Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Departamento de Engenharia Eletrônica e Sistemas

Mestrado em Engenharia Elétrica

Estudo, Melhorias e Implementação Computacional do Talus

Genaro Dueire Lins

Dissertação de Mestrado

Recife
Março de 1999

Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Departamento de Engenharia Eletrônica e Sistemas

Genaro Dueire Lins

Estudo, Melhorias e Implementação Computacional do Talus

Esse trabalho foi apresentado à Pós-Graduação em Engenharia Elétrica do Centro de Tecnologia e Geociências da Universidade Federal de Pernambuco como requisito parcial para a obtenção do grau de Mestre em Engenharia Elétrica.

ORIENTADOR : Prof. **Fernando Menezes Campello de Souza**

03 de março de 1999

Lins, Genaro Dueire

Estudo, melhorias e implementação computa - cional do Talus / Genaro Dueire Lins. - Recife : O Autor. 1999.

167 folhas : il., tab., gráf.

Dissertação (mestrado) - Universidade Federal de Pernambuco. CTG. Engenharia Elétrica, 1999.

Inclui bibliografia.

1. Algoritmos de otimização. 2. Algoritmos probabilisticos. 3. Otimização global. 4. Programação matemática. Título.

519.856	CDU (2.ed.)	UFPE
519.76	CDD (21.ed.)	BC99-58

Dedico este trabalho à minha mãe Gilka Dueire Lins e ao meu pai Zanoni Lira Lins (*In memoriam*), como reconhecimento pela dedicação à família e estímulo ao conhecimento.

Agradecimentos

Agradeço ao Prof. Fernando Menezes Campelo de Souza que com sua dedicação e seu entusiasmo tornou este trabalho possível. Sempre lembrarei dos seus ensinamentos e da forma paciente com a qual me orientou no decorrer desta tese.

Ao Prof. Francisco de Sousa Ramos, agradeço pelo seu incentivo durante todo o transcorrer do meu Mestrado.

A minha família, em especial, a minha mãe e a minha irmã Rilane Dueire Lins Miranda. agradeço pelo apoio sempre presente durante o transcorrer desta tese. Aos meus irmãos Rafael Dueire Lins e Zanoni Dueire Lins agradeço pelo exemplo acadêmico.

Resumo

Nesta tese estuda-se, aprimora-se e implementa-se em linguagem C++ um algoritmo probabilístico que faz parte do projeto Talus (concepção, desenvolvimento e construção de um computador estocástico de arquitetura paralela). Caracteriza-se a área de estudo e apresenta-se os algoritmos determinísticos e, um pouco mais detalhadamente, os algoritmos probabilísticos, com ênfase no *simulated annealing*. Estuda-se com mais detalhes os algoritmos genéticos Apresenta-se as melhorias no algoritmo Talus, destacando-se a estratégia para se calcular pontos de sela, e o *software* em C++. Resolve-se 28 problemas de otimização apresentados na literatura, alguns específicos para teste de algoritmos e outros oriundos da formulação de problemas práticos. A maioria dos problemas resolvidos apresentam elevado grau de complexidade. Faz-se uma comparação do algoritmo Talus com os algoritmos genéticos e sugere-se um novo operador genético, denominado o operador de mutação Talus.

Palavras Chaves : Otimização Global, Algoritmos Probabilísticos, Programação Matemática.

Baseado na equação 3.9, pode-se concluir que um alto valor médio de aptidão sozinho não é suficiente para uma alta taxa de crescimento de um dado esquema. Com esta equação chega-se também à importante constatação que constitui o Teorema do Esquema:

Desde que o comprimento de definição e a ordem de um esquema sejam baixos, a evolução acarreta um aumento exponencial do número de seus representantes, caso sua adequação permaneça acima da média ou, inversamente, um decréscimo exponencial, se sua adequação for inferior a média.

Diversos pesquisadores têm apresentado extensões a este teorema, permitindo o modelamento do efeito de outros operadores (como o do operador de especiação descrito adiante no item 4.9.3). Também, extensões do teorema têm sido propostas visando a possibilidades dos operadores genéticos acarretarem a criação de novos esquemas ou a eventualidade do operador de *crossover* não destruir um esquema mesmo quando o ponto de *crossover* estiver situado em um *locus* entre os alelos fixos extremos do esquema (o que ocorre, por exemplo, quando ambos os pais são representantes do esquema).

4.6.1 O conceito de Paralelismo Implícito

Um único indivíduo numa população representa, simultaneamente, 2^L diferentes esquemas (L sendo o número de genes no cromossomo). Considerando-se que a população não seja homogênea, o número total de esquemas representados na população ainda é maior.

Analisando-se a argumentação que levou ao estabelecimento do Teorema do Esquema, observa-se que, durante a evolução, todos os esquemas representados na população

competem uns com os outros, em paralelo, para conseguirem ser representados nas gerações subseqüentes.

Conclui-se, então, que mesmo na eventualidade de um algoritmo utilizar uma população pequena, o número de diferentes elementos de informação (os esquemas) processado é muito alto. Esta capacidade de processamento simultâneo de grande volume de informação, apresentada pelos Algoritmos Genéticos, foi por Holland denominada de **Paralelismo Implicito.**

Uma constatação interessante é que todos os esquemas são processados com a mesma "eficiência". Esquemas de comprimento de definição grande mesmo com alta adequação, têm pequena probabilidade de sobreviver a uma geração. Assim, é importante estimar o número de esquemas processados eficientemente.

4.7 **Implementação dos Operadores Genéticos Básicos**

Os operadores genéticos básicos (seleção, *crossover* e mutação) apresentados na descrição do SGA são implementações fiéis aos operadores ideais definidos por Holland. Entretanto, inúmeras outras implementações daqueles operadores têm sido propostas e, inclusive, novos operadores têm sido criados. Dependendo do tipo do problema são utilizados operadores específicos, os quais permitem a otimização da operação do algoritmo. Outras, são aplicações mais generalizadas.

Neste item são mostradas as principais implementações dos operadores genéticos básicos, bem como uma análise de um outro importante elemento da estrutura de um Algoritmo Genético: a estratégia de substituição da população.

É importante notar que as implementações apresentadas para os operadores *crossover* e mutação são voltadas à manipulação de genótipos binários. O emprego de outras formas de codificação, em geral exige a utilização de operadores genéticos específicos.

4.7.1 Seleção

A estrutura básica do Algoritmo Genético proposta por Holland pressupõe que o processo de seleção apresente duas características básicas: baixo favorecimento (viés, discrepância entre a probabilidade teórica de seleção de um indivíduo e a média efetiva obtida na prática, para diversas aplicações do operador de seleção) e pequena dispersão (amplitude da faixa de valores do número de vezes que um indivíduo é selecionado, efetivamente obtida na prática).

O algoritmo 4.5 descreve uma implementação do operador seleção, a **seleção por roleta,** também conhecida como seleção estocástica com reposição. Esta implementação apresenta alto favorecimento, mas alta dispersão, já que qualquer indivíduo pode ser selecionado um número arbitrário de vezes (o indivíduo de mais alta adequação pode não ser selecionado, enquanto o de mais baixa adequação pode ser selecionado todas as vezes, embora sejam eventos de mais baixa probabilidade).

Uma variação deste algoritmo foi proposta por Baker (1987). Na sua implementação de característica de baixo favorecimento é mantida mas, é reduzida a dispersão ao mínimo (a diferença entre o número teórico de vezes que o indivíduo deve ser selecionado e o efetivamente obtido não é superior a 1). Esta implementação, descrita no algoritmo a seguir, é denominada seleção estocástica universal. Pode-se notar que esta implementação apresenta uma analogia a uma roleta viciada com N marcadores (sendo N o número de indivíduos da população).

- ordenar os indivíduos segundo um critério arbitrário
- associar a cada indivíduo um valor de adequação acumulada, igual à somatória das adequações de todos os indivíduos anteriores a ele (incluindo ele próprio).
- calcular $\Delta f = \frac{1}{N} \cdot \sum f_i$
- gerar aleatoriamente um número com distribuição uniforme entre 0 e ***Af***
- formar uma população auxiliar, repetindo N vezes :
    - incluir na população auxiliar o primeiro indivíduo cuja adequação for maior ou igual ao número gerado
    - subtrair de todas as adequações acumuladas o valor *Af*
- os pares de indivíduos são selecionados aleatoriamente, sem reposição, nesta população auxiliar

Algoritmo 4.5

Uma terceira implementação do operador seleção é a **seleção determinística**. Como o próprio nome indica, esta implementação não inclui nenhum fator aleatório. Ela apresenta baixa dispersão, mas alto favorecimento, uma vez que os indivíduos de baixa adequação nunca são selecionados. A implementação deste algoritmo pode ser descrita como segue por Algoritmo 4.6.

- associar a cada indivíduo um número esperado de descendentes (n_d), igual à divisão da adequação do indivíduo pela adequação média da população

- inserir, numa população auxiliar , um número de cópias de cada indivíduo igual à parte inteira do respectivo n_d

- ordenar os indivíduos da população em ordem decrescente do valor da parte fracionária dos respectivos n_d

- copiar os primeiros indivíduos da população ordenada para população auxiliar, até que estas possuam N indivíduos

- os pares de indivíduos são selecionados aleatoriamente, sem reposição, nesta população auxiliar

Algoritmo 4.6

Uma última implementação, a **seleção com resíduo estocástico,** introduz um fator probabilístico, permitindo reduzir o favorecimento, embora acarrete um aumento da dispersão. Esta implementação é descrita no Algoritmo 4.7.

- associar a cada indivíduo um número esperado de descendentes (n_d), igual à divisão da adequação do indivíduo pela adequação média da população

- inserir, numa população auxiliar , um número de cópias de cada indivíduo igual à parte inteira do respectivo n_d

- para cada indivíduo, substituir n_d pela parte fracionária de n_d.

- utilizando os novos valores de n_d, selecionar "por roleta" tantos indivíduos quantos forem necessários para que a população auxiliar possua N indivíduos

- os pares de indivíduos são selecionados aleatoriamente, sem reposição, nesta população auxiliar

Algoritmo **4.7**

4.7.2 *Crossover*

Como mencionado anteriormente, o operador de *crossover é* responsável pela recombinação de características dos pais durante a reprodução, permitindo assim que as próximas gerações herdem essas características. Ele é considerado o operador genético predominante, por isso é aplicado com probabilidade dada pela taxa de *crossover* p_c que deve ser bem maior que a taxa de mutação. Este processo é bastante sensível à específica implementação.

No ***crossover* de um ponto**, (Algoritmo 4.4), um ponto de cruzamento é escolhido e a partir deste ponto as informações genéticas dos pais serão trocadas. As informações anteriores a este ponto em um dos pais serão ligadas às informações posteriores à este ponto no outro pai. Esta implementação consegue preservar eficientemente esquemas mais curtos (com pequeno comprimento de definição), mas é altamente destrutivo para esquemas longos. Numa situação limite, esquemas com comprimento de definição igual ao número de genes são sistematicamente destruídos (a menos que sejam representados por ambos os pais).

Uma variação desta implementação que embora mantendo a tendência de preservar esquemas mais curtos, permite também a preservação de esquemas mais longos, é o *crossover* **de dois pontos**, descrito no Algoritmo 4.8. Nesta implementação, são definidos aleatoriamente dois pontos de *crossover*, ou seja, os genótipos são divididos em três trechos. Cada filho contém **os** trechos inicial e final de um dos pais e o central do outro pai.

entradas : g_i, g_j : genótipo dos dois indivíduos

- aleatoriamente selecionar dois *loci* diferentes L_1, L_2 como pontos de *crossover,* com $1 \leq L_1 \leq L_2 \leq L$ (L = número de genes do cromossomo)
- quebrar o genótipo de cada pai em três trechos : o primeiro trecho contendo os alelos dos *loci* 1 a $L_1 - 1$, o segundo, os alelos dos *loci* L_1 a $L_2 - 1$ e o terceiro , os *loci* L_2 a L
- o genótipo do primeiro filho é formado pelo primeiro e terceiro trecho do primeiro pai e o segundo trecho do segundo pai; o genótipo do segundo filho é formado pelo primeiro e terceiro trecho do segundo pai e o segundo trecho do primeiro pai

Algoritmo 4.8

Uma característica básica das duas implementações descritas é a tendência de preservar informações que sejam codificadas "condensadamente" no cromossomo, ou seja, preservar blocos construtivos. Uma terceira implementação, o *crossover* **uniforme,** permite preservar informações codificadas de forma distribuída por todo cromossomo. Nesta implementação, descrita pelo Algoritmo 4.9, os alelos dos genótipos dos pais são tratados individualmente; o genótipo de cada filho é formado por alelos escolhidos aleatoriamente de um pai ou do outro pai.

entradas : g_i ,g_j : genótipo dos dois indivíduos

- gerar uma seqüência de L dígitos binários (padrão), em que o valor de cada dígito é escolhido aleatoriamente
- construir o genótipo de cada um dos fihos, repetindo para i variando de **1** a L a seguinte rotina:
  - se o *i-ésimo* dígito do padrão for igual a 0 fazer :
    - copiar o alelo de locus i do primeiro pai para o primeiro filho
    - copiar o alelo de locus i do segundo pai para o segundo filho
  - caso contrário, fazer:
    - copiar o alelo de locus i do primeiro pai para o segundo filho
    - copiar o alelo de locus i do segundo pai para o primeiro filho

Algoritmo 4.9

A figura 4.3 mostra a representação gráfica das três implementação do operador *crossover*.

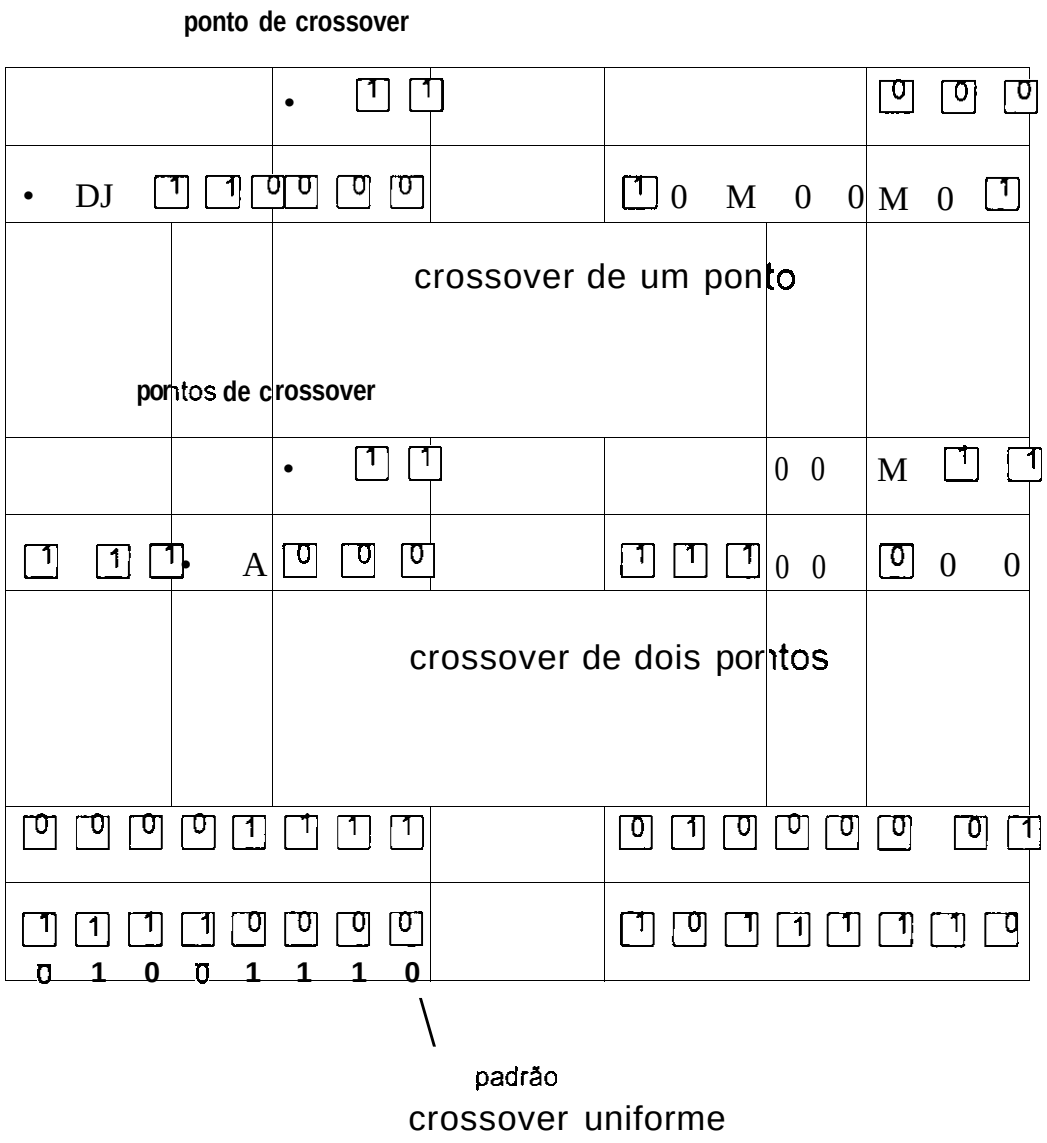


Figura 4.3

A função do operador de mutação é manter a diversidade em uma população, através **da** modificação aleatória dos genótipos da população. Desta forma, o operador de mutação permite que todos os pontos do conjunto viável tenha probabilidade diferente de zero de ser alcançado.

Como quase a totalidade dos Algoritmos Genéticos que utilizam genótipos codificados sob a forma binária empregam a mutação bit a bit. descrita no item 4.10, esta será a única implementação descrita neste trabalho.

4.8 Estratégia de substituição da população

No SGA, a população tem a totalidade de seus indivíduos substituída a cada geração. Esta estratégia pode levar a uma oscilação do valor máximo de adequação da população durante sua evolução, uma vez que a simples aplicação dos operadores genéticos não garante que algum indivíduo após uma geração tenha adequação igual ou superior às adequações dos indivíduos das gerações anteriores.

Uma forma empregada de resolver este problema é a utilização da **estratégia elitista.** Por esta estratégia, caso uma geração implique a redução do valor máximo de adequação **da** população, mantém-se na população seu melhor indivíduo, sendo desprezado o pior indivíduo após a geração. Esta estratégia geralmente permite uma considerável melhoria no desempenho de um Algoritmo Genético.

Abstract

This thesis is a study, improvement, and implementation in C++ of a probabilistic algorithm that is a part of the Talus Project (conception, development, and construction of a stochastic parallel computer). The field of study is characterized and deterministic algorithms are presented and, in greater detail the probabilistic algorithms, with emphasis on simulated annealing. Genetic algorithms are studied in greater detail. Improvements in the Talus algorithm are shown, highlighting the strategy for calculating saddle points, and a C++ software is also presented. A total of 28 optimization problems present in the literature, some specific for testing algorithms, and others arising from the formulation of practical problems are solved. Most of the problems solved show a high degree of complexity. Finally, there is a comparison of the Talus algorithm with genetic algorithms, and a suggestion for a new genetic operator, named the Talus Mutation Operator.

Key Words: Global Optimization, Probabilistic Algorithms, Mathematical Programming.

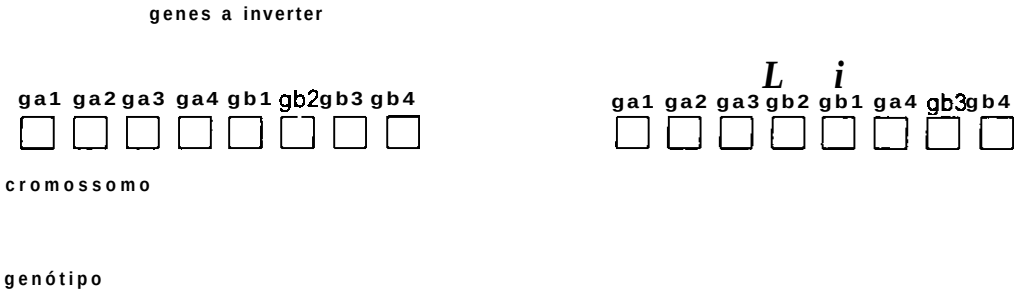
Uma outra estratégia desenvolvida é a substituição parcial da população . Esta estratégia implica na seleção de um número pequeno de indivíduos para reprodução. Estes indivíduos se reproduzem . gerando um número igual de descendentes. Estes descendentes substituem os indivíduos de menor adequação na população .

4.9 Outros operadores

Além dos operadores propostos inicialmente por Holland, outros operadores têm sido desenvolvidos, alguns deles voltados a aplicações específicas de Algoritmos Genéticos. A seguir apresentam-se alguns deles :

4.9.1 Inversão

Este operador, que é inspirado em um fenômeno biológico, permite que genes tenham seus *loci* alterados no cromossomo. É importante observar que este operador não atua no genótipo do indivíduo. mas sim sobre o cromossomo ; desta forma, cada indivíduo da população passa a ter um cromossomo diferente. A figura 3.3 ilustra a aplicação deste operador



O operador de inversão não é comumente utilizado, pois implica em um considerável acréscimo no tamanho das estruturas de dados para armazenamento dos indivíduos, uma vez que estas devem incluir uma representação do cromossomo específico; também, acarreta maior tempo de processamento, pois exige a manipulação do cromossomo de cada indivíduo, além da manipulação normal de seu genótipo.

4.9.2 Alteração da escala de adequações

A **Alteração da escala de adequações** visa solucionar os seguintes problemas que freqüentemente aparecem na implementação de um Algoritmo Genético:

- no início do processo evolutivo, a população apresenta alguns poucos indivíduos de alta adequação; estes indivíduos se reproduzem muito freqüentemente fazendo com que a população perca muito rapidamente sua diversidade; como conseqüência, a população apresenta tendência a convergir para uma solução ótima local (distinta da solução ótima global);

- com o decorrer do processo evolutivo a população tende a conter um número considerável de indivíduos de alta adequação alguns apenas ligeiramente menores que outros; como as adequações são aproximadamente iguais os indivíduos se reproduzem com aproximadamente a mesma freqüência e, conseqüentemente, a convergência da população é extremamente lenta.

O operador de **alteração de escala de adequações** modifica as adequações dos indivíduos da população, fazendo com que a relação entre as adequações máxima e média da

população se mantenha constante durante todo o processo evolutivo. como conseqüência de sua aplicação , um indivíduo excepcionalmente bom terá sua adequação reduzida nas primeiras gerações e aumentadas no final do processo evolutivo.

Goldberg (1989) descreve diversas implementações deste operador

4.9.3 Especiação

A aplicação de um Algoritmo Genético a um problema multimodal pode levar a duas situações distintas : a população converge para um único máximo ou a população não consegue convergir para máximo nenhum. com indivíduos próximos a máximos distintos cruzando entre si e gerando filhos de baixa adequação. Alternativamente, pode ser importante para determinada aplicação determinar todos os máximos do problema.

Através do operador de **especiação**, é possível manter dentro da população diversas **espécies**, ou seja subpopulações de indivíduos que apresentam características semelhantes; estas espécies evoluem independentemente, havendo pouca interação entre indivíduos de espécies diferentes. Idealmente. diferentes espécies convergem para máximos diferentes. Ded et al e Goldberg (1989) descrevem em detalhes este operador e sua implementação.

Apesar de, até o presente item, ter sido considerado que os genótipos eram codificados em forma binária, esta forma de codificação não é obrigatória. Algoritmos Genéticos que manipulam estruturas codificadas segundo alfabeto de maior cardinalidade têm sido desenvolvidos, apresentando freqüentemente bom desempenho.

Serão apresentadas a seguir as formas mais utilizadas de codificação no campo de Algoritmos Genéticos

4.10.1 Codificação Binária

A codificação segundo o alfabeto binário é, indiscutivelmente, a mais utilizada na área de Algoritmos Genéticos. A predominância do seu uso se deve ao fato dela simplificar a análise teórica dos algoritmos, além de permitir a construção de operadores genéticos simples. Uma característica muito importante desta forma de codificação é a sua capacidade de oferecer o número máximo por bit de informação. Esta característica pode ser observada pelo seguinte exemplo:

- considere-se que o fenótipo de uma solução possa assumir 16 diferentes valores;
- utilizando-se codificação binária, o correspondente cromossomo seria constituído de 4 genes;
- utilizando-se um alfabeto de cardinalidade 4, o correspondente cromossomo seria constituído de 2 genes;
- no caso da codificação binária, existem 3^4 possíveis esquemas, enquanto no caso da codificação quaternária, existem apenas 5^2 diferentes esquemas.

Uma classe particularmente importante de problemas a que Algoritmos Genéticos são aplicáveis é constituída por problemas de otimização numérica. Neste caso, o genótipo de cada indivíduo representa um conjunto de números . Existem inúmeros diferentes códigos binários que efetuam a codificação de valores numéricos inteiros; os mais utilizados são o código binário convencional e o código de Gray.

Idealmente, um código eficiente para aplicações de Algoritmos Genéticos deve possuir as seguintes características :

- dados dois valores algebricamente próximos, a distância de Hamming entre as palavras do código correspondentes aqueles valores do código deve ser pequena, e vice-versa

- dados dois valores algebricamente distantes, a distância de Hamming entre as palavras do código correspondentes aqueles valores do código deve ser grande, e vice-versa

O código de Gray permite a geração de palavras de código que apresentem a primeira condição da primeira característica. O código de Binário convencional não apresenta nenhuma das duas características.

Apesar dos códigos mencionados não apresentarem características ideais, outros códigos binários não são usualmente empregados para esta classe de problemas.

A representação em ponto **flutuante** apresenta algumas características bastante convenientes :

- os parâmetros que compõem a solução do problema podem ser representados de uma forma mais natural, mais próximo do domínio de definição do problema que no caso de codificação binária;

- esta forma de representação evita o problema da ocorrência de grande discrepância entre a distância algébrica e a distância de Hamming entre as representações de dois valores.

