicamente a busca de soluções ainda melhores.

Entretanto, para evitar que o processo volte a um ótimo local e entre em ciclo em sua forma mais primitiva, a pesquisa tabu classifica um determinado número de movimentos recentes, como sendo "movimentos tabus", os quais não serão possíveis serem repetidos por um determinado horizonte temporal. Nesse caso, o escape dos ótimos locais se produz de forma sistemática e não aleatória, ou seja, a pesquisa tabu mantém uma memória de eventos, ou outro elemento próprio das soluções de interesse visitadas em relação ao passado. Posteriormente usa a memória para alterar a vizinhança da solução atual, influindo no processo seguinte de busca. Atualmente o tema central do método consiste em tratar de explorar a memória adaptativa para assim controlar os processos de busca [GLO94].

As estruturas de memória podem ser de vários tipos. Uma solução pode ser armazenada de modo completo (memória explícita), ou somente parcialmente, guardando informações acerca de certos atributos das soluções ao passar de uma solução a outra (memória atributiva). Se trata de selecionar de modo estratégico não somente eventos de interesse para memorizar, senão também para esquecer, considerando não somente a qualidade das soluções mas também a possível influência desses atributos em relação à viabilidade ou estrutura das soluções.

A memória pode conter informação a respeito da freqüência e "recência" dos eventos. Os eventos ocorridos mais recentemente, se armazenam na memória da recência (curto prazo). Um subconjunto dos movimentos contidos na memória são classificados como tabus. Os elementos deste subconjunto serão excluídos durante um certo período de tempo até existirem somente alternativas viáveis.

Os elementos que não são excluídos devem possuir atrativos que aumentam a qualidade das soluções.

Um conceito relacionado é o uso de estratégias de listas de candidatos, as quais são usadas para melhorar e atualizar os melhores movimentos. Para grandes problemas, estratégias de listas de candidatos podem acelerar o processo significativamente.

Nas considerações a longo prazo, é de utilidade a memória baseada em freqüências, a qual contém informações relacionadas com o tempo que certos atributos pertenceram a soluções visitadas na pesquisa. Esse tipo de dado é básico para definir as estratégias de intensificação e diversificação [GLO94].

Em geral, podem ser mantidos algumas das melhores soluções encontradas. Quando decresce até um determinado ponto o ritmo ao qual se estão encontrando novas soluções melhores até então achadas, entram em jogo as estratégias de intensificação e diversificação. As de intensificação enfatizam atributos comuns às boas soluções achadas até esse instante. Neste caso, essas soluções podem ser elementos de um bloco regional já exploradas para proceder a uma busca mais intensiva naquela área. Em contrário, estratégias de diversificação consideram atributos pouco usados no passado, com o objetivo de dirigir a busca para novas regiões.

Estratégias de intensificação e diversificação podem integrar-se (*path relinking*) mediante a geração de soluções obtidas ao explorar as trajetórias que conectam as boas soluções. Uma estratégia de intensificação gera soluções obtidas a partir de soluções similares, enquanto uma diversificação utiliza soluções que são diferentes entre si.

Outro dos aspectos mais importantes da pesquisa tabu é a oscilação estratégica, a qual guia os movimentos até que se chegue a um limite que geralmente representa o ponto onde o método para. Em lugar de parar, o

procedimento permite cruzar ligeiramente esse limite modificando tanto a definição de vizinhança como o critério de avaliação voltando logo atrás e cruzando o limite desde a direção oposta. A repetição deste processo de cruzamento do limite proporciona uma marca adequada para combinar estratégias de intensificação e diversificação.

A pesquisa tabu, ao contrário dos métodos: têmpera simulada e algoritmos genéticos, põe sua ênfase nos procedimentos determinísticos em local de aleatórios, ou seja, pretende usar a inteligência e não a "força bruta". Ainda existe uma variante do método que usa regras probabilísticas para selecionar o melhor movimento em cada iteração [GLO94]. A filosofia da pesquisa tabu se baseia na crença de que a aceitação de uma má estratégia sistemática de busca é mais frutífera que a escolha de uma boa de tipo aleatório. Portanto, a busca tabu faz uso de estruturas especiais de memória e de estratégias de busca dinâmica cuidadosamente projetadas para guiar o processo de otimização de modo eficiente.

2.3.3. GRASP

GRASP é a mais recente das técnicas entre as metaheurísticas que aqui se apresentam. Foi desenvolvida originalmente por Feo e Resende [FEO89] ao estudar um problema de cobertura de alta complexidade combinatória. Cada iteração em GRASP (acrônomo de *Greedy Randomized Adaptive Search*) conta geralmente de dois passos: a fase de construção e o procedimento de busca local. No primeiro se constrói uma tentativa de solução, que logo é melhorada mediante um procedimento de intercâmbio até que se chegue a um ótimo local.

Na fase de construção, GRASP costuma manter uma lista de candidatos formada por elementos de alta qualidade. A solução inicial se constrói iterativamente considerando um elemento de cada vez.

Em cada iteração do procedimento construtivo, um elemento é eleito de forma aleatória da lista de candidatos para agregá-lo à lista como parte da solução que se constrói. A adição de um elemento à lista de candidatos se determina mediante uma junção de tipo devorador, enquanto a seleção de um elemento dessa lista de candidatos depende dos que foram eleitos previamente, ou seja é devorador, aleatório e adaptativo [KLI92].

Durante a utilização do algoritmo GRASP, se dá um equilíbrio computacional entre as fases de construção e de melhora dentro do processo global. Baseando-se na construção eficiente de uma solução inicial, pode conseguir-se uma redução no número de passos necessários para alcançar o ótimo local na fase de melhora. Sabe-se(e os resultados empíricos parecem demonstrá-lo) que o tempo total de cálculo em cada iteração GRASP pode reduzir-se fazendo bons projetos de procedimentos de construção [LAG94].

2.3.4. Algoritmos Genéticos

Os algoritmos genéticos são técnicas de busca baseada na mecânica da seleção natural e da genética [GOL87]. Sua estrutura é projetada baseando-se numa tentativa de abstração artificial do "algoritmo" de seleção da própria natureza, com o objetivo de que assim se consiga êxitos similares à capacidade de adaptação a um amplo número de ambientes diferentes, ou seja, os algoritmos genéticos são flexíveis e de âmbito geral, podendo ser aplicados em um número amplo de problemas.

Nos organismos biológicos, a informação hereditária é passada através dos cromossomos que contém a informação de todos os fatores hereditários, ou seja, os genes. Os genes por sua vez, estão compostos por um determinado número de valores (alelos). Vários organismos se agrupam formando uma população, e aqueles que melhor se adaptam, são os que mais possuem probabilidades de sobreviver e reproduzir-se. Alguns dos sobreviventes são selecionados pela natureza para serem cruzados e assim produzir uma nova geração de organismos. Esporadicamente, os genes de um cromossomo podem sofrer ligeiras mudanças (*mutações*). Ocorre ainda, a troca de alelos entre os genes, na tentativa de produzir invíduos melhores (*crossing-over*).

Com isto, os conceitos básicos dos algoritmos genéticos já podem ser apresentados. Em geral, os alelos são valores binários (0-1), representando valores das variáveis de decisão, que corresponderão com os genes. Os cromossomos representam, pois, a solução. Os cromossomos, por sua vez, não são mais que tiras de *bits*, e em sua forma mais simples, um organismo por um simples cromossomo.

As três operações básicas antes mencionadas (reprodução, *croossing-over* e mutação) vem sendo freqüentemente utilizadas neste tipo de algoritmos. Um algoritmo genético simples pode aplicar operadores de reprodução, cruzamento e mutação, a uma população e gerar novos organismos de modo iterativo. O processo de reprodução se guia através de uma função que mede o grau de adaptação (*fitness*) do cromossomo, sendo o processo de seleção do cromossomo por reproduzir dependente dos valores que essa função lhe designa: a valores maiores, maior probabilidade de sobrevivência. Os membros da nova geração são novamente emparelhados aleatoriamente.

Mediante a operação de cruzamento há o intercâmbio genético entre os pares. Um modo simples de fazer consiste em dividir cada uma das tiras de *bits* em duas (ou mais) subtiras e trocar os valores das subtiras entre si, obtendo

assim novos cromossomos de uma nova geração. As mutações consistem em alterar um (ou mais) *bit* de um cromossomo (de 0 para 1, ou vice-versa), com uma probalidade relativamente pequena. Todo esse processo se repete até que algum critério de parada seja atingido.

Os algoritmos genéticos, em suas versões mais clássicas, manipulam a informação em tiras de *bits*. A maioria das outras metaheurísticas geram uma solução simples a cada iteração. Entretanto os algoritmos genéticos trabalham com populações de soluções, gerando uma nova população a cada iteração (geração). Em geral são algoritmos neutros ao contexto, e não consideram nem exploram a informação do problema dentro de processo de busca. A geração de soluções, é pois, aleatória por natureza. Os algoritmos genéticos utilizam o sorteio para guiar a busca, repetindo durante um número relativamente grande de iterações operações bastante simples, que dão lugar a um grande volume de soluções.

CAPÍTULO 3

3. ALGORITMOS GENÉTICOS

3.1. A Teoria dos Algoritmos Genéticos

O termo genética, derivado do grego *gênesis* (geração, produção) foi introduzido em 1905 pelo biólogo inglês Willian Bateson para designar o ramo da biologia que estuda a hereditariedade e a variação dos organismos, apresentadas biologicamente. É uma ciência multidiciplinar pois tem vínculos estreitos com várias áreas do conhecimento como a química, a física, a matemática e a estatística [LIN80].

Na natureza, a evolução, em particular dos seres vivos, tem algumas características que motivaram John Holland a começar uma linha de pesquisa na área que se transformou no que se denomina Algoritmos Genéticos (AG). A habilidade de uma população de cromossomos em: (i) explorar um espaço viável em paralelo; (ii) se combinar com o melhores da população mediante o mecanismo da reprodução; (iii) trocar material genético usando a permutação (*crossing-over*), é algo intrínseco à evolução natural e é explorada pelos AGs.

Os algoritmos desenvolvidos inicialmente por Holland eram simples, gerando soluções satisfatórias a problemas que eram considerados como "difíceis" naquela época. No campo dos AGs há evoluções desde então, principalmente devido as inovações introduzidas na década de 80. Os AGs tem sido incorporados

a mecanismos cada vez mais elaborados, dando motivação a necessidade de resoluções, aproximadas, de problemas práticos de uma ampla variedade.

As características da evolução nos AGs, são descritas por Davis [DAV91], da seguinte maneira:

- A evolução é um processo que opera nos cromossomos no lugar dos seres vivos que eles codificam;
- os processos de seleção natural provocam que aqueles cromossomos que codificam estruturas com êxito se reproduzam mais freqüentemente do que aqueles que não o são;
- As mutações podem fazer com que os cromossomos dos filhos sejam diferentes que os dos pais, e os processos de recombinação podem criar cromossomos bastante diferentes nos filhos pela combinação de material genético dos cromossomos dos pais.
- A evolução biológica não tem memória.

A premissa dos AGs, está na publicação do livro de Holland em 1975 [HOL75]: "Adaptation in Natural and Artificial Systems" e no grande número de pesquisas que vem sendo desenvolvidas por aqueles investigadores que os utilizam como metaheurística para otimização, e que podem encontrar soluções aproximadas a problemas de grande complexidade computacional mediante um processo de "evolução simulada", em particular como um algoritmo matemático implementado em um computador. Devido ao trabalho original de Holland houve a denominação de cromossomos, que inicialmente ocorrera em forma de um algoritmo que manejava *strings* binárias, já que eles representavam um ponto no espaço de configurações do problema.

Como ocorre na evolução biológica, a evolução simulada é projetada para encontrar cromossomos cada vez melhores, mediante uma manipulação "cega" de seus componentes. O termo "cega" se refere ao trecho em que o processo não possui nenhuma informação sobre o problema do qual está tratando de resolver,

excetuando o valor da função objetivo. Na concepção original dos AGs, a função objetivo é a única informação para se avaliar o valor de um cromossomo. Por outro lado esta metaheurística esta baseada em integrar e implementar eficientemente as idéias fundamentais: (i) a facilidade de representação para codificar configurações do problema de otimização, e (ii) o poder de transformações simples para modificar e melhorar as configurações. As melhores são guiadas mediante um mecanismo de controle que permite a uma população de cromossomos evoluir com as probabilidades que os seres vivos tem. Um algoritmo genético pode ser visto como uma estrutura de controle que organiza e dirige um conjunto de transformações e operações, projetadas para simular os processos da evolução natural.

Na evolução dos seres vivos, um dos problemas que cada indivíduo encontra, cotidianamente, é a sua sobrevivência. Eles contam com habilidades natas, previstas em seu material genético. A nível dos genes, o problema é buscar aquelas adaptações benéficas num ambiente hostil. Devido em parte a seleção natural, cada espécie ganha uma certa quantidade de "conhecimento", o qual é codificado e incorporado na nova formação de seus cromossomos. Esta configuração se vê alterada pelas operações de reprodução. Algumas delas são mutações aleatórias, inversões de partes do cromossomo e do *crossing-over*, que é o intercâmbio do material genético proveniente dos cromossomos dos pais.

Nos AGs, as mutações aleatórias promovem certa variação e ocasionalmente alterações benéficas aos cromossomos. A "inversão" é um mecanismo que altera a situação dos genes nos cromossomos, permitindo que alguns genes, que se coadaptaram exitosamente, se localizem mais próximos em um cromossomo, o que por sua vez permite que se aumente a probabilidade de mover-se em conjunto durante o *crossing-over*. Este último é o mecanismo responsável pelo intercâmbio de material genético entre seus pais, para ser combinado nas estruturas filhas [DAV87].

Evidentemente, é a existência do *crossing-over* que governa a eficiência dos AGs e os destaca nitidamente dos outros tipos de metaheurística. Com ele, as características dos pais podem ser combinadas imediatamente quando se reproduzem. Portanto, a probabilidade desta combinação aumenta quando os pais tem um nível elevado de *fitness*, devido a reprodução com maior freqüência.

3.2. Composição de Um Algoritmo Genético

No intuito de encontrar uma solução ótima a problemas de otimização de grande escala, um algoritmo genético básico funciona da seguinte maneira. Uma população de cromossomos (os pais potenciais de uma nova população) se mantém ao longo de todo processo evolutivo. A cada um deles se relaciona um valor de *fitness* que advém do valor de adaptação na função objetivo. Cada cromossomo representa um ponto no espaço de configurações (espaço de busca) do problema.

Dois cromossomos (originalmente só restringidos a *string* binárias) são selecionados para serem os pais de uma nova solução, mediante o mecanismo de *crossing-over*. Nos algoritmos originais de Holland, um deles era eleito de acordo com seu valor de *fitness* (maior valor de *fitness* $\rightarrow$ maior probabilidade de ser eleito), enquanto o outro pai era eleito aleatoriamente. A operação de *crossing-over* resultava em duas configurações filhas (binárias), e depois se calculava o seu valor de *fitness*. Estas soluções substituiam duas soluções da população eleitas aleatoriamente. Este processo se repetia tantas vezes quanto se necessitava, até que alcançasse algum critério lógico para deter a simulação. Esta é uma breve descrição de um modelo muito simples de evolução que trata de incorporar conceito de sobrevivência e seleção do mais apto, assim como o da reprodução sexuada.

Desde o início da pesquisa em AGs, se tem dedicado grandes esforços a estudar as múltiplas variedades deste esquema básico dos AGs. Independentemente de como possa ser sofisticado o projeto do AG, existem cinco componentes que devem ser incluídos.

- Uma representação, em termo de cromossomos, das configurações do problema;
- Uma maneira de criar as configurações da população inicial;
- Uma função de avaliação, que permita ordenar os cromossomos de acordo com a função objetivo;
- Operadores genéticos que permitam alterar a composição dos novos cromossomos gerados pelos pais durante a reprodução;
- Definir parâmetros que o AG utiliza (tamanho da população, probabilidades associadas ao uso dos operadores genéticos, critério de parada, etc.)

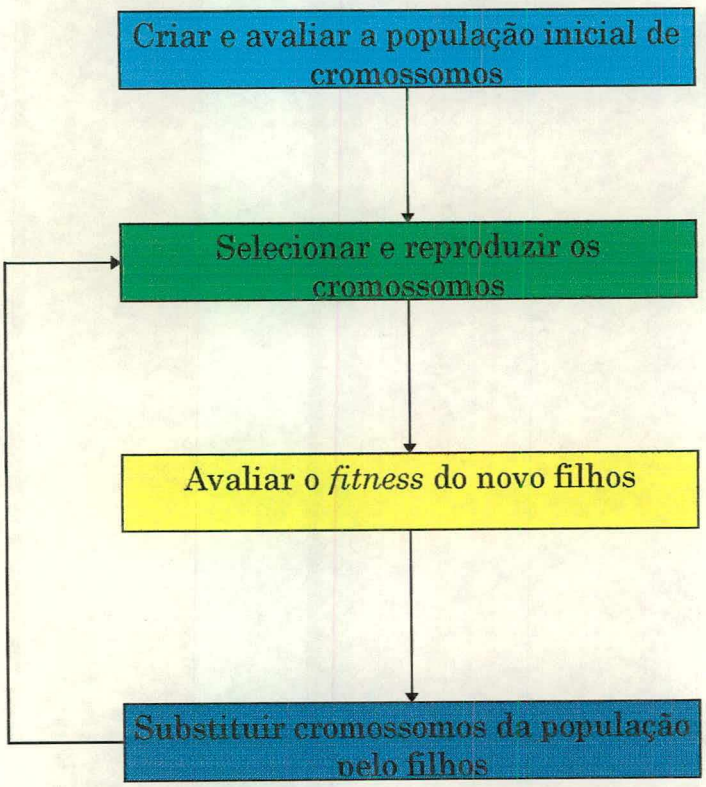


Figura 02. Esquema básico dos Algoritmos Genéticos

3.3. Elementos Básicos

Nos trabalhos de Holland e os realizados por seus seguidores os cromossomos são representados por *strings* binárias (**0-1**). Este tipo de configuração tem sido utilizado em muitos campos distintos.

A representação binária é uma codificação natural para um grande número de problemas, como por exemplo o problema da mochila. Neste problema a *string* 10011 pode representar a configuração: x_1= 1; x_2=0; x_3=0; x_4=1 e x_5=1.

Por motivos técnicos, as representações binárias algumas vezes não são eficientes em problemas práticos. Alguns pesquisadores tem começado a explorar o uso de outras representações [HEIN95], em geral em conexão com aplicações industriais dos AGs. Exemplo de outras representações inclui: *ordered lists* em *bin-packing*, *embbeded lists em prodution scheduling* e *variable-element lists* em *semiconductor layout* [LAG94].

No caso do TSP, uma rota pode ser representada por uma lista ordenada de cidades. Esta representação é mais natural pela natureza do problema, porém requer a criação de operadores genéticos específicos para ele. Entretanto, a seleção de uma representação está fortemente associada como projetos de operadores a utilizar em AG [MOS89], [MOS93].

No contexto da otimização combinatória, a dificuldade em utilizar *strings* binários também está associada com o fato de ter que encontrar uma representação na qual as *substrings* tenham um significado relevante no AG. Algumas das representações que são consideradas mais naturais vem apoiadas na noção do *schemata* que dá uma boa base matemática aos AGs [HOL75]. Alguns pesquisadores afirmam que se obtém bons resultados afastando-se dos

alinhamentos básicos da teoria padrão dos AGs e alguns chegam a firmar que o êxito mesmo, se deve a violação destes princípios.

Independentemente de como estas codificações são entendidas, seu propósito é identificar características similares no espaço de configurações, enquanto se mantém uma conexão mais próxima da representação. Em geral, diferentes codificações dão diferentes perspectivas e uma diferente resolução para investigar o espaço de configurações. Estas considerações estão relacionadas com o conceito de *deceptiveness* (problemas enganosos) [GOL89].

3.4. A População Inicial

A população inicial de um AG pode ser criada de diferentes maneiras. Embora pareça estranho, ao se defrontar com um problema novo, pode-se aprender muito inicializando esta população de maneira aleatória. Fazer evoluir uma população que já foi gerada aleatoriamente até chegar a ter uma bem adaptada (com configurações satisfatórias que são soluções aproximadas do problema de otimização), e um bom teste para saber como está funcionando a implementação do AG, devido as características essenciais e críticas da solução final, devem ser conseqüência deste processo evolutivo e das características dos métodos usados para gerar a população inicial.

Para algumas aplicações, especialmente as da indústria, pode ser conveniente iniciar usando métodos mais diretos. Estas técnicas incluem a perturbação de saída de algum algoritmo guloso (*weighted random initialization*), ou inicialização por perturbação de uma solução gerada por algum perito no problema. Uma técnica que combina as perturbações aleatórias com uma regra de seleção gulosa é denominada GRASP.

3.5. Avaliação do Nível de *Fitness*

A função utilizada para designar os valores do *fitness* dos membros da população, deve ser tal que dê como resultado uma boa discriminação. Este é o motivo pelo qual muitas implementações de AGs usam certo tipo de "normalização", no seguinte sentido: suponha-se que os resultados da evolução de uma função objetivo atinjam um valor de 120 pontos, para um indivíduo "muito bom", enquanto que um valor de 100 pontos represente um indivíduo "ruim". Se estes valores forem utilizados sem alteração como medidas dos valor do *fitness* na fase de reprodução, seria necessário muito tempo para que os descendentes do primeiro indivíduo influam mais do que os do indivíduo "ruim". A normalização deve ser empenhada na importância das melhoras, sem estragar todo material genético relevante presente nesta população.

Se a normalização está influenciada muito fortemente pelo melhor indivíduo, é muito possível que a busca se concentre naquela região do espaço de configurações próxima a ele, perdendo assim parte das vantagens do paralelismo intrínseco dos AGS. Por outro lado, se não há precisão para as melhores soluções, pode-se ter uma busca que é demasiado lenta.

É muito importante ao projetar uma função de avaliação, levar em conta as restrições do problema. A satisfação destas restrições podem ser levadas a cabo impondo penalidades em indivíduos que as violam criando decodificadores da representação, que evitem gerar indivíduos não viáveis. Uma função de avaliação que incorpora um termo de penalização tem a seguinte forma:

fitness=valor_objetivo_normalizado-penalização*medida_de_inviabilidade

Supondo que o melhor indivíduo tenha o maior valor. Se o valor da penalização é alto e o domínio do problema é tal, que é provável a produção de indivíduos não válidos, pode ocorrer então que quando se encontre um indivíduo

válido, este exerça influência sobre os outros e a população convirja para ele, independentemente da busca pelos melhores indivíduos. Isto pode ocorrer porque, possivelmente, os caminhos para melhores indivíduos possíveis requerem a produção de outros indivíduos, não possíveis como passos intermediários, e os valores de penalização tornam impossíveis que estas estruturas intermediárias se façam presentes na população, se diversifiquem e se reproduzam.

Uma maneira de ajustar dinamicamente o valor da penalização é incorporar o conceito de "oscilação estratégica", que foi originalmente concebido como uma maneira de introduzir a diversificação na pesquisa tabu. A forma mais direta é assumir encontrada uma medida de não-viabilidade, a qual quantifica a violação das restrições.

Uma vez avaliado o *fitness* de um indivíduo, se leva a cabo um teste para comparar esse valor com o *fitness* associado ao melhor indivíduo criado durante o processo evolutivo. Se o *fitness* do melhor indivíduo é superior ao do melhor encontrado até esse momento, então é substituído garantindo, assim, que ao término da simulação o melhor indivíduo encontrado estará na memória, apesar de não estar presente necessariamente na população final. A convergência do processo evolutivo em um indivíduo particular é geralmente usada como um critério de finalização do AG.

3.6. Seleção de Operadores Genéticos de *Crossig-over*

O propósito da seleção dos pais é incrementar a probabilidade de reproduzir membros da população que tenham bons valores na função objetivo. Uma maneira popular de visualizar a seleção é denominada *Roulette Wheel* (roleta), uma representação de uma roda que gira em cada indivíduo tem uma seção circular que é diretamente proporcional ao *fitness* do indivíduo [HEIN95].

0.295.709.2
Biblioteca Universitária
UFSC

Um passo de seleção então seria análogo a um giro dessa roleta, selecionando o indivíduo correspondente segundo o arco do círculo onde ele está.

Uma vez que os pais foram eleitos, se utiliza um operador genético como o *crossing-over*. As vezes, a eleição de um mecanismo de seleção adequado permite um tratamento matemático rigoroso como é visto em [SHA94].

O *crossing-over* é um procedimento pelo qual os seres vivos trocam parte de seu material genético para criar um novo organismo. Nos AGs, o *crossing-over* recombina o material dos cromossomos para gerar novos indivíduos. O operador original dos AGs, modelado por analogia com os processos da natureza, é o *one-point-crossing-over* (um ponto de *crossing-over*). Este operador elege aleatoriamente um ponto de ruptura de tal maneira que o material genético ao lado desse ponto é intercambiado entre os pais para criar os filhos.

Com o desenvolvimento no campo dos AGs, surgiram limitações do *one-point-crossing-over*, principalmente no que pode combinar características presentes nos cromossomos. Uma solução tem sido o uso do *two-point-crossing-over* (dois pontos de *crossing-over*). Neste caso, são elegidos dois pontos de ruptura aleatoriamente e é intercambiado o material genético entre ambos. Redin estudou o caso de *n-points-crossing-over* em [RED95].