Avaliações experimentais mostram que algoritmos que empregam esta forma de codificação podem ser altamente eficientes, permitindo a obtenção de resultados com alta precisão e em um número de gerações menor que no caso de codificação binária. **Entretanto,** esta última constatação não implica necessariamente um menor tempo de processamento, uma vez que operações em ponto flutuante exigem um esforço computacional maior que a simples manipulação de dígitos binários. Vale salientar também que o emprego desta forma de codificação implica na necessidade de utilização de operadores genéticos específicos.

Algoritmos Genéticos têm seu comportamento fortemente influenciados pela específica implementação adotada para os operadores genéticos, pelos valores dos parâmetros básicos (número de indivíduos na população e probabilidades de *crossover* e mutação) e pela específica regra de codificação empregada.

Diversos trabalhos têm sido desenvolvidos com o objetivo de quantificar a influência de cada um destes fatores sobre a eficiência de específicos Algoritmos Genéticos. Muitos destes trabalhos têm-se centrado na análise de desempenho do SGA. Neste caso observou-se que o algoritmo apresenta bom desempenho se a população for constituída por algumas dezenas de indivíduos, a probabilidade de *crossover* for alta, e a de mutação for baixa. Especificamente, De Jong (1993), Grefenstette (1986) e Schaffer (1995) propõe faixa de valores, apresentados na tabela abaixo, para parâmetros básicos do algoritmo.

	Número de indivíduos	probabilidade de *crossover*	probabilidade de mutação
De Jong	50 - 100	0.6	10^{-3}
Grefenstette	30 80	0,95 0,45	10^{-3} 10^{-3}
Scchaffer et al	20-30	0,75 - 0,95	0,005 - 0,01

Um fator que influencia marcadamente o comportamento da adequação máxima da população é a estratégia de substituição. A estratégia elitista, por sua simplicidade de implementação e por não alterar significativamente a estrutura de Algoritmo Genético proposta por Holland, é freqüentemente usada.

A maneira pela qual alguns parâmetros influem no comportamento dos Algoritmos Genéticos é apresentada a seguir:

- Tamanho da População - O tamanho da população afeta diretamente o desempenho global e a eficiência dos algoritmos genéticos. Com uma população pequena o desempenho pode cair, pois deste modo a população fornece uma pequena cobertura do espaço de busca do problema. Uma grande população geralmente fornecesse uma cobertura representativa do domínio do problema, além de prevenir convergências prematuras para soluções locais ao invés de globais. No entanto, para se trabalhar com grandes populações, são necessários maiores recursos computacionais, ou que o algoritmo trabalhe por um período de tempo muito maior.
- Taxa de Cruzamento - Quanto maior for esta taxa, mais rapidamente novas estruturas serão introduzidas na população. Mas se esta for muito alta, pode haver perda de estruturas de alta aptidão. Com um valor muito baixo, o algoritmo torna-se muito lento.
- Taxa de Mutação - Uma baixa taxa de mutação previne que uma dada posição fique estagnada em um valor, além de possibilitar que se chegue em qualquer ponto do espaço de busca. Com uma alta taxa a busca torna-se essencialmente aleatória.
- Intervalo de Geração - Controla a porcentagem da população que será substituída, mas com valores muito altos pode ocorrer perda de estruturas de alta aptidão. Com um valor baixo, o algoritmo pode tornar-se muito lento.

Algoritmos genéticos vêm sendo aplicados, com bom resultado, em uma ampla gama de problemas práticos. Para ilustrar a grande diversidade de aplicações, a seguir é mostrada uma lista de trabalhos apresentados no congresso **GALESIA** 95 * nos quais os Algoritmos Genéticos foram aplicados.

- escalonamento de acesso a discos rígidos em sistemas multi-processados
- estabelecimento de cronograma de manutenção em redes ferroviárias
- geração de conjuntos de dados para teste de sistemas computacionais
- otimização da arquitetura de computadores paralelos
- otimização da operação de sistemas de abastecimento de água
- otimização da operação de sistemas de sistemas de geração e transmissão de energia elétrica
- otimização da topologia de redes de telecomunicação
- otimização de métodos de compressão de imagens
- planejamento de redes de transmissão de energia elétrica
- planejamento de trajetória para robôs
- projetos para antenas de radar
- projeto de controladores PID
- projeto de filtros digitais
- projeto de motores de explosão

* GALESIA 95 - Fist IEE/IEEE International Conference on Genetic Algoritms in Engineering Systems : Innovatios and Applicatios - Sheffield- 12 a 14 de Setembro de 1995

O número de problemas práticos aos quais os Algoritmos Genéticos estão sendo aplicados com sucesso é bastante grande, como mostra o item anterior. No ambiente acadêmico, inúmero sistemas de programação voltados a construção e avaliação de Algoritmos Genéticos foram desenvolvidos (ver Barth (1992)). Uma grande variedade de programas já utilizam Algoritmos Genéticos como parte da sua rotina de programação (principalmente na área financeira). Começaram, também, a ser desenvolvidos sistemas de desenvolvimento voltados a Algoritmos Genéticos. Como exemplo, lista-se a seguir alguns destes sistemas comerciais:

- Flextoot GA[1] (Flexible Iteligence Groups LCC - EUA) - "toolbox" para MATLAB;
- Evolver [2] (Palisade Corp - EUA) - "add-in" para Microsoft Excel (aplicações de engenharia e finanças) ;
- Engeneer [3] (Logica Cambridge LTD - Inglaterra) - ambiente de desenvolvimento
- Partek [4] (Partek Incorporated - EUA) - sistema, baseado em Algoritmo Genético, para análise estatística e visualização de dados;
- Xpert Rule KBS [5] (Atar Software Limited - Inglaterra) - sistema, utilizando Algoritmos Genéticos, para construção e manipulação de bases de conhecimento .

[1] - www.flextool.com
[2] - www.palisade.com
[3] - www.wior.uni-karlsruhe.de/bibliothek/software_for_or/genetic_algorithms/com/aacom.html
[4] - www.partek.com
[5] - www.atar.com

No aspecto mais teórico do campo de Algoritmos Genéticos, pesquisas vêm sendo desenvolvidas em diversas áreas (vide Gordon & Whitley (1993)):

- Desenvolvimento de ferramentas matemáticas adequadas para uma precisa descrição do comportamento dos Algoritmos Genéticos (vide Goldberg & Richardson (1987). Fitzpatrick & Grefenstette (1988) - Uma linha de pesquisa que apresenta boas perspectivas de sucesso nesta linha é a utilização de Cadeias de Markov na modelagem dos algoritmos (vide Goldberg & Segrest (1987) e White & Flockton (1995)).

- Algoritmos Genéticos paralelos (vide Biachini (1995), De Jong (1993),Gondon &.Whitley (1993)) - esta linha se configura como altamente promissora, já que a manipulação de informações em um Algoritmo Genético é inerentemente paralela, ainda que o mesmo tenha sido implementado de forma seqüencial.

- Algoritmos Genéticos auto-adaptativos (vide Aisawa (1993), Lee (1993) e Xiao (1997)) algoritmos que modificam os valores de seus parâmetros, de forma a otimizar a eficiência do processo evolutivo.

- Algoritmos Genéticos híbridos (vide Lee (1993), Roberts & Turega (1995))- desenvolvimento de sistemas que utilizem, de forma integrada, Algoritmos Genéticos, redes neurais e técnicas de lógica difusa, entre outras técnicas;

- Co-evolução (vide Tang & Man (1996))- criação de múltiplas subpopulações, independentes, em uma mesma população, de forma tal que o desempenho dos

indivíduos de uma subpopulação seja dependente da evolução dos indivíduos das outra.

- Implementação de Algoritmos Genéticos em *hardware* (vide Sipper (1997)) - desenvolvimento de sistemas de processamento específicos para processamento de Algoritmos Genéticos.

Capítulo 5

Melhorias e Implementação do TALUS

5.1 Introdução

Um algoritmo probabilístico traduz-se por um processo estocástico. Pode-se notar que o algoritmo de busca aleatória pura é um processo estocástico puro. Ele não aproveita nenhuma informação da função objetivo além de uma comparação entre o valor da função numa iteração e na iteração seguinte. Se a probabilidade de resolver um problema numa única tentativa não é baixa, então o número médio de tentativas para efetivamente resolver o problema não será grande. Por outro lado, se esta probabilidade for muito pequena, o número médio de tentativas para efetivamente resolver-se o problema será demasiado grande. Será preciso construir um processo estocástico que incorpore informações sobre o problema. Esse processo deve ter uma capacidade de adaptação (uma espécie de código genético) que faça com que ele se dirija para a solução do problema no menor tempo possível.

Em Cavalcante (1996) e Cavalcante & Campeilo de Sousa (1999) foi introduzido o algoritmo de otimização global denominado TALUS. A partir de uma nuvem de pontos gerados aleatoriamente, e da função a ser otimizada, gera-se uma distribuição de probabilidade sobre os pontos dessa nuvem do espaço de busca. Esta distribuição indica qual deve ser o deslocamento de cada ponto da nuvem de forma a concentrá-la mais na região onde se localiza o ponto de

ótimo. Uma nova distribuição é gerada sobre os pontos da nuvem e assim sucessivamente. As médias dessas consecutivas distribuições convergem para o ponto de ótimo.

Na linguagem de teoria dos jogos, os algoritmos genéticos seriam algoritmos de "competição", onde os indivíduos mais inaptos são destruídos pelos mais aptos. O Talus seria considerado um algoritmo de "coalisão total" onde todos os indivíduos se beneficiam da informação de toda a população para melhorar sua própria adaptação.

Uma das principais características de um algoritmo probabilistico, e em particular no Talus, é a natureza inerentemente paralela destes métodos. Porém, nesta tese. será explorada uma outra característica fundamental deste: a **simplicidade de implementação.** Neste capítulo serão expostos os principais mecanismos que garantem o seu funcionamento, bem como um exemplo da implementação deste num caso unidimensional, que permite uma visualização mais clara do funcionamento do algoritmo.

5.2 O Algoritmo Talus - caso unidimensional

Entradas:

- função objetivo $f : \mathbb{R} \rightarrow \mathbb{R}$
- Número de iterações - *nite*
- Tamanho da núvem - *tnuv*
- Constantes - γ_1, γ_2, β, δ
- Espaço de busca - l_{inf}, l_{sup}

Passo 1. Faça $k = 1$;

Passo 2. Faça $x_1 = l_{inf}$ e $x_2 = l_{sup}$ e $x_i = l_{inf} + A.(l_{sup} - l_{inf})$, onde A é gerado através de uma função de densidade uniforme U[0,1]

Passo 3. Calcule $f(x_1), ..., f(x_{tnuv})$;

Passo 4. Faça $f^k_{max} = \text{Max}\{f(x_1), ..., f(x_{tnuv})\}$;

Passo 5. Calcule $F^k(x_i) = \dfrac{1}{1 + k^{2.\delta}.(f^k_{max} + 10^{-10})}$, $i = 1 \quad tnuv$

Passo 6. Calcule $p(x_i) = \dfrac{F^k(x_i)}{\sum_{j=1}^{tnuv} F^k(x_j)}$

Passo 7. Calcule $x_{med} = \sum_{i=1}^{tnuv} x_i.p(x_i)$ e $a = \left(\sum_{j=1}^{tnuv} x_i - \sum_{i=1}^{tnuv} x_i.p(x_i)\right)^{1/3}$

Passo 8. Faça $C_i = 0$ se $f(x_i) = f_{max}$ 1 caso contrário;

Passo 9. Faça $S_i = 1$ se $(x_{med} - a) > f(x_i)$ e -1 caso contrário ;

Passo 10. Calcule $y = r_i + \gamma_2$

Passo 11. Faça $q_i = \gamma \cdot C_i \cdot S_i \cdot (x_{med} - a - x_i)$

- se $l_{inf} \leq q_i + x_i < l_{sup}$ faça $x_i = q_i + x_i$,caso contrário
- se $r_i < \dfrac{A \cdot (l_{sup} - l_{inf})}{k \cdot \beta}$, faça $x_i = x_{med} + \dfrac{A \cdot (l_{sup} - l_{inf})}{k \cdot \beta}$, caso contrario
- $x_i = l_{inf} + \Delta .(l_{sup} - l_{inf})$

Passo 12. Se $k = nite$ pare, caso contrário vá para o Passo 2.

A idéia básica do Talus é considerar a função a ser maximizada como se fosse uma função densidade de probabilidade. O algoritmo é baseado numa seqüência de distribuições de probabilidade; ocorre uma progressiva diminuição da variância e a média vai convergindo para o ponto de máximo.

A cada iteração é atribuída uma probabilidade a cada ponto da nuvem. Esta probabilidade pode ser interpretada como a probabilidade do ponto de máximo estar numa vizinhança daquele ponto. Para que a função densidade de probabilidade seja construída é necessário, como no caso de Algoritmos Genéticos, adotar uma transformação que torne esta função sempre positiva. Por uma questão, de comodidade de implementação computacional optou-se pela função $F^k(x_i) = \frac{1}{1 + 10^{-10} + \left(f^k_{max} - f(x_i)\right)^{\delta}}$, $i = 1, \ldots, tnuv$ (o 10^{-10} é para o caso de $f^k_{max}=0$).

As probabilidades dos pontos da nuvem são dadas por $p(x_i) = F^k(x_i) / \sum F^k(x_j)$

É importante notar que esta função $F^k(x_i)$ foi construída de forma a respeitar a idéia intuitiva de que a probabilidade de na vizinhança do indivíduo de mais alta aptidão estar o ponto de ótimo cresce à medida que o número de iterações cresce; esta escala de crescimento é controlada pelo δ.

O Talus possui as seguintes bases para garantir seu funcionamento :

• A cada iteração o operador *curare* (nome comum a substâncias de origem vegetal que possui uma potente ação paralisante) C_i faz com que seja mantido para a próxima iteração o ponto de mais alta aptidão. Assim a seqüência de valores máximos, $f^1_{\max}, \ldots, f^k_{\max}$, é monotonicamente não decrescente e como por hipótese ,a função é cotada superiormente, a seqüência é convergente .

• A cada iteração toma-se como estimativa do máximo verdadeiro da função a diferença entre a média e a assimetria, isto é $x^r_{m\bar{e}d} a^k$. E através do passo $y \cdot C_i \cdot S_i \cdot (x^k_{m\bar{e}d}\, a^k - x_i)$ cada indivíduo é direcionado para o ponto estimado como ótimo, $x^k_{med}\ a^k$.

• Os passos iniciais são propositadamente pequenos para que pontos de alta aptidão na trajetória do ponto, em direção ao ponto mais apto, não sejam perdidos. Caso seja achado neste processo um ponto de aptidão mais alta o *curare* garante a permanência deste para a próxima iteração. A forma encontrada para implementar os passos variando com a iteração foi adotar uma função ganho_do_passo = $y = \frac{}{k+\gamma_2}$ que multiplica a correção necessária para cada ponto encontrar o ponto de ótimo, $Cj \cdot S_i \cdot (x^k_{med}\, a^k - x_i)$. Note que se $k_1 > k_2$ ter-se-á $\frac{k_1}{k_1+\gamma_2} > \frac{k_2 \cdot \gamma_1}{k_2+\gamma_2}$ e que $k \to \infty$ implica $\frac{}{k+\gamma_2} \to \gamma_1$.

Foi optado ter nuvem inicial obtida de uma distribuição aleatória por uma simples questão de comodidade de implementação. Não haveria grande mudança se optasse por esquadrinhar o espaço de busca em *tnuv* partes em cada eixo e tomar a nuvem inicial como todas

as interseções de todos os hiperplanos para o caso do $\mathbb{R}^n$. Porém este processo seria computacionalmente desvantajoso, exigindo uma série de passos específicos para se obter o reticulado uniforme. Ademais, o método probabilístico é inerentemente tolerante a falhas.

5.4 Fatores que influenciam no desempenho

O tamanho da nuvem tem implicações na precisão final e na probabilidade de que o algoritmo não perca o ótimo numa região estreita do espaço de busca. Porém, primeiramente, se se tem uma nuvem muito grande e o número de iterações for pequeno, as principais propriedades do Talus serão perdidas e ter-se-á quase uma busca aleatória pura. A segunda consideração é que ao se otimizar uma função de $f : \mathrm{R} \rightarrow \mathrm{R}$, está-se realizando para cada iteração cálculos e ordenações com *tnuv* pontos. Já se $f: \mathbb{R}^n \quad \mathbb{R}$, está-se realizando (como ver-se-á no próximo item) a cada iteração cálculos e ordenação com *tnuv* pontos, sendo estes cálculos repetidos para cada variável.

A escolha das constantes γ_1, γ_2 tem papel fundamental no desempenho do algoritmo. Contudo na maioria dos casos não se tem informações *a priori* suficientes para determinar os valores que otimizem o desempenho do algoritmo. Faz-se necessário um pequeno estudo das características da função, como saber sua entropia, variância e assimetria quando esta foi aplicada numa nuvem aleatória .

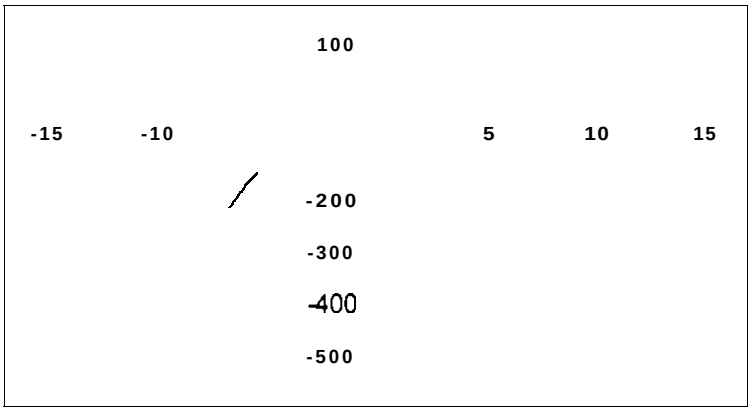
O δ controla a escala de crescimento. É um controle de quão rápido a probabilidade de na vizinhança do indivíduo de mais alta aptidão está o ponto de ótimo, esta probabilidade cresce à medida que o número de iterações cresce. As mesmas considerações que foram feitas em

relação a γ_1, γ_2 quanto ao estudo da função se aplicam em relação a δ, porém como o algoritmo tem se mostrado extremamente eficiente para $\delta=1$ numa ampla gama de funções este estudo é menos importante que no caso de γ_2.

A constante *P* tem seu papel vinculado na etapa que garante a permanência dos pontos dentro do conjunto viável e em particular próximo a média. Como a saída dos pontos do conjunto viável não deve ocorrer habitualmente seu papel é, neste ponto de vista de caráter secundário. No algoritmo toma-se *P=5*.

5.5 Funcionamento do Talus

Tome-se como exemplo o funcionamento do Talus para a função $f(x)=4.x.(1-x)$ $f:[-10,10]\rightarrow\mathbb{R}$ com $tnuv=8$, $nit=10$ e $\gamma_1=1$, $\gamma_2=1$

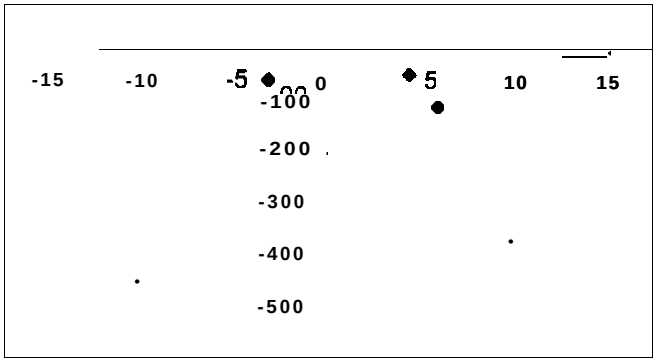


r

I n d i c e

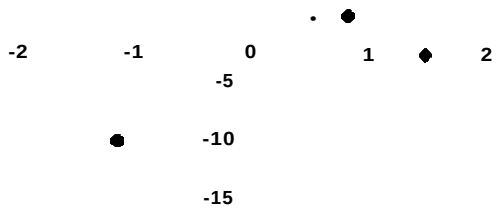
Inicialmente foi gerada a primeira nuvem de números aleatórios:

	$f(x)$
-10	-440
10	-330
2.24	-11.18
3.2	-28.13
3.96	46.99
5.55	-101.4
-3.39	-59.67
-0.88	-6.68



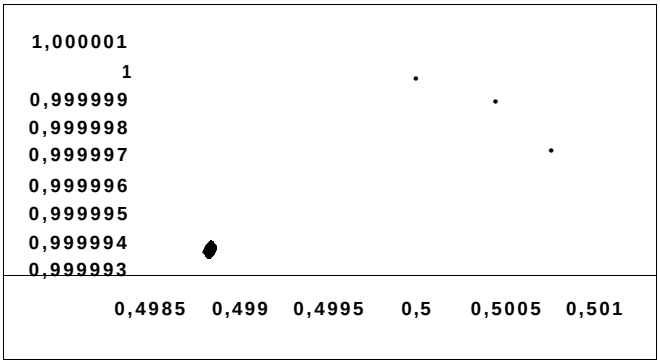
Já na segunda iteração pode-se ver que a nuvem já se concentra em torno do ótimo que é 0,5. Note-se as alterações nas escalas do gráfico.

x	y w
-1.15	-9.86
2.19	-10.37
2.31	-12.16
0.59	0.97
0.77	0.72
1.45	-2.58
-0.04	-0.2
0.56	0.99



Nuvem na segunda iteração e Gráfico

Para a última iteração tem-se:



Pode-se notar que como a função é praticamente simétrica, $x^k_{med} - \alpha^k$ torna-se uma aproximação muito boa do ponto de máximo e é por este motivo que a convergência é tão rápida.

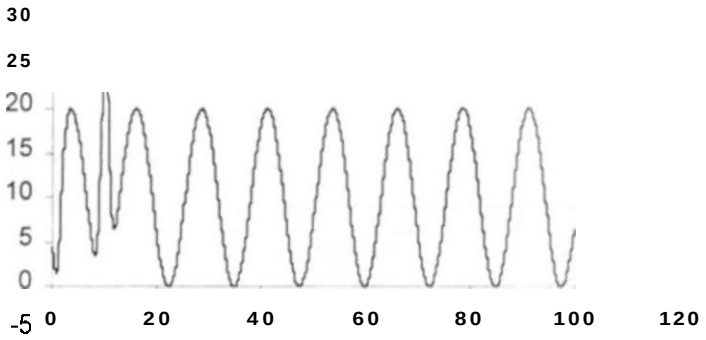
Analise-se agora o funcionamento do Talus para uma função não simétrica. Uma função com as características exigidas é a função de teste abaixo, que pode assumir uma faixa muito grande de configurações :

$$f(x) = b_1 e^{-(c_1-x)^{2\cdot m_1}} + b_2 e^{-(c_2-x)^{2\cdot m_2}} + b_3 x + b_5 (x - c_5)^{m5} e^{(-n_5(x-d_5))} + b_6 \sin(c_6 x - d_6)$$

com

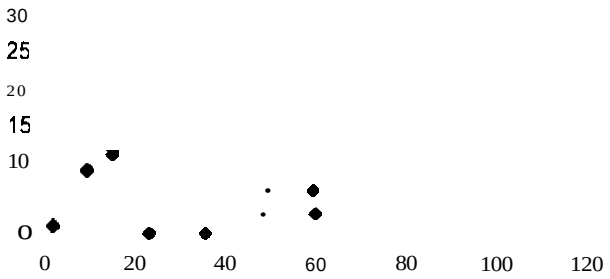
b_1	c_1	m_1	b_2	c_2	m_2	bs	b_4	b_5	c_5	m_5	n_5	d_5	b_6	c_6	d_6
28	10	1	-12	1	1	0	10	10	5	5	4	-5	10	0.5	1

O gráfico da mesma pode ser visualizada abaixo:

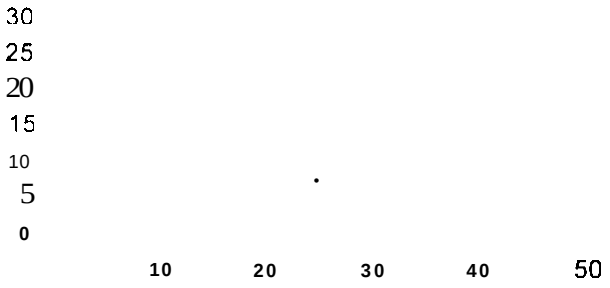


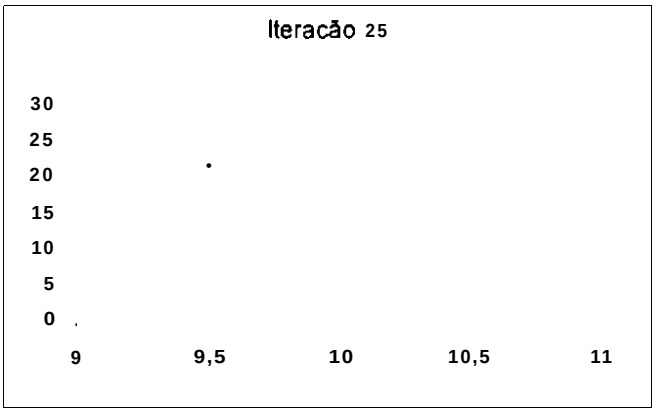
Para esta função será tomado $tmin = 25$ pontos , $nite = 30$ e $\gamma_1 = -1, \gamma_2 = 10$.

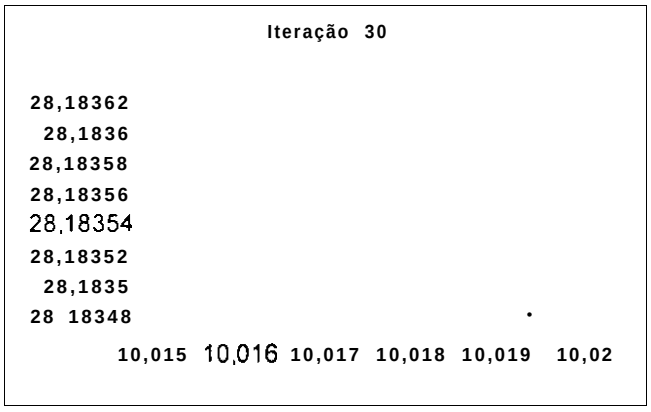
É possível visualizar a nuvem inicial e as nuvens da iteração **20, 25** e **30** pelos gráficos a seguir:



Iteração 20







O ponto de máximo desta função é 10.01742.