O mecanismo de *crossing-over* também apresenta suas restrições, o qual levou Ackley [ACK87], a desenvolver a "*crossing-over* uniforme": começando pelo primeiro *bit*, é elegido aleatoriamente um pai para que contribua com seu primeiro *bit* ao primeiro filho, enquanto que o segundo filho recebe o *bit* do segundo pai. Esse processo continua até que todos os *bits* tenham sido atingidos.

Outro processo importante nos AGs é a mutação, apesar de que está usualmente concebida como um operador cujo papel é secundário. No caso binário a mutação consiste em trocar com certa probabilidade, chamada de *mutation*

rate, o valor de um *bit*. Existem basicamente duas maneiras de implementá-lo. A primeira maneira troca o *bit* que o teste de probabilidade permite (se o i-ésimo *bit* vale 1 e passa pelo teste de probabilidade, ou seja, há mutação, a nova *string* conterá o valor 0 na i-ésima posição). Na segunda maneira, gera-se aleatoriamente um novo *bit* para substituir o *bit* que passou pelo teste de probabilidade. Portanto, 50% das vezes o novo *bit* não trocará, e o *mutation rate* será a metade da primeira maneira.

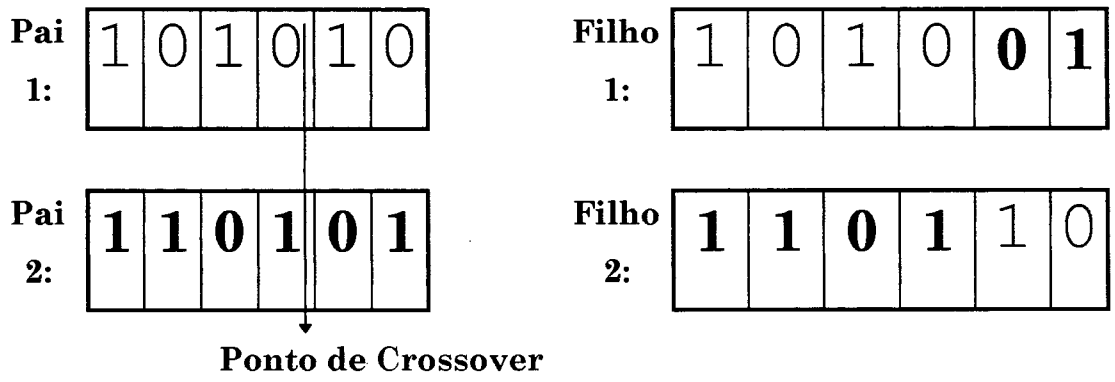


Figura 03. Exemplo de *One-Point-Crossing-Over*

Um operador de *crossing-over* que pode manipular *strings* não binários é o PMX (*Partilly-Matched-Crossing-over*). Dados dois cromossomos_pais, o operador copia uma *substring* de um dos pais diretamente nas mesmas posições do filho. As posições restantes são preenchidas com valores que ainda não tenham sido utilizados na mesma ordem em que se encontram em um dos pais.

3.7. O Teorema do *Schema* e o Paralelismo Intrínseco

A codificação binária foi originalmente usada por Holland para derivar o Teorema do *Schema*, considerado por muitos como uma base matemática no qual se apoiam os AGs. A teoria se baseia no desejo de um melhor entendimento do domínio do problema. É essencial estudar tanto as similaridades entre *strings*

como também de seu *fitness*. De alguma maneira, o interesse se reorienta de *strings* individuais a grupos de *strings* de grande *fitness*. Como Goldberg coloca, a análise de termos do *schemata* oferece uma descrição adequada para analisar perguntas do tipo: como pode uma *string* ser similar a outra? De que maneira uma *string* é um representante de outras classes de *strings* com similaridade em outras posições específicas?

Um *schema* está definido como um modelo (*template)* entre *strings*. Em uma representação binária, um *schema* é uma *string* do tipo ($\mathbf{e}_1$, $\mathbf{e}_2$, $\mathbf{e}_3$, ..., $\mathbf{e}_n$) com $\mathbf{e}_i \in \{0,1,*\}$ onde o símbolo * representa ou 0 ou 1. Um *schema* representa um subespaço de *strings* que tem os mesmos valores nas posições que não tem símbolo * (onde estão especificados). No entanto, o tamanho do subespaço é maior, quanto maior o número de símbolos * haja presentes no *schema*.

O teorema do *schema* troca a visão que se tem sobre uma busca em um espaço de configurações, em uma busca num conjunto de *schemata* os quais são instanciados por uma *string*. Devido a cada *string* ser uma instância de 2^n *schematas* possíveis, a sobrevivência de uma *string* se descobre bastante acerca da informação implícita do *fitness* do *schemata* correspondente. Esta é uma das pedras angulares da idéia chamada de "paralelismo intrínseco". A cada passo do processo evolutivo, as *strings* são selecionados da população com uma probabilidade relacionada com seu *fitness*. Esta seleção tem o efeito de incluir em cada geração representantes de um *schemata* particular proporcional a seu *fitness* médio. O *fitness* médio é uma quantidade artificial que só indica que *strings templates* são os mais visados para serem investigados. Os processo de seleção com controles adicionais, tem sido a base da teoria da convergência dos AGs. O teorema do *schema* é interpretado algumas vezes no sentido no que sugere que o paralelismo intrínseco dos AGs pode ser exponencial. Entretanto, isto não ocorre, porque alguns *schematas* não sobrevivem a reprodução, e o número de *schematas* diferentes considerados depende do tamanho da população.

Holland estimou que o número de *schematas* que estão sendo processados em cada geração que evolui numa população de m *strings* é $O(m^3)$. Esse processamento é geralmente interpretado como uma designação de tentativas que são quase ótimas. Em [MÜH88], se mostra que isto é certo somente para problemas simples de otimização.

3.8. Problemas "AG-difíceis" e o Conceito de "Engano"

Uma dificuldade central na teoria dos AGs é a caracterização de quais são aqueles problemas que são difíceis de otimizar utilizando esta metaheurística. A maioria das intenções se centram em torno do conceito de "engano" (em inglês: *deception*). Goldberg [GOL87] introduziu o conceito de "engano" nos AGs, junto com a definição do problema do "engano mínimo". Desde então, o "engano" tem sido usado como o principal elemento para a pesquisa de "problemas AG-difíceis".

As funções enganosas (*deceptive functions*) estão geralmente baseadas na hipótese de blocos de construção, a qual trata de explicar o funcionamento dos AGs. A versão estática desta hipótese é a seguinte:

- ***Static Building Block Hypothesis***: dada alguma partição de um hiperplano, que seja curta e de baixa ordem, um algoritmo genético converge ao hiperplano com o melhor *fitness* médio.

Por exemplo, caso se considere a seguinte partição de segunda ordem de um hiperplano:

Hiperplano	F(H)
H(a): 00******	1
H(b): 01******	3
H(c): 10******	10
H(d): 11******	7

Tabela 01- Partições de Segunda ordem

onde F(H) representa o *fitness* estático médio do *schema* H. A média é calculada considerando todos os pontos representados por H. Esta média é denominada estática porque se calcula independentemente do estado da população, em qualquer instante da evolução. No exemplo anterior, a hipótese de blocos de construção prediz que o ganho esperado será o hiperplano H(c). A hipótese implica que funções que são "AG-fáceis" tenham soluções ótimas, para aquelas em que os *schemas*, de baixa ordem, produzem uma média de *fitness* maior que a média associado com *schemas* sub-ótimos.

Por outro lado, a hipótese implica que funções para as quais os *schemas* de baixa ordem, associados com o ótimo tenham uma baixa média de *fitness*, representam um tipo "AG-difícil".

Ainda que a hipótese estática de blocos de construção tenha seu interesse de ponto de vista intuitivo, nunca foi experimentalmente provada, e por outro lado não segue os princípios do teorema dos *schemas*. Este teorema descreve o crescimento esperado de um hiperplano dentro de uma geração baseada em seu *fitness* médio observado, ou seja, a média é calculada baseando-se em amostras do hiperplano tomadas da população.

As observações apresentadas até aqui são provenientes, principalmente, do trabalho de Grefenstette [GRE93], que demonstra que alguns problemas altamente enganosos resultam fáceis de otimizar por meio de AGs, enquanto alguns problemas classificados como "AG-fáceis" por não possuírem "enganos", resultam quase impossíveis de otimizar. Portanto a classe de funções enganosas não são nem um subconjunto de problemas AG-difíceis, nem um superconjunto deles. O importante é entender que é impossível predizer o comportamento dos AGs, baseando-se em *fitness* médios estáticos dos hiperplanos. Mas é possível que exista uma correlação entre a ausência ou presença de engano e a dificuldade de otimizar funções usando AGs. Todavia, esta correlação ainda não foi estabelecida. Dentro do campo dos AGs, a caracterização de problemas difíceis, seguirá sendo

uma prioridade, a qual deverá levar em conta os riscos típicos dos AGs, incluindo a amostragem dinâmica.

3.9. Algoritmos Meméticos

Uma tendência relativamente nova no campo dos AGs é a incorporação de técnicas de busca local. Após alguns resultados desalentadores dos AGs ao ser aplicados a alguns problemas (especialmente os muitos difíceis) de otimização combinatória, os pesquisadores começaram a buscar novas maneiras para estender os AGs e obter resultados melhores. Um passo importante nessa direção foi dado por Mühlenbein [MÜH87] com o desenvolvimento de Algoritmos Genéticos Paralelos (AGPs), que permitem que os indivíduos na população melhorem seu *fitness* mediante melhoras iterativas. Nos AGPs também se permite que os indivíduos se selecionem entre eles por processos locais, o que facilita sua implementação em sistemas concorrentes. De certo modo, os AGs simples com busca local podem ser vistos como métodos sofisticados de descida múltipla. O processo de reinicialização está neste caso governado por regras genéticas, e a fase de descida é levada a cabo como de costume. O êxito destes métodos pode ser atribuído entre ter uma busca rápida e manter uma diversidade para evitar a convegência prematura. Outro método pode consistir em desenvolver uma busca chamada *Delta Coding* proposta por D. Whitley e seu colaboradores [WHI91].

Estas variações no conceito dos AGs foram denominadas globalmente de Algoritmos Meméticos (AMs) [MOS92], [HOF93], [RAD94]. A Técnica do Compartilhamento Sucessivo também pode ser encarada como um AM, pois sugere uma mudança. Porém não a nível de estrutura do AGs, mas sim no extermínio de nichos ecológicos dentro da função objetivo, mediante o uso de

funções auxiliares, evitando com isso a re-pesquisa em regiões onde ótimos já foram obtidos.

3.10. Nichos

A especialização permitida pela diferenciação sexual é assegurada na natureza através da espécie no seu nicho. Intuitivamente pode-se colocar que o nicho é o papel de um organismo num ambiente, e que espécies são uma classe de organismos com características comuns. Essa separação do ambiente e a distribuição dos organismos neste ambiente em subconjuntos diferentes é muito comum na natureza. Aproveitando-se desta idéia, esta seção apresentará como a indução de nichos e espécies podem colaborar com a otimização de funções usando os AGs.

3.10.1. Nichos Biológicos

Embora exista uma bem desenvolvida literatura na Biologia, tanto para nichos quanto para espécies, sua transferência para os AGs tem sido limitada [GOL89]. Como muitos outros conceitos e operadores, as primeiras teorias diretamente aplicáveis à busca genética são atribuídas a Holland [HOL75]. Para ilustrar os nichos e as espécies, Holland introduziu uma modificação do "problema bandido com duas armas"[1] com pagamento compartilhamento distribuídos (*appud* [GOL89]).

---

[1] O problema do bandido de duas armas propõe um dilema: um bandido possui duas armas, uma na mão direita outra na esquerda. O bandido (jogador) não sabe, porém, o retorno (vantagem) associado a arma da mão direita e a da mão esquerda. Sabendo que há diferenças entre os retornos; arma da direita $A_d = \mu_1$ com variança σ_1^2 e arma da esquerda $A_e = \mu_2$ com σ_2^2. O bandido não sabe mas $\mu_1 \geq \mu_2$ .17em explorar Como ele responderá corretamente qual a arma

Uma vez reconhecida a importância do compartilhamento para a formação de nichos, tem-se o principal da teoria necessária para que se entenda como isto funciona na otimização com AGs. Entretanto, outro fator observado na natureza é o *matching* (casamento), pois até agora o trabalho referia-se ao acasalamento aleatório entre os elementos da população. Isto é contrário à maioria dos exemplos biológicos. A observação de que as espécies improvavelmente se acasalarão com organismos diferentes dos seus, faz surgir uma questão: porque isto ocorre? Ainda de outra maneira: qual é a vantagem seletiva da regra do acasalamento governar o comportamento das espécies?

Quando a população de cadeias é repetidamente produzida, acasalada e cruzada, a prole resultante tende a serem *strings* relativamente inúteis. Goldberg [GOL89] coloca que espécies interacasaladas tendem a gerar proles de baixo desempenho. Se, por outro lado, alguma pressão puder ser mantida a fim de fazer com que indivíduos similares "casem-se" entre si, as gerações de proles infrutíferas podem ser reduzidas. Isso mostra a necessidade para que os AGs encoragem padrões para o acasalamento.

Podem surgir ainda situações, como a ilustrada na figura 04, onde é dada uma função de adaptação que possui dois picos no intervalo [a, b]. O pico localizado em x=b é ótimo global em relação ao pico em x=a. Usando um algoritmo genético tradicional básico (**apêndice 01**) na pesquisa dos ótimos deste problema, com uma população aleatoriamente distribuída no espaço do intervalo, ter-se-á estatisticamente, na maioria dos casos, mais elementos no sub-intervalo [a,c] do que no sub-intervalo [c,b]. Isto fará com que elementos pertencentes a [a,c] apareçam com mais freqüência ao se eleger elementos para a reprodução, e consequentemente menos elementos entre [c,b], criando com isso uma tendência da população, como um todo, de compartilhar mais a vizinhança ao redor de x=a

---

de mais valor (exploração) ao mesmo tempo que usa essa informação (aproveitamento, utilização).

e menos nos entornos de x=b. Esta classe de problemas é classificada como Problemas Enganosos ou Funções de Armadilha.

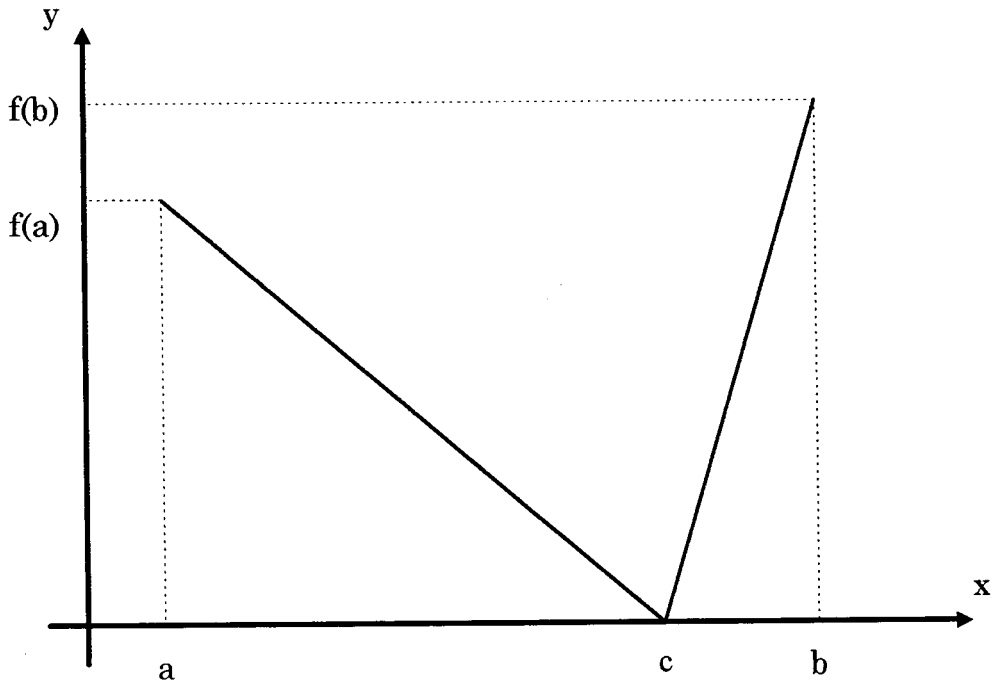


Figura 04. Função de Armadilha - picos nos extremos de um espaço unidimensional

CAPÍTULO 4

4. COLOCAÇÃO DO PROBLEMA

4.1. Introdução

Como nos trabalhos com *fitness sharing* de Goldberg e Richardson [GOL87], Deb [DEB89] e Deb e Goldberg [GOL89] assume-se que:

(i) Conhece-se, ou pode-se estimar, quantos máximos de interesse a função tem;
(ii) Que esses máximos de interesse se estendam por todo o espaço de pesquisa.

Todavia, idéias para superar essas limitações são discutidas na tese. Também não se assume que todos os máximos tem a mesma altura.

A seguir será feito um resumo das técnicas atuais de localização de múltiplas soluções em funções multimodais.

4.2. Iteração

A iteração é a técnica mais simplista que pode ser usada juntamente com algoritmos de otimização para produzir múltiplas soluções. Se o método de otimização incorpora conduta estocástica, então existem chances após sucessivas

rodadas de produzir diferentes soluções. Mas isto certamente é uma maneira pouco inteligente de abordar um problema de otimização. Por mais iterações que se executem, não existe a garantia da localização de todos os máximos de interesse.

Aplicando o método da iteração a uma técnica que produz uma solução simples (única), o que de melhor se pode esperar é que para localizar **p** máximos, o algoritmo seja rodado **p** vezes. Para compensar a duplicação de soluções, devido as iterações cegas, deve-se aumentar o número de iterações por um fator, que é chamado de fator de redundância R [JOL61]. Se todos os máximos tem a mesma probabilidade de serem determinados, pode ser demonstrado que R tem um valor dado por:

$$R = \sum_{m=1}^{p} \frac{1}{m}$$

Que pode ser aproximado para **p**>3, por $R \cong \gamma + \ln(\mathbf{p})$, onde $\gamma \cong 0{,}577$ (constante de Euler).

Este valor é relativamente pequeno. Entretanto se todos os máximos, não forem igualmente prováveis de se determinar, o valor de R deverá ser muito maior.

4.3. Subpopulações Paralelas

Outra técnica que pode produzir múltiplas soluções é um AG onde a população é dividida em múltiplas subpopulações que se desenvolvem em paralelo. Se não existir comunicação entre subpopulações, então esta maneira de resolver é diretamente equivalente a fazer iterações com pequenas populações,

várias vezes. Similarmente não há garantias que diferentes populações cheguem a convergir a diferentes máximos. A maioria das implementações usam algum grau de comunicação entre as subpopulações para permitir um bom "esparamamento" genético. Entretanto, o efeito das comunicações reduz a diversidade das soluções. Toda a população, eventualmente, tende a convergir para uma única solução [GRO85]. Esquemas de "companheirismo" local sofrem problemas similares [DAV91].

4.4. *Fitness Sharing*

Goldberg e Richardson [GOL87] descrevem a idéia do *fitness sharing*, que traduzindo poderia ser colocado como uma **divisão de adaptação**[1] em AGs. Sendo esta técnica um modo de promover subpopulações estáveis, ou espécies [2].

A idéia de dividir, vem da analogia com a natureza. Nos ecossistemas, existem muitos caminhos diferentes pelos quais os animais sobrevivem (pastando, caçando, no chão, em árvores, etc.) e diferentes espécies evoluem segundo o seu papel na natureza. Cada papel é referente ao seu nicho ecológico. Para cada nicho, os recursos físicos são finitos e devem ser divididos entre a população do nicho. Este fato fornece um desencentivo para que todas as espécies ocupem um único nicho, fazendo com que os seres vivos evoluam a ponto de formar subpopulações diferentes nos nichos. O tamanho de cada subpopulação é o reflexo da quantidade de recursos disponíveis em cada nicho.

---

[1] Divisão de adaptação, aqui, toma a idéia de compartilhar um mesmo ambiente.
[2] Em um trabalho anterior, tinha o objetivo de promover especificações existentes. Entretanto, na maioria dos casos a especificação foi empregada como um método para a manutenção da diversidade para melhor encontrar múltiplas soluções.

A analogia com a otimização de funções é a de que a localização de cada máximo representa um nicho; fazendo as divisões adequadas de adaptações, associado a cada nicho, consegue-se o encorajamento da formação de subpopulações estáveis e seu máximo. A divisão é realizada no princípio para que haja parâmetros que limitem cada nicho, e assim o limitem para que o compartilhamento seja feito entre os indivíduos daquele nicho. Conseqüentemente a propriedade concedida para um indivíduo será inversamente proporcional ao número de outros indivíduos no mesmo nicho. O "pagamento" total por um nicho é igual a altura de seu máximo. Sendo assim, máximos maiores sustentam populações proporcionalmente maiores.

Contudo, existe um problema fundamental: onde estão os nichos? Como é feita a decisão se dois indivíduos são do mesmo nicho e por isso devem ter seus valores de aptidão divididos? Para fazer isso corretamente, precisa-se saber onde cada nicho está e qual o tamanho dele. Obviamente não se consegue saber com antecedência onde os nichos estão. Goldberg e Richardson [GOL87] abordaram isto admitindo que se dois individuos estão muito próximos, dentro de uma distância conhecida como **raio do nicho**, então os valores da aptidão deverão ser compartilhados. Proximidade, neste caso, é medida pela distância em um espaço de parâmetros decodificados em uma dimensão única segundo uma métrica[3]. Deb e Goldberg [GOL89] mostraram um método para escolher o raio do nicho, o qual será discutido mais adiante.

Embora a técnica deles tenha sido bem sucedida, Smith, Forrest e Perelson [SMI92] destacaram suas desvantagens. Para calcular exatamente a aptidão de um indivíduo é necessário determinar sua distância a cada membro da população. A complexidade total de tempo dependerá do tempo necessário para que a execução do AG (básico) mais o tempo adicional necessário para realizar os

---

[3] Métrica num conjunto M é uma função $f:M\times M\rightarrow\Re$ que associa a cada $(x,y)\in M\times M$ um número real chamado de distância de x à y, tal que:
(i) $d(x,y)\geq 0$ e $d(x,y)=0\Leftrightarrow x=y$ (Positividade)
(ii) $d(x,y)=d(y,x)$ (Simetria)

cálculos do *fitness sharing*. A complexidade adicional é $O(N^2)$, onde **N** é o tamanho da população. Isto é uma desvantagem se **N** é grande, e **N** deve ser grande, caso se deseje encontrar muitos ótimos.

Goldberg [GOL89b]: para que um AG multimodal, que espera encontrar **p** soluções diferentes, necessita de uma população **p** vezes maior do que é necessário para um AG unimodal, isto porque um determinado tamanho da população é necessário para localizar e explorar cada máximo. Assim, se um AG tiver subpopulações estáveis para muitos máximos, o tamanho da população total deverá ser proporcional ao número de tais máximos.

O argumento acima assume que a população está igualmente espalhada entre todos os máximos em questão. Contudo, este pode não ser o caso, por duas razões. Primeira, sob o esquema do *fitness sharing*, altos valores de aptidão conterão proporcionalmente mais indivíduos do que os picos mais baixos. Segunda, pode haver picos na função do *fitness*, nos quais não se está interessado, mas que "aprisionam" uma porção da população. Este era um problema significativo para Goldberg, Deb e Horn [GOL92]. Esta distribuição irregular requer que **N** seja incrementado, para assegurar que até mesmo o menor pico de interesse contenha uma população suficientemente grande. O fator pelo qual **N** deve ser aumentado é denominado de *peak ratio* (ϕ). Se ρ é a proporção de picos que são de interesse, e f_{min} é o valor do menor pico de interesse, então ϕ é determinado por:

$$\phi = \frac{\text{Média do } \textit{fitness} \text{ de todos os picos}}{\rho . f_{min}}$$

O tamanho da população (total) exigido é $\mathbf{N=np\phi}$, onde **n** é o tamanho da população necessária para encontrar uma solução[4].

(iii) $d(x,z) \leq d(x,y)+d(y,z)$ (Desigualdade Triangular)

[4] Os métodos para escolher valores adequados para n são listados em [GOL89b]. Contudo, o valor ótimo de n depende da extensão do cromossomo.

Oei, Goldberg e Chang [OEI91] sugeriram várias melhorias para o algoritmo original do *fitness sharing*. Eles colocam que a complexidade computacional do *fitness sharing* pode ser reduzido de $O(N^2)$ para $O(Np)$ se a população for amostrada ao invés de calcular a distância entre os membros. Eles também descrevem que, em combinação com torneios de seleção, um parâmetro limiar de tamanho de nicho pode ser usado, para limitar o número máximo de indivíduos. Isto deveria evitar que os máximos fortes obtenham, significativamente, mais indivíduos em seu nicho em relação a outros.

Logo, o valor efetivo de ϕ estará entre 1 (se os picos sem importância tem baixo *fitness* próximos a f_{min}) e $1/\phi$ (se os picos sem importância tiverem aptidões próximas a f_{min}). Assumindo que estas sugestões são viáveis, pode-se calcular a complexidades do tempo como segue.

O tempo necessário para que o AG básico encontre um único pico, é proporcional ao número de gerações realizadas antes da convergência, isto é, aproximadamente ($\alpha N g_c$), onde α é o tempo para a avaliação de uma função, e g_c é o número de gerações até a convergência. O tempo adicional para o *fitness sharing* é de aproximadamente ($\beta A N p g_c$), onde β é o tempo de cálculo entre as distâncias de um indivíduo a outro, e A é o número de indivíduos a serem amostrados em cada nicho. Usando A=5 [OEI91] e estabelecendo $N=n\phi p$ a complexidade de tempo é:

$$C_{share}=O(n\phi p g_c(\alpha+5\beta p))$$

3.5. Outros Métodos

Glover [GLO89] descreve o método conhecido como Pesquisa Tabu, que pode ser usado no campo da otimização combinatorial. O método emprega a técnica da subida de encosta de acesso íngreme (*steepest-ascent hill climbing*) em uma solução de processo único e mantém o registro de movimentos feitos recentemente. Estes registros são usados para prevenir o retorno à soluções previamente exploradas. Este método permite avançar mesmo quando um ótimo local é alcançado.

A Técnica do Compartilhamento Sucessivo é conceitualmente similar, porém, ao invés de criar um caminho proibido (Tabu) para evitar a volta pelo mesmo caminho, são criadas regiões proibidas. Estas regiões tabu são centradas em torno de locais onde se encontram os ótimos locais ou globais.

Smith et al. [SMI92] descreve uma técnica que consegue promover a formação de nichos num sistema AG classificador. Eles demonstraram que ela tem um efeito similar ao *fitness sharing*, mas evita as limitações do número de picos que deve ser conhecido, que os picos devem estar uniformemente espalhados, e que a técnica é de complexidade de tempo $O(N^2)$. Infelizmente, a técnicas deles é somente aplicável em sistemas classificadores, não para otimização de funções multimodais.

CAPÍTULO 5

5. TÉCNICA DO COMPARTILHAMENTO SUCESSIVO

5.1. A Fundamentação Teórica da Técnica

5.1.1. Princípios Básicos

Ackley [ACK87] comparou um número de técnicas de otimização de funções, incluindo AGs e métodos de subida de encosta. Quando a técnica de otimização localiza um pico subótimo ele repetia a técnica até que o máximo global (sabido) tinha sido alcançado. Ele concluiu que para resolver problemas difíceis (com máximos locais), deve-se procurar até localizar um dos máximos e então com lições aprendidas, tentar novamente. Mas iterações cegas não aprendem lições. Quando se repete um AG convencional, tudo o que foi aprendido na pesquisa anterior é esquecido. A **Técnica do Compartilhamento Sucessivo** que é descrita está baseada na idéia de levar adiante conhecimento obtido em uma pesquisa para cada pesquisa subseqüente.

Uma vez que o máximo tenha sido encontrado, não há necessidade de procurá-lo novamente. O objetivo é eliminar esse pico da função de *fitness*, usando um método semelhante ao *fitness sharing*. Como foi mencionado anteriormente, o *fitness sharing* é complicado de ser executado quando a localização dos nichos é desconhecida. Após uma pesquisa do AG, a localização de

um dos nichos já é conhecida. Em segunda rodada do AG, assume-se que este nicho já está conceitualmente preenchido e que algumas poucas recompensas (disponíveis para qualquer indivíduo) podem estar dentro de sua área, e assim os indivíduos são forçados a convergir para um nicho desocupado, que por sua vez também estará conceitualmente preenchido na terceira rodada. Este processo pode continuar até que se decida (usando alguns critérios, tal como número de máximos esperados na função) que todos os máximos tenham sido localizados.

O algoritmo do *fitness sharing* trabalha, essencialmente, modificando dinamicamente o "panorama" da função de *fitness*, de acordo com a localização, em cada geração, de indivíduos na população. Isto é feito de tal forma como que para encorajar a dispersão de indivíduos por todo o "panorama", que leve para uma melhor exploração de todos os máximos. Ao invés disso, na **Técnica do Compartilhamento Sucessivo** funciona modificando o panorama da função de adequação de acordo com a localização de soluções encontradas em iterações anteriores. Isto é feito de tal maneira que desencoraje indivíduos a explorar áreas onde soluções tenham sido encontradas, e encorajando-os a encontrar um máximo em outro lugar.

No algoritmo do *fitness sharing*, uma vez que a população começa a convergir em picos múltiplos, faz com que cruzamentos entre espécies, de diferentes picos, passam a ser de provável improdutividade. Deb [DEB89] mostrou que um esquema de restrição de acasalamento era capaz de melhorar o algoritmo do *fitness sharing* por esta razão. Se o acasalamento entre indivíduos de diferentes picos for proibida, o desempenho seria melhorado ao invés de percorrer um número de populações de AGs menores separados; um explorando cada pico. Isto pode ser alcançado com um AG paralelo onde as subpopulações evoluem separadamente em cada processador, após um período inicial no qual evoluem juntos. Usando também algum algoritmo que assegure que nenhuma das subpopulações estejam convergindo para o mesmo ponto.

5.1.2. A Descrição da Técnica

Qualquer problema que envolva um AG necessita de uma função de adaptação. Para a **Técnica do Compartilhamento Sucessivo** também se necessita uma métrica. Esta métrica é uma função que a dados dois cromossomos associa um valor denominado distância entre os cromossomos. A medida de distância mais trivial é a métrica de Haming[1], embora como Deb e Goldberg [GOL89] mostraram, o algoritmo do *fitness sharing* funciona melhor se é usada uma medida de distância baseada no parâmetro decodificado do espaço. Uma razão para isso é que para - para cromossomos codificados em binários - grandes diferenças entre cromossomos podem corresponder pequenas diferenças no parâmetro do espaço (por exemplo o penhasco de Haming). Usando um parâmetro para determinar a distância entre dois cromossomos, evitamos qualquer distorção topológica apresentada pelo esquema de código.

Tipicamente o genótipo de um cromossomo representa o número de parâmetros (fenótipo) e eles são mapeados a um valor adequado. Se há k parâmetros, então cada indivíduo pode ser considerando como ocupando um valor adequado no espaço dimensional k. A distância entre dois indivíduos pode ser considerada como a distância Euclidiana[2] entre eles. Como foi sugerido por Deb [DEB89], se o tamanho dos parâmetros em diferentes dimensões forem muito diferentes, então pode-se normalizar cada parâmetro para valores entre 0 e 1antes que a distância euclidiana seja determinada. O algoritmo aqui exposto poderá ser trabalhado com qualquer métrica arbitrária, porém durante o desenvolver do mesmo na tese, será usada a métrica euclidiana.

---

[1] A métrica é da pela soma das diferenças absolutas entre os alelos de cromossomos, por exemplo: sejam os cromossomos $A=(10010)_2$ e $B=(11011)_2$, assim $d(A,B)=|1-1|+|0-1|+|0-0|+|1-1|+|0-1|=2$.

[2] A métrica euclidiana é dada por $d(x,y)=\left[\sum_{i=1}^{n}(x_i - y_i)^2\right]^{\frac{1}{2}}$

Tendo sido definida a função de adequação e a métrica de trabalho, o algoritmo é proposto como segue:

1. **Inicialização**: equiparar a função de adequação modificada com a função de adequação original;
2. **Execução**: utilizando um AG (ou outra técnica de pesquisa) sobre a função de adequação modificada, manter o registro do melhor indivíduo encontrado no percurso;
3. **Atualização**: atualizar a função de adequação modificada para dar uma depressão na região próxima do melhor indivíduo, produzindo uma nova função de adequação modificada;
4. **Controle e refinamento**: se a adequação original do melhor indivíduo excede o limiar da solução de interesse, então refinar a solução e apresentar como solução;
5. **Critério de parada**: se as p soluções ainda não foram encontradas, retornar a execução (2).

O termo sucessão surge porque a técnica consiste de uma seqüência de várias rodadas de AGs. O conhecimento da localização de nichos é propagado para subsequentes percursos na mesma seqüência.

A solução é um conjunto de **k** parâmetros que definem a posição de um máximo de interesse.

O máximo de interesse representa o limite da adequação mais baixo (de interesse). É assumido que há um número de **p** máximos de interesse, todos com *fitness* maior que o limiar de mínimo de solução. Seu valor é classificado individualmente para cada problema. Se nenhuma informação está disponível a respeito dos valores de aptidão dos máximos de interesse, o limiar da solução pode ser classificado como zero. Neste caso, a técnica é finalizada após os **p** primeiros picos tenham sido encontrados.

A função de adequação modificada M(**x**), para um indivíduo **x** é calculado da função de adequação original F(**x**), multiplicada pelas funções auxiliares de pico único. Inicialmente classifica-se $M_0(\mathbf{x})\equiv F(\mathbf{x})$. Ao fim de cada rodada de AGs, o melhor indivíduo s_n, encontrado no percurso é usado para determinar a função auxiliar de pico único $G(\mathbf{x},s_n)$. A função de adequação modificada é então atualizada de acordo com:

$$M_{n+1}(\mathbf{x})\equiv M_n(\mathbf{x})*G(\mathbf{x},s_n)$$

5.1.3. Funções Auxiliares

Várias formas para a função de auxílio $G(\mathbf{x},s_n)$ são possíveis; as funções testadas encontram-se a seguir e sua denominação vem do autor:

(i) A função de potência: $G_p(\mathbf{x},\mathbf{s})=\begin{cases}\left[\dfrac{d(x,s)}{r}\right]^{\alpha} & \text{se } d(x,s)<r, \text{ caso contrário}\\ 1\end{cases}$

(ii) A função exponencial: $G_e(\mathbf{x},\mathbf{s})=\begin{cases}e^{\left[\ln(m)*\frac{r-d(x,s)}{r}\right]} & \text{se } d(x,s)<r, \text{ caso contrário}\\ 1\end{cases}$

(iii) 1ª função auxiliar de Hein: $G_h(\mathbf{x},\mathbf{s})=\sqrt{\dfrac{d(x,s)}{\frac{0,1}{d(x,s)}+d(x,s)}}$

(iv) 2ª função auxiliar de Hein: $G_h(\mathbf{x},\mathbf{s})=\left|\dfrac{e^{d(x,s)}-e^{-d(x,s)}}{e^{d(x,s)}+e^{-d(x,s)}}\right|$

Nas quatro funções apresentadas r é o raio do nicho, d(**x**,**s**) é a distância entre **x** e **s**. Na função (i), α é o fator de potência, que determina quão côncavo ($\alpha>1$) ou quão convexo ($\alpha<1$) é a função auxiliar. Se $\alpha=1$ a função resulta linear. Quando $d(\mathbf{x},\mathbf{s})=0 \Rightarrow G(\mathbf{x},\mathbf{s})=0$. Na equação (ii) m é o limiar mínimo de interesse, assim quando $d(\mathbf{x},\mathbf{s})=0 \Rightarrow G(\mathbf{x},\mathbf{s})=m$, aonde m deverá ser maior que zero, porém $m<1$, além disso m também determina a concavidade da curva auxiliar. Quanto menores forem os valores de m maior será a concavidade da curva. Para os dois primeiros casos $d(\mathbf{x},\mathbf{s})\geq r \Rightarrow G(\mathbf{x},\mathbf{s})=1$. A terceira função não necessita deste expediente.

A atuação das funções auxiliares são apresentadas na **Figura 05**. A duas primeiras funções auxiliares (potência e logarítmica) com raio r=1. Na 1ª função auxiliar de Hein o raio tem valor infinito, porém na prática se verifica que ele pode ser controlado computacionalmente para um raio conveniente a cada problema. Na 2ª função auxiliar de Hein o raio r = e $\cong$ 2,718 (constante de Napier).

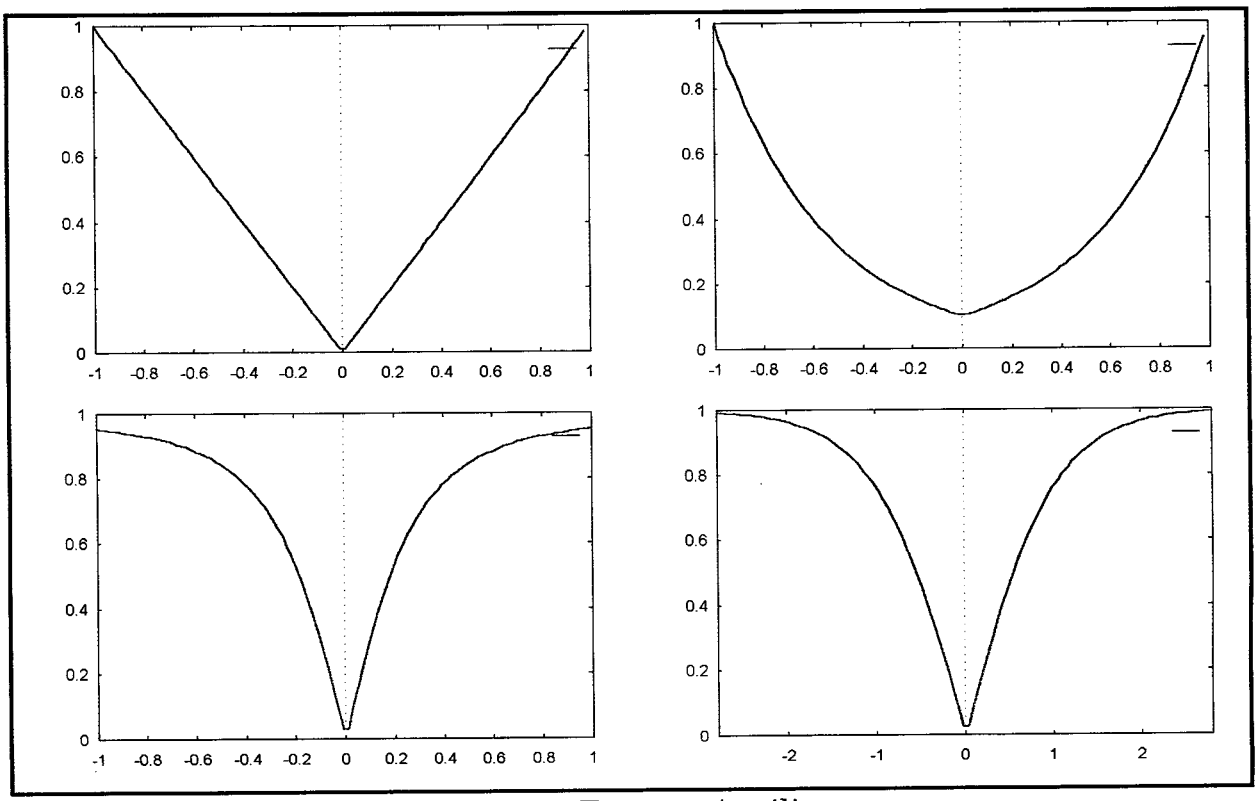


Figura 05. Funções Auxiliares

O *fitness sharing*, descrito por Goldberg e Richardson [GOL87], Deb [DEB89] e Deb e Goldberg [GOL89], realiza a tarefa de reduzir o *fitness* de um indivíduo dependendo de sua distância de outro elemento na população. De maneira similar a função auxiliar realiza a tarefa de reduzir o *fitness* de um indivíduo dependendo da sua distância de cada melhor indivíduo encontrado na rodada anterior.

A função ilustrada na **Figura 06** mostra a função original $f(x)=sen^2(2\pi x)$, da qual tome-se como conhecido o máximo em x = 0,25.

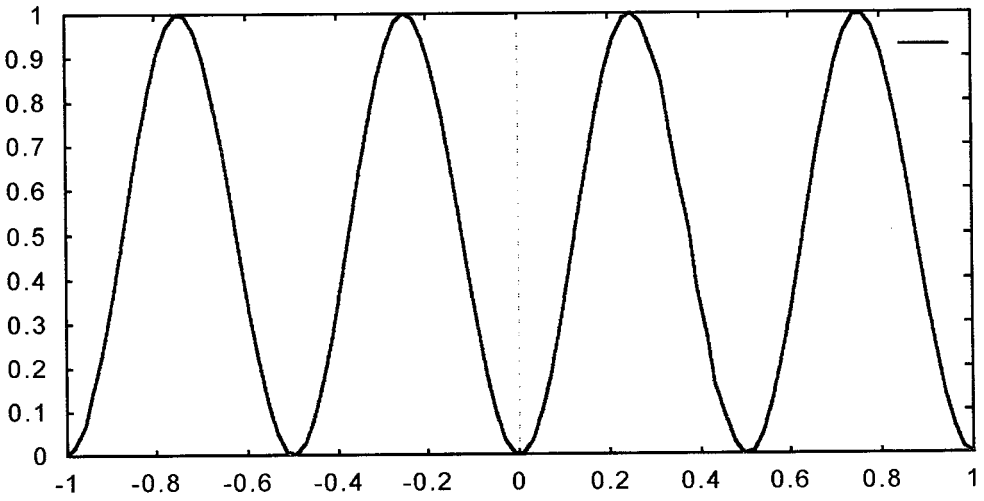


Figura 06. Função Original

Observe-se que na **Figura 07** o pico em x = 0,25 foi suprimido. Os tamanhos destes picos podem ser tornados arbitrariamente pequenos, usando, por exemplo, a 1ª função auxiliar de Hein ele pode ser completamente anulado[3]. Pequenas modificações nas outras três funções auxiliares, promovem efeitos semelhantes.

[3] Este anulamento formará "bicos" na função modificada, o que impedirá o uso de métodos analíticos (com derivadas) na discussão de seus pontos de máximos e/ou mínimos. Isto justifica o uso dos algoritmos genéticos, pois estes não necessitam do conceito de derivada

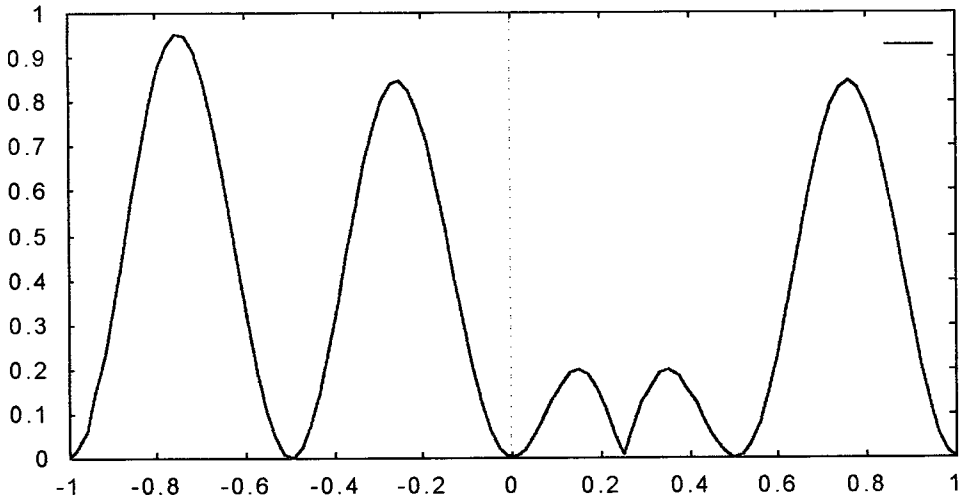


Figura 07. Função Modificada

Caso o AG escolha estes pequenos picos como sendo ótimos, da função modificada, numa rodada subseqüente, então este candidato a "máximo" pode ser submetido ao teste da derivada nula, em relação a função original. Isto não é nenhuma garantia de máximo, porém os procedimentos nos AGs são de maximização e assim pode-se supor válida a idéia.

Para que a função original não se modifique, fora do nicho a ser eliminado, pode-se optar em estabelecer que, valores de modificação gerados pelas funções auxiliares, quando próximos a 1 (um) sejam, então, tomados como sendo 1 (um), bastando que seja estabelecida uma pequena vizinhança no entorno do pico eliminado. Assim os picos da função não serão deslocados no espaço. Computacionalmente isto é bastante simples de se implementar.

Parta determinar o raio de um nicho, se usou-se o mesmo método como Deb propõe [DEB89], que é dado como segue; se cada **p** máximo na função é rodeado por uma hiperesfera de raio r, e que estas hiperesferas não se sobrepõe (intersecção nula), e que elas preencham o espaço k dimensional do problema, então o raio é dado por:

$$r = \frac{\sqrt{k}}{2 * \sqrt[k]{p}}$$

A técnica de Deb é simples, mas ela requer que se saiba, ou se estime, o número de máximos na função, e assume que todos os máximos estão igualmente distribuídos por todo o espaço de pesquisa. Claramente, há funções onde estas restrições não são satisfeitas.

5.2. Condições de Parada

Decidir quando parar uma fase do AG e iniciar outra não é tarefa fácil. Perto do fim de uma rodada qualquer de um AG tradicional, a eficiência na busca cai, porque a população torna-se mais homogênea (uniforme), motivando a exploração a confiar gradativamente em vantagens pela mutação [GOL89b]. Pode-se decidir o ponto do percurso em que se deve interromper uma rodada e iniciar outra. Um critério que pode ser adotado registra o *fitness* médio, ou seja, após h gerações sem contribuições ao *fitness* e a média pouco variante ($\bar{X} \leq \varepsilon$, por exemplo) e as rodada são então finalizadas. Na tese o critério utilizado está ligado ao número de gerações, ou seja, ao final de g_c gerações é escolhido o melhor valor entre os presentes como sendo um ótimo (local).

5.3. Refinamento da Solução

Devido a extensão dos cromossomos podem ocorrer problemas de refinamento da solução, ou seja, o estabelecimento das coordenadas exatas do pico máximo. Este refinamento pode ser feito com o uso dos Algoritmos Naturais (AN) descritos por Ricieri [RIE94], que fazem um tratamento também

cromossômico, porém sem o uso de *strings* binárias. Este procedimento ocorre após o término da busca por AGs.

5.3.1. Princípios de Otimização Natural

Otimização natural, ou simplesmente algoritmos naturais, é uma técnica que localiza máximos, ou mínimos de funções lineares e não-lineares, condicionadas a domínios numéricos formados por igualdades ou desigualdades. Isso graças a uma releitura matemática da seleção natural que é resumida por Ricieri na forma do seguinte algoritmo:

> "Na competição dos números pelo ótimo (*sobrevivência*) apenas aqueles que verificam o domínio (*cromossomo*) e que maximizam ou minimizam (*portadores de caracteríticas úteis*) a função (*organismo*) em dada iteração, originam (*linhagem*) novas iterações (*cruzamento*)."

Esse algoritmo elementar simula a transmissão dos traços hereditários numéricos nas sucessivas gerações de resultados (genótipo) oriundos de uma função.

Seguem sua nomeclatura e aplicação:

(a) Nomeclatura: as denominações, como nos AGs, forma extraídas da genética do cromossomo que é interpretado como o domínio da função a ser otimizada. Por analogia as hélices são as variáveis, enquanto as moléculas são os números que as variáveis podem assumir. A base, elemento de ligação das moléculas pertencentes as hélices (variáveis), será o resultado da função.