5.6 O Algoritmo Talus - caso multidimensional

Entradas: Função objetivo $f : \mathbb{R} \to \mathbb{R}^n$
Dimensão do espaço - *dim*
Número de iterações - *nite*
Tamanho da nuvem - *tnuv*
Constantes - γ_1, γ_2, P, δ
Espaço de busca - l^j_{inf}, l^j_{sup} $j=1,..,dim$

Faça $k = 1$;

P1. Faça $x^j_1 = l^j_{inf}$ e $x^j_2 = l^j_{sup}$ e $x^j_i = l^j_{inf} + A.(l^j_{sup} - l^j_{inf})$, onde $i = 1..., tnuv j = 1, ..., dim$ e A é gerado através de uma função de densidade uniforme U[0,1]. Observe que

P2. Calcule $f(x_1), ..., f(x_{tnuv})$;

P3. Faça $f^k_{max} = \text{Max}\{f(\vec{x}_1), ..., f(\vec{x}_{tnuv})\}$;

P4. Calcule $F^k(x_i) = \dfrac{1}{1+k^{2\cdot\delta} \cdot f^k_{\max} + 10^{-10}}$, $i = 1....fawv$

P5. Calcule $p(\vec{x}_i) = \dfrac{}{tnuv}$

P6. Calcule $x^j_{med} = \sum_{i=1}^{tnuv} x^j_i . p(x^j_i)$ e $\alpha^j = \left(\sum_{m=1}^{tnuv} x^j_m - \sum_{i=1}^{tnuv} x^j_i . p(x^j_i) \right)$
$j = 1....di$

P7. Faça $C_i = 0$ se $f(x_i) = f_{max}$ e 1 caso contrário;

P8. Faça $S_i = 1$ se $f(x^j_{med} - \alpha^j) > f(x_i)$ e -1 caso contrário;

P9. Calcule $\gamma = \dfrac{k \cdot \gamma_1}{k + \gamma_2}$;

P10. Faça $q^j_i = \gamma \cdot C_i \cdot S_i \cdot (x^j_{med} - \alpha^j - x^j_i)$

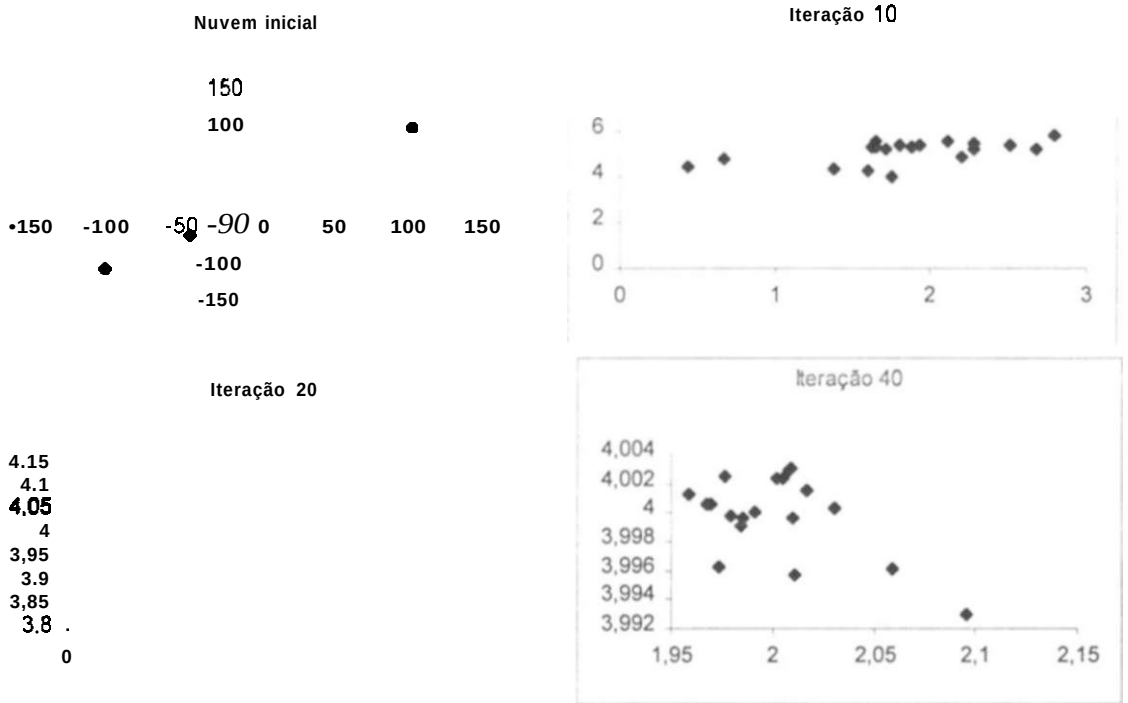
se $l^j_{\inf} \le q^j_i + x^j_i < l^j_{\sup}$ faça $x^j_i = q^j_i + x^j_i$, caso contrário

se $l^j_{\inf} < x^j_{med} + \dfrac{\Delta \cdot (l^j_{\sup} - l^j_{\inf})}{k \cdot \beta} < l^j_{\sup}$, faça $x_i = x_{med} + \dfrac{A \cdot (l^j_{\sup} - l^j_{\inf})}{k \cdot \beta}$, caso contrário $x^j_i = l^j_{inf} + \Delta.(l^j_{sup} - l^j_{inf})$ para $j = 1, ..., di$

P11. Se $k = nite$ pare, caso contrário vá para o Passo 2.

Pode-se notar que no caso multidimensional, na aptidão, é utilizada a informação do vetor completo. A partir daí é utilizada esta informação para o calculo das probabilidades dos pontos. A média e a assimetria se tornam vetores, de dimensão igual à dimensão do argumento da função, referentes à média e à assimetria para cada coordenada calculada com as probabilidades de cada ponto. O *curare* e o sinal mantêm as mesmas características que no caso unidimensional. Por fim o calculo do passo é equivalente ao caso unidimensional para cada coordenada de cada vetor.

Para uma visualização do seu funcionamento tome-se a função $f(x,y) = -(x-2)^4 - (y-4)^2$, cujo ponto de ótimo é (2,4). Para esta função será tomado $tnuv = 20$ pontos , $nite = 40$ e $\gamma_1 = 1$, $\gamma_2 = 10$. O Talus encontrou (2.02242 , 3.9998) como resposta.



com um espaço de busca dado por $[-100, 100] \times [-100, 100]$

5.7 Talus Versão - MinMax

Inicialmente deve-se notar que para fazer o Talus funcionar como um algoritmo de minimização é necessário faze-se as seguintes modificações:

Passo P3 pelo P3 $_m$: Faça $f^k_{min} = \text{Min}\{f(\vec{x}_1) \ldots f(x_{tmv})\}$;

Passo P4 pelo P4 $_m$: Calcule $F^k_{\min}(x_i) = \dfrac{}{1 + k^{2\omega_i}} \left(f(\vec{x}_i) - f^k_{\min} \right)$ .i

Passo P7 pelo P7 $_m$: Faça $C_i = 0$ se $f(\vec{x} = f_{min}$ e 1 caso contrário ;

Passo P8 pelo P7 $_m$: Faça $S_i = 1$ se $f(x^j_{med} - \alpha^j) < f(\vec{x}_i)$ e -1 caso contrário ;

A idéia básica para o problema de MiniMax é fazer com que o Talus, a cada iteração, resolva o como um problema de maximização e como um de minimização. O passo para cada variável seria o passo referente ao problema de maximização ou (exclusivo) referente ao problema de minimização dependendo do tipo de variável que se está tratando.

Esta vertente do Talus se mostrou simples e eficiente. incorporando de forma imediata as idéias básicas do algoritmo. A maneira de fazer a escolha do sentido de otimização foi através de um vetor V_j que assume o valor 1 caso o sentido seja de maximização e assume o valor 0 caso contrário .

5.7.1 Algoritmo Talus Versão - MinMax

Entradas: Função objetivo $f : \mathbb{R} \quad \mathbb{R}^n$
Dimensão do espaço - *dim*
Número de iterações - *nite*
Tamanho da nuvem - *tnuv*
Constantes - γ_1, γ_2 , *fi* , δ
Espaço de busca - l^j_{inf} , $l^j_{sup}\ j=1,..,dim$
Sentido de otimização de cada variável - V_j , $j=1,..,dim$

1. Faça $k = 1$;

2. Faça $x>_1 = l^j_{inf}$ c $x>_2 = l^j_{sup}$ e $x> = l^j_{inf} + A .(l^j_{sup} - l^j_{inf})$, onde $i = 1\ldots, tnuv\ j = 1\ldots$ dim e A é gerado através de uma função de densidade uniforme 1 I|0,1 |. Observe que

3. Calcule $f(\bar{x}\ \backslash)$,.... $f(x_{tnuv})$

** Parte do Algoritmo de Maximização **

4. Faça $f^k_{max} = \text{Max}\{f(x\backslash), ..., f(x_{tnuv})\}$;

5. Calcule $I^k_{\max}(x_i) \Big/ \left(1 + k^{2\cdot\delta} \cdot \left(f^k_{\max} - f(x_i)\right) \Big/ \left(f^k_{\max} + 10^{-10}\right)\right), i = 1, \ldots, tnuv$

6. Calcule $p^{\max}(\vec{x}_i) = {}^{I}\max(x_i)$

7. Calcule $x^{j,\max} = \sum_{i=1} x^j . p_{\max}(x^j_i)$

$$\text{tfmax} = \left(\sum_{m=1}^{tnuv} x^j_m - \sum_{i=1}^{tnuv} .p_{\max}(x^j_i)\right)^3\Big)^{1/3}, j = 1,\ldots,\text{di}$$

8. Faça $C^{\max} = 0$ se $f(\vec{x}_i) = f_{max}$ e 1 caso contrário ;

9. Faça $S^{max}_{i} = 1$ se $f(x^{j,\max}_{med} - a^{j}_{max}) > f(\vec{x}_i)$ e -1 caso contrário ;

** Parte do Algoritmo de Minimização

10. Faça $f^{k}_{min} = \text{Min}\{f(\vec{x}_1), ..., f($

Calcule $F^{k}_{\min}(x_i) = \dfrac{1}{1+k^{2\cdot\delta}}$ $\left(f(\vec{x}_i) - f^{k}_{\min}\right)$, $i = 1, \ldots, tmuv$

12. Calcule $p^{\min}(\vec{x}_i) = tmuv$

13. Calcule $x^{j,\min}_{med} = \sum^{tmuv} x^{j}_{i} \cdot p_{\min}(x^{j}_{i})$ e

$a^{j}_{\min} = \left(\sum_{m=1}^{tmuv} \left(x^{j}_{m} - \sum_{i=1}^{tmuv} x^{j}_{i} \cdot p_{\min}(x^{j}_{i})\right)^{3}\right)^{1}$, $j = 1, \ldots, di$

14. Faça $C^{\min}_{i} = 0$ se $f(\vec{x}_i) = f_{min}$ e 1 caso contrário ;

15. Faça $S^{\min}_{i} = 1$ se $(x^{j,m}_{me} - a^{j}_{\min}) > f(\vec{x}_i)$ e -1 caso contrário ;

16. Calcule $y = \overline{\quad}$

17. Se $V_j = 1$ faça $q^{j}_{i} = \gamma \cdot C^{\max}_{i} \cdot S^{\max}_{i} \cdot \left(x^{j,\max}_{med} - a^{j,\max} - x^{j,\max}_{i}\right)$

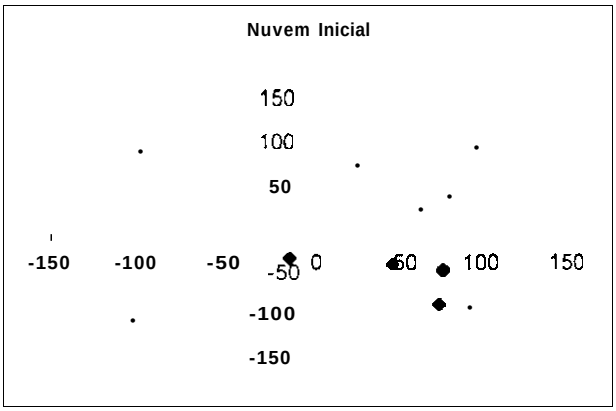
18. caso contrário faça $q^{j}_{i} = \gamma \cdot C^{\min}_{i} \cdot S^{\min}_{i} \cdot \left(x^{j,\min}_{med} - a^{j,\min} - x^{j,\min}_{i}\right), j = 1, .., dim$

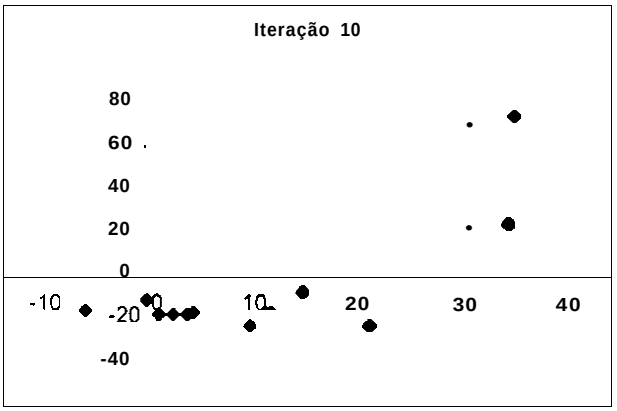
19. se $l^{j}_{\inf} < q^{j}_{i} + \quad < l^{j}_{\sup}$ faça $x^{j}_{i} = q^{j}_{i} + x^{j}_{i}$,caso contrário

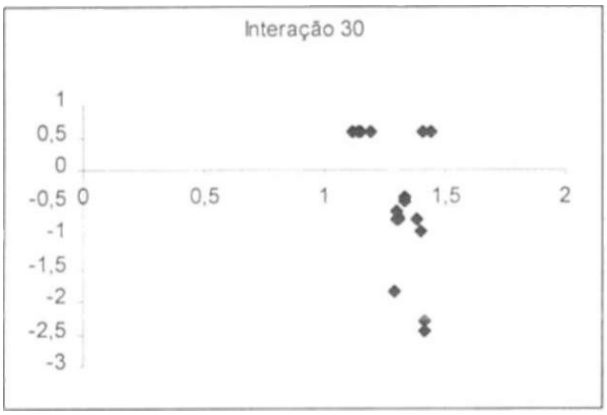
20. se $l^{j}_{\inf} \le x^{j}_{med} + \dfrac{\Delta \cdot (l^{j}_{\inf} - l^{j}_{\sup})}{k \cdot \beta} < l^{j}_{\sup}$, faça $x^{j}_{i} = x^{j}_{med} + \dfrac{\Delta \cdot (l^{j}_{\inf} \sim l^{j}_{\sup})}{}$, caso contrário $x^{j}_{i} = l^{j}_{inf} + A.(l^{j}_{inf} - l^{j}_{sup})$ para $j = 1....di$

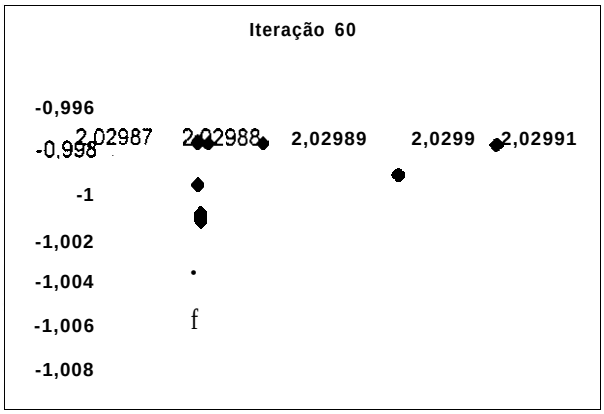
21. P121. Se $k = nite$ pare, caso contrário vá para o Passo 2.

Para ilustrar o uso do Talus com ferramenta para resolução de MiniMax tome-se a seguinte função $flx,y) = (x-2)^2 - (y+1)^2$. Vê-se que o ponto $x=2$ e $y=$ -\ é um ponto de sela. Para esta função será tomado $tnuv=20$ pontos , $nite = 100$ e $\gamma_1 = 1$, $\gamma_2 = 10$. Pode-se visualizar o funcionamento do Talus através das nuvens representadas pelos gráficos abaixo .



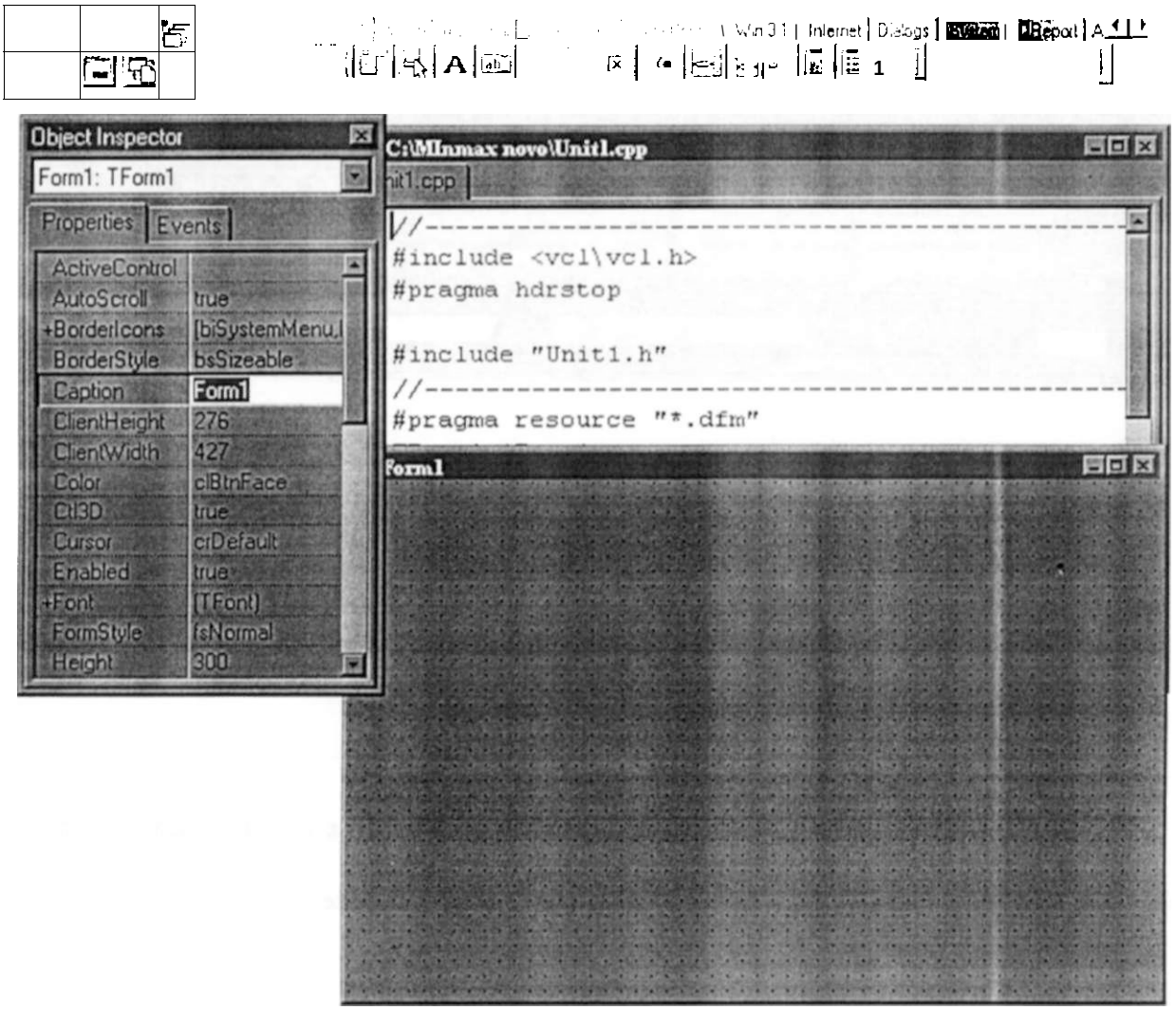




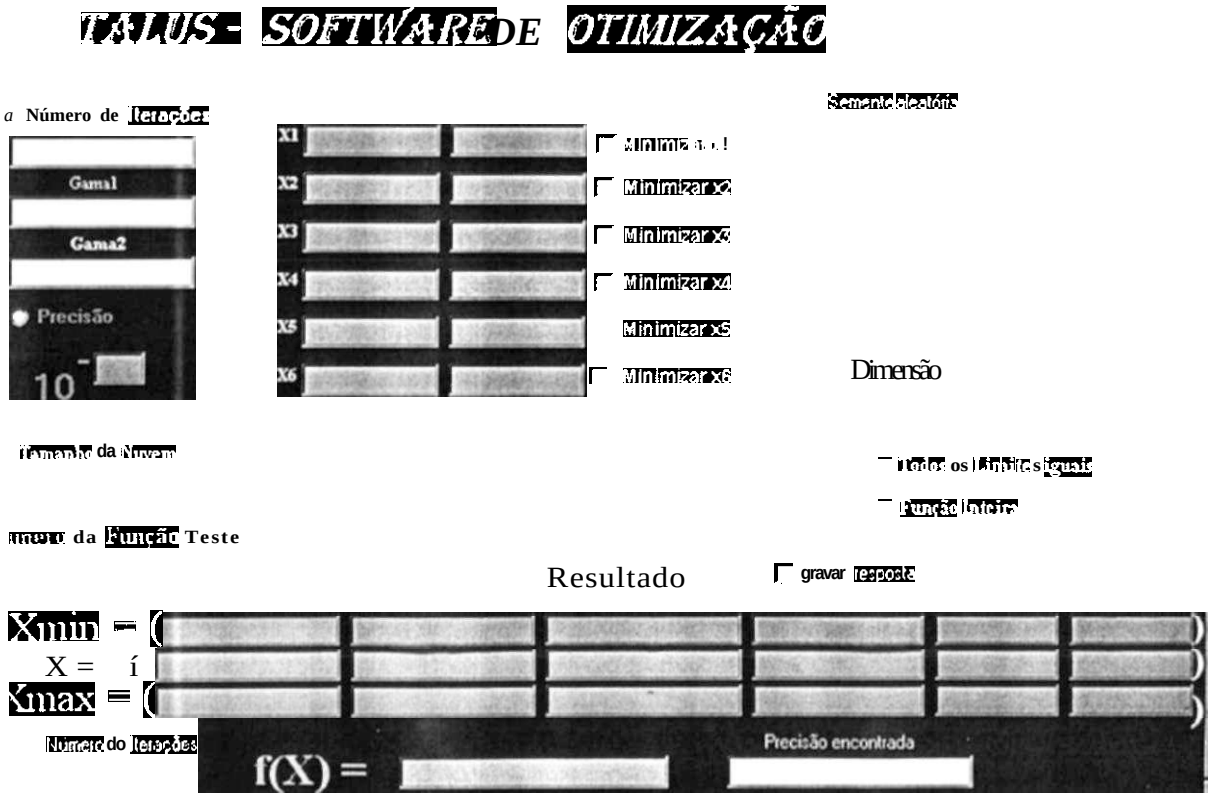



```
Função número=23
nite=100      tnuv=20     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=2.029871
 x2=-1.000007
   f=0.000892
```

Através de um estudo das principais características das linguagens de programação. foi constatada a adequação da Linguagem C++ para a implementação do algoritmo Talus. Inicialmente foi utilizado o compilador Borland C++, porém a amigabilidade do ambiente fornecido pelo Borland C++ para a execução do programa era muito pobre . Este inconveniente foi eliminado quanto se optou pelo uso do C++ Builder (também da Borland). O ambiente fornecido pelo compilador pode ser visualizado na figura abaixo:



Ao ser compilado o Talus apresenta a seguinte configuração (tela):



5.8.1 Considerações em relação ao Software Talus

Não foi de forma alguma preocupação central neste trabalho construir um software otimizado em termos de código.

A limitação quanto ao número de variáveis, 6 variáveis, é uma limitação apenas de apresentação. Na estrutura do programa o número de variáveis é flexível.

A entrada da função ser somente no código fonte se deve ao fato de que a construção de um "compilador" para leitura da mesma seria uma melhoria marginal em relação ao objetivo deste trabalho.

A escolha automática do γ_1 e do γ_2, como já foi mencionado, não **é** de forma alguma trivial. Então foi deixado livre sendo em quase todos os casos tomado $\gamma_1 = 1$ e $\gamma_2 = \frac{nite}{10}$.

Ao se optar pela precisão o programa só termina quando encontrar a precisão estabelecida. Isto é, quando $\frac{f(x_{med}^{k+1}) - f(x_{med}^{k})}{f(x_{med}^{k})} < \frac{x_{med}}{10^{-20}} < precisão\ estabelecida.$

O gerador de números aleatórios do C++ Builder é muito deficiente. Como uma forma de sanar este problema foi implementado o método da congruência multiplicativa para geração dos números aleatórios.