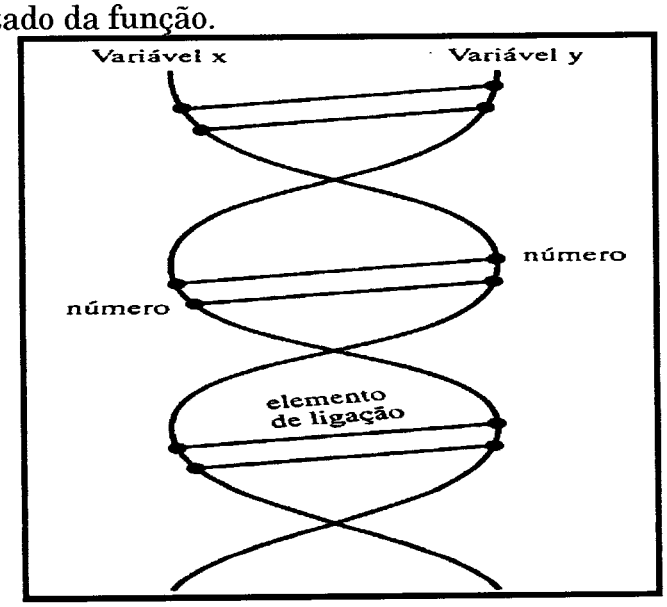


Figura 08. Domínio da Função

O domínio (códico genético cromossômico) de uma função linear ou não linear (organismo) na otimização natural admite dois tipos de iterações matemáticas (cruzamentos):

⇒ Iteração simples (cromossomo S)

⇒ Iteração composta (cromossomo C)

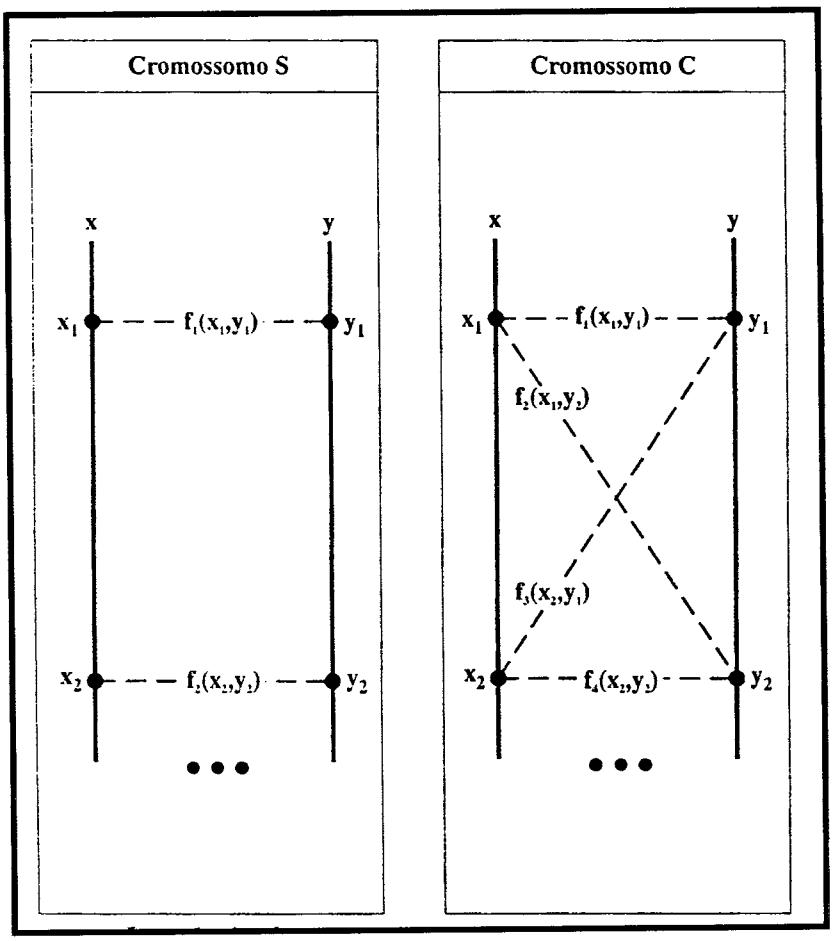


Figura 09. Cromossomo S - Iteração Simples;
Cromossomo C - Iteração Composta

(b) Aplicação: Seja, por exemplo, o problema:

$$\text{Max } f(x,y)=x^2y$$

sujeito a: $x+y=4$, com $x,y \in \Re$

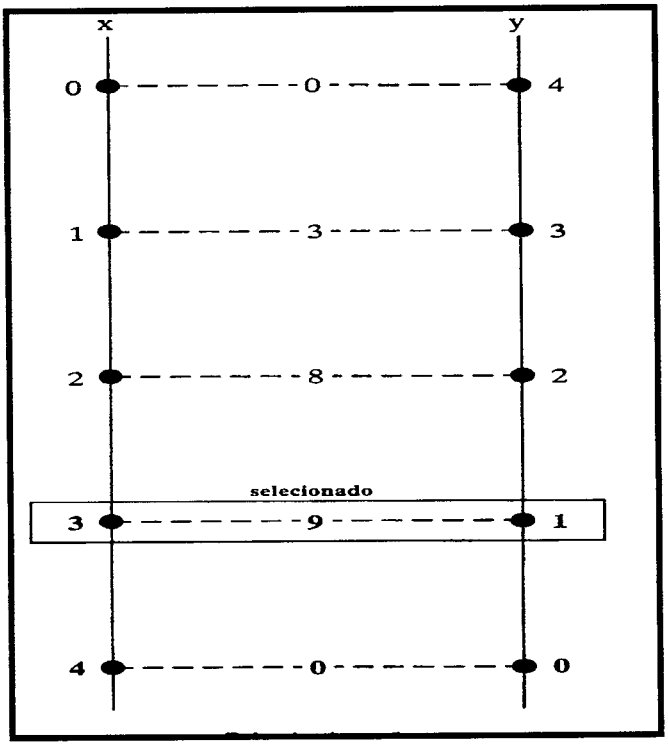


Figura 10. Primeira Iteração

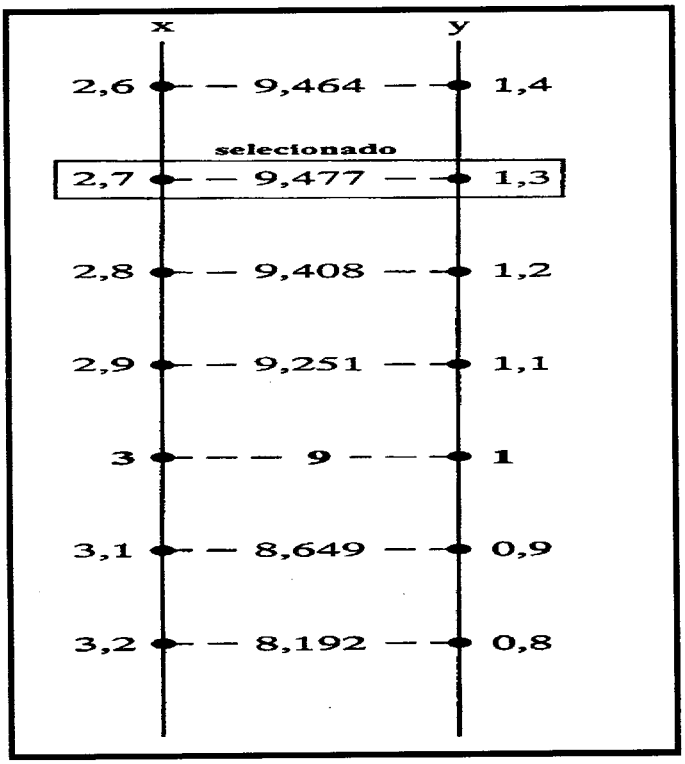


Figura 11. Segunda Iteração

O cromossomo numérico foi construído arbitrando valores para a variável x (0, 1, 2, 3, 4, ...) e verificando em x+y=4 o y correspondente (4, 3, 2,1, 0, ...). Na **Figura 10** é formado o primeiro cruzamento cujos resultados são (0, 3, 8, 9, 0, ...) foram obtidos de $f(x,y)=x^2y$ a ser maximizada.

Seleciona-se então o par x=3 e y=1 que produz a maior imagem (9) da função f(x,y), o qual dará origem ao próximo cruzamento matemático.

Para se contruir um novo cromossomo numérico (ampliação do anterior) que permitirá o segundo cruzamento, basta incrementar um dos valores do par selecionado de ±0,1, numa vizinhança ao redor de x e de y com um raio de 0,4, por exemplo) assim os valores irão desde x=2,6 até x=3,2 e y de y=1,4 até y=0,8. Os valores foram nesta ordem para que sua soma não ultrapasse 4. Assim repete-se o processo aplicando os outros pares ordenados x e y em f(x,y). Seleciona-se na segunda iteração x=2,7 e y=1,3 que produzem o maior valor f(x,y)=9,477, originando o terceiro cruzamento incrementado por ±0,01. Nesta última iteração o algoritmo chega a x=2,67 e y=1,33, com f(x,y)=9,481437. Caso seja necessário maior precisão pode-se incrementar os valores com valores cada vez menores: ±0,001, ±0,0001, etc. De modo geral esse algoritmo orienta o tipo de cruzamento que permite encontrar a trajetória numérica da otimização natural que conduz ao *ser* perfeito (x*, y*).

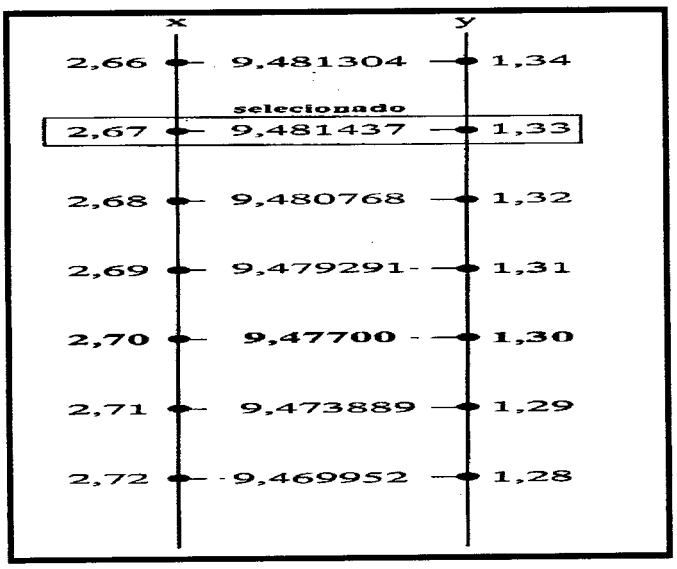


Figura 12. Terceira Iteração

5.4. Complexidade Computacional da Técnica

Para determinar a complexidade computacional de tempo, foi assumida que a performance de um AG, que tenta localizar qualquer de um dos vários máximos de uma função multimodal, é aproximadamente igual a aquela de uma um AG tentando localizar um máximo global de uma função unimodal. Para testar esta hipótese, foram executados 250 rodadas de um AG convencional, com **k** pontos de *crossing-over*, sobre as funções que estão colocadas no capítulo 6, com cromossomos de 30 bits binários, com tamanho de população igual a 20 elementos e com g_c=20 gerações como critério de parada (não foi computado o tempo de refinamento das soluções).

Comparou-se o número de avaliações nas funções testadas para localizar qualquer um dos máximos da primeira função com o número exigido para localizar o máximo global da segunda e terceira funções. O AG foi aproximadamente 2 vezes mais rápido. Não só isso, a exatidão das soluções foi melhorado por um fator 100.

Um AG tradicional que necessita uma população de n elementos para localizar o máximo de uma função unimodal e converge em g_c gerações, apresenta que a adição de mais máximos tende a reduzir o tempo de convergência e aumenta a exatidão das soluções encontradas. Por isso, parece falho assumir que a Técnica do Compartilhamento Sucessivo necessite uma população de igual tamanho, e que convirja em um número igual de gerações, quando localiza qualquer um dos máximos de uma função multimodal.

A técnica envolve múltiplos fases de um AG e a fase inicial tem uma complexidade de tempo idêntica àquele de um AG não-modificado. A medida que os picos são localizados, fases sucessivas do algoritmo tornam-se mais lentas devido a complexidade em calcular as funções auxiliares de modificação.

Isto aumenta linearmente com o número de picos encontrados. Usando a mesma representação numérica como na equação:

$$C_{share} = O(n\phi \mathbf{p} g_c[\alpha+5\beta\mathbf{p}])$$

A primeira fase tem a complexidade $O(\alpha n g_c)$. Na segunda fase em cada geração o algoritmo necessitará calcular a distância de cada indivíduo ao pico anteriormente encontrado na primeira fase. Este cálculo exige tempo adicional proporcional ao tamanho da população n. Na terceira fase, há dois picos prévios para calcular a distância, de maneira que o tempo adicional será proporcional a $(2n)$. Se o *fitness* de quaisquer picos não interessantes é pequeno comparado com aquele do menor pico interessante (isto é $\phi = 1$) um total de aproximadamente $\mathbf{p}$ fases são necessárias. O tempo total adicional é por isso proporcional a $(0+n+2n+3n+...(\mathbf{p}-2)n+(\mathbf{p}-1)n)$. Isto é $((\mathbf{p}-2)n*\mathbf{p}-1)/2)$, ou aproximadamente $(\mathbf{p}^2*n/2)$. O tempo total para o AG executar estes $\mathbf{p}$ percursos é proporcional a $(n\mathbf{p}g_c)$. Indicando as constantes de tempo como α e β, a complexidade de tempo total, $C_{sucessivo}$, pode ser colocada como:

$$C_{sucessivo}=O(n\mathbf{p}g_c[\alpha+0.5\beta\mathbf{p}]$$

Assim para $\phi = 1$ as duas complexidades são de forma semelhante; o *fitness sharing* exige uma fase com uma população de tamanho $n\mathbf{p}$, enquanto que a técnica do compartilhamento sucessivo exige $\mathbf{p}$ rodadas com uma população de tamanho n. Se os picos não interessantes têm um *fitness* próximo àquela do pico menos interessante, ambos os métodos são mais lentos; o *fitness sharing* exige um aumento no tamanho de população e a técnica do compartilhamento sucessivo exige rodadas adicionais. Nesse sentido, os dois métodos diferenciam somente pela técnica do compartilhamento sucessivo exigir 10 vezes menos cálculos de distância. Contudo, se g_c varia com o tamanho da população, a complexidade de tempo será bastante diferente.

Pondo $g_c = O(f(N))$, onde N é o tamanho da população, a melhora (aceleração) S alcançada pela técnica proposta é:

$$S = \frac{f(np)\left(\frac{\alpha}{\beta} + 5p\right)}{f(n)\left(\frac{\alpha}{\beta} + 0.5p\right)}$$

O número de gerações para convergência g_c é altamente dependente em ambas as técnicas e AG específico usado, mas as estimativas (aproximadas) feitas por Goldberg [GOL89b] dá o número de gerações para um indivíduo ótimo controlar a população como O(log N). Em um estudo de uma função discreta unimodal, Mühlenhein e Schlierkamp-Voosen [MÜH93] descobriram que g_c era independente de N quando nenhuma mutação era empregada para valores de N acima de um certo limiar. Para uma função contínua multimodal com mutação eles descobriram que g_c é $O(\log N)$. Outra estimativa é que g_c é $O(N)$ para certos problemas [MÜH92]. Todavia há um consenso de que as estimativas de g_c como $O(1)$, $O(\log N)$, ou $O(N)$ não diminuirá com N; neste princípio, a técnica do compartilhamento sucessivo será sempre mais rápida do que o *fitness sharing*.

O fator aceleração também dependerá no número de picos, p, relativo a proporção de tempo para avaliação do *fitness* e o cálculos das distâncias, α/β. Na maioria dos problemas, segundo Goldberg e Deb [GOL91], o cálculo do *fitness* leva mais tempo do que o cálculo de distância e tipicamente $\alpha/\beta >> 1$. Por isso para p pequeno ($p < n$ e $p << \alpha/\beta$) os fatores de aceleração para $g_c = O(1)$, $O(\log N)$, e $O(N)$ são :

$$S \approx 1$$

$$1 < S_{\log N} < 2$$

$$S_N \approx p$$

Para **p** grande (**p**>>α/β e **p**>>n), os fatores de aceleração são:

$$\mathbf{S}_1 \approx 10$$

$$\mathbf{S}_{\log}\,\mathbf{N} \approx 10\left(1+\frac{\log \mathbf{p}}{\log n}\right)$$

$$\mathbf{S}_N \approx 10\mathbf{p}$$

O fator aceleração sugere que a técnica do compartilhamento sucessivo será sempre melhor do que o *fitness sharing* tradicional e será igualmente útil em problemas onde **p** é grande.

CAPÍTULO 6

6. O CAMPO DE PROVAS

A Técnica do Compartilhamento Sucessivo foi testado em funções já anteriormente utilizadas por outros pesquisadores. Os testes se realizaram em três tipos de problemas:

⇒ Seção 6.1: funções multimodais com somente um único máximo global;
⇒ Seção 6.2: funções multimodais de codificação binária;
⇒ Seção 6.3: funções *"bump"*;

Os testes mais importantes recaem sobre as funções da seção 6.2. Nestas funções multimodais todos os máximos são de interesse. Os resultados mostram que, como a abordagem do *fitness sharing*, a técnica do compartilhamento sucessivo pode, eficientemente, localizar todos os máximos. Para os testes o AG tradicional foi baseado no SGA de Goldberg [GOL89a], onde ocorre a evolução produtiva, seleção estocástica remanescente, pesagem de *fitness* linear e **k** pontos de cruzamento.

6.1. Funções de Armadilha

As funções de armadilha são funções enganosas, as quais fazem com que um algoritmo genético evolua para um pico que não é o melhor, isto causado pela distribuição populacional no ambiente (espaço de busca).

6.1.1. Armadilha de Dois Picos

Na comparação feita por Ackley [ACK87] de várias técnicas de otimização, descobriu-se que uma função enganosa é muito difícil de ser otimizada. Um exemplo desta função pode ser dada como sendo:

$$f(x) = \begin{cases} \frac{-32}{3}x + 160, 0 \leq x < 15 \\ 40x - 600, 15 \leq x \leq 20 \end{cases}$$

O máximo global está em x=20 e tem um *fitness* f(x)=200. Há em x=0 um falso máximo, com um *fitness* f(x)=160. Os valores de $x \in [0, 15)$ conduzem o algoritmo para o pico falso. Esta função está implementada juntamente com os resultados no **anexo 01**.

Tradicionalmente o problema aqui não é encontrar ambos os picos, mas simplesmente evitar que o algoritmo opte por um pico falso em detrimento de um global.

6.1.2. Armadilha Central de Dois Picos

Ackley também menciona uma variação de uma função enganosa, que ele considera ser mais difícil de resolver. Exemplificando, pode-se apresentar:

$$f(x) = \begin{cases} 16x, 0 \leq x < 10 \\ -32x + 480, 10 \leq x \leq 15 \\ 40x - 600, 15 < x \leq 20 \end{cases}$$

Esta função (**anexo 02**) possui um máximo local em x=10, porém seu máximo global está em x=20.

6.2. Funções de Codificação Binária

Funções de codificação binária são aquelas nas quais o cromossomo, além de avaliar o *fitness* traz consigo mais parâmetros, como neste caso o sinal (+/-). Deb [DEB89] descreve várias funções que ele usou para testar o algoritmo do *fitness sharing*.

6.2.1. Máximos Iguais - F1

$$f_1(x)=sen^6(5\pi x),\ x\in[0,\ 1]$$

A função $f_1(x)$ foi dada por Deb [DEB89], a qual tem cinco máximos igualmente espaçados e de altura igual (**anexo 03**).

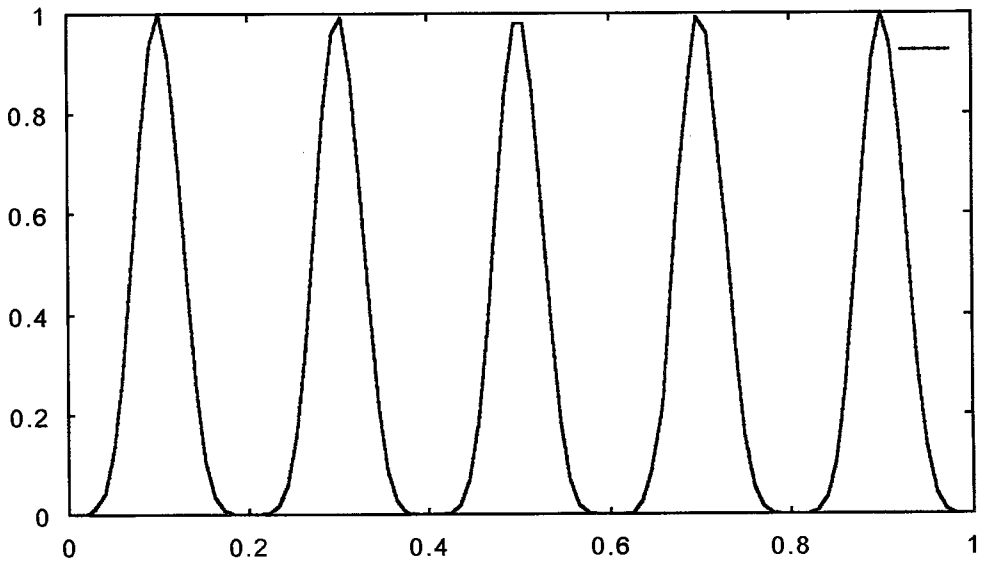


Figura 13. Máximos Iguais

6.2.2. Máximos Decrescentes - F2

$$f_2(x)=e^{\left[-2\ln(2)\left(\frac{x-0,1}{0,8}\right)^2\right]}\operatorname{sen}^6(5\pi x),\ x\in[0,1]$$

Esta função possui uma configuração espacial parecida com a função $f_1(x)$, porém $f_2(x)$ possui máximos que estão em ordem decrescente (**anexo 04**).

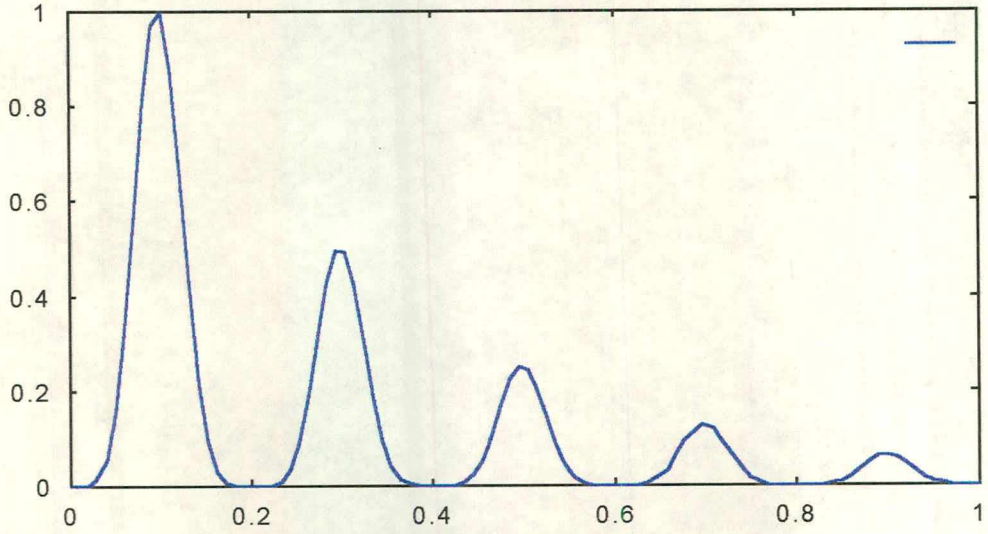


Figura 14. Máximos Decrescentes

6.2.3. Máximos Irregulares - F3

$$f_3(x)=\operatorname{sen}^6[5\pi(x^{3/4}-0,05)],\ x\in[0,1]$$

Novamente a função é muito semelhante a função $f_1(x)$, porém os máximos estão irregularmente espalhados (**anexo 05**).

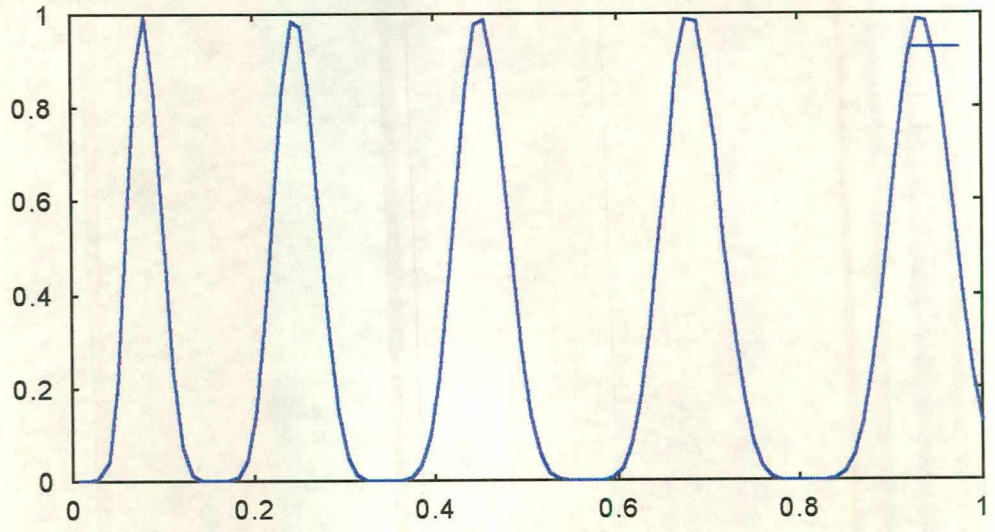


Figura 15. Máximos Irregulares

6.2.4. Máximos Decrescentes Irregulares - F4

$$f_4(x)=e^{\left[-2\ln(2)\left(\frac{x-0,1}{0,8}\right)^2\right]}\operatorname{sen}^6[5\pi(x^{3/4}-0,05)],\ x\in[0,1]$$

Aqui os máximos decrescem exponencialmente **(anexo 06)**.

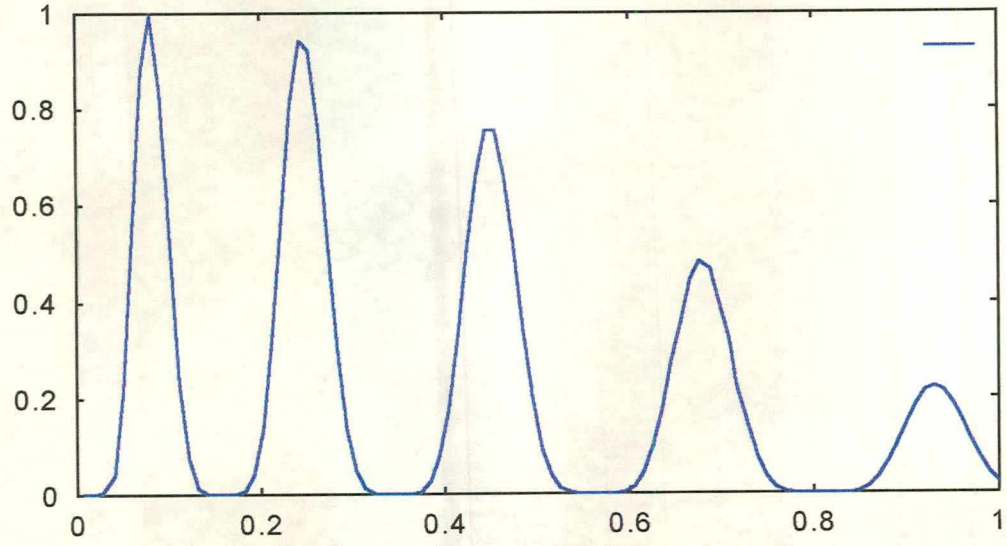


Figura 16. Máximos Decrescentes Irregulares

6.2.5. Função de *Himmelblau* - F5

A função de *Himmelblau* é uma função de duas variáveis, a qual Deb modificou para ser um problema de maximização [DEB89].

$$f_5(x)= 200-(x^2+y-11)^2 - (x+y^2-7)^2 \text{ , } x\in[-6,6]; y\in[-6,6]$$

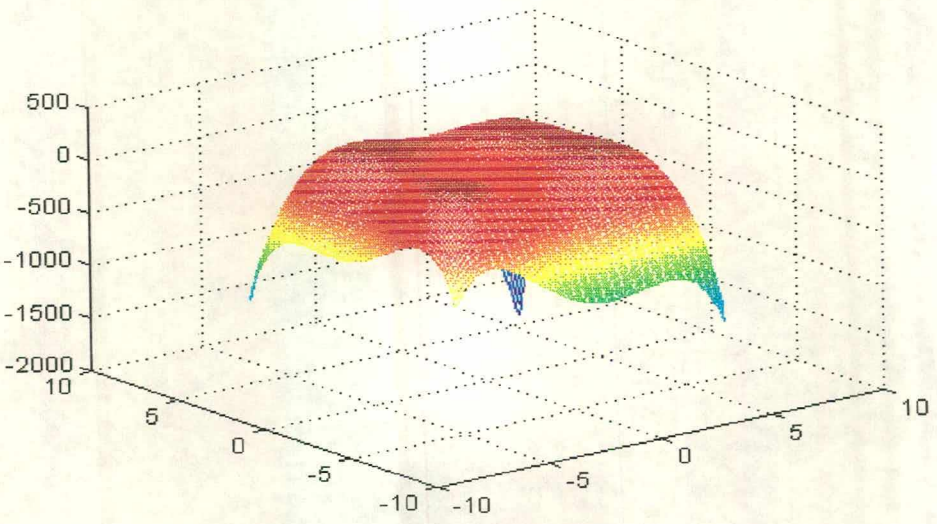


Figura 17. Função de *Himmelblau*

Esta função (**anexo 07**) possui quatro máximos de igual altura f(x,y)=200 nas coordenadas (x_1; y_1)=(3,58; -1,86); (x_2; y_2)=(-2,815; 3,125); (x_3;y_3)=(3; -2) e (x_4; y_4)=(-3,78; -3,28).