5.8.2 Descrição do código fonte

Bibliotecas do C++ necessárias para o funcionamento do programa

```
#include <vcl\vcl.h>
#pragma hdrstop
#include <stdio.h>
#include<math.h>
#include<stdlib.h>
#include "Pmimnov.h"
```

Declaração das Variáveis

```
double FKa(double fm,double fq.int u);

int nrand (int l,int sem);
double Rrand(int sem);
```

```
FILE *arq1;
```

Entrada de dados

```
sem=atoi(Esem->Text.c_str());

fmax2=fmax1 =-10000000.0;
fmin2=fmin1=10000000;

switch(dim)
 {
case 0:
xmax[0]=li[0]=atof(E5->Text.c_str());
ls[0]=atof(E6->Text.c_str());
break;
case 1:
xmax[0]=li[0]=atof(E5->Text.c_str());
ls[0]=atof(E6->Text.c_str());
```

Testes de seleção do sentido de otimização

```
i f (CheckBox1 ->Checked) {V1 [0]= 1; V2[0]=0;}
else {V1[0]=0;V2[0]=1;}
if (CheckBox2->Checked) {V1 [ 1 ]= 1;V2[ 1 ]=0;}

if (CheckBox6->Checked){V1[5]=1;V2[5]=0;}
else {V1[5]=0;V2[5]=1;}
```

Geração da Nuvem inicial

```
for(k=0;k<=dim;k++)
{
nuv[0][k]=li[k];
nuv[1][k]=ls[k];
fprintf(arq,"%f\n", nuv[0][k]);
fprintf(arq,"%f\n", nuv[ 1 ][k]);
for(l=2;l<=tnuv;l++)
```

Gera nuvem para caso Discreto

```
   {if (CB2->Checked)
{nuv[l][k]=nrand(ls[k],sem)-nrand(li[k],2*dim*sem)+li[k];}
else
{nuv[l][k]=Rrand(sem*dim)*(li[k]+(ls[k]-li[k])*rand()/RAND_MAX);}
```

```
fprintf(arq,"%f\n",nuv[l][k]);
   }


for (k=0;k<=100000;k++)
{

kk=kk+1; ..... Testa o número de iterações----------

if((RBI->Checked)&&(k>=nint)) break;

                                        /* Parte de Minimização */

soma1=0.;
                for(n=0;n<=dim;n++)
                 {
                    xmed1[n]=0.;
                    assim1[n]=0.;
                 }
                                        Cálculo do fmin
      for(l=0;l<=tnuv;l++)
       {
         if(funcao(nuv[l],nf)>fmax1)
          {  fmax1=funcao(nuv[l],nf);}
         if((funcao(nuv[l],nf)<fmin1))
            {fmin1 =funcao(nuv[l],nf);}
       }
                                        Cálculo do Curare
      for(l=0;l<=tnuv;l++)
       {
       soma1=soma1 +FKi(fmin1,funcao(nuv[l],nf),k);
       cur1[l]=(funcao(nuv[l],nf)>fmin1)? 1:0;
        }
                        Cálculo da Média e das probabilidades de cada ponto
      for(n=0;n<=dim;n++)
          for(l=0;l<=tnuv;l++)
         {
           p1[l]=FKi(fmin1,funcao(nuv[l],nf),k)/soma1;
           xmedl[n]=nuv[l][n]*pl[l]+xmed1[n];
           }
         fn1 =funcao(xmed1,nf);
                                        Cálculo da Precisão
```

```
if (((fn1-fv1)/(fv1+pow(10,-15)))>0) o=((fn1-fv1)/(fv1+pow(10,-15)));else
o=-((fn1-fv1)/(fv1+pow(10,-15)));
if((RBP->Checked)&&(o<pow(10,prec))&&(k>=2))-- Teste de termino do programa
break;
```

Cálculo da Assimetria

```
for(n=0;n<=dim;n++)
    for(l=0;K=tnuv;l++)
    {
     assim1[n]=pow((nuv[l][n]-xmed1[n]),3.0)*pl[l]+assim1[n];
    }

 for(n=0;n<=dim;n++)
{sx=(assim1[n]>0)?1:-1;
assim1[n]=sx*pow(fabs(assim1[n]),1./3.);}
```

Teste para o caso do programa não ter encontrado a precisão desejada e gama estar próximo de 0

```
if((RBP->Checked)&&(gama<0.001))
{k=1;}
 gama=(doublc)gamal *(k+1)/(k+1+gama2);

    for(n=0;n<=dim;n++)
    AX[n]=xmed1[n]-assim1[n];
```

Cálculo do Sinal

```
    for(l=0;l<=tnuv;l++)
    { s1[l]= (funcao(AX,nf)<=funcao(nuv[l],nf)) ?1.:-1; }
```

Parte do algoritmo destinada a Maximização

```
soma2=0.:

            for(n=0;n<=dim;n++)
            {
               xmed2[n]=0.:
               assim2[n]=0.;
             }


    for(l=0;l<=tnuv;l++)
```

. . . . . Simétrica a parte anterior

CÁLCULO DO PASSO

```
for(n=0;n<=dim;n++)
{
      for(l=0;l<=tnuv;l++)
    {.....seleciona se a função é inteira.....-
       if(CB2->Checked)
          { if((k<=3)&&(((cur1[l]==1)&&(V1[n]==1))||((cur2[l]==1)&&(V2[n]==1))))
 .....se a função for inteira nos pontos não max faça nas primeiras iterações o passo igual
a 1 na direção do máximo.....
passo=(gama*(curl [l]*sl [l]*(xmedl [n]-assim 1 [n]-nuv[l][n])*V1[l]+cur2[l]*s2[l]*(xmed2[n]-ass
im2[n]-nuv[l][n])*V2[l])>0)? 1:-1;
                  arredondamento do passo para funções inteiras
else passo = arredond(
gama* (cur1 [l] * s 1 [l] * (xmed1[n]-
im2[n]-nuv[l][n])));
                }

          else
Passo tomado como combinação do passo de max e do de min .Porém V1[n] =0 ->V2[n]=1 e
vice-versa
passo =
gama*(cur1[l]*sl [l]*(xmedl [n]-assim1[n]-nuv[l][n])*V1[n]+V2[n]*cur2[l]*s2[l]*(xmed2[n]-ass
im2[n]-nuv[l][n]));
      q=(double)nuv[l][n]+passo;
              Testa se o ponto acrescido do seu passo saiu do espaço de busca
          if ((q>=li[n])&&(q<=ls[n]))
              { nuv[l][n]=q;}
          else
          { if (CB2->Checked) nuv[l][n]=arredond(xmed1[n]*V1[n]+xmed2[n]*V2[n]+
(-1+2*rand()/RAND_MAX)*(ls[n]-li[n])/((k+1)*beta));
              Se saiu joga o ponto aleatoriamente próximo ao ponto médio
            else nuv[l][n]=xmed1[n]*V1[n]+xmed2[n]*V2[n]+ (-1+2*rand()/RAND_MAX)
*(ls[n]-li[n])/((k+l)*beta);

              if ((nuv[l][n]>=ls[n])||(nuv[l][n]<=li[n]))
        Se novamente saiu joga o ponto aleatoriamente dentro do espaço de busca
              {  if(CB2->Checked)
nuv[l][n]=arredond(li[n]+(ls[n]-li[n])*rand()/RAND_MAX);
                 else  nuv[l][n]=li[n]+(ls[n]-li[n])*rand()/RAND_MAX;
              }
          }
```

Retorna os valores encontrados para f

```
E23->Text=(funcao(xmed,nf));
```

Encontra os limites da nuvem na última iteração

```
for(k=0;k<=dim;k++)
{xmax[k]=-10000.;
xmin[k]=1000000.;
 for(l=0;l<=tnuv;l++)
  {
if(nuv[l][k]>=xmax[k])
xmax[k]=nuv[l][k];
 if(nuv[l][k]<=xmin[k])
xmin[k]=nuv[l][k];
  }
```

Retorna os valores encontrados para Xmed Xmax Xmin

```
 switch (dim)
   {
   case 0:

  if(CB2->Checked)
   {E17->Text=aredond(xmed[0]);
   fprintf(arq1,"x1=%f    f=%f \n", aredond(xmed[0]),funcao(xmed,nf));}
   else
   {fprintf(arq1,"x1=%f    f=%f \n", xmed[0],funcao(xmed,nf));
   E17->Text=xmed[0];
   Emin1->Text=xmin[0];
   Emax1 ->Text=xmax[0];}
    brcak;
```

Funções a serem otimizadas

```
double funcao(double x[10],int nf)
{
double m1,m2,m5,n5,b1,b2,b3,b4,b5,b6,c1,c2,c5,c6,d5,d6,V,ax1,ax2,ax3,ax4;
int i;
switch (nf)
{
case 1:
b1=28.;


ax3=-n5*(x[0]-d5);
ax4=pow(x[0]-c5,m5);
 V=(b1*exp(ax1)+b2*exp(ax2)+b3*x[0]+b4+b5*ax4*exp(ax3)+b6*sin(c6*x[0]-d6));
```

```
break;
case 2:


}
return V;
}
```

Função Normalizadora para máximo

```
double FKa(double fm,double fq,int u)
{return (double)( 1./(1.+(u+1)*(u+1)*(fm-fq)));}
```

Função Normalizadora para mínimo

```
double FKi(double fm,double fq,int u)
{return (double)(1./(1.+(u+1)*(u+1)*(fq-fm)));}
```

Função que faz o arredondamento

```
double arredond (double nx)
 {if (fmod(nx,1) >0.5)
 return  floor(nx)+1;
 else return floor(nx);
 }
```

Implementação do método da congruência multiplicativa para geração de números pseudo - aleatórios

```
double Rrand(int sem)
   {  double r;
   int n;
   r=123456789;
   for (n=0;n<=sem;n++)
   r=fmod(r* 100003,pow( 10,10));
   if(r/pow(10,10)<=l) return (r/pow(10,10));
   else return (1);
   í
```

As partes complementares ao programa podem ser encontradas no Anexo 1.

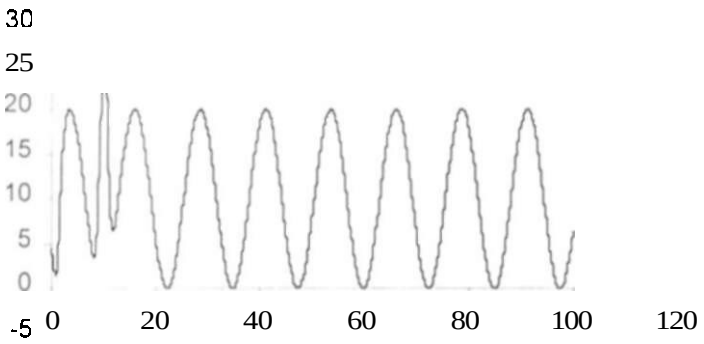
A funções apresentadas aqui são especialmente construídas para tesar a acurácia dos algoritmos de otimização em achar o ponto de ótimo. Neste capitulo será apresentado o resultado da aplicação do Talus neste tipo de função. Estas podem ser testadas no mesmo através do número da respectiva função.

Função 1: Maximize $f(x)$ $x \in [0, 100]$

$$f(x) = b_1 e^{-(c_1 - x)^{2 \cdot m_1}} + b_2 e^{-(c_2 - x)^{2 \cdot m_2}} + b_3 x + b_5 (x - c_5)^{m5} e^{(-n_5 (x - d_5))} + b_6 \sin(c_6 x - d_6)$$

fix) com os seguintes parâmetros:

$b,$	c_1	m_1	b_2	c_2	m_2	bs	b_4	b_5	Cs	m_5	n_5	d_5	b_6	c_6	d_6
28	10	1	-12	1	1	0	10	10	5	5	4	-5	10	0.5	0.1



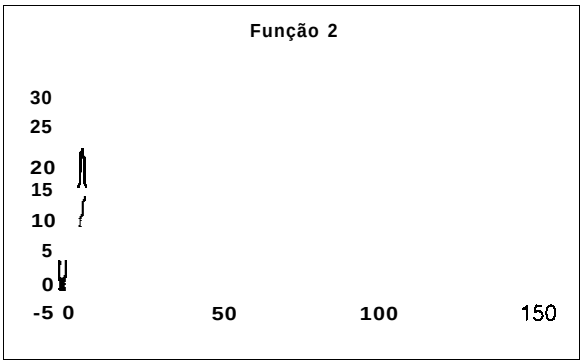
Solução Talus :

```
Função número=l
nite=100    tnuv=20    gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=10.017422    f=28.183596
```

Função 2 :Maximize f(x) ,$x \in [0, 100]$

fix) com os seguintes parâmetros:

$b,$	c_1			c_2	nu	b_3	b_4		c_5		n_5	d_5		$Cr,$	d_6
10	5	1	-12	1	1	0	10	10	5	5	4	-5	10	0.1	0.1



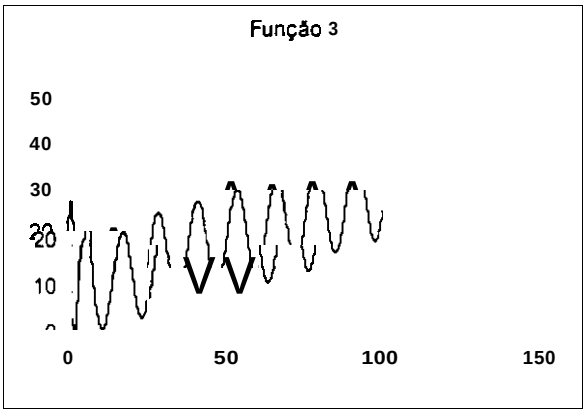
Solução Talus:

```
Função número=2
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=5.046062     f=23.915372
```

Função 3 : Maximize f(x) $[0, 100]$

fix) com os seguintes parâmetros:

b_1	c_1	m_1	b_2	c_2	m_2	b_3		b_5	c_5	m_5	n_5	d_5	b_6		
10	5	1	-12	1	1	0.2	10	10	5	5	4	-5	10	0.1	0.1



Solução Talus

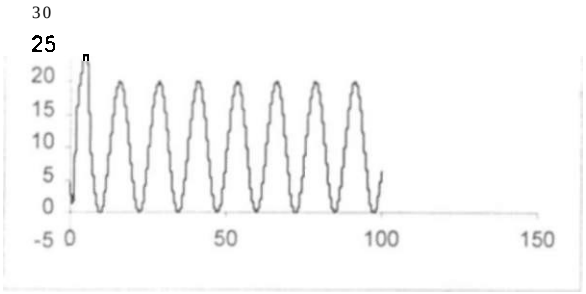
```
Função número=3
nite=100     tnuv=20     gama1=1.000000  gama2=1.000000 precisão=0.000000
x1=91.386208     f=38.269238
```

Função 4 : Maximize $f(x), x \in [0, 100]$

$f(x)$ com os seguintes parâmetros:

b_1	c_1	m_1	b_2	c_2	m_2	b_3	b_4	bs	c_5	m_5	n_5	d_5		c_6	d_6
10	5	1	-12	1	1	0	10	10	5	5	4	-5	10	0.5	0.1

Função 4



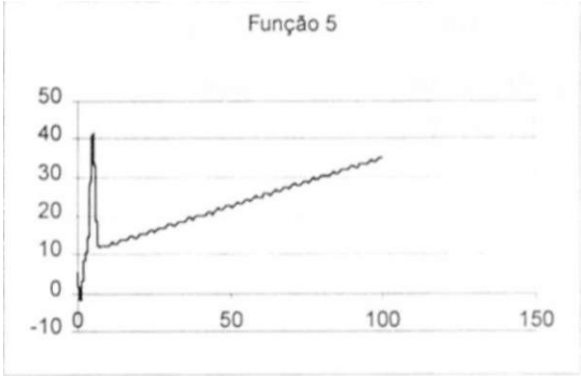
Solução Talus :

```
Função número=4
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=4.825835     f=27.071581
```

Função 5 : Maximize fí» $x \in [0, 100]$

fix) com os seguintes parâmetros:

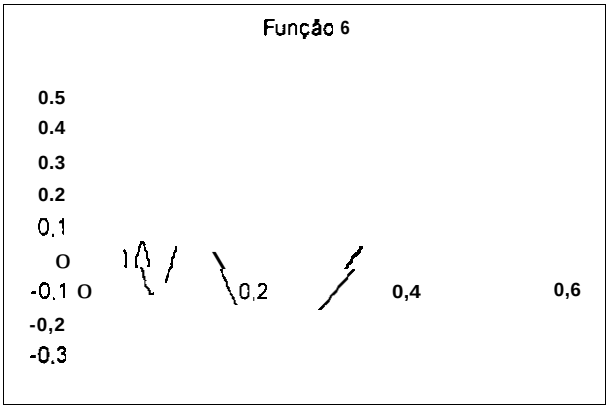
b_1	c_1	m_1	b_2	c_2			b_4	b_5	c_5	m_5		d_5		c_6	d_6
30	5	1	-12	1	1	0.25	10	10	5	5	4	-5	0.35	2	2



Solução Talus:

```
Função número=5
nite=100     tnuv=40     gama1=1.000000  gama2=15.000000 precisão=0.000000
x1=5.002416     f=41.596453
```

Função 6 : Minimize $f(x) = x.sen(1/x)$ $x \in [0,05, 0,5]$ $x^* = 0,223$ (Hanssoun (1995) pag.418 423) _____



Solução Talus:

```
Função número=6
nite=100     tnuv=20     gama1=1.000000  gama2=30.000000 precisão=0.000000
x1=0.222548     f=-0.217234
```

Função 7 : Minimizar $f(x_1,x) = 100 \cdot (x_2 - x_1^2 + ^2(1 - x_1)^2$

Proposta em Himmelblau (1972) pag.394 problema 2

Ponto inicial : $X^\circ = [-1, 2.1]^7$

Solução : $x^* = [1, 1]$

Solução Talus:

```
Função número=7
nite=100     tnuv=50     gama1=1.000000  gama2=15.000000 precisão=0.000314
x1=1.000121
 x2=1.000246
   f=0.000000
li=-10;
ls=10;
```

Função 8 : Minimizar $f(x_1, x_2) = 4 \cdot (x_1 - 5)^2 + (x_2 - 6)^2$

Ponto inicial : $x^0 = [8, 9]^7$

Solução: $x^* = [5, 6]^7$, $f(x^*) = 0$

Problema 25 **pag.** 426 **de Himmelblau** (1972)

```
Função número=8
nite=100     tnuv=100     gama1=1.000000  gama2=10.000000 precisão=0.418375
x1=5.000204
 x2=6.001161
   f=-0.000002
li=-10;
ls=10;
```

Função 9 : Minimizar $f(x_1, x_2) = (x_1 \cdot x_2) \cdot (1 - x \setminus)^2 \cdot [1 - x_1 - x_2 (1 - x)^5]$

Ponto inicial $x^0 = [-1.2, 1]^7$ ***fix")*** $= 26.656$

Solução : $x^* = [1, -]^T$ ***ou*** [0, ***-Y ou*** $[-, 0]^T$ ***fix')*** $= 0$

Problema 27 pag. 427 de Himmelblau (1972)

Soluções Talus:

```
Função número=9
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=0.491425
 x2=0.000000
   f=0.000000
li=-10;
ls=10;
```

```
Função número=9
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=0.000122
 x2=0.001814
   f=0.000000
li=-10;
ls=10;

Função número=9
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=1.000000
 x2=3.266151
   f=0.000000
li=-10;
ls=10;
```

Função 10 : Minimize $f(x_1, x_2, x_3) = (x_1 + 10 \cdot x_2^2)^2 + 5 \cdot (x_3 - x_4 + (x_2 2 \cdot x_3)^4 + 10 \cdot (x_1 - x_4$

Ponto inicial : $x^0 = [3, -1, 0, 1]^T \; fix^0) = 215 \; ; x^0 = [1, 1, 1, 1]^T \; f(x^0) = 125$

Solução : $x^* = [0, 0, 0, 0]^T \quad f(x^*) = 0$

Problema 26 pag. 427 de Himmelblau (1972)

Solução Talus :

```
Função número=10
nite=100     tnuv=100     gama1=1.000000  gama2=10.000000 precisão=0.296615
x1=-0.008192
 x2=0.000725
  x3=0.004024
 x4=0.004131
 f=0.000001
li=-5;
ls=5;
```

Função 11: Minimize $f(x_1, x_2) = 0.5 \cdot x_1^2 + 0.5 \cdot [1 - \cos(2 \cdot x_1)] + x_1^2$

Alguns mínimos locais e 1 mínimo global $x^* = [0, 0]^T$

Solução Talus:

```
Função número=11
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=92.117639
x1=0.000005
 x2=0.000007
   f=0.000000
li=-5;
ls=5;
```

Função 12 : Minimizar $f(x_1,x_2 \doteq (x_2 - x_1^2)^2 (\ - x_1)^2$

Solução :$x^* = [1,1]^T \quad f(x^*) = 0$

Solução Talus:

```
Função número=12
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=0.438657
x1=1.000003
 x2=1.000010
   f=0.000000
li=-10;
ls=10;
```

Função 13 : Minimizar $f(x_1,x_2) = (x_2 - x_1^2)^2 + 100 \cdot (1 - x_1)^2$

Proposta cm Himmelblau (1972) pag. 195

Ponto inicial : $x^0 = [-1, 2.1]^T$

Solução :$x^* = [1,1] \ f(x^*) = 0$

Solução Talus:

```
Função número=13
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=5.156862
x1=0.999984
 x2=0.999681
   f=0.000000; li=-10;ls=10
```

Função 14 : Minimizar $f(x_1,x_2)$

$$f(x_1,x_2) = [1.5 - x_1 \cdot (1 - x_2)]^2 + [2.25 - x_1 \cdot (1 - x_2^2)]^2 + [2.625 - x_1 \cdot (1 - x_2^3)]^2$$

Proposta cm Himmelblau (1972) pag. 195

Solução :$x^* = [3, 1/2] \ f(x^*) = 0$

Solução Talus:

```
 Função número=14
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=0.472773
x1=2.999975
 x2=0.499991
   f=0.000000
```

Função 15 : Minimizar $fix\backslash,X2) \ = 4 \cdot x_1^2 + 4 \cdot x_2^2 - 4 \cdot x_1^2 \cdot x_2^2 - 12x_2$

Proposta cm Himmelblau (1972) pag. 195

Solução $:x^* = [1,2]$ *fix>)* = -\2

Solução Talus:

```
Função número=15
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=0.999998
 x2=2.000000
   f=-12.000000
li=-10;
ls=10;
```

Função 16 : Minimizar *ftx\xi)* $= \sum_{.51}^{r\,5} i \cdot \cos[(i+1) - x_1 + i] \quad \sum i \cdot \cos[(i+1) \cdot x_1 + i]$

Para $-10 < x_1 < 10$ *e* - $10 < x_2 < 10$ (*Shubert's function*)Stuckman (1992)

Solução 760 mínimos locais sendo 18 mímimos globais com f_{mm}= - 186.73

Solução Talus :

```
Função número=16
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=-7.708314
 x2=-7.083507
   f=-186.730909
```

```
 Função número=16
nite=100     tnuv=20     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=-7.083506
 x2=-1.425128
   f=-186.730909
```

Função 17 : Maximizar *fix\,x,)* $= \cos(x_1)\ \cos(x_2) \cdot e^{-[(x_1-\pi)^2+(x2-}$

Para $-10 \leq x_1 < 10$ *e* - $10 < x_2 < 10$ (*Easom's function*)Stuckman (1992)

Solução $x^* = [\pi, \pi]$ $f(x^*) = -1$

Solução Talus :

```
Função número=17
nite=100     tnuv=50     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=3.141589
 x2=3.141535
   f=1.000000
```

A comparação imediata com um método de otimização determinístico através da relação erro e número de iterações pode trazer um viés a favor dos métodos probabilisticos. Este falso desempenho muito superior se deve à nuvem que não foi levada em conta no processo. Porém, uma comparação, mesmo sabendo que esta é pouco rigorosa, cabe neste trabalho.

Função de **Shubert**

% erro / Número de iterações

Método	20	50	100	200	500	1,000
Stuckman	91.22	89.32	86.62	86.61	86.63	86.61
Mockus	87.97	87.7	87.15	87.15	86.65	86.65
Perttunen	12.01	12.01	0.39	0.01	0	0
Torn	88.32	87.37	87.01	87.01	87.01	87.61
Sim.Annealing	113	113	113	113	113	113
Zinlinskas	86.67	86.67	86.67	86.67	86.67	*
Shaltenis	88.01	87.23	87.23	87.23	87.23	*

* CPU insuficiente

% erro / Número de iterações

Talus(*tmav=20)*	20	50	100	200	500	1,000
	0.06	$2,18.10^{-3}$	$9,44.10^{-4}$	0	0	0

Função de Easom

% erro / Número de iterações

Método	20	50	100	200	500	1,000
Stuckman	100	100	100	100	100	100
Mockus	100	100	100	100	100	100
Perttunen	54.4	21.66	6.19	2.43	2.15	*
Torn	100	100	100	100	100	100
Sim.Annealing	100	100	100	100	100	100
Zinlinskas	100	100	100	100	100	*
Shaltenis	100	100	100	100	100	*

* CPU insuficiente

% erro / Número de iterações

Talus(*tnuv=20)*	20	50	100	200	500	1,000
	0.8	$5{,}72.10^{-3}$	$4{,}39.10^{-4}$	0	0	0

Tabelas de Stuckman (1992).

Pode-se ver o excelente desempenho do Talus agora de forma comparativa. Vale notar que a nuvem adotada tem apenas 20 pontos o que representa uma nuvem muito pequena.

5.11 O Talus na perspectiva genética

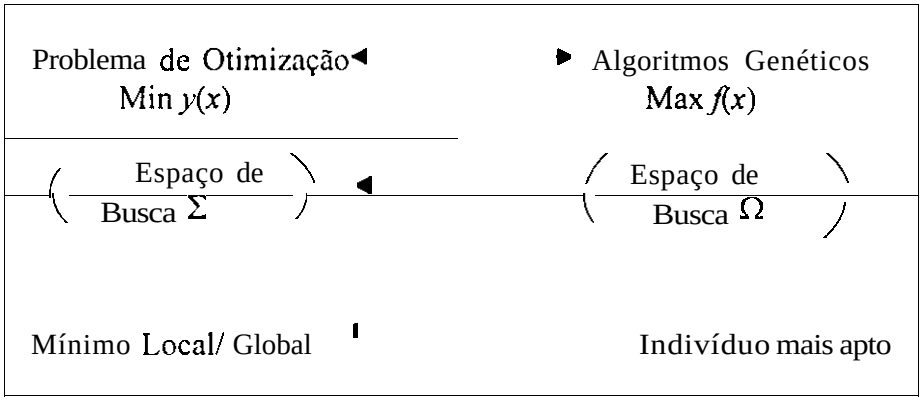
No Talus a informação de toda a população é coletada através da média e da assimetria. É aproveitada mais informação que o simples valor da função de aptidão de cada indivíduo. Vale salientar também que, nenhum dos indivíduos e tirado da população; a estes é dada a oportunidade de aumentar a sua aptidão através de uma busca do ponto de maior valor da população.

Em Hassoun (1995) (Exemplo 8.5.2, pag 447) a função 6 é minimizada através de um algoritmo genético. Para facilitar a compreensão do funcionamento do algoritmo genético utilizado, será transcrita aqui toda a sistemática e as considerações pertinentes.

Como exemplo do processo de solução considere a função mostrada na figura baixo c o seguinte problema de otimização.

Minimize $y(x) = x \cdot sen\{ 1\ lx)$

sujeito a x e [0.05 ,0.5]



Esta função possui dois mínimos locais $0,058$ e $x \cong 0,091$, bem como um mínimo global $x \cong 0,223$. O algoritmo genético padrão será usado para obter o mínimo global desta função.

A primeira coisa a notar é que o espaço de busca aqui é o de valores reais $\Sigma = [0.05, 0.5]$. Como mencionado antes, alguma transformação é necessária para codificar/decodificar o espaço de busca real Σ para algum espaço de *strings* binárias Ω.Neste

exemplo , o espaço de busca binário consiste de *strings* de 6 *bits* ,i.e. $\Omega = \{0,1\}$, foi usado com a transformação decodificada dada por : $D(s) = 0,05 +$ $\cdot (0.05 - 0.5)$

onde $d(s)$ é a representação decimal binária de 6 *bits* . Por exemplo , a representação decimal de 000 0 1 1 é 3 , assim $D(0000\,1\,1) \text{S } 0.071$; Como era esperado os dois pontos extremos são mapeados da seguinte forma $D(0\,0\,0\;0\,0\,0) = 0.05$ e $D(1\,1\;1\;1\,1\,1) = 0.5$

Para estabelecer uma função de aptidão apropriada, recorde-se que o problema minimizar $y(x)$ eqüivale a maximizar $f(s)$ Então algum tipo de transformação é requerida aqui. Neste exemplo , a seguinte transformação é usada , $f(s) = 1 - z \cdot sen\left(\frac{1}{z}\right)$, onde $z = D(s)$.

Antes de se aplicar o algoritmo genético padrão, valores de M (número de elementos da população inicial), p_c (taxa de *crossover*) e p_m (taxa de mutação) devem ser escolhidos. Os valores destes parâmetros são em geral determinados empiricamente por algumas simulações de ensaio. Contudo, uma primeira forma tenta escolher e p_m nos intervalos $0.6 \leq p_c \leq 0.99$ e $0.001 \leq p_m < 0.01$ respectivamente. Para M, resultados empíricos indicam que $n \leq M \leq 2n$ (n é número de *bits* da *string*) é uma boa escolha (Alander ,1992) [ver também Reeves (1993)]. Em adição à escolha dos parâmetros, algum critério de parada ou critério de convergência é requerido. Embora diferentes critérios de convergência possam ser usados ,o critério utilizado aqui é parar quando a população é suficientemente dominada por uma *string*. Neste caso, a convergência é admitida quando uma única *string* representa 80% ou mais da população.

Duas situações serão descritas abaixo. Na primeira situação, uma população inicial foi gerada uniformemente dentro do espaço de busca , e então como, usualmente, os operadores genéticos de seleção, *crossover* e mutação foram aplicados antes do critério de convergência ser encontrado. Os seguinte parâmetros foram usados :M=10 , p_c=0.8 e p_m=0.01, e os resultados das simulações são mostrados na tabela abaixo.

	População $S(t)=\{s_1,...,s_{10}\}$	Valor Decodificado $x = D(s)$	Aptidão $1 - y(x)$
$t = 0$	000010 (1)	0.064	0.994
	000110 (1)	0.092	1.091
	001010 (1)	0.12	0.892
	010001 (1)	0.17	1,063
	011001 (1)	0.226	1.217
	100000 (1)	0.275	1,131
	100111 (1)	0.324	0.981
	101110(1)	0.373	0.833
	110101 (1)	0.423	0.704
	1111100(1)	0.472	0.597
$t = 1$	010101 (1)	0.198	1.185
	110101 (1)	0.423	0.704
	101010(1)	0.345	0.916
	001010 (1)	0.12	0.892
	011001 (3)	0.226	1.217
	010001 (3)	0.17	1.064
$t = 2$	101001 (1)	0.338	0.938
	011001 (4)	0.226	1.217
	111001 (1)	0.451	0.64
	010001 (3)	0.17	1.063
	110101 (1)	0.423	0.704
$t = 3$	010001 (4)	0.17	1.063
	011001 (6)	0.226	1.217

Nesta tabela ,uma lista da população é mostrada por geração no tempo t=0,1,2 e 3 .O valor real da decodificação é também mostrado , bem como os valores de aptidão associadas. O número dentro dos parênteses ao lado de cada *string* mostra o número de vezes que determinada

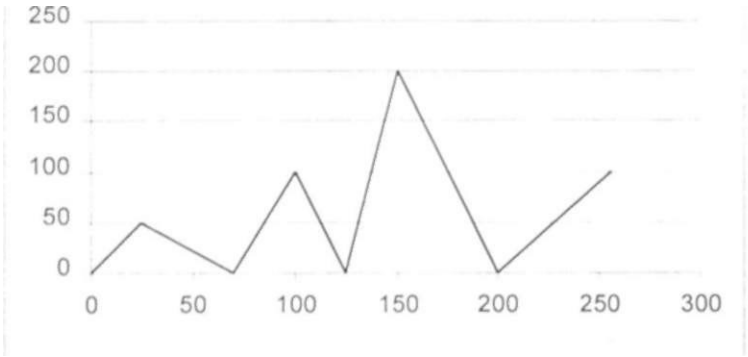
string aparece na população. Note como a população converge para uma população dominada por *strings* de alta aptidão. Depois de quatro iterações ($t = 4$), a população é dominada pela *síring* $s^* = 0\ 1\ 1\ 00\ 1$, e a população tem convergido. A *síring* é decodificada para o valor $x = 0.23$, o qual é muito próxima da solução ótima global $x^* \simeq 0,223$. Note que uma melhor acurácia pode ser obtida pelo uso de um decodificador mais preciso do espaço de busca real. Isto é, pelo uso de um espaço de busca do algoritmo genético com uma *síring* com maior dimensão, por exemplo, $\Omega = \{0,1\}^{10}$, ou $\Omega = \{0,1\}^{20}$, quando se exige mais precisão.

A segunda simulação do problema demostra o caso onde a mutação é necessária para a obtenção da solução ótima global. Nesta simulação, a população inicial foi criada com todas as *strings* próximas do ponto extremo esquerdo $x = 0.05$. As crescentes taxas de mutação e *crossover* são usadas para encorajar a diversidade da população. Com esta ajuda o algoritmo genético ramifica-se para explorar a totalidade do espaço de busca. De fato se a mutação não fosse usada (i.e. $p_m = 0$), então a solução global pode nunca ser achada pelo Algoritmo Genético. Este é o caso da população inicial dominada pelo esquema $0\,00\,0\#\#$, o qual não é um desejável obstáculo de construção ,porque a aptidão do esquema é relativamente baixa. Aplicando seleção e *crossover* não ajudará porque nenhum novo esquema será gerado. Os resultados das simulações são mostrados na tabela 5.2. Neste caso, o Algoritmo Genético levou 44 iterações para convergir para solução $s^* = \mathbf{01}\ 1\,00\,0$, com o correspondente valor real $x = 0.219$.

	População $S(t)=\{s_1 \ldots s_{10}\}$	Valor Decodificado $x = D(s)$	Aptidão 1 -$y(x)$
$t = 0$	000000 (3)	0.05	0.954
	000001 (1)	0.057	1.055
	000010 (3)	0.064	0.994
	000011 (3)	0.071	0.929
$t=5$	000010 (2)	0.064	0.994
	000110(1)	0.922	1.091
	011010(2)	0.233	1.213
	001011 (1)	0.127	0.873
	100010(1)	0.289	1.09
	010010(1)	0.177	1.103
	001000 (1)	0.106	0.999
	001110(1)	0.148	0.935
t =30	111000 (1)	0.444	0.656
	010010 (4)	0.177	1.103
	000010 (1)	0.064	0.994
	011010(2)	0.233	1.213
	010000 (2)	0.163	1.021
$t = 44$	010100(1)	0.191	1.164
	011000(9)	0.219	1.217

É interessante comparar-se esta solução com a obtida usando o Talus, convertendo os pontos das nuvens para a mesma codificação binária com 6 bits usada em Hassoun (1995).

Uma "engenharia reversa" para descobrir os mecanismos genéticos pelos quais o Talus poderia ser interpretado pode ser realizada. Com este objetivo, foi implementada a seguinte função e estudado o comportamento genético das nuvens geradas a cada iteração pelo Talus.



$$f(x) = \begin{cases} 2x & se \quad 0 < x < 25 \\ -x + 75 & se \quad 25 < x < 75 \\ 4x - 300 & se \quad 75 < x \leq 100 \\ -4x + 500 & se \quad 100 < x < 125 \\ 8x - 1000 & se \quad 125 < x < 150 \\ -4x + 800 & se \quad 150 < x \leq 200 \\ 2x - 400 & se \quad 200 < x \leq 255 \end{cases}$$

				64	32	16	8
4			0	0	0	0	0
x	f(x)	f(x)/fmax		representação binária (8 bits)			
ite=0							
0	0	0.000	0		**0**		
255	110	0.267	1		1		
208	16	0.039	1		0		
117	32	0.078	0				
127	16	0.039	0		1		
164	144	0.350	1				
97	88	0.214	0		1		
69	6	0.015	0		O		
130	412	1.000	1		O		
ite= 0							
1	2	0.005	0		O		
254	108	0.259	1		1		
207	14	0.034	1		O		
116	36	0.086	0		1		
128	24	0.058	1		O	O	
164	144	0.345	1		1	O	
96	84	0.201	0		1	O	
70	5	0.012	0		O	O	
130	417	1.000	1		O	O	
ite= 1							
2	4	0.010	0		O	O	
253	106	0.256	1		1	1	
206	12	0.029	1		O	O	
117	32	0.077	0				
129	32	0.077	1		O	O	
164	144	0.348	1		1	O	
95	80	0.193	0		O	1	
71	4	0.010	0		O	O	
130	414	1.000	1		O	O	
ite= 2							
3	6	0.014	0		O	O	
254	108	0.255	1		1	1	
205	10	0.024	1		O	O	
118	28	0.066	0		1		
130	40	0.095	1		O	O	
164	144	0.340	1			O	
96	84	0.199	0			O	
72	3	0.007	0			O	
130	423	1.000	1			0	

	ite= 3										
	4	8	0.019	0	0	0	0	0	1	0	0
	253	106	0.248	1	1	1	1	1	1	0	1
	204	8	0.019	1	1	0	0	1	1	i)	0
	119	24	0.056	0	1	1	1	0	1	1	1
	131	48	0.1 12	1	0	0	0	0	0	1	1
	164	144	0.336	1	0	1	0	0	1	0	0
	97	88	0.206	0	I	1	0	0	0	0	1
	73	2	0.005	0	1	0	0	1	0	0	1
Média	131	428	1.000	1	0	0	0	0	0	1	1
	ite= 4										
	69	6	0.009	0	1	0	0	0	1	0	1
	164	144	0.211	1	0	1	0	0	1	0	0
	192	32	0.047	1	1	0	0	0	0	0	0
	140	120	0.176	I	0	0	0	1	1	0	0
	147	176	0.258	1	0	0	1	0	0	1	1
	164	144	0.211	1	0	1	0	0	1	0	0
	126	8	0.012	0	1	i	1	1	1	1	0
	112	52	0.076	0	1	1	1	0	0	0	0
Média	139	682	1.000	1	0	0	0	1	0	1	1
	ite= 5										
	104	84	0.085	0	1	1	0	1	0	0	0
	158	168	0.170	1	0	0	1	1	1	1	0
	174	104	0.105	1	0	1	0	1	1	1	0
	145	160	0.162	I	0	0	1	0	0	0	1
	147	176	0.178	1	0	0	I	0	0	1	1
	158	168	0.170	1	0	0	1	1	1	1	0
	137	96	0.097	1	0	0	0	1	0	0	1
	129	32	0.032	1	0	0	0	0	0	0	1
Média	144	988	1.000	1	0	0	1	0	0	0	0
	ite= 6										
	125	0	0.000	0	1	1	1	1	1	0	1
	153	188	0.167	1	0	0	1	1	0	0	1
	162	152	0.135	1	0	1	0	0	0	1	**0**
	147	176	0.156	1	0	0	1	0	0	1	1
	147	176	0.156	1	0	0	1	0	0	1	1
	153	188	0.167	1	0	0	1	1	0	0	1
	143	144	0.128	1	0	0	0	1	1	1	1
	138	104	0.092	1	0	0	0	1	0	1	**0**
Média	146	1128	1.000	1	0	0	1	0	0	1	0

	ite= 7										
	140	120	0.085	1	0	0	0	1	1	0	0
	153	188	0.133	1	0	0	1	1	0	0	1
	158	168	0.119	1	0	0	1	1	1	1	0
	151	196	0.138	1	0	0	1	0	1	1	1
	151	196	0.138	1	0	0	1	0	1	1	1
	153	188	0.133	1	0	0	1	1	0	0	1
	149	192	0.136	1	0	0	1	0	1		1
	146	168	0.119	1	0	0	1	0	0	1	0
Média	150	1416	1.000	1	0	0	1	0	1	1	0
	ite= 8										
	146	168	0.111	1	0	0	1	0	0	1	0
	152	192	0.126	1	0	0	1	1	0		0
	154	184	0.121	1	0	0	1	1	0	1	0
	151	196	0.129	1	0	0	1	0	1	1	1
	151	196	0.129	1	0	0	1	0	1	1	1
	152	192	0.126	1	0	0	1	1	0	0	0
	150	200	0.132	1	0	0	1	0	1	1	0
	149	192	0.126	1	0	0	1	0	1	0	1
Média	151	1520	1.000	1	0	0	1	0	1	1	1
	ite= 9										
	148	184	0.117	1	0	0	1	0	1	0	0
	150	200	0.127	1	0	0	1	0	1	1	0
	151	196	0.125	1	0	0	1	0	1	1	1
	150	200	0.127	1	0	0	1	0	1	1	0
	150	200	0.127	1	0	0	1	0	1	1	0
	150	200	0.127	1	0	0	1	0	1	1	0
	150	200	0.127	1	0	0	1	0	1	1	0
	149	192	0.122	1	0	0	1	0	1	0	1
Média	150	1572	1.000	1	0	0	1	0	1	1	0

Tendo em vista o bom desempenho do Talus e as diferenças salientadas é possível propor um operador genético que tente incorporar alguma das boas características do Talus. Este pode ser implementado através de uma variante do operador de mutação fazendo uma mutação diferenciada bit a bit .

O operador de mutação Talus pode ser implementado da seguinte forma :

Faça $p'_m = 10^{-3}$ (probabilidade normal de mutação) se no *locus i* o bit do indivíduo for igual ao do indivíduo de mais alta aptidão, caso contrário faça $p'_m = 1 - \frac{f_{max}}{f_{max}}$.

Esta sugestão tenta reproduzir, em termos binários o que o Talus faz no espaço *n*-dimensional.

5.12 Melhorias feitas no algoritmo Talus e em sua implementação

Pode-se citar como principais avanços :

- O Software Talus escrito em **C++** que da maior percepção das qualidades do algoritmo e permite amplas possibilidades de aplicações com eficácia, eficiência e comodidade .

- Na versão anterior. isto é .em Cavalcanti (1996) $y = \gamma_1 \cdot e^{k}$, c nesta versão $y =$. Esta mudança elimina a função exponencial, muito cara computacionalmente, mantendo o sentido desejado para o γ.

Também, na versão anterior $F^k(x_i^k) e$ .Aqui a expressão foi substituida por $F^k(x_i) = \frac{}{1 + 2^{-\left(\frac{f_{max} - f(x_i)}{f^k_{max} + 10^{-10}}\right)}}$, $i = 1 \ldots tmuv$. Esta mudança tem a mesma justificativa que a anterior, eliminado também o número excessivo de parâmetros.

- Implementação do mesmo no sentido de minimização.

- Implementação do mesmo para o caso de pontos de sela. Esta nova vertente do Talus tinha sido proposta porém não tinha sido elaborada nem implementada.

- Implementação do mesmo para Programação Inteira. O mesmo aconteceu na versão de pontos de sela.

5.13 Conclusões

Mesmo sendo o Talus um algoritmo paralelo por concepção, este tem se mostrado eficiente e flexível na resolução de uma ampla gama de problemas de Programação Matemática, rodando em máquina de von Neumann.

O algoritmo é inerentemente tolerante a falhas por causa da facilidade com que os pontos fora da curva podem ser detectados estatisticamente e eliminados do processo, sendo os erros menores absorvidos pela nuvem de pontos pois o algoritmo trabalha sempre com os valores médios.

Como será mostrado no próximo capítulo a aplicabilidade do Talus na resolução de problemas do mundo real é muito grande. Sabe-se contudo que não é em máquinas de pequeno porte que será mostrada a capacidade do algoritmo para resolver problemas muito complexos e de ordem elevada do mundo real. Contudo alguns problemas complexos apresentados na literatura serão resolvidos no Capítulo 6 .

Capítulo 6

Exemplos de Aplicações do TALUS

6.1 Introdução

As aplicações apresentadas neste capítulo têm o objetivo de ilustrar o grau de universalidade do algoritmo estudado nesta tese. Os exemplos abordados incluem programação linear, programação inteira, equações diofantinas, programação não linear (com restrições) e estratégias minimax. Nos problemas não lineares não convexos o algoritmo apresenta vantagens comparativas, mesmo nas arquiteturas computacionais de hoje (Von Neumman). Uma arquitetura paralela desenvolvida tendo em vista a estrutura do algoritmo poderá vir a propiciar uma vantagem comparativa do algoritmo na grande maioria dos problemas de otimização.

Problemas (química, elétrica e jogos) são exemplos de situações complexas do chamado mundo real (muitas não linearidades, funções transcendentais, pontos de sela e ordem elevada), onde o algoritmo mostra toda a sua potencialidade.

6.2 Programação Linear

Tendo em vista a eficácia e a eficiência dos algoritmos dedicados a solucionar problemas de otimização linear (principalmente o método Simplex, vide Chvátal (1983)) este item tem como objetivo simplesmente mostrar a versatilidade do Talus.

Como o Talus não resolve diretamente um problema de otimização com restrições, têm-se que incorporar as mesmas ao funcional objetivo. Será utilizada aqui o método das penalidades para resolver o Problema 1.

6.2.1 Problema 1 :

$$\text{Maximize } 2x_1 + 5x_2$$
$$\text{sujeito a } x_1 + 4x_2 < 24$$
$$3x_1 + x_2 \leq 21$$
$$x_1 + x_2 < 9$$
$$x_1, x_2 > 0$$

Problema 2.2 [Chvátal (1983)] pag.26.

$$x_1^* = 4, x_2^* = 5, f^* = 33$$

Assim têm-se o seguinte funcional objetivo :

$$fix) = 2x_1 + 5x_2 - \frac{1}{\epsilon} e^{5\cdot(x_1+4x_2-24)} - \frac{1}{5} e^{4\cdot(3x_1+x_2-21)} - \frac{1}{\epsilon} e^{4\cdot(x_1+x_2-9)}$$

Solução Talus (Função 19):

```
 Função número=19
nite=100      tnuv=30     gama1=1.000000   gama2=10.000000 precisão=0.000000
xl = 4.000166
 x2=4.999971
   f=32.600000
```

6.3 Programação Inteira

Considere o modelo :

$$\text{otimize } \sum^{n} c_j \cdot x_j$$

$$\text{sujeito a } \sum \cdot x_j < h_i \text{ para } i = 1, 2 \ldots m$$

$$x_j > 0 \quad \textbf{para } j = 1, 2 \ldots n$$

x_j de valor inteiro para j =1,2, ...,p n)

Este tipo de otimização é conhecido como um problema de programação inteira (ou diofantina ou, ainda, discreta). Quando $p=n$ o modelo é chamado programação inteira pura ,caso contrário é chamado de programação inteira mista.

6.3.1 Importância dos problemas dc programação inteira

Wagner (1986) indica os seguintes exemplos da importância da programação inteira:

- Utilização de equipamentos - Pode-se, por exemplo, definir x_j como unidades de equipamento que devem operar durante o horizonte de planejamento do modelo.
- Custo de preparação - Pode-se, por exemplo, querer considerar uma atividade que esteja sujeita a um custo fixo C_i sempre que o nível correspondente a $x_i>0$, onde C, independe do nível real de x_i .
- Tamanho dos Lotes - Em algumas situações do planejamento da produção, pode-se querer restringir o nível de x_j a ou $x_i \geq L_j$.
- Decisões "sim-ou-não" - Por exemplo, para especificar situações tipo "ou-ou".

6.3.2 Problema 2

Ultimo Teorema de Fermat :Achar x,y e z tal que $x^3+y^3=z^3$

Formulação : $\underset{x_1,x_2,x_3}{Min}\ (x_1^3+x_2^3-x_3^3)^2$

Espaço de busca : $x_1,x_2x_3\ e\ [0,1000]$

Solução Talus:

```
Função número=20
nite=100    tnuv=900    gama1=1.000000  gama2=12.000000 precisão=0.000000
x1=150.000000
 x2=144.000000
```

```
x3=73.000000
f=1.028145
```

```
Função número=20
nite=100     tnuv=900     gama1=1.000000  gama2=12.000000 precisão=0.000000
x1=103.000000
 x2=64.000000
  x3=94.000000
  f=1.00000
```

```
Função número=20
nite=100     tnuv=900     gama1=1.000000  gama2=12.000000 precisão=0.000000
x1=144.000000
 x2=71.000000
  x3=138.000000
  f=1.094285
```

Obs. Estes resultados foram encontrados depois de várias tentativas.

Também tem-se : $6^3 - 8^3 = 9^3 - 1$. $720^3 + 242^3 = 729^3 - 1$. $729^3 + 244^3 = 738^3 + 1$

6.3.3 Problema 3

Achar x, y e z tal que $x^7 + y^7 = z^7$

Formulação : $\underset{x_1,x_2,x_3}{Min}$ **fctj** $+ x_2^7 - x_3^7)^2$

Espaço de busca : $x_1, x_2, x_3 \in [0, 1000]$

Solução Talus:

```
 Função número=21
nite=100     tnuv=999     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=1.000000
 x2=2.000000
  x3=0.000000
  f=15121.601541
```

```
Função número=21
nite=100     tnuv=999     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=53.000000
```

```
 x2=53.000000
  x3=1.000000

  f=1.0000000

Função número=21
nite=100      tnuv=500     gama1=1.000000  gama2=10.000000 precisão=0.000000
x1=41.000000
 x2=41.000000
  x3=1.000000
  f=1.0000000
```

Obs. A complexidade do problema força o Talus achar somente soluções triviais. Para resolver este problema seria necessário aumentar sensivelmente o número de iterações, juntamente com γ_2

6.3.4 Problema 4

Mordell (1969) (Teorema 7, p.268)

Achar x_1, x_2 tal que $\left(\frac{x_1(x_1-1)}{2}\right)^2 =$

Formulação : $Mm \left[\left(\frac{x_1(x_1-1)}{2}\right)^2 - \frac{x_2(x_2+1)}{2}\right]^2$

Espaço de busca : $x_1, x_2 \in [-100, 100]$

Soluções : $(x_1, x_2) = (1,1), (2,2), (3,3), (4,9), (-1,-1), (-3,-8), x = 0.1, y = 0,1$; isto significa que os únicos números triangulares que são quadrados de números triangulares são $0, 1, 36$.

Solução Talus :

```
 Função número=22
nite=15      tnuv=999     gama1=1.000000  gama2=1.500000 precisão=0.469583
x1=0.000000
 x2=1.000000
   f=0.002288

Função número=22
nite=15      tnuv=999     gama1=1.000000  gama2=1.500000 precisão=0.000000
x1=1.000000
 x2=0.000000
   f=0.008396

Função número=22
nite=15      tnuv=999     gama1=1.000000  gama2=1.500000 precisão=0.000000
x1=1.000000
```

```
x2=1.000000
  f=0.001535
```

6.3.5 Problema 5

Mordell (1969) p.287

Conjectura (Erdös, Straus)- Existem x, >0 , x_2>0, x_3>0 inteiros que satisfazem a equação $\frac{4}{n} = \frac{1}{x_1} + \frac{1}{x_2} + \frac{1}{x_3}, n > 3$. Já foi demostrado para $n < 10^7$.