6.3. Funções "*BUMP*"

As funções *bump* não possuem nenhuma definição própria. Elas formam um grupo de funções, encontradas na literatura [KEA95a], que se caracterizam pelo seu grande número de picos ótimos (máximos e mínimos), de igual ou diferentes alturas pelo espaço de busca. Para a realização de testes de otimização, foram utilizadas as três seguintes funções *bump*:

6.3.1. Função BUMP-1

Max z= **abs**($\cos^4 x+\text{sen}^4 y-2(\cos^2 x \cos^2 y)$), $x \in [0,10]$, $y \in [0,10]$

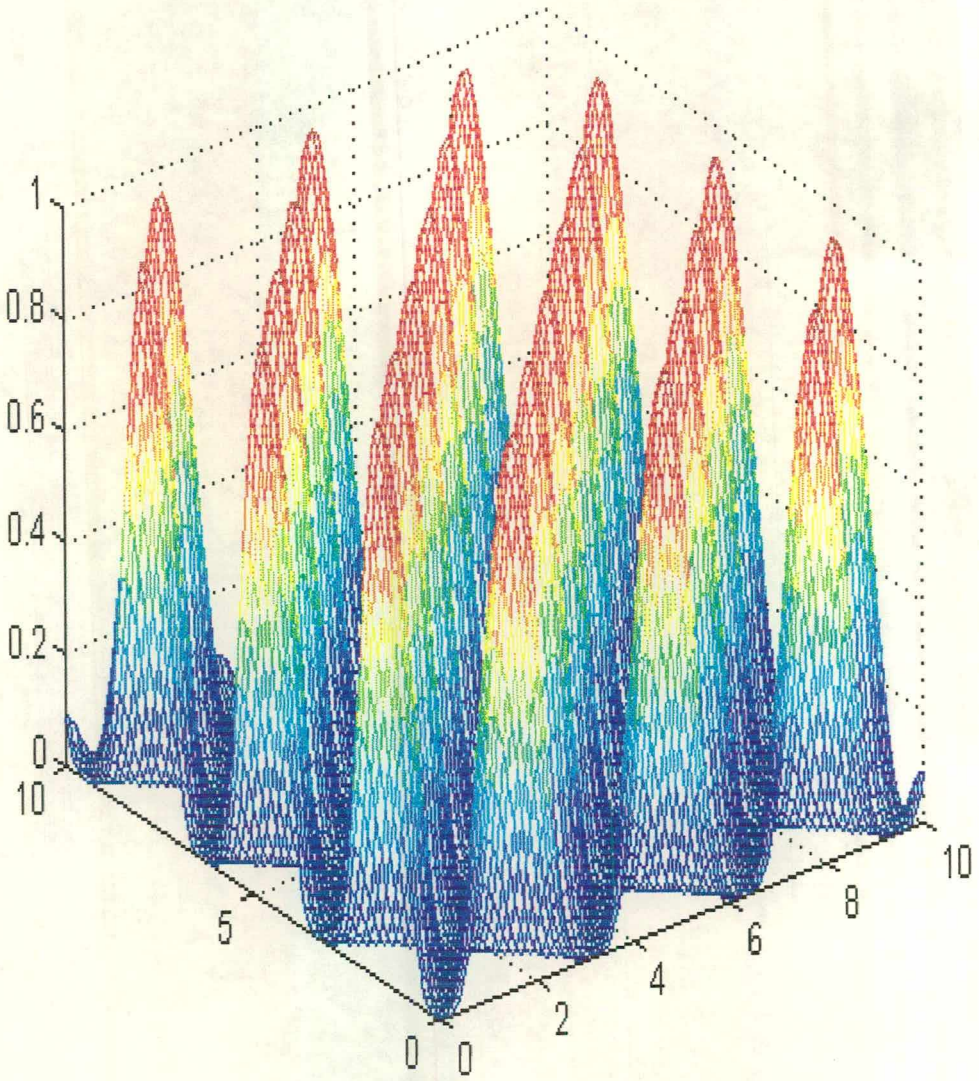


Figura 18. Superfície *BUMP*-1

Cromossomo: 40*bits*
População (***n***): 50
Gerações (g_c): 50
Pr[cross]= 90% (dois pontos de *crossing-over*)
Pr[mutação]= 1%
p Ótimos de Interesse=18
Implementação: **anexo 08**

6.3.2. Função BUMP-2

$$\text{Max } z = \mathbf{abs}\left(\sum_{i=1}^{4}\cos(ix) + \cos(iy) - 2\prod_{j=1}^{2}\cos(ix).\cos(iy)\right),\ x\in[0,10],\ y\in[0,10]$$

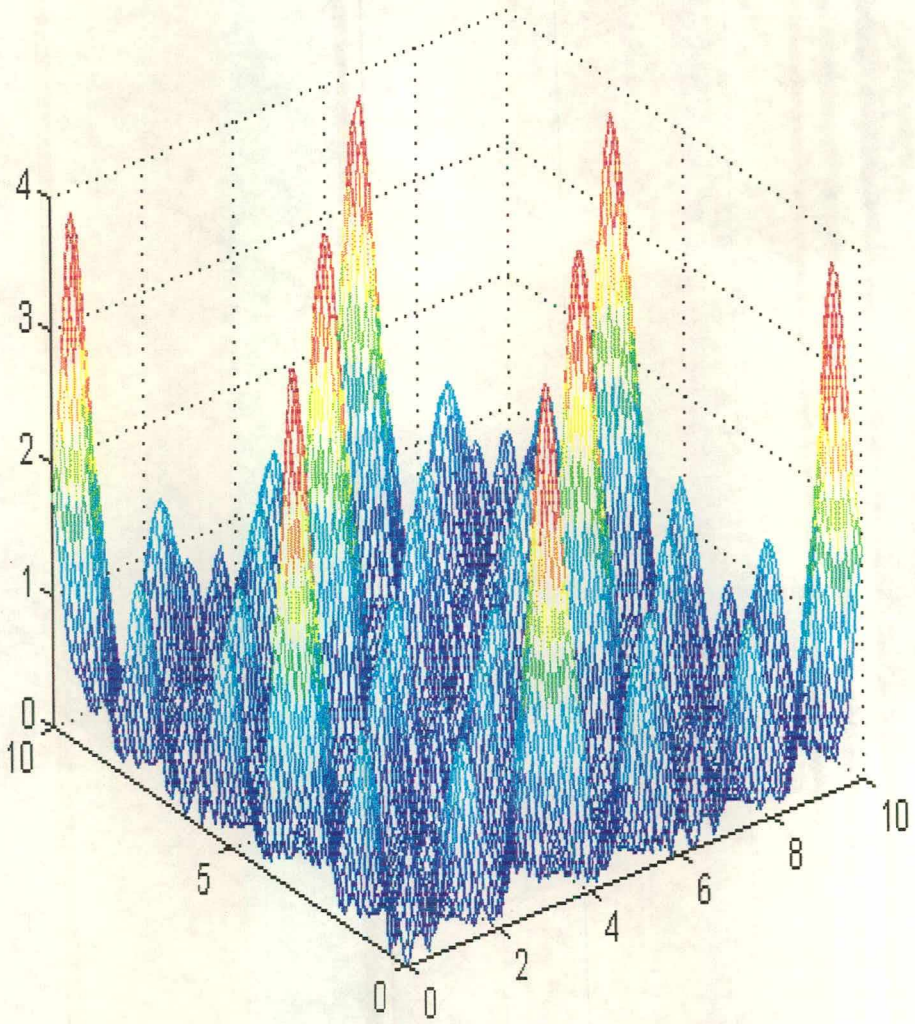


Figura 19. Superfície *BUMP*-2

Cromossomo: 40*bits*
População (***n***): 50
Gerações (g_c): 50
Pr[cross]= 90% (dois pontos de crossing-over)
Pr[mutação]= 1%
p Ótimos de Interesse= 7
Implementação: **anexo 09**

6.3.3. Função BUMP-3

$$\text{Max } z = \sum_{i=1}^{20} \textbf{abs}(\text{sen } x_i . \text{sen } x_{i+1} + 0{,}1 x_i x_{i+1} \text{ sen}(x_i + x_{i+1}) \cos(x_i + x_{i+1}) + \cos x_i \cos_{i+1})$$

com $x_i \in [0,10]$, i=1, ..., 20.

Cromossomo= 420*bits*
População (*n*)= 100
Gerações (g_c)= 100
Pr[cross]= 90%
Pr[mutação]= 1%
p Ótimos de Interesse= 420
Implementação: **anexo 10**

CAPÍTULO 7

7. OS RESULTADOS

7.1. As Medidas e o Desempenho

Objetivamente, a mensuração do desempenho de um AG não é uma questão trivial e pouco progresso tem sido relatado nesta área. Fatores a serem levados em conta são:

⇒ Velocidade: quão rapidamente (tempo de CPU ou número de avaliações da função) o algoritmo completa;
⇒ Exatidão: quão próximo(s) a(s) solução(ões) encontrada(s) está/estão da solução(ões) ótima(s);
⇒ Taxa de Sucesso: que proporção de gerações converge para uma solução ótima do que para uma solução sub-ótima e;
⇒ Consistência: quanta variação nos três critérios anteriores pode ser esperada de uma geração para a próxima.

Dependendo de como se ajustam os parâmetro de um AG, pode-se geralmente controlar incrementos na exatidão ou incremento na taxa de sucesso. Mas é difícil reunir todos estes conceitos em um único fator, levando em conta todos os sub-fatores, porque o interesse relativo destes critérios varia com a aplicação. Neste trabalho usou-se uma medida que combinou velocidade e taxa de sucesso como meios iniciais de comparação entre algoritmos. Também exatidão e consistência foram avaliados. Cada um destes termos é definido mais além.

Assumiu-se que o objetivo de executar um AG é encontrar um conjunto completo de soluções, isto é, um conjunto de todos os máximos de interesse de uma função. Para qualquer exemplo usando qualquer método de solução do problema, sempre há uma possibilidade de que um conjunto solução completo não seja encontrado. Para coletar estatísticas de desempenho neste caso o algoritmo é simplesmente repetido cegamente tantas vezes quantas foram necessárias para coletar um conjunto completo de soluções, considerando as soluções encontradas em todas as repetições juntas. Esse método assumiu naturalmente que um conjunto solução é reconhecido assim que ele é encontrado. Como já foi mencionado anteriormente, após g_c gerações o melhor elemento da população é tomado como ótimo local e assume-se um nicho de raio r, que será penalizado, ao seu redor.

A Taxa de Sucesso é a proporção de sucessões que surgem com o conjunto de soluções desejados. Para baixas taxas de sucesso pode-se esperar em ter que repetir o algoritmo várias vezes antes de um conjunto completo de soluções venha a ser obtido.

Avaliações Esperadas é uma medida de quantas avaliações de cada função, em média, pode-se esperar repetir para encontrar um conjunto completo de soluções. O número de avaliações é efetuado em todas as gerações de uma sucessão. Numa eventual falha de uma sucessão em determinar uma solução, repetiu-se o algoritmo e continuou-se a contar o número de avaliações desempenhadas.

A Consistência do algoritmo é expressa em termos de desvio-padrão de uma amostra na medida das avaliações esperadas.

A exatidão foi expressa em termos do erro quadrático médio (RMS) nas soluções encontradas. O valores foram calculados determinando a distância entre cada solução produzida pelo algoritmo e a solução exata mais próxima. Esta distância é então fixada e o valor do **RMS** é a raiz quadrada da média destes valores. Esta medida dá uma indicação do erro provável numa solução específica.

7.2. Funções de Armadilha

A seguir tem-se a comparação entre um algoritmo genético tradicional, a técnica da pesquisa genética repetida por Ackley e a Técnica do Compartilhamento Sucessivo. Por serem as funções de armadilha altamente enganosas o AG tradicional e a técnica proposta por Ackley se mostraram ineficazes na localização de um ótimo global [DEB89], em relação a Técnica do Compartilhamento Sucessivo, que teve um desempenho nitidamente superior.

Algoritmo	Taxa de Sucesso	Avaliações Esperadas
Pesquisa Genética (Ackley)	0,00%	>1000000
AG Tradicional	0.01%	2.300.000
Técnica do Compartilhamento Sucessivo	77.60%	4.900

Tabela 02. Comparação de Resultados em Funções Enganosas

7.2.1. Armadilha de Dois Picos

Os resultados dos testes de Ackley no problema da armadilha usando uma técnica conhecida com pesquisa genética repetida são apresentados na **tabela 02** juntamente com um resumo dos próprios resultados na mesma tabela usando um AG tradicional repetido (para comparação) e a Técnica do Compartilhamento Sucessivo. O algoritmo de pesquisa genética de Ackley tinha uma população de 50 elementos, probabilidade de *crossing-over* 100% e probabilidade de mutação

entre 1,75% e 2,5%. Ackley realizou mais de 50 testes, mas abandonou testes adicionais tão logo o algoritmo atingia 1 milhão de avaliações. Para verificar, apenas, quão bom (ou mau) um AG era em seu problema, foram experimentados neste trabalho um AG tradicional com parâmetros mais convencionais (população =50, Pr[cross]=80% , Pr[mutação]=1%), em um número de testes muito maior. Dentre as 92 mil gerações somente 12 encontraram uma solução ótima global. Geralmente estas surgem na primeira geração, que implica que uma busca casual teria tido a mesma eficiência. Com os mesmo parâmetros básicos como o AG tradicional, o AG munido da técnica do compartilhamento sucessivo determinou a solução ótima em 194 dentro dos 250 testes, mantendo uma média de 4900 avaliações[1].

Os parâmetros básicos usados com o AG, munido da técnica do compartilhamento sucessivo, foram: n= 50, Pr[cross]= 80%, Pr[mutação] =1%, função de potência, e α =2. O raio do nicho, determinado pela equação: $r = \frac{\sqrt{k}}{2 * \sqrt[k]{p}}$, foi 5.

Uma sucessão típica obtida para encontrar quatro picos, foi a seguinte para x: **0**(11), **5**(1), **10**(1) e **20**(16). Os valores entre parêntesis são os números de gerações (típicas) nas quais as soluções foram encontradas. Os picos em 5 e 10 são artificiais que a técnica introduziu ao se utilizar a função de modificação. Estes picos artificiais, não são picos na função não modificada, e a técnica de rejeição destes picos será discutida no capítulo 8.

A quarta fase de uma sucessão geralmente localiza o máximo em x=20 após, aproximadamente, 16 gerações. Contudo, em algumas sucessões, a quarta fase foi interrompida muito cedo. Isto acontece se um elemento "muito bom", mas sub-ótimo, surge por acaso em uma geração anterior. Este surgimento aumenta a

---

[1] O método de Ackley, conhecido como SIGH, resolveu o problema da armadilha em apenas 780 avaliações. Contudo este resultado surpreendentemente bom, deu-se pelo fato de que o SIGH inclui a teoria da indução que torna-o especialmente bom neste problema específico.

média do *fitness* de uma geração anterior, que, em períodos, estimula a fase a ser encerrada mais cedo, antes de ter convergido no máximo global. Quando o AG determina um valor sub-ótimo como solução, a técnica modifica a região vizinha deste ponto, danificando o ótimo global que está neste entorno, tornando difícil encontrá-lo em fase subseqüentes.

Testes extensivos foram realizados usando a técnica para avaliar os quatro tipos de funções auxiliares de modificação, 250 sucessões foram realizadas em cada caso. Cada sucessão teve um limite superior fixado em seis fases para determinar o ótimo global. As **tabelas 03** e **04** apresentam o desempenho das medidas em cada caso. Muitos valores foram arredondados para dois dígitos significativos, para evitar passar a impressão de uma falsa precisão.

Função Auxiliar	Taxa Sucesso	Média Rodadas	Avaliações Esperadas	Margem Confiança	σ
α=0.5	15%	38.6	29000	±6.000	19000
α=1	44%	13.1	11000	±1.400	7300
α=2	78%	5.7	4900	±400	2900
α=4	79%	5.6	4700	±400	2900
α=8	73%	6.3	5200	±420	2900

Tabela 03. Avaliação das Funções Enganosas Usando a Função de Modificação de Potência

Função Auxiliar	Taxa Sucesso	Média Rodadas	Avaliações Esperadas	Margem Confiança	σ
m=0.1	38%	14.4	14000	±1.900	9500
m=0.01	79%	5.5	4900	±380	2700
m=0.001	72%	6.4	5600	±500	3400
m=0.0001	74%	6.1	5400	±530	3700

Tabela 04. Avaliação das Funções Enganosas Usando a Função de Modificação Exponencial

Os resultados mostraram, claramente, que o bom desempenho pode ser obtido de uma série de funções auxiliares de modificação, desde que a função auxiliar seja suficientemente côncava. Sugere-se na função auxiliar de potência o uso dos valor de α=2, 4 e 8. Nas funções exponenciais sugere-se m=0,01, 0,001 e 0,0001. Estatisticamente, as funções auxiliares de modificação, pouco diferem nos seus resultados em termos de número de avaliações esperadas. Para a maioria dos testes foi utilizado α=2.

Nas experiências iniciais o número de rodadas por sucessão foi fixado em 6. Foram tentados também valores de 4, 12 e 24 (**tabela 05**). Mais uma vez, aumentar máximos de fases/seqüência teve efeitos positivos e negativos. Havendo ótimos próximos entre si, isto pode levar a execução muitas fases extras para localizar o máximo global.

Assumindo um número maior de rodadas por sucessão pode levar ao aumento do número de avaliações esperadas. Por outro lado, uma "perda não tão próxima" pode ser capaz de localizar o máximo global dando algumas fases extras - que podem ser mais rápidas num todo, do que desistir e iniciar uma nova seqüência .

Máximo rod./seq.	Taxa Sucesso	Média Rodadas	Avaliações esperadas	Margem Confiança	σ
4	80%	5,0	4900	±260	1900
6	78%	5,7	4900	±400	2900
12	88%	6,3	5100	±400	3000
24	90%	7,1	6400	±1100	9100

Tabela 05. Variação das rodadas/seqüência nas funções de armadilha

As funções auxiliares de Hein, citadas em 5.1.3, como opção de modificação não foram incluídas, no relatório, pois tiveram um desempenho abaixo das funções exponenciais com α=0,5 e das de potência com m=0,1. Devido a sua concavidade ocorre a formação de picos "fictícios" próximos aos removidos.

7.2.2.Armadilha Completamente Enganosa

Usando funções de dois picos, onde o máximo local correspondia a 99,95% da altura do máximo global, após 250 testes não descobriu-se nenhuma diferença estatisticamente significante nos resultados (4700±300 avaliações). Isto não foi surpreendente pois as três primeiras fases da sucessão não encontraram dificuldades em localizar o máximo local em 0 e dois máximos ainda em 5 e 10. estes máximos foram adequadamente suprimidos pela função auxiliar, tornando o trabalho de localizar o máximo global em x=20, não mais difícil que anteriormente.

7.2.3. Armadilha de Dois Picos Centrais

Novamente usando ao técnica do compartilhamento sucessivo, não foi mais difícil do que no problema da armadilha original. Em condições similares à original, esta foi resolvida em uma média de 3000±230 avaliações.

7.3. Funções de Codificação Binária

Usou-se para as funções de **F1** a **F4** uma população de 20 elementos e para a função **F5** uma população de 30 elementos. Os outros parâmetros foram **Pr**[cross]=90%, **Pr**[mutação]=1%. As gerações foram em número de 20. O raio do nicho utilizado para as funções **F1** a **F4** foi 0,1. Para a função de *Himmelblau* foi 4,24. Os resultados das 250 sucessões (novamente arredondadas para dois dígitos) são apresentados na **tabela 06**.

Função	Taxa Sucesso	Média Rodadas	Avaliações Esperadas	Margem Confiança	σ	RMS
F1	99	5,1	1900	±74	600	0,0043
F2	90	5,6	3300	±140	1100	0,0075
F3	100	5,2	1900	±79	630	0,0039
F4	99	5,1	3000	±66	530	0,0041
F5	76	6,1	5500	±530	3700	0,20

Tabela 06. Resultado das Funções F1, F2, F3, F4 e F5

Estes resultados demonstram que, para problemas unidimensionais de **F1** a **F4**, virtualmente cada seqüência encontra 5 máximos, e pode-se esperar que todas as 5 soluções sejam encontradas em pouco mais de 5 rodadas. No problema bidimensional, **F5**, a técnica alcança uma alta taxa de sucesso, com 75% das sucessões encontrando os quatro máximos. A taxa de sucesso mais baixa aumenta o número de fases exigidas. O valor do **RMS** para **F5** aparece alto em comparação com **F1** a **F4** porque o espaço de parâmetros é maior. Todos os valores do **RMS** são menores que 10% do raio do nicho, mostrando que os múltiplos máximos de uma função podem ser localizados em algumas das fases com uma exatidão razoável.

Em adição, par **F2**, foram realizados testes com vários valores de α (**tabela 07**). Bem como nas funções de armadilha, descobriu-se que com α=0,5 a 1 os resultados não foram tão bons em relação ao uso de α=2, 4 e 8, com os quais a técnica alcançou bons resultados.

α	Taxa Sucesso	Média Rodadas	Avaliações esperadas	Margem Confiança	σ	RMS
0,5	2%	306	190000	±120000	120000	0,0090
1	75%	6,6	4000	±290	2000	0,0074
2	90%	5,6	3300	±140	1100	0,0075
4	96%	5,2	3100	±89	700	0,0070
8	100%	5,1	3000	±68	550	0,0065

Tabela 07. Resultados da variação de α na função F2

7.4. Resultados das Funções *BUMP*

As funções *bump*, apresentadas no item 6.3., constituem um teste crucial para qualquer algoritmo de otimização. Keane [KEA95a] introduziu e ilustrou as dificuldades e vantagens no uso de AGs, têmpera simulada, estratégias de evolução e programação evolucionária. Keane conclui em [KEA95b] que os AGs se mostraram os mais eficientes na determinação dos ótimos em funções *bump*.

Aproveitando-se das conclusões de Keane, foram realizadas as implementações das funções mencionadas em 6.3.1, 6.3.2 e 6.3.3. Mantendo 20 *bits* por variável, pontos de *crossing-over* na quantidade igual da dimensão do problema, probabilidade de *crossing-over* de 90%, mutação de 1%.

Não foi efetuada nenhuma avaliação mais detalhada destes modelos, contudo constatou-se ser imperativo aumentar o número de gerações devido ao grande espaço de busca e quantidade elevada de picos, possibilitando a total formação de nichos e consequente determinação do ótimo. O refinamento das soluções, no caso das funções *bump*, é aconselhável, pois o grande número de picos e conseqüente grande uso das funções auxiliares, modifica em muito a função original. Assim a retirada, do espaço de busca, de um ótimo (local ou global) deve ser realizado com grande precisão, pois uma falha neste processo pode impossibilitar a determinação de outro pico ainda não localizado.

Os resultados, da implementação dos AGs, foram checados por exaustão, com ε=0,001, na varredura da região possível.

CAPÍTULO 8

8. DISCUSSÃO E CONCLUSÃO

Os resultados, apresentados no capítulo 7, mostram que a Técnica do Compartilhamento Sucessivo consegue dar bons resultados. No caso das funções de armadilhas, geralmente consideradas serem "quase" insolúveis por um AG tradicional, a técnica localizou todos os máximos no espaço de parâmetro. Localizou também o máximo global sem dificuldade. No caso das funções de codificação binária, o algoritmo do *fitness sharing* usado por Deb [DEB89], determinou todos os máximos num tempo razoável. Infelizmente estes resultados não podem ser diretamente comparados com os de Deb, pois o objetivo do trabalho de Deb foi mostrar que o *fitness sharing* pode evoluir e manter estável subpopulações em todos os máximos de uma variedade de funções multimodais. Deb não tentou otimizar seu algoritmo com respeito a velocidade de convergência ou exatidão dos resultados.