Formulação : *Min* $\frac{4}{n} - \frac{1}{x_1} - \frac{1}{x_2} - \frac{1}{x_3}$, $n = 10^8$

Espaço de busca : $x_1, x_2, x_3 \in [10^7, 10^{10}]$

Solução Talus:

```
Função número=23
nite=1000     tnuv=500     gama1=1.000000  gama2=100.000000 precisão=0.005893
x1=221736974.740126
 x2=9164119438.469707
 x3=3465418868.395746
  f=0.000000
x1=2813643607.000000
 x2=212297449.000000
  x3=6781830237.000000
  f=0.000000
x1=350772973.000000
 x2=385104189.000000
  x3=962654047.000000
  f=0.000000
x1=1471629756.000000
 x2=1997478012.000000
  x3=114663214.000000
  f=0.000000
```

6.4 **Problema** 6 : Processo **Químico**

Himmelblau (1972) p.412

Fonte: M. A. Efroymson, Esso Reseach and Engineering Co. (cited in Colville, IBM N.Y. Sci. Center Rept. 320-2949, June ,1968)

$$\text{Minimize } \sum_{i=1}^{4} c(x_i) + \sum_{i=5}^{6} 100c(x_i)$$

sujeito a $t_3 - 300 > 0$

$t_4 - 300 > 0$

$280 - T_5 > 0$

$250 - T_6 > 0$

Onde :

$c(x_i) = 2.7 \cdot x_i + \mathbf{1300}$

$T_1 \sim \frac{0.028x_1 + 300}{1 + 0.0001425x_1}$ $\qquad T_5 = \frac{t_2 + (70 - t_2)e^{-a_2}}{1 - 0.8e^{-a_2}}$

$t_1 = 500 - T_1$ $\qquad t_4 = 350 + (t_2 - T_4)e^{a_4}$

$a_2 = -0.0001665 \cdot x_2$ $\qquad T_{j2} = 0.873 + 0.274$

$T_{t_2} = \frac{200 - 350 \cdot e^{-a_2}}{1 - 1.5e^{-a_2}}$ $\qquad a_5 = 0.000375 \cdot x_5$

$t_2 = 300 + (200 - T_2)e^{a_2}$ $\qquad Ts = 80 + (T_{j2} - 80)e^{-a_5}$

$a_3 = (0.085)(9.36)10^{-5}x_3$ $\qquad T_{j1} = 0.7T_1 + 0.3T_2$

$T_3 = \frac{t_1 + (29.75 - t_1)e^{-a_3}}{1 - 0.915e^{-a_3}}$ $\qquad T_6 = 80 + (T_{j1} - 80)e^{-a_6}$

$t_3 = 350 + (t_1 - T_3)e^{a_3}$ $\qquad a_4 = 0.00025x_4$

Condição Inicial:

$x^0 = [8000 \quad 3000 \quad 14000 \quad 2000 \quad 300 \quad 10]^T$

$f(x^0) = 459100$

Resultado encontrado pelo método da tolerância flexível:

$x^* = [11884 \quad 3288 \quad 20000 \quad 4000 \quad 114.18 \quad -155.03]^T$

$f(x^*) = 250799.9$

Valores de $f(x)$ para pontos de ótimo citados em Coville:

Condições Iniciais viáveis	Método	Condição inicial não viável'
255,303.5	Gradiente generalizado	266.754
389,858	Pop-360	
132,518	Optim (Mobile Oil)	125,578

Condição inicial não viável

$x^0 = [8000 \quad 3000 \quad 10000 \quad 2000 \quad 200 \quad 10]$

Resultado Talus :

```
Função número=24
nite=1000      tmuv=500     gama1=1.000000   gama2=100.000000 precisão=1000000
x1=11230.503027
 x2=2606.498190
  x3=16103.786556
 x4=3715.045646
 x5=74.556727
 x6=-171.839931
f=330440.989749
```

6.5 **Problema 7 : Otimização** de uma **rede** elétrica

[Himmelblau (1976)] p.413

Fonte : P.Huard, *Electricité de France* , Directions des Études et Recherches (cited in Colville, IBM N.Y. Sci. Center Rept. 320-2949, June ,1968)

$$\text{Minimize } f_1(x_1) + f_2(x_1)$$

sujeito a

$$f_1(x_1) = 30x_1 \quad para \quad 0 < x_1 < 300$$
$$f_1(x_1) = 31x_1 \quad para \quad 300 < x_1 < 400$$
$$f_2(x_2) = 28x_2 \quad para \quad 0 \le x_2 < 100$$
$$f_2(x) = 29x_2 \quad para \quad 100 < x_2 < 200$$
$$f_2(x_2) = 30x_2 \quad para \quad 200 \le x_2 < 1000$$
$$x_1 = 300 - \frac{x_3 x_4}{131078}\cos(148577 - x_6) + \frac{0.90798x_3^2}{131078}\cos(147588)$$
$$x_2 = -\frac{x\ x}{131078}\cos(148577 + x_6) + \frac{0.90798x_4}{131078}\cos(147588)$$
$$x_5 = 300 - \frac{x_3 x_4}{131078}\sin(148577 + x_6) + \frac{0.90798x_4^2}{131078}\sin(147588)$$
$$0 = 200 - \sin(148577 - x_6) + \frac{0.90798x_3^2}{131078}\sin(147588)$$
$$0 \le x_1 \le 400$$
$$0 \le x_2 < 1000$$
$$340 \le x_3 \le 420$$
$$340 < x_4 < 420$$
$$-1000 \le x_5 < 1000$$
$$0 \le x_6 < 0.5236$$

	Alta Precisão	Precisão Moderada
	107.81	201.78
	196.32	100
x^*_3	373.83	383.07
x^*_4	420	420
x^*_5	21.31	-10.907
x^*_6	0.153	0.07314
fix)	89,275.888	8,853.44

Solução Talus :

```
Função número=25
nite=1000    tnuv=100    gama1=1.000000  gama2=100.000000 precisão=0.000000
x1=98.861859
 x2=196.462863
  x3=377.894483
 x4=407.600503
 x5=12.965745
 x6=0.316486
```

6.6 **Problema 8 : Planejamento de Levantamento Oceanográfico**

Fonte : Tese de Doutorado de Luiz Viana Nonato intitulada Aplicação de Algoritmos Genéticos no Planejamento de Levantamento oceanográfico. (Escola Politécnica da USP).

Otimizar o posicionamento do braço mecânico bi-articulado, controlado pelos ângulos α e P.

$$\textit{Max fia ,ß)} = \frac{4 - \sqrt{(x_p - (\cos\alpha + \cos(\alpha+\beta)))^2 + (y_p - (\sin\alpha + \sin(\alpha+\beta)))^2}}{4} \quad \text{com}$$

$x_p = 1, y_p = -1$.

Solução Algoritmos Genéticos: $(\alpha , P) = (270^o, 90^o)$ e $f(\alpha^*, \beta^*) = 0.982$

Solução Talus

```
Função número=26
nite=100    tnuv=100    gama1=1.000000  gama2=10.000000 precisão=0.000001
x1=4.709109
 x2=1.588455
   f=0.996306
```

Para Θ - A - [0. 1] onde Θ é o conjunto de estados da natureza e A o espaço das ações. Para L = (tf - a)^{2}função perda. Tem-se

$$\text{tf} = 1 \rightarrow L = 1 - 2a + a^2 \text{ para } a = 1 \rightarrow L = 0 \text{ para } a = 0 \rightarrow L = 1$$

$$\text{tf} = 0 \rightarrow L = cr \text{ para } a = 1 \rightarrow L = 1 \text{ para } a = 0 \rightarrow L = 0$$

logo a solução tem que ser randomizada. Considere-se

$$\Gamma(a)\Gamma(\beta)$$

Sabe-se que $\int \text{TF}^{\cdots 1}(1 - \text{tf})^{\cdots} d\theta = \frac{\Gamma(a+\beta)}{\Gamma(a)\Gamma(\beta)} = B(a,\beta)$ e que $\Gamma(x+1) = x \cdot \Gamma(x)$. Tem-se

$E_a[(\theta - a)^2] = \text{tf}^2 - 2 \cdot \text{tf} \cdot \quad + \text{tf } a^2)$. Também $E\{a) = \quad$ e $E(a^2) = \frac{a \cdot (a+1)}{(a+\beta)(a+\beta+1)}$

Portanto : $\quad - a)^2 \ ] = 0^2 - 20 \cdot \frac{a}{a+\beta}$ fa^fa-

Assim o problema é então :

$$\underset{a,\beta}{Min}\ \underset{\theta}{Max}\ \text{tf}^2 - 20 \cdot \frac{a}{a+\beta} + \frac{a \cdot (a+1)}{(a+\beta)(a+\beta+1)} \text{ para tf} \in [0,1], a, \beta \text{ e } [0.01,20] \text{ ou}$$

$$Min\ Max\ \text{ri-2xj} \cdot \frac{x_1}{x_1 + x_2} + \frac{x_1 \cdot (x_1+1)}{(x_1+x_2)(x_1+x_2+1)} \text{ para } x_3 \in [0,1], x_1, x_2 \in [0.01,20]$$

Solução Talus :

```
 Função número=27
nite=1595      tnuv=100     gama1=1.000000  gama2=100.000000 precisão=0.000006
x1=13.474529
 x2=0.332525
 x3=0.029776
  f=0.896768
```

```
Função número 27
nite=1595     tnuv=100     gama1=1.000000  gama2=100.000000 precisão=0.000006
x1=13.474529
 x2=0.332525
 x3=0.029776
  f=0.896768

Função número=27
nite=1000     tnuv=999     gama1=1.000000  gama2=100.000000 precisão=0.044937
x1=1.232234
 x2=19.116610
 x3=0.446758
  f=0.151817


 Função número=27
nite=1000     tnuv=100     gama1=1.000000  gama2=100.000000 precisão=0.000078
x1=7.767819
 x2=7.789259
 x3=0.885879
  f=0.164534
```

Para todos estes problemas o tempo de execução não ultrapassou cinco minutos, com o algoritmo "rodando" num microcomputador tipo PC com 64Mb RAM, 166MHz e 2 Gb de disco rígido. Os métodos determinísticos e mesmo os estocásticos como o ***simulated** annealing* e os algoritmos genéticos, têm se mostrado mais lentos e muitas vezes ineficazes nestes tipos de problema.

7 Conclusões, Comentários e Sugestões

Capítulo 7

Conclusões, Comentários e Sugestões

7.1 Introdução

A demanda por algoritmos eficazes e eficientes para a resolução de problemas de programação matemática é muito grande. Isto deve-se principalmente ao fato de que os problemas de otimização aparecem em muitos contextos de interesse. Minimização de perdas em linhas de sistemas de potência, minimização de consumo de combustível em motores a explosão estimação, deteção e decodificação de sinais em problemas de telecomunicação, processamento de imagem, alocação de recursos em sistema produtivos, roteamento em sistemas de transporte, controle de processos, redes neurais, estatística, são apenas alguns dos exemplos que evidenciam a importância do problema.

Desde a época Fermat que pensou no problema de máximos e mínimos de uma função de $\mathbb{R}$ em em termos de tangentes paralelas ao eixo das abcissas, passando pelo cálculo diferencial e integral inventado por Newton e Leibniz *(Nova Methodus proMaximis et Minimus)*, até o método Simplex, na segunda guerra mundial, e os métodos de programação não linear (condição de Kuhn-Tucker, Lagrangeanos, etc), e os métodos modernos da otimização global (Pardalos (1995)), a busca tem sido intensa. Além destes problemas citados acima cumpre citar problemas de programação matemática em espaços de dimensão infinita, ou seja, os problemas de controle ótimo, que são desdobramentos do cálculo das variações, área do conhecimento que

teve início com o braquistócrono de Bernoulli. Em termos de algoritmo, esses problemas podem ser escritos como problemas de programação não linear.

Por outro lado, é importante lembrar que muitos problemas de matemática podem ser expressos em termos de um problema de otimização. Um exemplo clássico é o das equações algébricas. Outro problema que pode ser formulado assim é o de encontrar soluções para equações diferenciais.

7.2 Algoritmos de otimização determinísticos e probabilísticos

Entre os algoritmos determinísticos o de maior sucesso e destaque é sem dúvida o método Simplex para a solução de problemas de programação linear. A sua limitação é obvia : o método só se aplica para modelos lineares (funcional objetivo e restrições lineares). No caso da programação não linear, os métodos existentes não tem, nem de longe, a mesma eficácia e eficiência do método Simplex. Ademais, em geral várias hipóteses restritivas são necessárias para que a maioria dos algoritmos funcionem a contento, como por exemplo diferenciabilidade e convexidade. Freqüentemente isto faz com que o algoritmo fique "preso" em algum ótimo local do problema real. Também, estes algoritmos são complexos e de difícil implementação, exigindo grande esforço computacional.

Os algoritmos probabilísticos, como foi chamado à atenção nesta tese, não têm essas limitações, e vêm tendo uma presença cada vez maior nas aplicações. Destacam-se *simulated annealing* e os algoritmos genéticos. Como citado no Capítulo 4, o *simulatedannealing*, que tem uma base matemática (e física) mais estabelecida, em geral é lento. Os algoritmos genéticos apesar de terem sido lançados em 1975 ainda não têm uma base matemática que permita maiores

desenvolvimentos, e também não são rápidos pelo menos nas arquiteturas convencionais. Estes dois tipos de algoritmos têm sido alvo de muitos estudos recentes.

7.3 O Talus : Desenvolvimentos e Resultados

Sendo o primeiro randomizador sobre o qual se tem registro na história da humanidade, Talus foi o nome adotado para este projeto Campello de Souza (1987). O algoritmo teve sua versão preliminar apresentada em Ribeiro (1996). Os resultados foram suficientemente promissores para que o assunto fosse mais explorado e aprofundado. Os resultado obtidos neste trabalho confirmaram as expectativas. Problemas apresentados na literatura com objetivo de testar algoritmos foram todos resolvidos pelo Talus, incluindo problemas do tipo *needle in the haystack* (agulha no palheiro). Ademais, vários problemas com diversos graus de complexidade, buscados na literatura também foram resolvidos. O problema de encontrar pontos de sela também foi abordado com sucesso, o que permite o tratamento numérico de problemas de minimax e jogos. Tal exploração foi possível graças ao desenvolvimento e elaboração de um *software* para o algoritmo, implantado em linguagem C+ + de fácil utilização. No total foram resolvidos 32 problemas. Não houve preocupação em resolver problemas de ordem muito elevada, embora o programa não tenha esta limitação. Rapidamente, funções de argumentos de ordem mais elevada podem ser incorporadas ao programa. Preferiu-se, nesta tese. explorar uma classe maior de problemas, e neste processo, introduzir melhorias e avanços no algoritmo.

A comparação do Talus com os algoritmos genéticos possibilitou uma outra visão do algoritmo, evidenciando uma boa coerência entre os dois enfoques. Essa comparação suscitou uma atitude de "engenharia reversa" que inspirou a sugestão de um novo operador genético, o

operador de mutação Talus, que será útil inclusive para a concepção e implementação de uma arquitetura paralela.

7.4 Sugestões para Futuros Estudos

Como sugestões para pesquisas no assunto pode-se citar as seguintes :

- Otimizar a relação entre a precisão desejada, o tamanho da nuvem e o valor de γ_2.
- Explorar o uso de outros construtos, tais como a entropia e momentos das variáveis aleatórias.
- Especificar a relação entre a probabilidade de perder um ponto de ótimo e o tamanho do espaço de busca, o tamanho da nuvem e o valor de γ.
- Analisar a viabilidade de se introduzir uma mutação genética no Talus.
- Fazer um estudo do Talus sobre o ponto de vista de algoritmos genéticos ("engenharia reversa").
- Estudar o sugerido operador mutação Talus dentro do contexto dos algoritmos genéticos.
- Implementar o Talus em arquiteturas paralelas existentes.
- Conceber e desenvolver uma arquitetura paralela baseada no funcionamento do Talus.

Referências Bibliográficas

AISAWA,A.N and WAH,B.W.(1993) Dynamic control of genetic algorithms in noisy environments. In : International Conference on Genetic Algorithms, 5., Urbana-Champaign,. Proceeding. San Mateo , Morgan.

AVRIEL, Mordecai (1976). Nonlinear programing : analysis and methods, Prentice-Hall, New Jersey .

AVRIEL,M. (1976) . Nonlinear Programming : Analysis and Methods, Prentice-Hall

BAKER,J.E. (1987) Reducing bias and inefficiency in the selection algorithms. In: International Conference on Genetic Algorithms ,2., Cambridge .Proceedings.Hillsdale, Lawrence Erlbaum,p14-21

BARTH, N. (1992), Oceanographic experiment design II: genetic algorithms. Jornalof Physical Oceanography,v9.,p.434-43

BAYES,T (1783) An essay towards solving a problem in the doutrine of chances. *Phil. Trans. Roy.Soc.* Vol 53,p.370-418

BIANCHINI,R.;BROWN,C.M.;CIERNIAK;MEIRA,W.(1995)Combining distribuited populations and periodic centralized selections in coarse-grain parallel genetic algorithms.In :PEARSON,D.W.; STEELE,N.C.;ALBRECHT,R.F. , artificial neural nets and genetic algorithms,Wien,Spring,p483-6

BINDER, K.,(1992). Large-scale simulations in condensed matter physics - the need for a teraflop computer, *Int. J. Mod. Phys.* **C 3,** 565

BOYER, C.B. (1974). História da matemática ;tradução: GOMIDE, E.F. São Paulo, Edgard Blücher.

CAMPELLO DE SOUSA , F. (1987) ;Projeto Talus : Um computador estocástico paralelo; Departamento de Eletrônica e Sistimas ,UFPE.

CAMPELLO, R.Eduardo e MACULAN,Nelson ,(1994). Algoritmos e Heurísticas :desenvolvimento e avaliação de performance, Niterói, RJ: EDUFF.

CAVALCANTE,José Ranière Ribeiro (1996) . Algoritmo probabilístico para otimização global, Dissertação (mestrado) - Universidade Federal de Pernambuco, CTG, Engenharia Elétrica .

CHAVÁTAL. V. (1983) Linear programming. New York, W.H. Freeman and Company.

CHERRI PANCAKE *et al.*, Specification of Baseline Development Environment, Section 3, Component BDE-3i, in *Guidelinesfar Writing System Software and Tools Requirements for Parallel and Clustered Computers.*
Available at
http://www.nero.net/pancake/SSTguidelines/baseline.html

CODDINGTON, P.D.,(1994). Analysis of random number generators using Monte Carlo simulation, *Int. J. Mod. Phys.* C **5,** 547

DAVIS, L.(1987). A genetic algorithms and simulated annealing :an overview. In: DAVIS, L. Genetic Algorithms and simulated annealing. London, Pitman.p. 170-204.

DE JONG,K. and SPEARS.W. (1993) On the state of evolutionary computation. In: International Conference on Genetic Algorithms,5., Urbana-Champain. Proceedings. San Mateo. Morgan Kaufmann. p 618-23

DEB, K. GOLDBERG, D.E. (1989) An Investigation of niche and species formation in genetic function optimization . In International Conference on Genetic Algorithms,3.,San Mateo. Proceedings. San Mateo. Morgan Kaufmann, p 42-50.

DUDEWICZ, E.T. and RALLEY,T.G.,(1981) *The Handbook of Random Number Generation and Testing with TESTRANDComputer Code,* American Science Press, Columbus, Ohio.

FERRNBERG,A.M., LANDAU,D.P.and WONGY.J., (1992). Monte Carlo simulations: Hidden errors from ``good" random number generators, *Phys. Rev. Lett.* 69, 3382

FINE,T.L, (1999). Feedforward Neural Network Methodology. Monograph Series in Statistics. New York: Springer-Verlag (forthcoming)

FITZPATRICK.J.M.;GREFENSTETTE,J.J.(1988). Ganetic algorithms in nisy environments.Machine Learning. v3.n. 2/3 , p. 101 -20,1988

GALESIA (1995) . Fist IEE/IEEE International Conference on Genetic Algoritms in Engineering Systems : Innovations and Applicatioms - Sheffield - 12 a 14 de Setembro de 1995

GIBSON. G.M. (1995) An argument against the principle of minimal alphabet. In : PEARSON, D.W. ; STEELE, N.C.;ALBRECHT,R.F. Artificial neural nets and genetic algorithms, Wien, Springer-Verlag

GNEDENKO, B.(1968), Theory of Probability, Chelsea.

GOFFE,W.L.; FERRIER.G.D. and ROGERS, J., (1994) ."Global Optimization of Statistical Functions with Simulated Annealing".Journal of Econometrics 60 (1/2) : 65-69.

GOLDBERG, D. E. (1989) . Sizing populations for serial and paralel genetic algorithms. In : International Conference on Genetic Algorithms,3., San Mateo. Proceedings. San Mateo, Morgan Kaufmann, p. 70-9.

GOLDBERG, D. E.(1989) . Zen and the art of genetic algorithms. In: International Conference on Genetic algorithms ,3., San Mateo. Proceedings. San Mateo, Morgan Kaufmann, p. 80-5.

GOLDBERG,D.E and SEGREST, P. (1987) . Finite markov chain analysis of genetic algorithms. In: International Conference on Genetic Algorithms. 2. , Cambridge. Proceedings. Hillsdale, Lawrence Erlbaum .

GOLDBERG,D.E. and RICHARDSON, J (1987). Genetic algorithms with sharing for multimodal function optimizaction . In: International Conference on Genetic Algorithms, 2., Cambridge. Proceeding.Hillsdale,LawTence Erlbaum , p.41-9

GOLDBERG,D.E.(1989). Genetic algorithms in search, optimization and machine learning, Reading, Addison-Wesley.

GOPAL, V. and BIEGLER, L.T. (1998), Large Scale Inequality Constrained Optimizaction and Control, IEEE Control Systems, Volume 18, Number 6, December. pp59-68.

GORDON,V.S. and WHITLEY,D. (1993) .Serial and paralel genetic algorithms as function optimizers . In : International Conference on Genetic Algorithms.5.,Urbana-Champaign.Proceedings. San Mateo,Morgan Kaufmann ,1993.p177-90

GRASSBERGER P.,(1993). On correlations in `good" random number generators, *Phys. Lett. A* 181,43

GRAYBEAL. W. and PODCHLU. (1980) .Simulation: principles and methods .Cambrige, Winthrop.

GREFENTETTE,J.J. (1986) Otimization of control parameters for genetic algorithms. IEEE Transactions on Systems,Man and Cybernetics,v16,n. 1.. p.122-28, Jan (1986)

HACKING. IAN (1975), The emergence of Probability, Cambridge University Press.

HALD. A. (1990). A History of Probability and Statistics and Their Aplications, before 1750, Wiley Interscience.

HASSOUN, M.H.(1995) . Fundamentals of Artificial Neural Networks, A Bradford Book, The MIT Press.

HIMMELBLAU, David M. (1972), Applied nonlinear programming , McGRAW-Hill Book company.

HIRIART-URRUTY,J.B.(1995). Conditions for global otimality. Kluwer Academic Publishers

JAMES, B.R. (1996) Probabilidade: Um curso em nível intermediário (segunda edição), Rio de Janeiro. Instituto de Matemática Pura e Aplicada.

KIRKPATRICK,S.;GELATT JR., C.D.; VECCHI, M.P. (1983) Opptimazation by simulated annealing.Science, V.220. n.4598,p.671-9, May 1983.

KNUTH D.E, (1981)*The Art of Computer Programming Vol. 2: Seminumerical Methods* (second edition). Addison-Wesley. Reading. Mass.,.

LEE, M.A. ; TAKAGI .H. (1995) . A framework for studying the effects of dynamic crossover,mutation and populaction sizing in genetic algorithms,5. ,In: Lecture notes in artificial intelligence,1011.Berlin. p.111-26.

LIMA.E.L (1976). Curso de análise. Rio de Janeiro, Instituto de Matemática Pura e Aplicada, CNPq.

LIMA,E.L (1981). Curso de análise. Volume 2, Rio de Janeiro, Instituto de Matemática Pura e Aplicada, CNPq.

M.Mascagni and D.H.Bailey, Requirements for a parallel pseudorandom number generator, Supercomputing Research Center technical report, unpublished.

MARSAGLIA,G.A.,(1985) A current view of random number generators, in *Computational Science and Statistics: The Interface,* ed. L.Balliard, Elsevier, Amsterdam,.

MAS-COLELL, A..Whinston. M. D. and Green. J. R. (1995) . Microeconomic Theory, Oxford : Oxford University Press.

MATEUS, Geraldo Robson e LUNA, H.P.L. (1986). Programação não linear , V Escola de Computação . Belo Horizonte : UFMG.

MORDELL. L.J. (1969), Diophantine Equations, Academic Press. 314.

NAYLOR,T.H.; BALINTFY,J.L. ; BURDIK,D.S.;CHU,K.(1971) .Técnicas de simulação em computadores. Editora Vozes Ltda em colaboração com a Editora da Universidade de São Paulo.

NONATO.L.V.(1997) . Tese de Doutorado intitulada "Aplicação de Algoritmos Genéticos ao Planejamento de Levantamentos Oceanográficos", Escola Politécnica da USP.

O'BEIRNE. T.H., (1961), Puzzles and Paradoxes, The New Scientist, n° 238, 8 June, p. 598.

PARDALOS.P.M. (1995), Handbook of Global Optimization, Kluwer Academic Publishers, Netherlands.

RIBEIRO, Clovis Augusto (1973). Implementação de um sistema de algoritmos de programação não linear, Tese de Mestrado, COPPE .

ROBERTS,S.G.and TUREGA,M. Evolving neural network structures:an evaluation of encoding techniques. In :PEARSON,D.W. and STEELE,N.C. ;ALBRECHET,R.F. Artificial neural nets and genetic algorithms

RUELLE, D. Chaotic evolution and strange attractors, Cambridge,Cambridge University Press

S.K.Park and K.W.Miller, Random number generators: Good ones are hard to find, *Comm.* ACM 31:10, 1192 (1988).

SALAMON, P.;NULTON, J.D.;ROBISON,J.; PETERSEN, J.; RUPPEINER, J.;RUPPEINER, G. and LIAO, L. (1988) . Simulated annealig with constant thermodynamic speed, Computer Physics Communications , 49, p. 423-428

SCHAFFER,J.D.; CARUANA,R.A.; ESHELMAN,L.J.; DAS,R. (1995). A study of control parametres affecting online performace of genetic algorithms for funtion otimization. In: International Conference on Genetic Algorithms,3.,San Mateo. Proceedings. San Mateo, Morgan Kaufmann, p. 51-60

SIPPER,M. ; SANCHEZ,E. ; MANGE, D.;TOMASSINI,M.;PÉREZ-URIBE, A;STAUFFER,A. (1997),A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. IEEE Transactions on Evotionary Computation, Vol.1 ,n.l, p. 3-17,Apr..

STUCKMAN,B.E; EASON,E. E. (1992), A Comparison of Bayesian / Sampling Global Optimization Techniques, IEEE Transactions on Sytems , Man, and Cybernetics , Vol 22, n.5, September / october .

SZU, H. (1986) .Fast simulated annealing, In: Neural Networks for Computing (Snowbird, 1986), J.S. Denker,ed., p.420-425. American Institute of Physics, New York.

TANG,K.S.; MAN,K.F. and KWONG,S. (1996), Genetic Algorithms and their applications, IEEE signal processing magazine, November, pp. 23-28

USPENSKY, J. V. (1937), Introduction to Mathematical Probability, McGraw-Hill

VATTULAINEN,L, ALA-NISSILA,T. and KANKAALA,K.,(1994)., Physical tests for random numbers in simulations, *Phys. Rev. Lett.* 73, 2513.

VATTULAINEN,I., ALA-NISSILA,T. and KANKAALA,K.,(1995). Physical models as tests of randomness, *Phys. Rev.* E 52, 3205

VIDYASAGAR,M.,(1998), Statistical Learning Theory and Ramdomized Algorithms for Control , IEEE Control Systems, Volume 18, Number 6, December, pp69-84.

Capítulo 1

Introdução

1.1 Antecedentes

Desde tempos imemoriais os homens têm usado os mais elaborados rituais para ajudá-los na tomada de decisões. Sacrificaram animais, "leram" as estrelas, ofereceram bebidas aos santos e observaram o vôo das aves, entre outros procedimentos, na busca de uma sistemática que fosse eficaz e eficiente como apoio às suas decisões. Freqüentemente os indivíduos e instituições colocam a sua fé em provérbios e regras práticas concebidas para aliviar um pouco do trabalho de adivinhação das tarefas do dia a dia. A conjuntura econômica do Brasil de hoje é um bom exemplo da necessidade de métodos racionais e científicos que possam servir de base para a tomada de decisões. Hoje em dia, a gerência do processo decisório usa, de fato, pelo menos nos meios mais sofisticados, um ritual mais científico : os modelos matemáticos implementados no computador.

Não é difícil perceber que a mente humana, desassistida, seria incapaz de levar em consideração todos os inúmeros fatores e suas complexas interrelações, presentes na operação de empresas privadas e públicas, no projeto de uma aeronave, no controle de tráfego urbano ou telefônico, e na condução de um governo.

WAGNER, H.M.(1986). Pesquisa Operacional.Prentice/Hall do Brasil. 2ª edição.

WHITE,M.S. and FLOCKTON.S.J.(1995) Modeling the behavior of genetic algorithms .In: IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems: Innovations and applications. 1., Sheffield,1995. Proceeding. London,IEE,p 349-56.

XIAO, J.: MICHALEWICZ.Z.: ZHANG,L.; TROJANOWSKI,K. (1997) Adaptive evolutionary planner / navigator for mobile robots. IEEE Transactions on Evotionary Computation. Vol.1 ,n.1, p. 18-28,Apr..

O desenvolvimento da engenharia eletrônica propiciou a elaboração de computadores cada vez mais potentes e isso suscitou o aparecimento e evolução de um grande número de técnicas de programação matemática. um termo cunhado por Robert Dorfman em 1950 .

Hoje em dia. a expressão Programação Matemática é uma referência genérica que abrange programação linear. programação inteira. programação não linear, programação convexa. programação dinâmica. fluxos ótimos em redes, controle ótimo, programação estocástica. etc. Cumpre lembrar que a palavra programação aqui significa planejamento, organização de tarefas. etc.: não tem nada a ver com programa de computador (*software*).

Essas técnicas aplicam-se numa grande diversidade de contextos, o que garante um contínuo e renovado interesse no seu conhecimento e desenvolvimento.

Umoutro termo muito usado. com o mesmo sentido de programação matemática, é otimização. Pode ser conceituada como uma disciplina que trata o problema de como escolher a melhor alternativa de ação dentre as muitas possíveis decisões num ambiente complexo da chamada vida real. A sistemática envolve tipicamente quatro estágios ou etapas. Primeiro desenvolve-se um modelo matemático que represente o problema em pauta. Isto inclui a escolha das variáveis. a coleta de dados pertinentes, a formulação da função a ser otimizada (maximizada ou minimizada dependendo do caso), e o arranjo das variáveis e dos dados (parâmetros) num conjunto de relações matemáticas. tais como igualdades e desigualdades, que caracterizam as restrições ou vínculos do problema. O segundo estágio dá seqüência ao processo por intermédio de uma análise do modelo matemático e da escolha de uma técnica numérica para a sua solução.

O terceiro estágio consiste em efetivamente implementar o método numérico e, não apenas encontrar a solução, mas também fazer uma análise de sensibilidade da solução com respeito à variação dos parâmetros do modelo. Esta terceira etapa é implementada com o uso de computadores digitais e o uso e desenvolvimento de *software* adequados. O quarto estágio consiste em retornar ao problema inicial do "mundo real" e estudar a adequação da resposta encontrada com a realidade à qual o modelo deve representar.

A maior parte do desenvolvimento se deu nos chamados algoritmos determinísticos. Apesar de grandes avanços e sucessos, os métodos determinísticos fazem suposições que limitam muito o seu alcance. Entre estas hipóteses pode-se citar a linearidade, a diferenciabilidade e a convexidade das funções envolvidas. São hipóteses deste tipo que permitem a construção de condições necessárias de otimalidade que levam à construção dos métodos numéricos de obtenção de soluções ótimas.

Os métodos probabilísticos não sofrem deste tipo de restrições e consequentemente têm seu alcance muito maior. Dentre estes têm se destacado o ***simulated annealing*** e os algoritmos genéticos, principalmente nos chamados problemas de otimização global (em contraposição à otimização local que tipicamente se consegue com os algoritmos determinísticos). Uma referência muito abrangente da área de otimização é Pardalos (1995).

A demanda por algoritmos eficazes e eficientes de otimização tem se intensificado recentemente, principalmente no desenvolvimento de novas técnicas de modelagem como por exemplo, nas redes neurais artificiais (vide Fine (1999)).

Nesta tese estuda-se. aprimora-se c implementa-se em linguagem C++ um algoritmo probabilístico de otimização global chamado Talus, parte de um projeto do Prof. Fernando Campcllo para concepção, construção e desenvolvimento de um computador paralelo, rápido, que opere com *software* simples, seja tolerante a falhas, estocástico e mais dedicado a problemas matemáticos (programação matemática, controle de processos, processamento de imagens, telecomunicações, simulação, etc.) (vide [Campello de Souza (1987)]).

1.2 Desenvolvimento da Tese

No Capitulo 2 é feita uma apresentação do problema geral de otimização e uma descrição rápida dos algoritmos determinísticos para a sua solução, e chamada a atenção para as hipótese básicas e as limitações desses algoritmos.

No Capítulo 3 são apresentados os algoritmos probabilísticos. Inicialmente são lembrados alguns conceitos básicos de convergência de seqüências numéricas e de funções. Em seguida é apresentado o método da busca aleatória pura. Depois trata-se a questão de geração de números aleatórios e pseudo aleatórios e do chamado método de Monte Carlo. Finalmente são expostos alguns algoritmos probabilísticos em particular o *simulated annealing*, que tem recebido muita atenção recentemente.

O Capitulo 4 trata exclusivamente de uma classe específica de algoritmos probabilísticos que também tem recebido muita atenção recentemente; os algoritmos genéticos. A abrangência maior da apresentação teve o intuito de facilitar a comparação desta classe de algoritmo com o Talus.

No Capítulo 5 são apresentadas melhorias e a implementação do Talus em linguagem C++. É feita uma apresentação detalhada do algoritmo, salientando inclusive as melhorias, e vários problemas apresentados na literatura são resolvidos. Mostra-se inclusive como encontrar pontos de sela. Através de um estudo rápido do tipo "engenharia reversa", procura-se entender o funcionamento do Talus sob a perspectiva dos algoritmos genéticos. Tal abordagem suscitou a sugestão de um novo operador genético : o operador de mutação Talus, que poderá ser útil na implementação de uma arquitetura paralela inspirada no algoritmo.

O Capitulo 6 trata da solução de problemas complexos apresentados na literatura, inclusive alguns oriundos de formulação de problemas do chamado mundo real em vários contextos. Estes incluem problemas não convexos de programação não linear com restrições em espaços de dimensão seis, equações diofantinas e minimax. Evidencia-se o bom desempenho do algoritmo em todos os problemas abordados.

Finalmente no Capítulo 7 são apresentadas algumas conclusões, comentários e sugestões para trabalhos futuros.

Capítulo 2

Algoritmos Determinísticos

2.1 Introdução

Um problema de Otimização pode ser enunciado como segue:

dado : domínio do problema : A
função de avaliação $f: A \rightarrow \mathbb{R}$

encontrar $x_m \in A$ *tal que* $\forall\, x \in A \quad f(x_m) \geq f(x)$

Como exposto em Boyer (1974), sabe-se que, historicamente. Fermat, em 1629 foi o primeiro a descrever uma técnica de reconhecimento do ponto de ótimo: o método de máximos e mínimos. Para curvas polinomiais da forma $y = f(x)$ ele comparou o valor de $f(x)$ com o valor $f(x + E)$ num ponto vizinho. Em geral, estes valores serão bem diferentes mas, num alto ou baixo de uma curva lisa, a variação será quase imperceptível. Portanto, para achar máximos e mínimos Fermat igualava $f(x)$ com $f(x+E)$, e embora os valores não fossem exatamente iguais, eram quase iguais. Quanto menor E, mais próximo do verdadeiro valor era o resultado encontrado; por isto Fermat além de dividir tudo por E, fazia $E=0$. Usando a notação matemática de hoje, a operação era equivalente a tomar o limite da relação entre os acréscimos e igualá-lo a zero, isto é, $\lim \frac{f(x+E)-f(x)}{E} = 0$.

A idéia de Fermat ganhou uma maior generalidade com o advento do Cálculo Diferencial e Integral. Tanto Newton quanto Leibniz perceberam que a primeira derivada se anula num ponto de ótimo local e Fizeram considerações em relação ao sinal da segunda derivada para evidenciar um ponto como máximo, mínimo ou de inflexão.

Jacques Bernoulli observou que não necessariamente em um ponto de ótimo a primeira derivada é nula podendo esta assumir um valor "infinito" ou mesmo indeterminado.

Lagrange forneceu um algoritmo elegante e simétrico para fornecer o ponto de ótimo de uma função $f(x)$ sujeita aos vínculos $g_i(x) = 0$. A idéia foi incorporar os vínculos à função objetivo $f(x)$ usando multiplicadores que hoje levam o seu nome, tornando o problema original em um problema desvinculado.

Modernamente, utiliza-se como critério de otimalidade as condições mais recentemente chamadas de Karush - Kuhn - Tucker, vide Mas-Colell et al. (1995). Porém, estas condições não fornecem um algoritmo para encontrar pontos de ótimo e sim condições às quais, se o ponto for de ótimo, este deve satisfazer.

Métodos numéricos determinísticos podem ser utilizados para encontrar dentre todas as soluções possíveis a melhor delas em um determinado problema. Estes métodos geram soluções a cada passo, e portanto, requerem a utilização do computador. Isto acarreta algumas dificuldades, tais como obter uma solução inicial para o início do processo iterativo. Outra dificuldade advém do uso do computador, tal como escalonamento de variáveis, para reduzir os erros numéricos computacionais.

Diante da necessidade de solução de um determinado problema e considerando que mais de um algoritmo pode ser utilizado, deve-se escolher o mais adequado. Conceitos básicos neste assunto podem ser encontrados em Campello & Maculan (1994). Duas alternativas se colocam:

- o algoritmo deve ser simples, fácil de codificar e depurar;
- o algoritmo deve ser eficiente e robusto, ou seja, resolver o problema utilizando o mínimo de recursos computacionais e em particular deve executar o mais rápido possível todos os seus passos para uma grande classe de problemas.

Porém ainda falta uma definição rigorosa do que vem a ser um algoritmo eficiente, ou ainda, como avaliar a eficiência de certo algoritmo. Himmelblau (1972) afirma que a eficiência de um algoritmo depende do tipo de problema a ser resolvido, do grau de preparação a ser realizado pelo usuário e da disponibilidade de informação sobre a região de vetores viáveis.

Este aponta alguns critérios a serem considerados:

1. Tempo requerido em uma série de testes (tempo de execução ou número de iterações);
2. Tamanho do problema (dimensão, número de restrições de desigualdades, número de restrições de igualdade);
3. Precisão da solução relativa ao vetor ótimo x^* e/ou relativa a $f(x^*)$;
4. Simplicidade de uso (tempo requerido para introduzir os dados e funções no computador);

5. Simplicidade do programa de computador para executar o algoritmo;

6. Capacidade de resolver problemas do mundo real.

Em geral interessam as taxas de crescimento do tempo e do espaço necessário para resolver instâncias cada vez maiores do problema em pauta .

Dentre os parâmetros utilizados, para avaliar o desempenho de um algoritmo, destacam-se o tempo de execução e a memória utilizada. O tempo de execução é sem sombra de dúvida o parâmetro mais usual na avaliação da eficiência de algoritmos.

O tempo de execução depende dos seguintes fatores principais:

do programador;

- dos dados de entrada;
- da qualidade do código gerado pelo compilador;
- do *hardware* utilizado;

da complexidade do algoritmo implementado.

A função que associa o tempo de execução de um algoritmo ao comprimento ou tamanho dos dados de entrada denomina-se complexidade em tempo de algoritmo. Em outros casos, quando o interesse **é** examinar a quantidade de memória para acomodar os dados de entrada e executar o processo, define-se a complexidade em espaço do algoritmo.

Um estudo detalhado da função de complexidade pode ser encontrado em Campello & Maculan (1994).

Conforme exposto em Campello & Maculan (1994), o fato de a função T(.), denominada complexidade local, estar fortemente relacionada com os dados de entrada nos obriga a trabalhar com uma complexidade no pior caso. Entretanto, na maioria dos casos, é difícil obter **T(.)**, mesmo considerando o pior caso. Recorre-se então ao conceito de complexidade assintótica.

Para exprimir a complexidade assintótica de um algoritmo, utiliza-se a notação 0(.) (Big- Oh), introduzida por Paul Bachmann (1894) (citado em Campello & Maculan (1994)). Então :

T(*n*) é da ordem $O(f(n))$ se existem 3 constantes positivas k e n_0, tais que $T(n) < k.f(n)$, para $n > n_0$.

Assim, um algoritmo é de ordem polinomial se for de ordem $O(p(n))$ para alguma função polinomial $p(n)$,caso contrário, será de tempo superpolinomial, isto é, exponencial ou de ordem mais elevada.

Pelo critério conhecido como *The Criterion of Polynomial Boundness*, desenvolvido por Jack Edmonds (1965) (citado em Campello & Maculan (1994)), um algoritmo pode ser considerado eficiente quando requer um número de passos que pode ser limitado por uma função polinomial no tamanho do problema. Uma questão relevante é saber se dado um determinado problema para o qual pode-se construir uma família de algoritmos que o resolvem, existirá dentre estes algum que seja polinomial. Problemas para os quais existem algoritmos polinomiais são

considerados tratáveis, caso comprovadamente não possam ser resolvidos por algoritmos polinomiais são rotulados de intratáveis.

Os problemas comprovadamente tratáveis, isto é, com algoritmos polinomiais conhecidos, constituem a classe P, enquanto os que podem ser resolvidos por algoritmos enumerativos cuja busca pode ser feita em árvore com profundidade limitada por funções polinomiais no tamanho da entrada do problema, comporão a classe NP. É intuitivo que $P \subseteq NP$.

A existência de uma classe ampla de problemas denominada por Cook (1971) e Karp (1972) (citados em Campello & Maculan (1994)), de NP-Completa impulsionou os estudos de complexidade e teoria de algoritmos bem como o desenvolvimento de heurísticas . Nesta categoria estão incluídos todos os problemas para os quais:

- não se sabe se algum deles pertence à classe P;
- se um deles pertencer a classe P , então todos os problemas em NP pertencem a P, o que implica P = NP.

O fato de um problema de programação discreta ser classificado como NP-Completo é aceito como forte evidência da não existência de algoritmos polinomiais para sua resolução e conseqüentemente como uma justificativa para a utilização de algoritmos enumerativos ou para desenvolvimento ou utilização de heurísticas.

Um estudo detalhado da complexidade, bem como caracterizar as classes P e NP pode ser encontrado em Campello & Maculan (1994).

As condições de otimalidade podem ser encontradas de forma detalhada em Cavalcante (1996) e Mateus & Luna (1986). Neste trabalho serão descritos de forma sucinta os principais algoritmos deterministicos para que se possa nos capítulos precedentes fazer comparações com os Algoritmos de Otimização Probabilísticos em geral e em particular com o Talus. Um estudo detalhado destes algoritmos pode ser encontrado em Mateus & Luna (1986), Himmelblau (1972) e Avriel (1976).

2.4.1. Otimização Desvinculada

O problema de Otimização desvinculada é definido como segue :

Seja f : $\rightarrow\mathbb{R}$. Encontrar, se existir um vetor $x^* \in \mathbb{R}^n$, tal que Vx e $\mathbb{R}^n$, $f(x^* \geq f(x)$ ou, de forma resumida, Maximize $f(x)$.

Existe uma diferença fundamental entre os diversos Métodos de Otimização Determinísticos : a utilização ou não no algoritmo das derivadas da função objetivo a ser otimizada. Em muitos casos não se conhece a expressão analítica da função objetivo, ou conhece-se a expressão porém o cálculo do gradiente é extremamente penoso face à complexidade da função, ou quando se quer obter facilidade no preparo das informações iniciais advindas da não utilização de gradientes. Nestas circunstâncias os métodos que não utilizam derivadas são preferíveis aos que usam. Embora, em geral, a eficiência dos métodos que não usam gradiente seja inferior a dos que usam, existem algoritmos que não usam derivadas que possuem desempenho superior ao de vários que delas fazem uso, ver Ribeiro (1973).

Apresenta-se a seguir. sucintamente, as características mais importantes **de** alguns métodos de ambos os tipos com e sem derivadas .

2.4.1.1 Métodos sem derivadas

i. Procura Direta

De grande simplicidade conceitual. é implementado alterando-se o valor **de uma** variável ao longo do tempo enquanto as outras são mantidas constantes. **até que um** mínimo ou máximo seja alcançado. Evidentemente este método de busca direta pura como apresentado acima é de pouca eficiência. contudo suas variantes procuram suprir suas maiores ineficiências (ver Himmenblau (1972) e Avriel (1976)).

ii. Método de Fibonacci (seção áurea)

O algoritmo pode ser descrito da seguinte maneira: dados uma precisão $\varepsilon > 0$, **um** ponto inicial x_0 e uma direção de descida S, ao longo da qual deve ser realizada a busca, procurar determinar um $\lambda > 0$ tal que $|\lambda - \lambda^*| \leq \varepsilon$, onde $\lambda^* > 0$ é um valor de λ **tal que** $f(x + \lambda^* \cdot S) = \min\{f(x + \lambda \cdot S)\}$. O método determina um intervalo inicial $[a, b]$ tal que $\lambda^* \in [a, b]$ e a partir de divisões sucessivas desse intervalo, determina o valor T (divisão áurea). Para este método ser aplicado é necessário que a função seja convexa. pois se tal não ocorrer não se pode afirmar nada sobre o valor de x_{k+1} a partir de x_k.

iii. Método de Davies - Swann - Campey - Povvell

Determina dentro de uma precisão preestabelecida, um valor de aproximado para um ponto de mínimo unidirecional λ usando extrapolação e interpolação. Este método também necessita da condição de convexidade para a função objetivo de modo a poder obter garantir que o ponto iterado não seja pior que o anterior. Esse método utiliza estimativas quadráticas a partir de informações de determinados pontos e valores da função nesses pontos.

iv. Método de Cauchy

E um método derivado do método do gradiente, no qual o gradiente é aproximado por diferenças finitas. O algoritmo efetua um calculo do gradiente da função utilizando vetores canônicos $e_i \in \mathbb{R}^n$. A direção de busca é essa aproximação do gradiente com sinal trocado. Então $f(x)$ deve ser, pelo menos, continuamente diferenciável.

v. Método de Powell

Atinge o mínimo de uma função quadrática com Hessiana positiva definida, em um máximo de n iterações, através de sucessivas buscas unidirecionais ao longo de uma série de direções conjugadas partindo de um ponto inicial x_0.

O método se fundamenta no fato que o mínimo de uma função quadrática é encontrado ao longo de cada p ($p < n$) direções conjugadas em um estágio de procura ; pode-se notar que é necessário apenas um passo em cada direção para que o mínimo seja alcançado nas n direções (ver Himmelblau (1972)).

vi. Método do poliedro flexível (Nelder - Mead)

O algoritmo se baseia no fato de que um poliedro regular de n + 1 vértices em $\mathbb{R}^n$ é um simplex. Então dados n + 1 pontos formando um simplex, pode-se avaliar $f(x)$ em cada um desses pontos e naquele vértice para o qual obtemos maior valor de *fix)* é feita uma reflexão do centróide do simplex. Esse vértice pode ser substituído pelo novo ponto e um novo simplex é obtido. Prosseguindo neste processo e utilizando técnicas de redução gradativa do simplex e de evitar ciclagem na vizinhança do ponto de mínimo, obtém-se um método de desempenho comparativamente melhor, cuja única exigência é a expressão analítica da função objetivo.

2.4.1.2 Métodos com derivadas

Apesar de em geral serem mais eficientes que os métodos sem derivadas, estes métodos apresentam, segundo Himmelblau (1972), os seguintes inconvenientes para implementação:

- Em problemas com número médio de variáveis torna-se muito trabalhoso ou mesmo impossível prover funções analíticas para as derivadas em algoritmos de gradiente ou de derivadas segunda;

- Técnicas de otimização baseadas no uso de derivadas de primeira e segunda ordens requerem uma quantidade relativamente grande de problemas de preparação pelo usuário antes de colocar o problema no algoritmo.

i. Método do Gradiente

E baseado no fato de que o gradiente calculado em qualquer ponto do domínio de $f(\cdot)$ aponta para a direção de máximo crescimento da função. Se se estiver a procura do máximo deve-se andar na direção do gradiente, caso contrário, anda-se no sentido contrário do mesmo. Logo, a direção S_i (i indicando a i-ésima iteração) é determinada por $S_i = -\nabla f(X_i)$ e o novo ponto é obtido através da relação $X_{i+1} = X_i + \cdot S_i = X_i - \lambda \nabla f(X_i)$, onde é o passo, que é obtido pela busca direcional.

ii. Método ***steepest descent***

Como já dito anteriormente a direção contrária ao gradiente é de descida. O gradiente normalizado de $f(\cdot)$, isto é, a direção do gradiente, é definida em x_k por $\boldsymbol{Si} = \frac{\nabla f(x_k)}{|\nabla f(x_k)|}$, fornece a transição de x_k para dada por $x_{k+1} = x_k - \frac{\lambda_k}{|\nabla f(x_k)|} = x_k - \lambda_k^* \cdot \nabla f(x_k)$ onde é o tamanho ótimo do passo.

iii. Método de Newton

O método de Newton utiliza a derivada segunda realizando uma aproximação quadrática de $f(x)$ utilizando a expansão em série de Taylor. Então a função $f(x)$ será considerada igual a

$$f(x_k) + \nabla^T f(x_k) \cdot (x - x_k) \quad \frac{1}{2} \cdot (x - x_k)^T \cdot \nabla^2 f(x_k) \cdot (x - x_k) \qquad (2.1)$$

onde $\nabla^2 f(x_k)$ é a matriz Hessiana de $f(x)$ em x_k.

A direção de procura S do método de Newton é escolhida substituindo $(x - x_k)$ da Equação (2.1) por $\Delta x_k = (x_{k+1} - x_k)$ e a aproximação quadrática de $f(x)$ em termos de Δx_k é igual a

$$f(x_k) + \nabla^T f(x_k)\ \Delta x_k + \frac{1}{2} \cdot [\Delta x_k]^T \cdot \nabla^2 f(x_k) \Delta x_k \qquad (2.2)$$

0 mínimo de $f(x)$ na direção de Δx_k é obtido diferenciando-se $f(x)$ em relação a cada uma das componentes de Δx eigualando-se o resultado a zero. obtendo-se

$$\Delta x_k = [\nabla^2 f(x_k)]\nabla f(x_k) \qquad (2.3)$$

onde $[\nabla^2 f(x_k)]^{-1}$ representa a inversa da Hessiana em x_k. Substituindo a equação (2.3) na equação (2.2) se obtém

$$x_{k+1} = x_k - [\nabla^2 f(x_k \cdot \nabla f(x_k) \qquad (2.4)$$

se $f(x)$ for quadrática o mínimo será alcançado em um único passo. Caso contrário a equação (2.4) deve ser modificada com o intuito de introduzir o parâmetro para o tamanho do passo ficando da forma

$$x_{k+1} = x_k - \lambda_k \cdot \frac{\nabla^2 f(x_k)^{-1} \cdot \nabla f(x_k)}{\nabla^2 f(x_k)] \cdot \nabla f(x_k)} \qquad (2.5)$$

iv. Método de Fletcher - Reeves (Gradientes conjugados)

Em termos de convergência é inferior ao de Newton, todavia o fato de não utilizar derivadas segundas faz sua eficiência eqüivaler a daquele.

O algoritmo é operacionalizado com a geração de uma seqüência de direções S_i que são combinações lineares entre $-\nabla f(X_i)$ e as direções anteriores são de tal forma que se a função objetivo for quadrática, as direções geradas pelo algoritmo são conjugadas (ver Ribeiro (1973) e Avriel (1976)).

Seja x_0 o ponto inicial e $S_0 \in \mathbb{R}^n$ a primeira direção de busca dada por $S_0 = -\nabla f(x_0)$ define-se então as direções S_i, i =1,2, ...de forma recursiva como $S_{i+1} = -\nabla f(x_i + 1) + w_i \cdot S_i$,onde w_i é dado por $w_i = \frac{\langle \nabla f(x_{i+1}}{\quad}$. Demonstra-se que este método converge para funções estritamente convexas e duas vezes diferenciáveis.

v. Método de Davison - Fletcher - Powell

Este método aproxima a inversa da matriz Hessiana da função, diminuindo drasticamente o volume de cálculos envolvidos na obtenção da matriz Hessiana e posteriormente em sua inversa.