A Técnica do Compartilhamento Sucessivo não tenta, simultaneamente, manter subpopulações em cada máximo. Ao invés disso endereça diretamente as tarefas mais úteis de localizar cada máximo. Os resultados mostram que usando a Técnica do Compartilhamento Sucessivo, os **p** picos de uma função como em **F1** a **F5** conseguem ser localizados em um pouco mais que **p** fases de um AG tradicional. A velocidade final e exatidão destas soluções depende da qualidade de implementação de um AG básico específico. Como em [DEB89], aqui foram feitas poucas tentativas para otimizar o desempenho destas medidas. Ao invés disso, os resultados demonstraram que a Técnica do Compartilhamento Sucessivo requer menos fases do que repetição cega, e a análise teórica mostrou que a complexidade de tempo é menor (logo, melhor) estando acima do algoritmo do *fitness sharing*. Finalmente, ambos os métodos: Técnica do Compartilhamento Sucessivo e o *fitness sharing* contam com uma certa regularidade no campo de

pesquisa. Eles assumem que regiões próximas aos máximos de interesse não contém outros máximos de interesse. Contudo, a validade deste posicionamento depende não somente (como seria esperado) do esquema de código usado, mas também de como definimos proximidade. Como foi apresentado na seção 5.1.2, a Técnica do Compartilhamento Sucessivo por si só não depende de qualquer métrica específica, mas o sucesso do algoritmo com certeza será afetado se uma métrica adequada for usada. Esta conclusão aplica-se igualmente para o algoritmo do *fitness sharing*.

8.1 Problemas Com o Raio dos Nichos

Se um raio inadequado é empregado para o nicho, surgem problemas. Para simplificar, aqui não se usou a hipótese de Deb [DEB89], que tratava de que os máximos estariam igualmente distribuídos por todo o campo de pesquisa. Esta hipótese leva a equação do cálculo do raio a um valor único para o raio do nicho que é usado para todos os nichos nos problema. Em geral os máximos não estarão igualmente distribuídos, logo subestimou-se o tamanho de alguns nichos e superestimou-se o tamanho de outros. O problema ao qual isto conduz também ocorre com o *fitness sharing* .

8.1.1 Nicho Pequeno

A modificação da função objetivo pelas funções auxiliares faz surgir novos picos. Este efeito modificador será mínimo se o raio do nicho for muito pequeno, ou se uma função auxiliar côncava for ineficientemente usada (por exemplo $\alpha<2$ ou as funções auxiliares de Hein). O resultado disto fará com que fases extras, possivelmente, sejam exigidas para pesquisar e remover os novos picos criados. Um problema similar esta ilustrado na **Figura 21**. A função original dada por $f(x)=sen^2(2\pi x^2)$ (**Figura 20**), dando dois picos de igual altura mas largura desproporcional. Usando a equação de cálculo do raio do nicho, resulta em r=0,25.

Quando a função auxiliar, como α=2, é aplicada a um pico mais amplo, dois novos picos são introduzidos , próximo a x=0,3 e x=0,6. Em problemas com dimensionalidade mais alta, muito mais do que dois picos podem ser introduzidos. Os picos introduzidos geralmente estarão a uma distância um pouco menos do que o raio do nicho da solução encontrada. Não importa que tipo de função auxiliar é usada (potência, exponencial ou as de Hein), sempre serão introduzidos desta maneira.

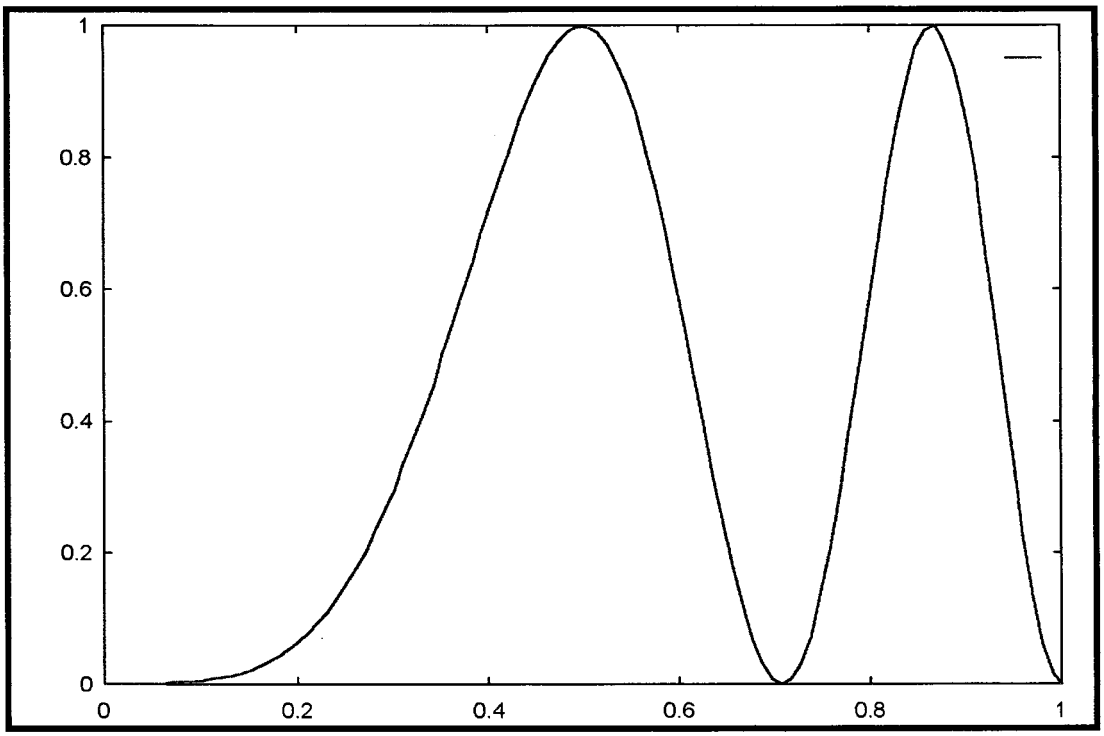


Figura 20. Função Original

8.1.2. Nicho Grande

A superestimação de um raio de nicho também causa problemas. A **Figura 21** mostra a mesma função como na **Figura 20**, mas com o raio de nicho duplicado. O uso de um raio de nicho maior, com α=2 os picos introduzidos estão ainda em evidência. Pior do que isto, o pico adjacente foi afetado. Sua altura foi significantemente reduzida, e a sua posição deslocou um pouco. A redução da altura pode levar um pico a não ser reconhecido como a solução se a solução limiar for fixada muito alta.

Com um valor mais alto de α, o efeito no pico adjacente e ainda mais drástico . Este efeito demonstra porque não se deve usar um valor de α muito maior, pois com isso arrisca-se perde máximos de interesse ou obtendo uma solução altamente incorreta caso o raio do nicho for superestimado.

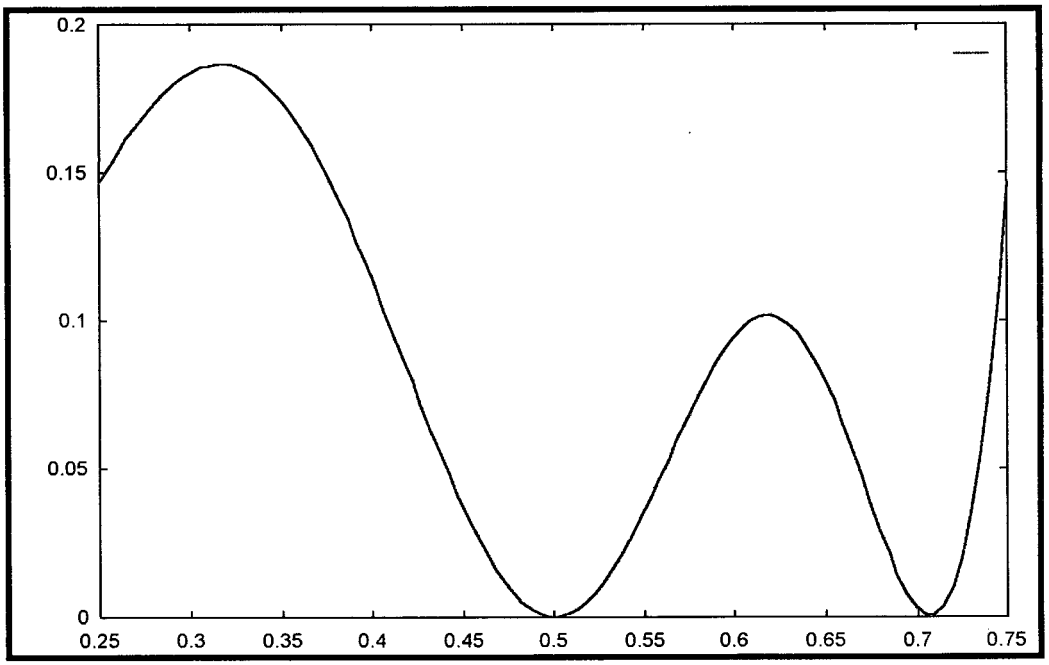


Figura 21. Função modificada, r=0,25 e α=2

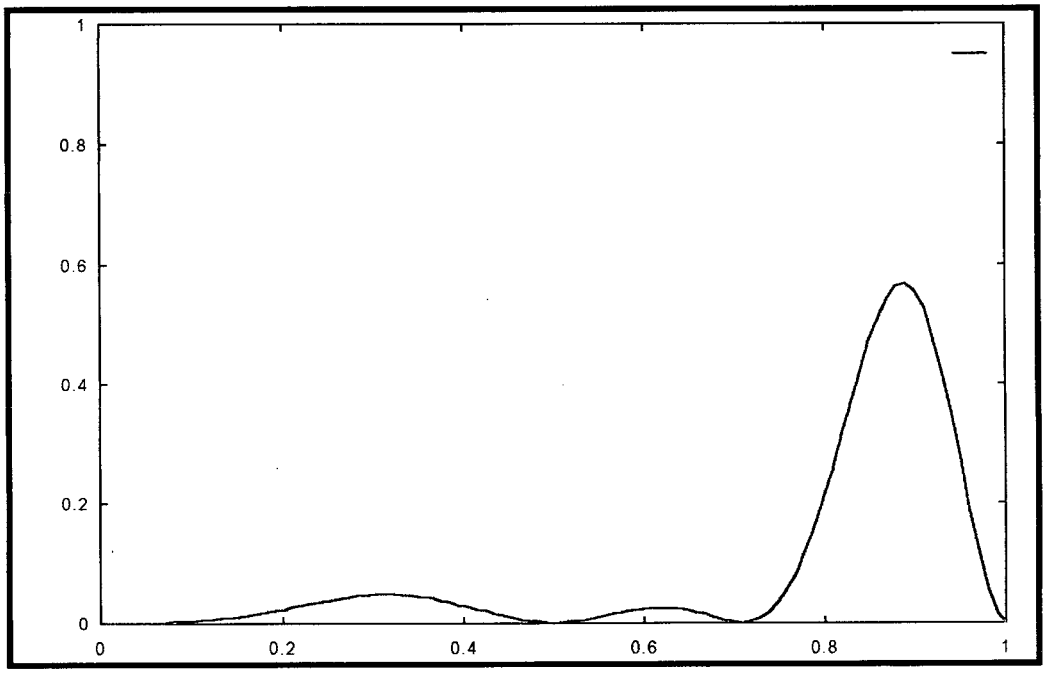


Figura 22. Função Modificada, r=0,5 e α=2

8.1.3. Soluções

Felizmente, soluções razoáveis, para ambos problemas, podem ser criadas. Os picos introduzidos, freqüentemente, são muito menores do que picos verdadeiros, e seus efeitos não serão percebidos. Se eles são de tamanho moderado, eles necessitam de fases adicionais na seqüência de suprimi-los. Em casos extremos, um pico introduzido pode ser grande o suficiente para ser confundido por um pico verdadeiro. Tais "soluções" falsas não são difíceis de detectar. Primeiro, elas estão localizadas a uma distância do pico verdadeiro que é sempre igual, ou ligeiramente menor do que o raio do nicho. Segundo, desempenhar uma técnica de pesquisa local, tal como a Otimização Natural com a função original (não-modificada), levará a uma falsa solução a convergir numa solução verdadeira.

O emprego de uma técnica de pesquisa local também supera a inexatidão apresentada pelo deslocamento da posição de um pico adjacente quando um raio de nicho muito grande é usado. A única solução para perda de picos através do emprego de raio muito grande é usar um raio menor de nicho. De fato, a melhor abordagem para todos estes problemas é obter um raio de nicho correto. Esta abordagem será discutida com detalhas na seção 8.3.

Problemas similares surgem com o *fitness sharing* quando o raio do nicho é muito grande ou pequeno, mas as soluções neste caso podem não ser tão simples. Por exemplo, se indivíduos estão presos em picos introduzidos por raio muito pequenos, uma população maior seria exigida para preencher os "nichos" adicionais. Isto não é fácil de detectar ou implementar. Com a Técnica do Compartilhamento Sucessivo, contudo, estes "nichos" adicionais são automaticamente preenchidos pela fases complementares.

8.2. Outros Problemas

Vários outros problemas podem ser identificados no momento da implementação, principalmente relacionados aos tipos de problemas que afligem os AGs em geral. Estes problemas fizeram surgir três sintomas.

- Imprecisão: soluções não são completamente seguras;
- Não finalização (incompleto): Seqüências as vezes fracassam em encontrar todas as soluções;
- Fases extras: mais fases são freqüentemente exigidas do que o número de soluções.

As causas e soluções com esses sintomas serão consideradas abaixo.

8.2.1. Imprecisão

A imprecisão é um problema padrão em todos os AGs tradicionais [JONG85]. Isto estimula o uso dos Algoritmos Meméticos, ou seja, os AGs híbridos, no qual um AG encontra soluções aproximadas, então usando uma técnica de pesquisa local, tal como a escalada de encostas (empregando a função de propriedade original) para dar um resultado mais preciso [MOS95]. Este problema também surge e pode ser resolvido da mesma forma se um raio de nicho muito grande é usado, ou se a função auxiliar é muito côncava, como discutido na seção 8.1.2.

8.2.2. Não Finalização

A não finalização é principalmente um efeito colateral de localizar soluções muito imprecisas. A dificuldades de exatamente definir quando parar uma determinada fase (veja seção 3.4) significa que ocasionalmente uma fase é interrompida antes de ter propriamente convergido. Isto geralmente leva a uma localização determinada imprecisa de um máximo. Freqüentemente o pico não será considerado como uma solução porque o *fitness* neste ponto está abaixo da

limiar de solução. Mas esta solução imprecisa está provavelmente dentro do nicho, assim o efeito da modificação origina um *fitness* fictício no pico verdadeiro a ser reduzido. Isto torna o pico verdadeiro menos provável de ser encontrado em fases subsequentes, logo a seqüência pode alcançar seu limite de fases máximas superiores sem localizar o verdadeiro pico.

Há duas soluções para este problema. Primeiro pode-se tentar evitar interromper fases antes de elas ter convergido. Quanto mais conservador se for em julgar a convergência, mais aumenta-se o número médio de gerações necessárias por fase. O efeito global pode ser aumentar o número médio de avaliações exigidas. As vezes é melhor ter um algoritmo que talvez falhe 25% das vezes mas é rápido, do que ter um que sempre tem sucesso mas é muito mais lento. A segunda solução é a mesma solução que a do problema de imprecisão: usar uma técnica de pesquisa local. Estas abordagem solucionam ambos os problemas de uma só vez, sem necessariamente adicionar muito ao número de avaliações exigidas.

8.2.3. Fases Extras

Há três motivos para fases extras. Primeiro, fases extras podem ser causadas pela convergência em máximos locais não desejados na função original. Fases adicionais nem sempre localizam um máximo com um *fitness* acima de limiar de interesse (logo o pico encontrado não conta como a solução). Convergência em máximo locais, novamente, é um problema padrão em todos os AGs; está para ser descoberto um AG que não tenha esta falha, assim, fazer uso do conhecimento que se obteve ao localizar um máximo local é valioso. A Técnica do Compartilhamento Sucessivo faz exatamente isto e assegura que fases subsequentes na mesma seqüência não convirjam novamente nos mesmos máximos locais já, anteriormente, encontrados. Desse modo uma deficiência de um AG tradicional torna-se imperceptível com o uso da técnica descrita. É claro que não é a forma mais eficiente de encontrar todos os máximos de interesse, mas

pelo menos pode-se estar certo de que fases sucessivas em uma seqüência tem um risco decrescido de ficar preso em máximo sem interesse da função original.

A segunda causa é que a fase pode convergir em um máximo local na função de objetivo modificada, como foi discutido na seção 8.1.1.

A terceira causa é a não conclusão exigindo iterações extras no algoritmo. Este efeito seria eliminado se o problema da não conclusão fosse resolvido.

8.3. Trabalhos Futuros sobre o Tamanho do Nicho

De modo geral, é impossível escolher um valor único do raio do nicho que será correto para todos os máximos de interesse, uma vez que nichos (isto é, máximos) em um problema não serão em geral todos do mesmo tamanho. Para adequadamente discutir a questão do raio do nicho, precisa-se permitir que o raio do nicho seja dimensionado individualmente para cada máximo.

Para o *fitness sharing* de Deb [DEB89], assumiu que o tamanho do nicho, forma e distribuição da população devem ser feitos com antecedência. Ele assume que todos os nichos são do mesmo tamanho e forma (hiperesferica) e que os nichos estão igualmente distribuídos por todo o espaço de pesquisa. Naturalmente, estas são hipótese razoáveis quando não se sabe nada sobre a topografia do espaço de pesquisa.

Uma vantagem da técnica do compartilhamento sucessivo, contudo, é que a informação sobre a topografia da função de propriedade é fortemente intensificada de fase em fase. Pode-se iniciar com certas hipóteses, mas não é necessário usa-las do princípio ao fim.

Ao invés de assumir um tamanho e forma estabelecida para todos os nichos, um novo algoritmo, melhor, permitiria parâmetros específicos serem armazenados para cada nicho. Ao invés de assumir que cada nicho seja hiperesférico, uma forma hiperelipsoidal seria mais comum em dando em contrapartida um "raio" diferente em cada dimensão. Como cada pico esta descoberto, seu tamanho, em cada dimensão, poderia ser estimado e registrado juntamente com o pico. Esta abordagem, supõe-se, tornaria o algoritmo mais rápido e mais preciso.

8.4. Conclusão

A tese descreveu uma nova técnica para localizar todos os máximos de uma função multimodal. Ela envolve múltiplas fases de um AG, cada um encontrando um máximo. Após cada máximo ter sido encontrado, a função objetivo é modificada para evitar que fases subseqüentes re-pesquisem regiões próximas da área problema.

Uma vantagem do compartilhamento sucessivo é que ela representa uma extensão simples para técnicas de otimização unimodais existentes. É uma técnica complementar que pode ser usada com qualquer tipo de AG, ou com outras técnicas, tal como a Têmpera Simulada. Comparado com as repetições simples (cegas) a técnica do compartilhamento sucessivo é capaz de evitar redescoberta de soluções que já tenham sido encontradas. Isto torna a técnica do compartilhamento sucssivo mais rápida por um fator de pelo menos $\log p$, onde p é o número de picos na função.

O *fitness sharing*, técnica introduzida por Goldberg e Richardson e investigada mais tarde por Deb, resultou ser problemática, pois caso a função tenha muitos picos, exige-se uma grande população para o AG. Em adição, o simples conceito de um raio de nicho, de valor único, limita a aplicação *fitness*

sharing em problemas onde os máximos estão igualmente espalhados por toda a área de pesquisa.

A Técnica do Compartilhamento Sucessivo que hora se apresentou obteve um bom desempenho nas funções que Deb usou para testar seu fitness sharing. Embora ele sofra de problemas similares àqueles do fitness sharing, populações menores podem ser usadas. Isto reduz a complexidade de tempo - por um fator aproximadamente 10(1+log **p**/log ***n***) quando **p** é grande, assumindo que o número de gerações necessárias para a convergência é ***O***(log ***n***) (onde ***n*** é o tamanho da população e **p** é o número de picos na função objetivo).

Com a exploração das funções auxiliares, as descobertas parecem ser relevantes também para o *fitness sharing*. A forma exata da curva auxiliar não é importante, o único fator essencial é que ela deva ser razoavelmente côncava. A concavidade é exigida para evitar que quaisquer picos introduzidos se tornarem muito grandes. O raio do nicho usado é importante por motivos similares.

A técnica do compartilhamento sucessivo **prova** ser um método útil quando aplicado na solução de problemas práticos que requerem a localização de ótimos múltiplos de funções multimodais.

APÊNDICE

ALGORITMOS GENÉTICOS TRADICIONAL

Um algoritmo genético (AG) pode ser descrito como um mecanismo que imita a evolução genética de uma espécie. Uma vantagem óbvia é o paralelismo intrínsico, mas ele vai mais adiante, deixando soluções evoluírem independentemente em paralelo: interações fazem soluções, ou seja, mistura um elemento com outro e produz "filhos" que esperançosamente conservam as melhores características de seus parentes. Os AGs podem ser vistos como uma forma de busca local. Em um senso mais generalizado, pode-se dizer que a solução, de um AG, está na proximidade de uma solução ótima, que pode ser explorada mais na proximidade por um conjunto de populações: isto é um tanto diferente da união das aproximações individuais obrigadas pelas interações. O principal operador usado para gerar e explorar a proximidade de um população e selecionar um a nova geração são: a seleção, *crossing-over*, e mutação. Estes operadores estão descritos na tese. Note-se que a literatura dos AGs é rica em termos emprestados da genética, as vezes com um pouco de pedantismo: aqui limita-se o uso do jargão da genética ao mínimo.

A primeira peculiaridade do AG é que os operadores genéticos não fazem operações, diretamente, no espaço da solução: soluções tem que ser codificadas com limite de tamanho de cadeias (*strings*). Desta forma há pequenas diferenças com a otimização clássica, comuns nas mesmas situações, como problemas de programação matemática com variáveis 0 e 1 (zero e um). Assim o código natural de uma solução é uma *string* de *bits*, contendo os valores de cada uma das variáveis booleanas em alguma ordem predefinida, lembrando porém, que nem sempre o código mais fácil de entender, é o mais apropriado. A partir de agora nas estruturas dos AGs, quando nós estiver colocado "solução", entenda-se "código de representação

de uma solução". Nas literaturas dos AGs a representação da cadeia de uma solução é chamada de "cromossomo". A característica associada com cada coordenada da cadeia é um "gene" e cada posição na cadeia é um "*locus*" (nos casos simples, cada gene é associado com um *locus*.

Um AG começa com uma população inicial, aleatoriamente escolhida, com N soluções e se reproduz formando novas populações (filhos) do mesmo tamanho N em cada iteração. Depois de (n+1) iterações é obtida a $X^{(n)}$ geração (enésima geração). As soluções geração-a-geração são obtidas fazendo-se, a cada iteração há cruzamentos (crossing-over), entre os pares de cromossomos geneticamente mais aptos, e sujeitando os filhos produzidos a mutações genéticas. Esta prole substituirá os atuais indivíduos na população corrente, podendo piorá-la em algum momento, porém, comprovadamente irá melhorá-la com o passar das iterações. A mutação, é geralmente produzida em uma pequena porção dos filhos. Entrando em maiores detalhes, pode-se dizer que, cada solução corrente da população $\{x_1, x_2, \ldots, x_n\}$ é avaliada pela grau de aptidão (*fitness*), que pode ser simplesmente o seu valor dentro da função objetivo, no caso de uma maximização de um problema de programação matemática.

A seleção dos melhores indivíduos de uma dada população é feita de acordo com suas aptidões, porém não por um meio de um caminho determinístico: os casais de soluções são extraídos aleatoriamente, para que se recombinem, trocando as população corrente por uma nova que, probabilisticamente, tem sua aptidão aumentada. As chances de um elemento (solução) ser um reprodutor ou não, pode der dado pela expressão:

$$\frac{F(x_i) - F_{min}}{\sum_{j=1}^{n} [F(x_i) - F_{min}]}$$

onde $F_{min} = \min\{F(x_j), j = 1,2,\ldots,n\}$.

Outra maneira de se calcular as chances de reprodução de cada indivíduo é pela freqüência relativa para todo x_i, com i=1,2,...,n.

Pares dos indivíduos selecionados são então submetidos (com alguma probabilidade $\aleph$) para a operação de *crossing-over*. A maneira mais comum de uso do *crossing-over* é o chamado "2-ponto de *crossover*", que é aplicado como segue:

Consideremos duas soluções correntes codificadas em *string* de *bits* de tamanho 8, e que os seguintes pares de indivíduos são selecionados.

0 1 0 1 1 0 0 1
1 1 0 0 0 1 1 0

Sejam as posições: 3ª e a 5ª, como as escolhidas ao acaso e os caracteres entre estas posições (inclusive) são permutados, produzindo duas soluções (dois filhos). Em nosso exemplo, permutamos os caracteres nas 3ª, 4ª e 5ª posições em ambas cadeias (*strings*).

0 1 | 0 1 1 | 0 0 1 0 1 | 0 0 0 | 0 0 1

➔

1 1 | 0 0 0 | 1 1 0 1 1 | 0 1 1 | 1 1 0

Cada filho é então submetido a mutações, com uma probabilidade μ. O simples operador de mutação consiste em escolher a posição ao acaso e substituir o caracter na posição por outro caracter do nosso alfabeto: {0,1}. Por exemplo, trabalhando com uma *string* de *bits* de 8 posições:

0 1 0 1 1 **0** 0 1

e fazendo uma mutação na sexta posição produziremos:

0 1 0 1 1 **1** 0 1

O passo final na geração de uma nova população é a substituição dos "maus" indivíduos da população corrente, pelos filhos.

Feito isto temos a nova geração, e o algoritmo geralmente para após um número de gerações pré-definido.

UM EXEMPLO DE ALGORITMO GENÉTICO HAPLÓIDE

Como um sumário, apresenta-se uma descrição esquemática de um típico AG. Note que a função F (aptidões) estará sendo maximizada em um espaço X de soluções codificadas.

Muitas variações do esquema básico podem ser encontradas na literatura, porém o utilizado na tese é este:

Algoritmo Genético **(AG)**

INICIALIZAÇÃO:

Selecione uma população inicial : $X^{(1)} = \{x_1^{(1)}, x_2^{(1)}, \ldots, x_N^{(1)}\} \subseteq X$

A partir de agora repetir os passos: abaixo n vezes (n=1,2,...), e $X^{(n)}$ indica a população corrente.

PASSO 1) Selecione aleatoriamente elementos em $X^{(1)}$, para a formação de 2M casais (costumeiramente N=2M), porém concedendo aos elementos de melhor adaptação, chances maiores de reprodução.

Sejam $\{y_j, j = 1,2,...,2M\}$, os 2M indivíduos escolhidos de $X^{(n)}$.

PASSO 2) **Crossing-Over**

Para k=1,...,M, o operador de crossing-over é aplicado aos pares (y_{2k}, y_{2k+1}), nas posições P_1 e P_2 (podendo ser $P_1 = P_2$), com probabilidade $\aleph$: produzindo M pares de filhos (z_{2k}, z_{2k+1}), que são idênticos aos seus parentes com probabilidade $(1-\aleph)$.

PASSO 3) **Mutação**

Para j=1,...,2M, o operador de mutação e aplicado ao elemento z_j com probabilidade μ, porém o gene mutante é escolhido aleatoriamente: esta produção de 2M mutações possíveis, gera os filhos w_j,j=1,...,2M , que são idênticos aos elementos z_j com probabilidade (1-μ).

PASSO 4) Substituição dos maus indivíduos:

Trocar os 2M elementos da geração n: $X^{(1)} = \{x_1^{(1)}, x_2^{(1)}, \ldots, x_N^{(1)}\} \subseteq X$, pelos 2M indivíduos $W^{(1)} = \{w_1^{(1)}, w_2^{(1)}, \ldots, w_N^{(1)}\} \subseteq X$, que são na verdade nossa nova geração, ou seja $X^{(2)} = \{x_1^{(2)}, x_2^{(2)}, \ldots, x_N^{(2)}\} \subseteq X$.

CRITÉRIO DE PARADA

Se n for igual ao número de iterações inicialmente fixado, então pare, caso contrário continue repitindo os passos de 1 a 4.

4. UM EXEMPLO DIDÁTICO

ilustrou-se acima um conjunto de complicados procedimentos, porém com o uso de em um simples exemplo emprestado de (GOL89, p. 14 sq.]

apresenta-se uma simulação de um AG simples, usando como representação cromossômica de *strings* binárias, ou seja, os valores 0 (zero) e 1 (um).

Problema: Maximizar $F(x) = x^2, x \in [0,31] \subset Z$

Codificando o Parâmetro

Como já se comentou, o códico a ser usado é o binário, como o intervalo sugere valores entre 0 e 31, logo verica-se que possuímos no máximo um cromossomo de comprimento igual a cinco. Como chegou-se a isso:

31/2= 15; resto **1**

15/2= 7; resto **1**

7/2= 3; resto **1**

3/2= **1**; resto **1**

Daqui concluímos que $(31)_{10} = (11111)_2$

Prova real: $1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 31$

INICIALIZAÇÃO

Por motivos didáticos escolheu-se uma população inicial de 6 elementos. Para obter-se uma população inicial fez-se, uso de uma seqüência de número aleatórios, e dela estabeleceu-se a seguinte regra:

$$n < 0.5 \Rightarrow bit = 0$$
$$n > 0.5 \Rightarrow bit = 1$$

Assim se compôs a população inicial, tomando os 30 primeiros números da seqüência (o artigo usa o quadro em linha) de números aleatórios expressa a seguir:

0.3470	0.9933	0.6565	0.8638	0.2744	0.0417	0.9026	0.8576	0.7662	0.6435
0.2777	0.8186	0.2834	0.2119	0.1372	0.7281	0.5025	0.7263	0.1118	0.1496
0.8717	0.3706	0.2956	0.2487	0.7677	0.9595	0.2899	0.9815	0.8123	0.9513
0.4236	0.1802	0.7331	0.5744	0.5426	0.3385	0.0712	0.6438	0.0224	0.5869
0.4916	0.3022	0.4115	0.5958	0.7397	0.9883	0.2121	0.1685	0.0400	0.6176
0.7209	0.1307	0.9836	0.2097	0.1009	0.3779	0.2026	0.4450	0.1065	0.1896
0.3771	0.8883	0.1345	0.1688	0.3716	0.8598	0.5210	0.4175	0.5493	0.1917
0.2586	0.8066	0.5592	0.3518	0.5529	0.4498	0.1436	0.6700	0.8932	0.7413
0.8636	0.6518	0.5569	0.5662	0.0886	0.6130	0.7201	0.5431	0.6464	0.1199

i	cromossomo
1	**0 1 1 1 0**
2	**0 1 1 1 1**
3	**0 1 0 0 0**
4	**1 1 1 0 0**
5	**0 0 0 1 1**
6	**0 1 0 1 1**

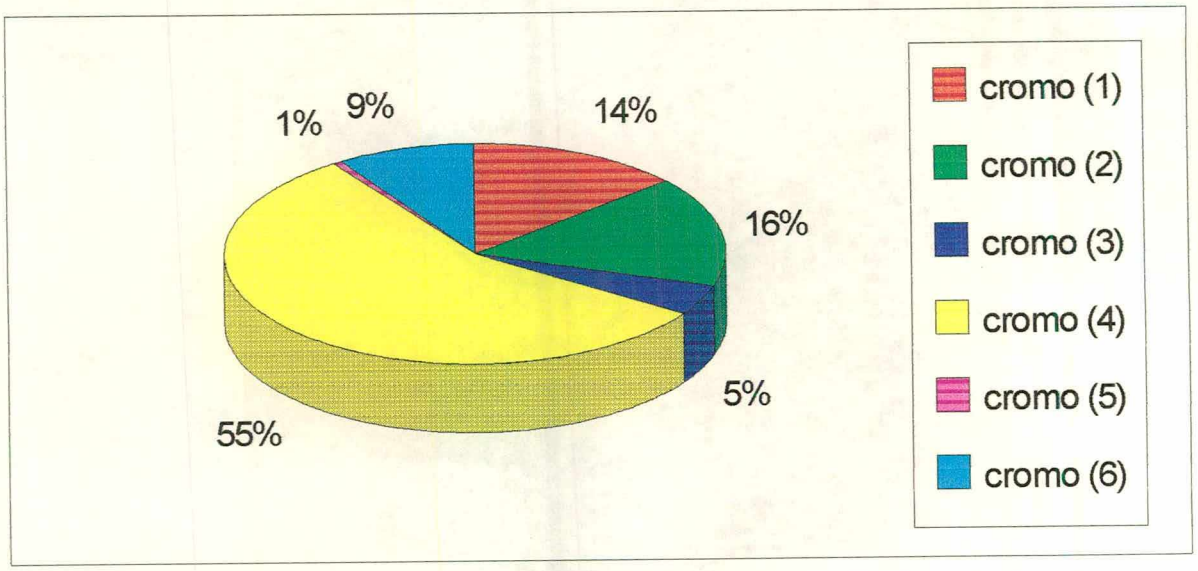
PASSO 1

A reprodução é um processo na qual as *strings* são avaliadas pelo seu grau de adaptação (*fitness*), dada pelos valores substituídos na função objetivo. Intuitivamente, pode-se pensar na função F(x) como alguma medida de proveito, utilidade ou firmeza que desejamos maximizar. Assim, quanto maior o grau de adaptação ao meio de uma dada palavra, maior será sua probabilidade de participar da reprodução. Esta operação é claro, é uma versão da seleção natural, que segue a Lei de Darwin sobre a sobrevivência dos mais aptos. Veja o quadro abaixo:

i	genótipo	fenótipo	*fitness*	proporção	acumulado
1	0 1 1 1 0	14	196	14%	14%
2	0 1 1 1 1	15	225	16%	30%
3	0 1 0 0 0	8	64	5%	35%
4	1 1 1 0 0	28	784	56%	91%
5	0 0 0 1 1	3	9	0%	91%
6	0 1 0 1 1	11	121	9%	100%
soma			1399		
média			**233.17**		
máximo			784		

Determina-ses agora "quem-cruzará-com-quem". Isto é feito usando a seqüência de números aleatórios, já utilizada anteriormente.

As chances de um cromossomo ser escolhido para formar "par" com outro (podendo ser consigo mesmo), são as seguintes:



Escolhendo números aleatórios:

(i) Se o número aleatoriamente escolhido estiver no intervalo [0,0.14), então o cromossomo 1 é reprodutor;

(ii) Se o número aleatoriamente escolhido estiver no intervalo [0.14, 0.30), então o cromossomo 2 é reprodutor;

(iii) Se o número aleatoriamente escolhido estiver no intervalo [0.30,0.35), então o cromossomo 3 é reprodutor;

(iv) Se o número aleatoriamente escolhido estiver no intervalo [0.35, 0.90), então o cromossomo 4 é reprodutor;

(v) Se o número aleatoriamente escolhido estiver no intervalo [0.90,0.91), então o cromossomo 5 é reprodutor;

(vi) Se o número aleatoriamente escolhido estiver no intervalo [0.91,1.00], então o cromossomo 6 é reprodutor.

PASSOS 2-3-4

Consideremos a probabilidade de *crossing-over*: 100% e a probabilidade de mutação como sendo 0%, isto para facilitar a análise. Em caso de valores diferentes, devemos recorrer a tabela de números aleatórios, para obter a confirmação ou não de cruzamento, e ainda, se há ou não mutação na prole. É útil também saber que, costumeiramente, a taxa de cruzamento é alta: 70%, 80%, 90%,... . A taxa de mutação é, geralmente, baixa: 0.5%, 1%, 2%,... . O algoritmo pode ainda ser construído, de modo, que estas taxas variem geração-a-geração, o que da ao algoritmo uma boa dinâmica.

Observação: A mutação em sistemas genéticos artificiais proteje contra uma eventual perda de material genético, como por exemplo: ficar preso dentro de um ótimo local, trabalhar unicamente com números pares (ou ímpares), etc.

Quanto ao *crossing-over* vamos estabelecer apenas um local de ocorrência, fazendo:

$0 \leq$ número aleatório < 0.25 primeira posição

$0.25 \leq$ número aleatório < 0.5 segunda posição

$0.5 \leq$ número aleatório < 0.75 terceira posição

$0.75 \leq$ número aleatório ≤ 1 quarta posição

	genitores	*crossing-over*	prole	fenótipo	adaptação
1	4 e 4	posição 2	**1 1** | ***1 0 0***	28	784
2	4 e 4	posição 2	***1 1*** | **1 0 0**	28	784
3	2 e 4	posição 4	**0 1 1 1** | ***0***	14	196
4	2 e 4	posição 4	***1 1 1 0*** | **1**	29	841
5	2 e 6	posição 4	**0 1 1 1** | ***1***	15	225
6	2 e 6	posição 4	***0 1 0 1*** | **1**	11	121
média					**491.83**
máx.					841

Observe-se que a média progrediu de 233.17 para 491.83, o que mostra que a nova população é geneticamente mais apta, do que a anterior.

ANEXO 01

Os programas computacionais utilizados para a avaliação das funções testadas, usaram um mesmo esquema básico, variando apenas a função objetivo e o espaço de busca.

A forma de apresentação das respostas também é padrão, variando apenas em detrimento do número de soluções pedidas.

Neste sentido será feita apenas a apresentação do programa que soluciona o problema 6.1.1, usando algoritmos genéticos. Em seguida são apresentadas as soluções encontradas.

Nos problemas restantes são apresentadas somente as soluções de cada problema, visto que o esquema básico de resolução é o mesmo.

```
{

   Programa......: SGA621.pas
   Autor : Nelson Hein                    Data...: 23/08/98
}
program SGA621;
uses crt,dos;

const
   MAXPOP    = 100;          { Tamanho máximo da população }
   MAXSTRING = 30;          { Tamanho máximo do cromossomo ( maior locus )}


type
   Allele     = boolean;                          { Allele = posição do bit }

   Chromosome = array[1..MAXSTRING] of Allele;  { cadeia de alelos ( bits )
}

   Individual = record
                   Chrom   :Chromosome;       { Genótipo = string de bits }
                   X       :real;             { Fenótipo = inteiro sem sinal
}
                   Fitness :real;             { Valor da função objetivo }
                   Parent1,
                   Parent2,
                   xSite   :integer;          { pais e ponto de cruzamento }
                end;

   Population = array[1..MAXPOP] of Individual;  { uma população }
   Vet        = array[1..MAXPOP] of Real; { Resultado das Funcoes aux }

var
   GD, GM, Y, A, B, Z : integer;      { Variáveis de auxilio }
   Raio               : real;         { Valor do Parametro do raio Limite }
   VetAux             : Vet;          { Resultado das funcoes }
   OldPop, NewPop     : population;   { Duas populações não-sobrepostas}
   PopSize,                           { tamanho da população }
   LChrom,                            { comprimento do cromossomo }
   Alfa,
   IX,
   Gen,                               { variável contadora do no. de
gerações }
   nFuncao,                           { Função escolhida para penalizar }
   MaxGen             : integer;      { no. total de gerações }
   PCross,                            { probabilidade de um overcross }
   PMutation,                         { probabilidade de uma mutação }
   SumFitness         : real;         { soma das adequações de uma população
}
   NMutation, NCross  : integer;      { Número de mutações e de overcross }
   Avg, Max, Min      : real;         { Estatísticas: média, máximo, mínimo}
   FileName           : string[12];    { Nome do arquivo de saida }
   Arquivo            : File;
   Buffer             : text;         { Ponteiro para arquivo texto de saída
}
   Melhor, AuxFitNess : Real;
   Param1,
   Alfa1              : Real;
   S1                 : String;
   H,M,S,Hund         : Word;
   nPontos            : Integer;

function LeadingZero(w : Word) : String;
var
  s : String;
begin
```

```
  Str(w:0,s);
  if Length(s) = 1 then
    s := '0' + s;
  LeadingZero := s;
end;

{ Fun‡Æo de exponenciacao }
function pot( base : real; exp : Integer):real;
var i : integer;
result : real;
begin
   if exp = 0 then
      result:= sqr(base)
   else
   Begin
      if exp <> 1 then
      Begin
         result := 1.0;
         for i := 1 to exp do
            result := result*base;
      End
      Else
         result := base;
   End;
   pot := result;
end;

{ Funcao de adequa‡„o do algoritmo que retorna o Sený(2*pi*x) }
function ObjFunc( X:real ):real;
Var Potec :real;
    I :Integer;
begin
   Potec := pot(sin(5 * pi * X),6);
   for I := 1 to IX do
      if abs( X - VetAux[I] ) < Raio then
         Case nFuncao of
            1:Potec := Potec * pot(abs( (X - VetAux[I])/Raio),Alfa);
            2:Potec := Potec * Exp(ln(Param1) * (( Raio - abs( X -
VetAux[I] ))/Raio));
         End;
   ObjFunc := Potec;
end;

{
  Fun‡„o para calcular o fen¢tipo de um indiv¡duo,
  pegando a string de bits do cromossomo e retornando
  o seu valor em decimal.
  Parametros: Chrom ®- Cromossomo a ser calculado
              LBits ®- Tamanho do cromossomo
  Retorno...: fen¢tipo ( o valor em decimal )
}
function Decode( Chrom: Chromosome; LBits: integer ):real;

var
   J              :integer;        { Contador Auxiliar          }
   Accum,                          { Acumulador da adequa‡„o    }
   VlrBin         :real;           { Contador bin rio           }
begin
    Accum  := 0;
    VlrBin := 1;
    for J := 1 to LBits do
    begin
        if Chrom[j] then
           Accum := Accum + VlrBin;
        VlrBin := VlrBin / 2;
    end;
```

```
    if ((Accum > 1) or (Accum < -1)) then
       Accum := 1 / Accum;

    Decode := Accum;
end;

{
  Fun‡„o: Retornar o valor verdadeiro se valor aleat¢rio
  sorteado ocorreu com sucesso numa distribui‡„o de
  Bernoulli de probabilidade probability.
  Parametros: Probability ®- Probabilidade
  Retorno...: um valor boleano True ou Falso
}
function Flip( Probability:real ):boolean;
begin
    if ( Probability = 1.0 ) then
        Flip := true
    else
        Flip := ( random <= Probability ); { Se o n£mero aleat¢rio for
menor  }
end;                                       { igual a probabilidade retorna
.t.}


{
  Fun‡„o: Retorna um n£mero inteiro aleat¢rio, com distribui‡„o
  equiprov vel, tal que Low <= Rnd <= High.
  Parametros.: Low  ®- Limite inferior     geralmente , passado como 1
               High ®- Limite superior     geralmente , passado o Tamanho
do
                                           cromossomo
  Retorno....: Se Low >= High retorna Low sen„o,
               Retorna um inteiro qualquel
}
function Rnd( Low, High: integer ):integer;
begin
   if ( Low < High ) then
      Rnd := random( High - Low + 1 ) + Low
   else
      Rnd := Low;
end;

{
  Retorna o n£mero de um indiv¡duo selecionado na popula‡„o, sorteado em
  propor‡„o ao seu valor de adequa‡„o na popula‡„o.
  Finalidade: implementar o operador de reprodu‡„o, favorecendo os mais
  adequados.
  Parametros:  PopSize    ®- Tamanho da popula‡„o
               SumFitness ®- Soma dos valores de adequa‡„o da popula‡„o
               Pop        ®- popula‡„o de onde se far  a sele‡„o
}
function Select( PopSize:integer; SumFitness:real; Pop:population
):integer;
var
   J               : integer;
   Rand, PartSum   : real;
begin
   PartSum := 0;
   Rand    := random * SumFitness;
   J       := 0;
   repeat
      J := J + 1;
      PartSum := PartSum + Pop[J].Fitness;  { fornece a distr. acumulada }
   until ( PartSum >= Rand ) or ( J >= PopSize );

   Select := j;
```

```
end;


{
  Sorteia um n£mero aleat¢rio para ver se h  muta‡„o, de acordo com a
  probabilidade PMutation. Se houver, o valor l¢gico do alelo , trocado
  e o contador NMutation , incrementado.
  Parametros..: AlleleVal ®- Posi‡„o qualquer dentro do cromossomo
                PMutation ®- Probabilidade de ocorrer muta‡„o
                NMutation ®- Quantidade de muta‡"es j  ocorridas
  Retorno.....: o novo valor do alelo.
}
function Mutation( AlleleVal: Allele; PMutation: real;
                   var NMutation: integer ): Allele;
var
    Mutate : boolean;
begin
    Mutate := Flip( PMutation );
    if ( Mutate ) then
    begin
       NMutation := NMutation + 1;
       Mutation  := not( AlleleVal );
    end
    else
       Mutation := AlleleVal;
end;


{
  Gera dois cromossomos descendentes a partir de dois cromossomos pais,
  atrav,s do crossing-over. O local do crossover , pr,viamente sorteado.
  Durante a c¢pia dos segmentos, em cada alelo s„o tamb,m sorteadas ( e
  eventualmente efetuadas ) poss¡veis muta‡"es.
  Parametros..: Parent1   ®- Cromossomo Pai 1   ( Pai  1 )
                Parent1   ®- Cromossomo Pai 2   ( Pai  2 )
                Child1    ®- Cromossomo Filho 1 ( Descendente 1 )
                Child2    ®- Cromossomo Filho 2 ( Descendente 2 )
                LChrom    ®- Comprimento do cromossomo
                NMutation ®- N$ de muta‡"es
                JCross    ®- Variavel de aux. com o comprimento do
cromossomo
                                conte£do : Posi‡„o onde ocorreu o cruzamento
                PCross    ®- Probabilidade de Crossing-Over
                PMutation ®- Probabilidade de muta‡„o
}
procedure Crossover( var Parent1, Parent2, child1, child2:Chromosome;
                     var LChrom, nChrom, NMutation, JCross:integer;
                     var PCross, PMutation:real );
var
      j : integer;
begin

    if ( Flip( PCross ) ) then
    begin
        JCross := Rnd( 1, LChrom-1 );
        NCross := NCross+1;
    end
    else
       JCross := LChrom;

   { For para testar de um at, o JCross sorteado por RND os Genes
     verificando se ha muta‡„o.
     Se Flip( Pcross ) for False n„o havera muta‡„o.
   }
   for j:=1 to JCross do
   begin
```

```
      child1[j] := Mutation( Parent1[j], PMutation, NMutation );
      child2[j] := Mutation( Parent2[j], PMutation, NMutation );
   end;

   { For para completar os genes n„o testados acima fazendo o crossing-over
     observe que o filho 1 recebe o resto do pai 2 e vice-versa a partir do
     ponto onde haver  o crossing-over
   }
   if ( JCross <> LChrom ) then
      for j := JCross + 1 to LChrom do
      begin
          child1[j] := Mutation( Parent2[j], PMutation, NMutation );
          child2[j] := Mutation( Parent1[j], PMutation, NMutation );
      end;
end;


{
  Gera pares de descendentes para a nova popula‡„o aos pares, a partir de
  pares de pais selecionados da popula‡„o velha. Para cada descendente,
  completa as informa‡"es de adequa‡„o de acordo com o seu Gen¢tipo, quais
  s„o os pais, e em que ponto ocorreu o crossover.
}
procedure Generation;
var
   J,                      { Variável auxiliar                          }
   Mate1,                  { Individuo1 para cruzamento                 }
   Mate2,                  { Individuo2 para cruzamento                 }
   JCross   : integer;     { Posição do cruzamento "Crossing-Over"      }
begin
    J:=1;
    repeat
       Mate1 := Select( PopSize, SumFitness, OldPop );
       Mate2 := Select( PopSize, SumFitness, OldPop );
       Crossover( OldPop[Mate1].Chrom, OldPop[Mate2].Chrom,
                  NewPop[J].Chrom, NewPop[J+1].Chrom,
                  LChrom, NCross, NMutation, JCross, PCross, PMutation );

      { Atribui‡„o das demais variáveis da Nova Popula‡„o }
      with NewPop[J] do
      begin
          X       := Decode( Chrom, LChrom );
          Fitness := ObjFunc( X );
          Parent1 := Mate1;
          Parent2 := Mate2;
          xSite   := JCross;
       end;

      with NewPop[J+1] do
      begin
          X       := Decode( Chrom, LChrom );
          Fitness := ObjFunc( x );
          Parent1 := Mate1;
          Parent2 := Mate2;
          xSite   := JCross;
      end;

      inc( J, 2 )
   until ( J > PopSize );
end;

{
  Produz as seguintes estat¡sticas a partir de PopSize e Pop ( popula‡„o
  sendo considerada )
  Parametros...: PopSize    ®- Tamanho da popula‡„o
                 Max        ®- M ximo da adequa‡„o
```

```
                 Avr         ®- M,dia da adequa‡„o
                 Min         ®- M¡nimo da adequa‡„o
                 SumFitness ®- Soma de todos os valores de adequa‡„o
                 Pop         ®- Popula‡„o antiga
}
procedure Statistics( PopSize:integer; var Max, Avr, Min, SumFitness: real;
                      var Pop: population );
var
      j : integer;

begin
    SumFitness := Pop[1].fitness;
    Min        := pop[1].fitness;
    Max        := pop[1].fitness;
    for J := 2 to PopSize do
       with Pop[J] do
       begin
          SumFitness := SumFitness+Fitness;
          if ( Fitness > Max ) then
          Begin
             Max    := Fitness;
             Melhor := X;
          End;

          if ( Fitness < Min ) then
              Min := Fitness;
       end;

    Avg := ( SumFitness / PopSize );
end;


{ Procedimento de entrada dos dados }
procedure initdata;
var
    ch     :char;
    j      :integer;
begin
   clrscr;
   writeln( '  ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿' );
   writeln( '  ³  Algor¡tmo Gen,tico ¯ÄÄ> SGA 6.2.1 - Vers„o 1.0   ³' );
   writeln( '  ³  Autor........: Nelson Hein                       ³' );
   writeln( '  ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ' );
   writeln;
   writeln( '     ___________      Inicializa‡„o dos dados   ®®®®®®®®®®' );
   writeln;

   write( 'Entre nome arquivo de sa¡da............: ' );
   readln( FileName );

   write( 'Entre tamanho da popula‡„o.............: ' );
   {PopSize := 20;}
   readln( PopSize );

   writeln( 'Entre comprimento do cromossomo.........: 30' );
   LChrom := 30;
   {readln( LChrom );}

   write( 'Entre N£mero M ximo de gera‡”es........: ' );
   {MaxGen := 20;}
   readln( MaxGen );

   writeln( 'Entre probabilidade de crossing-over....: 90%' );
   PCross := 0.90;
   {readln( PCross );}
```

```
  writeln( 'Entre probabilidade de muta‡„o .........: 01%' );
  PMutation := 0.01;
  {readln( PMutation );}

  write( 'Indique o n§ de pontos de interesse ....: ' );
  readln( nPontos );

  write( 'Indique qual fun‡Æo utilizar (1,2)......: ' );
  {nFuncao := 1;}
  readln( nFuncao );

  write( 'Entre raio limite para aplicar formula..: ' );
  Readln( Raio );

  If nFuncao = 1 Then
  Begin
     Write( 'Entre Com o Valor de Alfa..............: ' );
     Readln(Alfa1);
     If Alfa1 < 1 Then
       Alfa:=0
     Else
     Begin
       str(Int(Alfa1),S1);
       val(S1 ,Alfa,Gm);
     End;
  End
  Else
  Begin
     write( 'Entre Com [ M ] para aplicar formula....: ' );
     Readln(Param1);
  End;
  writeln;
  writeln( 'Aguarde o processamento... ' );
  randomize;     { Para sempre gerar uma seq□^ncia de n§s aleat¢rios
difer.}
  NMutation := 0;    { N§ de muta‡"es ocorridas        }
  NCross    := 0;    { N§ de crossing-over ocorridos }
  FileName := FileName + '.Out';
  assign( Buffer, FileName );
  rewrite( Buffer );

end;

procedure initreport( var Buffer:text );
begin
   writeln( Buffer,' ÚÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄ¿' );
   writeln( Buffer,' ³                                                ³' );
   writeln( Buffer,' ³  Algoritmo Gen‚tico ¯¯ÄÄ> SGA621 - Vers„o 1.0  ³' );
   writeln( Buffer,' ³                                                ³' );
   writeln( Buffer,' ÀÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÄÙ' );
   writeln( Buffer );
   writeln( Buffer, '   Parƒmetros assumidos:' );
   writeln( Buffer );
   writeln( Buffer, '   Arquivo de sa¡da -----------> ', FileName, '.out'
);
   writeln( Buffer, '   Tamanho da popula‡„o -------> ', PopSize );
   writeln( Buffer, '   Comprimento do cromossoma --> ', LChrom );
   writeln( Buffer, '   N£mero m ximo de gera‡"es --> ', MaxGen );
   writeln( Buffer, '   Probabil. de crossover -----> ', PCross:5:3 );
   writeln( Buffer, '   Probabil. de muta‡„o -------> ', PMutation:5:3 );
   writeln( Buffer, '   Fun‡Æo Utilizada -----------> ', nFuncao:3 );
   writeln( Buffer, '   Valor de Alfa --------------> ', Alfa1:5:3);
   writeln( Buffer, '   Valor do Raio --------------> ', Raio:5:3);
   writeln( Buffer, '   Valor de M -----------------> ', Param1:5:3);
   writeln( Buffer, '   Pontos de Interesse --------> ', nPontos:2);
   GetTime(h,m,s,hund);
```

```
   Writeln( Buffer, '   Inicio do Processo ---------> ',LeadingZero(h),':',
            LeadingZero(m),':',LeadingZero(s),'.',LeadingZero(hund));
   writeln( Buffer );
   writeln( Buffer );
end;


{ Inicializa os dados para a primeira popula‡„o do processo. }
procedure initpop;
var
  j, i : integer;
begin
  for j := 1 to PopSize do
     with OldPop[j] do
     begin
        for i := 1 to LChrom do
            Chrom[i] := Flip( 0.5 ); { 50% chance para 1 e para 0 }
        x       := Decode( Chrom, LChrom );
        fitness := ObjFunc( x );
        Parent1 := 0;
        Parent2 := 0;
        xsite   := 0;
     end;
end;


procedure initialize( var Buffer:text );
begin
   initdata;
   {initpop;}
   statistics( PopSize, Max, Avg, Min, SumFitness, OldPop );
   initreport( Buffer );
end;


{ Escreve os Genes de um cromossomo como uma lista de 0s e 1s.}
procedure writeChrom( var Buffer:text; Chrom:Chromosome; LChrom:integer );
var
   j : integer;
begin
   for j := LChrom downto 1 do
       if ( Chrom[j] ) then
          write( Buffer, '1' )
       else
          write( Buffer, '0' );
end;

begin
   Melhor := -999;
   for Z := 1 to nPontos do
       VetAux[Z] := 0;

   IX := 0;
   initialize( Buffer );
   for Z := 1 to nPontos do
   begin
      initpop;
      statistics( PopSize, Max, Avg, Min, SumFitness, OldPop );
      Gen := 0;
      repeat
         Gen := Gen + 1;
         Generation;
         statistics( PopSize, Max, Avg, Min, SumFitness, NewPop );
         OldPop := NewPop;   { avan‡a uma gera‡„o }
      until ( Gen >= MaxGen );
      writeln( 'Concluiu. Veja resultados no arquivo de sa¡da.',Z:2 );
```

```
      IX := IX + 1;
      VetAux[IX] := Melhor; { Guardar o melhor resultado das Gera‡äes 'A'}
   End;
   For Z := 1 to nPontos do
   Begin
      Raio := pot(sin(5 * pi * VetAux[Z]),6);
      Writeln( Buffer, 'Resultado ',Z:2,'  ',VetAux[Z]:5:10, ' -- Fun‡Æo
Objetivo.: ',Raio:3:10 );
   end;
   GetTime(h,m,s,hund);
   Writeln(Buffer,'Termino.: ',LeadingZero(h),':',
            LeadingZero(m),':',LeadingZero(s),
            '.',LeadingZero(hund));
   close( Buffer );
end.
```

RELATÓRIO PROGRAMA 611.PAS

Algoritmo Genético SGA611

Parâmetros assumidos:

```
Arquivo de saída -----------> 611.Out
Tamanho da população -------> 50
Comprimento do cromossoma --> 30
Número máximo de gerações --> 30
Probabil. de crossover -----> 0.900
Probabil. de mutação -------> 0.010
Função Utilizada ----------->   1
Valor de Alfa --------------> 2.000
Valor do Raio --------------> 5.000
Valor de M -----------------> 0.000
Pontos de Interesse --------->  4
Inicio do Processo ---------> 15:18:24.55
```

```
Resultado  1  19.92187 -  196.87462
Resultado  2   0.50245 -  154.64050
Resultado  3   5.50780 -  101.25010
Resultado  4  10.58593 -   47.08344
```

ANEXO 02

RELATÓRIO DO PROGRAMA 612.PAS

```
Algoritmo Genético SGA612

   Parâmetros assumidos:

   Arquivo de saída -----------> 612.Out
   Tamanho da população -------> 50
   Comprimento do cromossoma --> 30
   Número máximo de gerações --> 30
   Probabil. de crossover -----> 0.900
   Probabil. de mutação -------> 0.010
   Função Utilizada ----------->   2
   Valor de Alfa --------------> 0.000
   Valor do Raio --------------> 5.000
   Valor de M -----------------> 0.010
   Pontos de Interesse --------->  4
   Inicio do Processo ---------> 15:30:22.15


Resultado  1  19.95163 -  198.06522
Resultado  2   9.96036 -  159.36575
Resultado  3   4.96112 -   79.37792
Resultado  4  16.25670 -   50.26808
```

ANEXO 03

RELATÓRIO DO PROGRAMA 621.PAS

```
Algoritmo Genético SGA621.PAS

   Parâmetros assumidos:

   Arquivo de saída -----------> 621.Out
   Tamanho da população -------> 50
   Comprimento do cromossoma --> 30
   Número máximo de gerações --> 30
   Probabil. de crossover -----> 0.900
   Probabil. de mutação -------> 0.010
   Função Utilizada ----------->  1
   Valor de Alfa --------------> 4.000
   Valor do Raio --------------> 0.100
   Valor de M -----------------> 0.000
   Pontos de Interesse --------->  5
   Inicio do Processo ---------> 15:31:00.59


Resultado  1  0.7000048906 -- Função Objetivo.: 0.9999999823
Resultado  2  0.4998649713 -- Função Objetivo.: 0.9999865038
Resultado  3  0.9012110726 -- Função Objetivo.: 0.9989148449
Resultado  4  0.2985031735 -- Função Objetivo.: 0.9983427660
Resultado  5  0.1003658343 -- Função Objetivo.: 0.9999009371
Termino.: 15:31:12.35
```

ANEXO 04

RELATÓRIO DO PROGRAMA 622.PAS

```
Algoritmo Genético  SGA622.PAS


  Arquivo de saída -----------> 622.Out
  Tamanho da população -------> 50
  Comprimento do cromossoma --> 30
  Número máximo de gerações --> 30
  Probabil. de crossover -----> 0.900
  Probabil. de mutação -------> 0.010
  Função Utilizada ----------->  2
  Valor de Alfa --------------> 0.000
  Valor do Raio --------------> 0.100
  Valor de M -----------------> 0.001
  Pontos de Interesse --------->  5
  Inicio do Processo ---------> 15:31:56.51


Resultado  1   0.31361132 - 0.789
Resultado  2   0.10950832 - 0.935
Resultado  3   0.50226494 - 0.702
Resultado  4   0.69842318 - 0.460
Resultado  5   0.89764557 - 0.251
```

ANEXO 05

RELATÓRIO DO PROGRAMA 623.PAS

```
Algoritmo Genético SGA623.PAS

    Arquivo de saída -----------> 623.Out
   Tamanho da população -------> 50
   Comprimento do cromossoma --> 30
   Número máximo de gerações --> 30
   Probabil. de crossover -----> 0.900
   Probabil. de mutação -------> 0.010
   Função Utilizada ----------->   1
   Valor de Alfa --------------> 4.000
   Valor do Raio --------------> 0.100
   Valor de M -----------------> 0.000
   Pontos de Interesse --------->  5


Resultado  1    0.68138051451 -> 0.9999992047
Resultado  2    0.93138611079 -> 0.9972889803
Resultado  3    0.24493038841 -> 0.9975035142
Resultado  4    0.07959732041 -> 0.9999846288
Resultado  5    0.45120467059 -> 0.9997928861
```

ANEXO 06

RELATÓRIO DO PROGRAMA 624.PAS

```
Algoritmo Genético SGA624.PAS

   Parâmetros assumidos:


   Arquivo de saída -----------> 624.Out
   Tamanho da população -------> 50
   Comprimento do cromossoma --> 30
   Número máximo de gerações --> 30
   Probabil. de crossover -----> 0.900
   Probabil. de mutação -------> 0.010
   Função Utilizada ----------->  2
   Valor de Alfa --------------> 0.000
   Valor do Raio --------------> 0.100
   Valor de M -----------------> 0.010
   Pontos de Interesse --------->  5


Resultado  1   0.07574 -    0.97561
Resultado  2   0.25005 -    0.94327
Resultado  3   0.43262 -    0.64074
Resultado  4   0.67980 -    0.48215
Resultado  5   0.93449 -    0.22123
```

ANEXO 07

RELATÓRIO DO PROGRAMA 625.PAS

```
Algoritmo Genético SGA625.PAS

   Parâmetros assumidos:

   Arquivo de saída -----------> 625.Out.out
   Tamanho da população -------> 50
   Comprimento do cromossoma --> 30
   Número máximo de gerações --> 50
   Probabil. de crossover -----> 0.900
   Probabil. de mutação -------> 0.010
   Função Utilizada ----------->   1
   Valor de Alfa --------------> 4.000
   Valor do Raio --------------> 4.240
   Valor de M -----------------> 0.000
   Pontos de Interesse --------->  4
   Inicio do Processo ---------> 15:34:52.82


It is now 15:35:19.35
Resultado  1   X -> 2.5664062500   Y -> 2.1977539063 - 194.9329731300
Resultado  2   X -> -2.7050781250   Y -> 2.5300292969 - 187.7550771800
Resultado  3   X -> -4.2270507813   Y -> -4.0515136719 - 165.1552812000
Resultado  4   X -> 4.0625000000   Y -> -4.1398925781 - -3.5349146214
```

ANEXO 08

RELATÓRIO DO PROGRAMA 631.PAS

```
Algoritmo Genético SGA631.PAS

   Parâmetros assumidos:

   Arquivo de saída -----------> 631.out
   Tamanho da população -------> 50
   Comprimento do cromossoma --> 30
   Número máximo de gerações --> 50
   Probabil. de crossover -----> 0.900
   Probabil. de mutação -------> 0.010
   Função Utilizada ----------->  1
   Valor de Alfa --------------> 4.000
   Valor do Raio --------------> 1.520
   Valor de M -----------------> 0.000
   Pontos de Interesse --------> 18
   Inicio do Processo ---------> 15:36:27.57


It is now 15:38:42.41
Resultado  1   X -> 1.5700000000   Y -> 3.1396166992 - 0.9999909229
Resultado  2   X -> 4.7103833008   Y -> 3.1396166992 - 0.9999841458
Resultado  3   X -> 6.2322827148   Y -> 1.6248779297 - 0.9890087567
Resultado  4   X -> 9.4280493164   Y -> 7.8496166992 - 0.9999404923
Resultado  5   X -> 9.4247167969   Y -> 4.7158666992 - 0.9999758037
Resultado  6   X -> 1.5402050781   Y -> 0.0303833008 - 0.9962866665
Resultado  7   X -> 4.6908203125   Y -> 6.3033666992 - 0.9982560238
Resultado  8   X -> 1.5791992187   Y -> 6.2796166992 - 0.9998333232
Resultado  9   X -> 3.1415332031   Y -> 1.5696166992 - 0.9999972099
Resultado 10   X -> 3.1370507813   Y -> 4.7158666992 - 0.9999345552
Resultado 11   X -> 0.0135644531   Y -> 1.5571166992 - 0.9992579304
Resultado 12   X -> 6.2835668945   Y -> 7.8558666992 - 0.9999926018
Resultado 13   X -> 1.5520874023   Y -> 6.2496166992 - 0.9970493573
Resultado 14   X -> 1.5743334961   Y -> 9.4258666992 - 0.9999726064
Resultado 15   X -> 7.8538330078   Y -> 9.4196166992 - 0.9999466797
Resultado 16   X -> 0.0015332031   Y -> 7.8496166992 - 0.9999571939
Resultado 17   X -> 7.8474340820   Y -> 6.2858666992 - 0.9998998831
Resultado 18   X -> 3.1393505859   Y -> 7.8558666992 - 0.9999828394
```

ANEXO 09

RELATÓRIO DO PROGRAMA 632.PAS

```
Algoritmo Genético SGA632.PAS

    Parâmetros assumidos:

   Arquivo de saída -----------> 632.out
   Tamanho da população -------> 50
   Comprimento do cromossoma --> 30
   Número máximo de gerações --> 30
   Probabil. de crossover -----> 0.900
   Probabil. de mutação -------> 0.010
   Função Utilizada ----------->  2
   Valor de Alfa --------------> 0.000
   Valor do Raio --------------> 1.520
   Valor de M -----------------> 0.010
   Pontos de Interesse --------->  7
   Inicio do Processo ---------> 15:39:07.29


It is now 15:39:36.78
Resultado  1   X -> 9.4567871094   Y -> 0.0183105469 * 5.9830668445
Resultado  2   X -> 6.2734032892   Y -> 6.3002276572 * 5.9961394166
Resultado  3   X -> 3.0920410156   Y -> 6.2889536475 * 5.9748501574
Resultado  4   X -> 0.0164794922   Y -> 3.1919054165 * 5.9693372307
Resultado  5   X -> 6.2541597208   Y -> 0.0042724609 * 5.9914004665
Resultado  6   X -> 6.2438576085   Y -> 9.4134521484 * 5.9678299947
Resultado  7   X -> 6.2794290530   Y -> 3.1876290680 * 5.9785755238
```

ANEXO 10

RELATÓRIO DO PROGRAMA 633.PAS

Algoritmo Genético SGA633.PAS

Parâmetros assumidos:

```
Arquivo de saída -----------> 633b.Out
Tamanho da população--------> 130
Comprimento do cromossoma---> 84
Número máximo de gerações---> 10
Probabil. de crossover------> 0.900
Probabil. de mutação--------> 0.100
Função Utilizada------------>   1
Valor de Alfa --------------> 4.000
Valor do Raio --------------> 2.000
Valor de M -----------------> 0.000
```

It is now 15:09:51.71

Primeira Solução			
Resultado	1	X ->	8
Resultado	2	X ->	6
Resultado	3	X ->	7
Resultado	4	X ->	3
Resultado	5	X ->	2
Resultado	6	X ->	4
Resultado	7	X ->	3
Resultado	8	X ->	9
Resultado	9	X ->	7
Resultado	10	X ->	9
Resultado	11	X ->	14
Resultado	12	X ->	12
Resultado	13	X ->	3
Resultado	14	X ->	5
Resultado	15	X ->	9
Resultado	16	X ->	8
Resultado	17	X ->	11
Resultado	18	X ->	15
Resultado	19	X ->	14
Resultado	20	X ->	0
Resultado	21	X ->	14

84.42837623000

Última Solução			
Resultado	1	X ->	14
Resultado	2	X ->	1
Resultado	3	X ->	4
Resultado	4	X ->	11
Resultado	5	X ->	3
Resultado	6	X ->	9
Resultado	7	X ->	3
Resultado	8	X ->	0
Resultado	9	X ->	15
Resultado	10	X ->	11
Resultado	11	X ->	15
Resultado	12	X ->	14
Resultado	13	X ->	5
Resultado	14	X ->	14
Resultado	15	X ->	15
Resultado	16	X ->	2
Resultado	17	X ->	15
Resultado	18	X ->	8
Resultado	19	X ->	2
Resultado	20	X ->	2
Resultado	21	X ->	2

2525.66319670000

REFERÊNCIAS BIBLIOGRÁFICAS

[AAR89] AARTS, E.H.L., *Simulated Annealing and Boltzmann Machines: a Stochastic Approach to Combinatorial Optimization and Neural Computing. Wiley*, New Iork, 1989.

[ACK87] ACKLEY, D.H. An *Empirical Study of Bit vector Function Optimization*, Genetic Algorithms and Simulated Annealing, London, Pittman, 1987.

[BALL81] BALL, M., M. MAGAZINE, *The Design and Analysis of Heuristics*, Network, vol II, 1981.

[BARR95] BARR, R., B. GOLDEN, J. KELLY, M. RESENDE, W. STEWART, *Designing and Reporting on Computational Experiments with Heuristic Methods*, Journal of Heuristics, vol 1. nº. 1, 1995

[CAM94] CAMPELLO, R.E., MACULAN, N. *Algoritmos e Heurísticas: Desenvolvimento e Avaliação de Performance*, Niterói, EDUFF, 1994.

[CHR75] Christofides, N. Graph Theory - An Algorithmic Approach, London, Academic Press, 1978.

[COOK71] COOK, S.A. *The Complexity of Theorem-Proving Procedures.* Proceedings of The Third Anual ACM Symposium on The Theory os Computing. Association fos Computing Machinery, 1971.

[DAV87] DAVIS, L., *Genetic Algorithms and Simulated Annealing.* Pitman, London, 1987.

[DAV91] DAVIS, L., *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, New York, 1991.

[DEB89] DEB, K. Genetic Algorithms in Multimodal Fuction Optimization, Master's thesis, University of Alabama, 1989.

[EIL77] EILON, S., *More Against Optimization*, Omega, vol 5, 1977.

[FEO89] FEO, T.A., M.G.C. RESENDE, *A Probabilistic Heuristic for a Computanionally Dificult Set Covering Problem.* Operations Research Letters, 8:67-71, 1989.

[GLO94] GLOVER, F., *Tabu Search Fundamentals and Uses.* University of Colorado at Bouder, 1994.

[GOL87] GOLDBERG, D.E. , J. RICHARDSON. *Genetic Algorithm With Sharing for Multimodal Fuction Optimization.* Proceedings of the Second International Conference on Genetic Algorithms, Pittsburb, 1987.

[GOL89] GOLDBERG, D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, Reading, Massachusetts, 1989.

[GOL89b] GOLDBERG, D.E. *Sizing Populations for Serial and Parallel Genetic Algorithms.* Proceedings for the Third International Conference on Genetic Algorithms, San Mateo, 1989.

[GOL92] GOLDBERG, D.E., K. DEB, J. HORN. *Massive Multimodality, Deception, and Genetic Algorithms*, Parallel Problem Solving From Nature, Amsterdam, 1992.

[GRE93] GREFENSTETTE, J.J., *Deception Considered Harmuful*, En D. Whitley, editor, Foudations of genetic Algorithms 2, p.72-91, San Mateo, 1993.

[GRO85] GROSSO, P. *Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interation in a Multimodal Model.* PhD Thesis, University of Michigan, Ann Arbor, 1985.

[HEIN94] HEIN, N. *Otimização Global Determinística - Um Algoritmo.* Dissertação, UFSC, 1994.

[HEIN95] HEIN, N. *Um Modelo Didático de Otimização Genética,* Boletim do Departamento de Matemática da FURB, Blumenau, 1995.

[HEIN95b] HEIN, N., *Um Algoritmo Genético para o Problema do Caixeiro Viajante,* Boletim do Departamento de Matemática da FURB, Blumenau, 1995.

[HOL75] HOLLAND, J.H. *Adaptation in Natural and Artificial Systems,* University of Michigan Press, Ann Arbor, 1975.

[HOM94] HOMER, S., M. PEINADO. *On the Performance of Polinomial-Time Clique Algorithms on Very Large Graphs.* Technical Reports 94-001, Boston University, Cambridge, 1994.

[HOF93] HOFMANN, R. *Examinations of the Algebra of Genetic Algorithms.* PhD thesis, Technique Universität München, 1993.

[JOL61] JOLLEY, L.B.W. *Summation of Series,* Dower, 1961.

[KEA95a] Keane A. A *Brief Comparasion of Some Evolutionary Optimization Methods.* Proceedings of the Conference on Adaptive Computing in Engineering Design ans Control 94, ed. Parmee, Plymouth, 1995.

[KEA95b] Keane, A. *Genetic Algorithm Optimization of Multi-Peack Problems: Studies in Convergence and Robustness,* Artificial Intelligence in Engineering, 1995.

[KLI92] KLINCEWICZ, J.G., A. RAJAN. *Using GRASP to Solve the Component Grouping Problem.* Technical Report, Holmdel, 1992.

[LAG94] LAGUNA, M., T.A. FEO, H. C. ELROD. *A Greedy Randomized Adaptive Search Procedure for the 2-Partition Problem,* Operation Research, 42:677-687, 1994.

[LEN82] LENSTRA, J.K., A.H.G. RINNOOY KAN. *An Aprraisal of Computing Complexity of Operations Research,* European Journal of Operation research, vol 11, 1982

[LEW78] LEWIS, H.R., C.H. PAPADIMITRIOU, *The Effiency of Algorithms,* Scietific American, vol 238, nº 1, 1978.

[LIE78] LIEBERMANN, G.J., HILLIER, F.S. *Introduction to Operations Reseach,* 2 ed. San Francisco, Holden-Day, 1974.

[LIN80] LINHARES, S., F. GEWANDSZNAJDER, *Biologia das Populações - Genética, Ecologia e Evolução*: 2º grau, São Paulo, Ática, 1980.

[MOS89] MOSCATO, P. , *On Genetic Crossover Operators for Relative Order Preservacion.* California Institute of Technology, Pasadena, 1989.

[MOS92] MOSCATO, P., M. G. NORMAN. *A Memetic Approach for the Traveling Salesman Problem - Implementation of a Computational Ecology for Combinatorial Optimization os Messa-Passing Systems.* Parallel Computing and Transputer Applications. Amsterdam, IOS Press, 1992.

[MOS93] MOSCATP, P. *An Introdudction to Population Approaches for Optimization and Hierachical Objective Functions: A Discussion on the Role of Tabu Search,* Annal of Operations Research, Vol 41, 85-121, 1993.

[MÜH87] MÜHLENBEIN,H., D. SCHLIERKAMP-VOOSEN. *Predictive Models for the Breeder Genetic Algorithms,* Evolutionary Algorithms, nº 1 p.25-50, 1987.

[OEI91] OEI, C.K. D.E. GOLDBERG, S. SHANG. *Tournament Selection, Niching and the Preservation of Diversity,* University of Illinois at Urbana Champaign, Departament of General Engineering, 1991.

[RAD94] RADCLIFFE, N.J.Fitness *Variance of Formae and*

Performance Prediction, Foudations of genetic Algorithms, 1994.

[RED95] REDIN, J.C. *Programação Matemática Via Algoritmos Genéticos - Uma Modificação na Estrutura do Crossing Over*, TCC-BCC-FURB, 1995.

[REE93] REEVES, C.R. *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scietific Pub., Oxford, 1993.

[RIE94] RICIERI, A.P. Otimização natural - A Genética dos Máximos e Mínimos. São Paulo, Editora Prandiano, 1994.

[SHA94] SHAPIRO, J.L., A. PRUGEL-BENNETT. Analysis of Genetic Algorithms usisng Statical Mechanics. Psysical Review Letters, 1994.

[SIL80] SILVER, E.A., R.V. VIDAL, D. de WERRA. *A Tutorial on Heuristics Methods*, European Journal of Operations Research, vol5, 1980.

[SMI92] SMITH, R.E., S. FORREST, A.S. PERELSON. *Searching for the Diverse, Cooperative Populations With Genetic Algorithms*, University of Alabama, 1992.

[WHI91] WHITLEY, D., K. MATHIAS, P. FITZHORN. *Delta Coding: An Iterative Search Strategy for Genetic Algorithms*, Proceedings of the Fourth Interantional Conference of genetic Algorithms, San Mateo, 1991.