A matriz direcional inicial, geralmente, é escolhida igual à identidade. Transformações graduais de direção de gradiente vão se processando a cada passo, para direções de Newton, extraindo-se desse fato as boas fases do método de Cauchy e de Newton.

2.4.2. Otimização Vinculada

O problema geral de Otimização Vinculada pode ser definido da seguinte forma:

Sejam as funções $f:\mathbb{R}^n \rightarrow$R , $g:\mathbb{R}^n \rightarrow$R , $h:\mathbb{R}^n \rightarrow \mathbb{R}^p$, sendo f, g ,h continuamente diferenciáveis. Encontrar, se existir, um vetor $x^* = (x_1, x_2, .., x_n)$ no conjunto V de pontos viáveis, V definido como o conjunto de todos os pontos $x \in \mathbb{R}^n$ tais que $g(x \geq b$ e $h(x) = 0$, tal que $\forall x \in V$, $f(x^*) \leq f(x)$. Esse problema pode ser reescrito de forma mais compacta como:

$$\begin{array}{ll} \underset{x}{\textit{Minimize}} & f(x) \\ \textit{Sujeito a} & g(x) > b \\ & h(x) = 0 \end{array}$$

2.4.2.1 Método sem derivadas

Método de tolerâncias flexíveis:

Esse processo é uma extensão do método do poliedro flexível já visto anteriormente e é devido a Paviani e Himmclblau. Como não necessita de derivadas da função objetivo e das restrições, agiliza bastante a preparação dos dados por parte do usuário tendo este um bom desempenho em relação aos diversos problemas utilizados para teste.

Adota-se um critério de tolerância de violação de restrições durante sua execução, pois o poliedro flexível sofre contrações e expansões ao longo de seu processamento, todavia, tende a diminuir de tamanho a medida que se aproxima do ponto procurado. O algoritmo gera uma seqüência positiva não crescente $\{\phi_k\}$, $k \in \mathbb{N}$ que é utilizada como critério de tolerância e também como critério de parada. Para um estudo dos detalhes operacionais do algoritmo (ver Cavalcante (1996)).

i. Método das penalidades

A idéia é reduzir a solução do problema de otimização vinculado em uma seqüência de problemas de otimização desvinculada da seguinte forma:

Min $\{f(x)+p_i(x) : x \in i = 1,2,\ldots$ Onde $Ih : \mathbb{R}^n \to \mathbb{R}, i = 1,2,\ldots$ são funções penalidades.

Este método admite duas variantes básicas: penalidades exteriores e penalidades interiores; a primeira adiciona a / W um custo positivo se $x \notin V$ forçando a solução do problema a se aproximar de V e na segunda forçam-se a estas soluções a permanecer no interior de V. O método das penalidades admite ainda uma combinação de penalidades (interiores e exteriores) de forma a gerar um método misto podendo com isso usufruir das vantagens de ambos.

ii. Método de direções viáveis

Esse método se destina a resolver o problema

Minimizar $f(x)$, $x \in V = \{x : g_i(x) < 0, i = 1,2 \ldots m\}$ e as funções $f(x)$ e $g_i(x)$ $i = 1,2,\ldots,m$ são continuamente diferenciáveis.

Dado um ponto $x_i \in V$, determina-se uma semi-reta $\{x : x = x_i + \lambda \cdot S_i\}$ passando pelo interior de V, onde $S_i \in \mathbb{R}^n$ e nesta semi-reta escolhe $x_{i+1} = x_i + \lambda \cdot S_i$ tal que $f(x_{i+1} \leq f(x_i)$. Assim este método pode ser utilizado para resolver o problema anterior apenas se o interior for não vazio, desse modo somente serão admitidos vínculos do tipo de igualdade se estes forem lineares.

liste método é uma extensão do processo de Cauchy. com a diferença que aqui x_{i+1} calculado a partir de x_i deve estar dentro da região viável.

2.4.1. Limitações dos Métodos Determinísticos

1. Geralmente exigem um bom comportamento (convexidade, diferenciabilidade) da função objetivo e algumas vezes das restrições.

2. Nos métodos com derivada ocorre uma instabilidade numérica muito acentuada em torno do ponto de ótimo pois $\nabla f(x) \approx 0$. Quando o método utiliza Hessiana ,este problema se acentua pois é necessário lidar com matrizes mal condicionadas que deverão ser invertidas, podendo com isto levar o algoritmo a divergir (erros de *overflow*).

3. Caráter inerentemente seqüencial destes algoritmos

4. Geralmente os algoritmos determinísticos têm o inconveniente de ziguezaguear em torno do ponto de ótimo. Corrigir este problema aumenta sensivelmente a complexidade computacional.

5. Os algoritmos determinísticos encontram apenas máximos (ou mínimos) locais. Para garantir extremos globais é necessário estabelecer hipóteses altamente restritivas; estas hipóteses normalmente não são obedecidas na maioria dos casos práticos.

Capítulo 3

Algoritmos Probabilísticos

3.1 Histórico

O método de estimação do valor de π foi o primeiro algoritmo probabilístico que se tem registro. Criado pelo naturalista francês Conde de Buffon (Georges Louis Leclere 1707-1788), este método consistia em arremessar ao acaso uma agulha de comprimento L em uma tábua limitada por duas linhas paralelas distanciadas por D. Como a probabilidade da ocorrência do evento, caracterizado pela agulha cortar uma das retas, é $P=(2.L/\pi.D)$, pode-se obter um estimador de π bastante preciso. Na segunda metade do século XIX um grande número de pessoas executou o experimento de Buffon tendo sido publicado em 1873 *On cm experimental determination of* π por A Hall como resultado deste experimento.

Em 1899 Lord Rayleigh mostrou que um passeio aleatório unidimensional sem barreiras incorporadas pode fornecer uma solução aproximada para uma equação diferencial parabólica .

Em 1931 A.N. Kolmogorov mostrou uma inter-relação entre processos estocásticos Markovianos e certas equações diferenciais .

No começo do século XX, a escola de Estatística Britânica já investigava um rudimentar método de Monte Carlo.

Durante a segunda guerra o método de Monte Carlo foi usado como ferramenta de pesquisa no trabalho da Bomba Atômica. Este trabalho envolveu diretamente simulações de

problemas probabilisticos ligados à difusão aleatória de neutrons. Neste estágio da pesquisa von Neumann e Ulam refinaram o método ainda conhecido como método da "roleta russa". Em cerca de 1948 Fermi. Metropolis e Ulam obtiveram o Método de Monte Carlo estimando os autovalores da equação de Schroedinger. Estes creditaram o nome do método ao matemático Jonh von Neumann.

Em cerca de 1970. com o desenvolvimento téorico da computação, o método de Monte Carlo adquiriu mais precisão. A força principal deste método se deve ao fato de que o esforço computacional no Método de Monte Carlo é proporcional à dimensão da função na qual foi aplicado o método. Já no restante dos métodos determinísticos este esforço varia com um polinômio de grau igual à dimensão da função. Ver-se-ão os detalhes do método no item 3.4.

Pela sua importância histórica e pelos *insights* que oferece vale a pena detalhar o método de estimação do valor de π conhecido como a "agulha de Buffon".

3.2 A Agulha de Buffon

() livro de Jacob Bernoulli. *Ars Conjectandi*, publicado postumamente em 1713, representa a primeira tentativa de exposição científica da teoria da probabilidade como um ramo separado da ciência matemática. Na *Pars Quarta* do citado livro é enunciado e demostrado o famoso, importante e seminal teorema, que exigiu, segundo o próprio autor, vinte anos de trabalho para ser concluído, e é conhecido também como Lei Fraca dos Grandes Números. O enunciado original (Hald (1990)) é o seguinte :

Teorema (Bernoulli (1713)) : Sejam r e s dois inteiros positivos e faça $p=r\,I\,(r+s)$ e $t=r+s$. Para qualquer número positivo real c, tem-se

$$P_r\left\{|h_n - p\frac{1}{t}\right\} > \frac{c}{c+1} \qquad (3.2.1)$$

para $n = kt$ suficientemente grande, isto é, para $k > k(r,s,c)$ onde $k(r,s,c)$ é o menor inteiro positivo que satisfaz à

$$k(r,s,t) \geq \frac{m(r+s+1)-s}{r+1} \qquad (3.2.2)$$

e m é o menor inteiro positivo que satisfaz à

$$\frac{\overline{\quad\quad}}{\log[(r+1)/r]} \qquad (3.2.3)$$

Obs: Para um dado p pode-se escolher t tão grande quanto se queira de forma que o intervalo para h_n torne-se arbitrariamente pequeno. Abstraindo-se o arredondamento para inteiros, k e portanto n são funções lineares de $\log c$, o que significa que o lado esquerdo de (3.2.1) tende para 1 quando c, e portanto n, tende para infinito.

A demonstração deste teorema pode ser encontrada em Hald (1990), Campello de Souza (1998) e Uspensky (1937).

O teorema de Bernoulli, como qualquer outra proposição matemática, é uma dedução a partir de premissas ideais. Até que ponto estas premissas podem ser consideradas uma boa aproximação da "realidade" só pode ser decidido através de experimentos.

Buffon jogou uma moeda 4040 vezes e obteve 2048 caras e 1992 coroas. Supondo-se que essa moeda fosse ideal, justa, teria-se uma probabilidade de 1/ 2 tanto para cara quanto para coroa. As freqüências relativas obtidas por Buffon foram

$= 0,507$ para cara

$4040 = 0,493$ para coroa

e elas diferem muito pouco das correspondentes probabilidade, 0,5. Neste caso as conclusões que poderiam ser obtidas do teorema de Bernoulli foram verificadas de uma maneira satisfatória.

Segundo Gnedenko (1968), o estatístico inglês Karl Pearson lançou uma moeda 12000 vezes e obteve 6019 caras. A freqüência relativa de cara foi 0.5016. Numa outra ocasião, ele lançou uma moeda 24000 vezes, obtendo cara 12012 vezes. Neste caso, a freqüência relativa de caras foi de 0,5005. Em todos os experimentos citados, a freqüência relativa desviou-se muito pouco da probabilidade 0,5.

Muitos experimentos com moedas foram realizados, e sempre verificou-se uma concordância satisfatória com a teoria. Também foram feitos muitos experimentos com cartas de baralho (jogos de baralho) e loterias, e a mesma concordância se manifestou. Vários relatos podem ser encontrados em Uspensky (1937).

Um dos testes experimentais mais surpreendentes do teorema de Bernoulli foi feito em relação a um problema considerado pela primeira vez por Buffon. Numa cartolina são traçadas várias retas paralelas eqüidistantes, e uma agulha muito fina que é menor (mais curta) do que a distância entre as linhas, é jogada ao azar na cartolina. Denota-se por l o comprimento da agulha e por h a distância entre os seguimentos paralelos de reta, a probabilidade de que a agulha interceptará uma das linhas (a outra possibilidade é que a agulha ficará completamente contida dentro de uma faixa qualquer entre duas linhas vizinhas) foi encontrada como sendo

$2l$

A figura a cima ilustra o aparato experimental.

A coisa que chama mais a atenção nesta expressão é que ela contém o número $\pi = 3,14159\ldots$, um número irracional transcendente que expressa a relação entre a circunferência de um círculo e o seu diâmetro.

Suponha que se joga a agulha muitas vezes e que se conta o número de vezes que ela corta as linhas. Pelo teorema de Bernoulli pode-se esperar que a freqüência relativa das interseções não vai diferir sensivelmente da probabilidade teórica, de sorte que, igualando-as, tem-se uma maneira de obter um valor aproximado de π .

Segue-se aqui uma exposição feita em Uspensky (1937). Uma série de experimentos deste tipo foi realizada por R. Wolf, astrônomo em Zurich, entre 1849 e 1853. Nos seus experimentos a largura das faixas era de 45 *mm*, e o comprimento da agulha era de 36 ***mm.*** Então a probabilidade teórica de interseção é $\frac{72}{45\pi} = 0,5093$.

A agulha foi lançada 5000 vezes e ela cortou as linhas 2532 vezes; a freqüência relativa foi portanto $\frac{2532}{2500} = 0,5064$.

A concordância entre os dois números é muito satisfatória. Se, confiando no teorema

de Bernoulli. escreve-se a equação aproximada $= 0,5093$, encontrar-se-ia o valor 3,1596 para π por menos do que 0,02.

Num outro experimento do mesmo tipo relatado por De Morgan no seu livro *"Budget of Paradoxes"* (1872). Ambrose Smith, em 1855, fez 3204 tentativas com uma agulha cujo comprimento era $\frac{3}{5}$ da distância entre as linhas. Aconteceram 1213 interseções nítidas, e 11 "contatos" nos quais foi difícil decidir. Se considera-se metade dos contatos como interseções, obter-se-ia 1218 interseções em 3204 tentativas, o que daria o número 3,155 para π. Se todos os contatos forem considerados como interseções, o resultado seria 3,1415 , muito mais próximo ao verdadeiro valor de π.

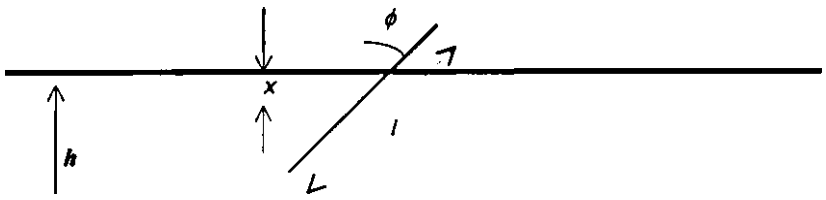
Num excelente livro intitulado "*Calculo delle Probabilitá*" vol. 1, pag. 183, 1925, de autoria de G. Castelnuovo, é feita uma referência ao experimento realizado pelo Professor Reina com uma agulha de 3 *cm* de comprimento que foi lançada 2520 vezes, sendo de 6 *cm* a distância entre as linhas. Levando em consideração a espessura da agulha, a probabilidade de interseção encontrada foi de 0,345. enquanto que os experimentos forneceram uma freqüência relativa de interseção de 0,341.

Serão expostas aqui duas demonstrações para o método da agulha de Buffon. Uma no espírito da demonstração original do teorema de Bernoulli, onde se pode usar a definição

clássica de probabilidade, e outra onde se usa a teoria de Kolmogorov, baseada na teoria da medida e integração de Lebesgue.

3.2.1 Demonstração Clássica

Seja h a largura da faixa entre duas linhas e $l<h$ o comprimento da agulha. A posição da agulha pode ser determinada pela distância x do seu ponto médio até a linha mais próxima e o ângulo agudo ϕ formado pela agulha e uma perpendicular traçada do ponto médio à linha. A figura abaixo ilustra a situação.



É aparente que x pode variar de 0 até $h/2$ e que ϕ varia entre os limites **0** e $\pi/2$, Não se pode definir da maneira clássica a probabilidade da agulha cortar a linha, porque existem infinitos casos (infinito não enumerável) com respeito à posição da agulha. Entretanto , é possível tratar este problema como limite de outro problema com um número finito de casos possíveis, onde a definição clássica de probabilidade pode ser aplicada.

Suponha que $h/2$ é dividida num número arbitrário m de partes iguais $\delta = h/2m$, e o ângulo reto $\pi/2$ em m partes iguais $\varpi = \pi/2n$. Suponha ainda que a distância x pode ter apenas os valores

$0, \delta, 2\delta, \ldots, m\delta$

e o ângulo ϕ os valores

$0, 0), 2\omega, \ldots, n\omega.$

Tem-se então

$$N = (m+1)(n+1)$$

casos possíveis com respeito a posição da agulha, e é razoável assumir que estes casos são igualmente verossímeis. Para encontrar o número de casos favoráveis, note-se que a agulha corta uma das linhas se x e ϕ satisfazem à desigualdade

$$x < \frac{l}{2} \cos\phi$$

O número de casos favoráveis portanto, é igual ao número de pares de inteiros i ,j que satisfazem à desigualdade

$$i\delta < \frac{l}{2} \cos j\varpi \qquad (3.2.4)$$

supondo-se que i pode assumir apenas os valores $0.1 \ldots m$ c j apenas os valores $0,1,2 \ldots n$. Como por hipótese $l < h$, o maior valor de i que satisfaz à condição (3.2.4) é menor do que m e pode-se então considerar a exigência de que i deve ser menor ou igual a m. Para um dado valor de j existem $k+1$ valores de i satisfazendo a (3.2.4) se k denota o maior inteiro que é menor do que

$$\frac{l}{2\delta} \cos j\varpi \ .$$

Noutras palavras, k é um inteiro determinado pelas condições

O número de possíveis valores para *i* correspondente a um dado *j* pode portanto ser representado por

$$m_i = \quad \cos j\varpi + \nu$$

onde ν_i pode depender de j, mas para todo j é maior ou igual a zero e menor do que **1,** $(\forall j \ 0 \leq \nu_i < 1)$. Tomando a soma de todos os m_j's correspondentes a j=0,1,2,...,n ,obtém-se o número de casos favoráveis.

$$M = \frac{1}{2\theta}(1 + \cos 1\varpi + \cos 2\varpi + \ldots + \cos n\varpi) + n\theta$$

onde tf novamente é um número satisfazendo às desigualdades

$$0 < \theta < 1.$$

Mas, como é bem conhecido,

$$1 + \cos\varpi + \cos 2\varpi + \ldots + \cos n\varpi = \frac{1}{2} + \frac{\sin\left(n + \frac{1}{2}\right)\varpi}{2\sin\frac{\varpi}{2}}$$

ou, como $\varpi = \frac{\pi}{2n}$

$$1 + \cos\varpi + \cos 2\varpi + \ldots + \cos n\varpi = \frac{1}{2} + \frac{1}{2}\cot\frac{\varpi}{2},$$

e portanto

Dividindo-se isto *por N*= $(m+1)(n+1)$ substituindo-se δ e ϖ pelas suas expressões

$$\delta = \frac{\quad}{2m} \qquad , \qquad \varpi = \frac{\quad}{2A?}$$

obtém-se a probabilidade procurada para o problema com um número finito de casos,

$$\frac{M}{N} = \frac{l}{2h} \cdot \frac{m}{m+1} \cdot \frac{\cot\frac{\pi}{4n}}{n+1} \quad \frac{l}{2h} \cdot \frac{m}{m+1} \cdot \frac{1}{n+1} \quad \frac{nO}{(n+1)(m+1)^*}$$

A probabilidade no problema de Buffon será obtida fazendo m , n crescerem na última expressão. Como

$$\lim \text{———} = 1.$$

$$\lim \text{———} = \lim \text{———} = 0 \quad \text{e}$$

$$m \to \infty \qquad m \to \infty$$

$$\lim_{n \to \infty} \qquad - \pi$$

tem-se $\lim_{\substack{n \to \infty \\ m \to \infty}} \frac{M}{N} = \frac{2l}{h\pi}$.

Chega-se portanto à expressão da probabilidade

$$p = \frac{2l}{h\pi}$$

no problema da agulha de Buffon.

A agulha de Buffon é o problema mais antigo que trata das probabilidades geométricas. Foi mencionado por Buffon, o famoso naturalista francês do século 18, nos Anais da Academia de Ciências de Paris (1733) e mais tarde reproduzido com a sua solução no livro de Buffon intitulado *"Essai d'arithimetique morales",* publicado em 1777.

A demonstração a seguir tem por trás toda a força da teoria axiomática de probabilidade de Komogorov, baseada na teoria da medida e integração de Lebesgue. Uma medida é uma função de conjunto não negativa e σ-aditiva. O domínio da medida tem que ser uma a-álgebra. No caso da reta real, por exemplo, este domínio é a menor σ-álgebra que contém todos os intervalos abertos, e os conjuntos deste domínio são conhecidos como conjuntos de Borel da reta real. Esta σ-álgebra está propriamente contida no conjunto das partes da reta real. Não se pode construir uma medida no conjunto das partes da reta real. Emile Borel (1871-1956) foi um matemático francês que introduziu estes conceitos, provou em 1909, um teorema hoje conhecido como Lei Forte dos Grandes Números e indicou o caminho que levou a Henri Lebesgue (1875-1941), outro matemático francês, à sua teoria da medida e integração. Kolmogorov (Andrei Nikolaevich Kolmogorov (1903 - 1987), matemático russo) baseou-se em todos esses conceitos na elaboração da sua teoria axiomática da probabilidade, publicada em 1933.

Com a moderna teoria da integração, a probabilidade avança muito com relação à definição clássica, que é circular. e os métodos combinatóricos de cálculo. Agora pode-se abordar com mais rigor o caso do infinito não enumerável. Retome-se o problema de Buffon.

Observe-se a figura abaixo

Figura 3.1

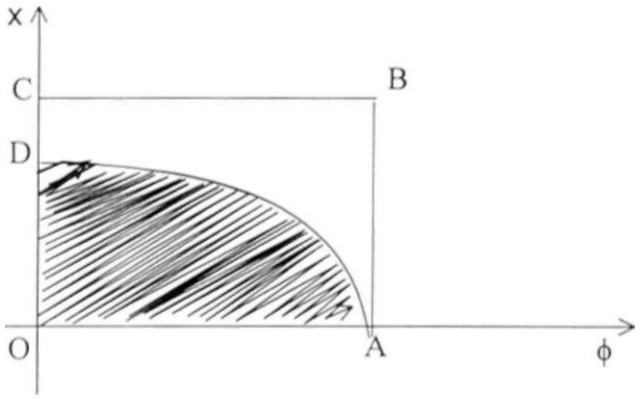


Figura 3.2

Determine-se a posição da agulha pela distância OP= *x* do seu ponto médio até alinha mais próxima. e pelo angulo ϕ entre OP e a agulha. As variáveis *x* e ϕ podem ser consideradas como independentes. Ademais; *x* e ϕ variam respectivamente entre 0 e $h\,I\,2$, 0 e π $I\,2$. Como uma hipótese, assume-se que as distribuições de probabilidades de *x* e ϕ são uniformes. O domínio de *x* , ϕ é o retângulo OABC (figura 3.2) com OA = $\pi/2$,OC = $h\,I\,2$. A agulha intercepta uma das linhas se

$$x < \frac{l}{2}\cos\phi$$

e então o ponto x, ϕ situa-se na área hachurada sob a curva

$$x = \frac{l}{2}\cos\phi \text{ (figura 3.2).}$$

Como a distribuição de x, ϕ é uniforme, a probabilidade procurada será dada por

$$p = \frac{\text{Área } OAD}{\text{Área } OABC}$$

Mas

$$\text{Area } OAD = \int_0^{\pi/2} \frac{l}{2}\cos\phi \cdot d\phi = \frac{l}{2}$$

$$\text{Área } OABC =$$

e consequentemente $p = \frac{2}{\pi}$.

Existe outra solução e uma extensão do problema de Buffon cujos detalhes podem ser encontrados em Uspensky (1937) e Gnedenko (1968).

O problema de Buffon é o ponto de partida para certos problemas de teoria da artilharia a qual leva em consideração a dimensão do projétil.

A fórmula deduzida acima tem sido usada na determinação empírica de uma aproximação de π. A tabela abaixo mostra os resultados de alguns desses experimentos (ver Gnedenko (1968)).

Experimentador	Ano	Número de lançamento da agulha	Valor experimental de π
Wolf	1850	5000	3.1596
Smith	1855	3204	3.1553
Fox	1894	1120	3.1419
Lazzarini	1901	3408	3.1415929

Os resultados de Fox e Lazzarine estão abertos ao ceticismo. De fato, no experimento de Lazzarine, o valor de π foi obtido corretamente até seis casas decimais. Mas, uma mudança de um no número de interseções (o número m) afeta pelo menos a quarta casa decimal se n (o número de lançamentos) é menor do que 5000. De fato, como $h>l$,

Então existe apenas um valor de m que poderia ter resultado no valor de π encontrado por Lazzarini. A probabilidade de se ter exatamente m interseções pode ser computada aproximadamente pela fórmula

$$P_n(m) \approx \frac{1}{\sqrt{2\pi np(1-p)}} e^{-\frac{(n-np)^2}{2np(1-p)}},$$

que é a aproximação gaussiana da distribuição binomial. Se se supõe que $h = 2l$, então para o experimento de Lazzarini encontra-se que para qualquer valor de m,

$$P_n(m) < \frac{1}{\sqrt{2\pi np(1-p)}} \approx 0,03.$$

Portanto, a probabilidade de se obter o resultado de Lazzarini é menor do que 1/30. Uma discussão interessante sobre o experimento pode ser encontrada em O'Beirne(1961).

O método da agulha de Buffon é interessante e ilustra aspectos importantes dos algoritmos probabilísticos.

3.3 Convergência

3.3.1 Conceitos Básicos

Def.3.1 - Um conjunto X chama-se finito quando é vazio ou quando existe , para algum $n \in N$, uma bijeção $\phi : I_n \rightarrow$ onde In={ 1,2,3,...,n}={p∈N; 1< p <n}.

Def.3.2 - Um conjunto X é dito infinito quando este não é finito. Ou seja X é infinito quando não é vazio e alem disto não existe uma bijeção ϕ : A, $\rightarrow X$.

Def 3.3 - Um conjunto X é dito enumerável quando é finito ou quando existe uma bijeção $f : \mathbb{N} \rightarrow X$. No segundo caso, X diz-se infinito enumerável e, pondo-se $f(1) = x_1, f(2) = x_2, ..., f(n) = x_n$ tem-se $X=\{x_1, x_2, ..., x_n\}$

Def.3.4 - Uma norma no espaço vetorial é qualquer função real II : $E \rightarrow \mathbb{R}$ que cumpra as condições abaixo para $x,y \in E$, $a \in \mathbb{R}$.

N1. $|x+y| < |x| + |y|$

N2. $|a.x| = a.|x|$

N3. $x \neq 0 \rightarrow |x|>0$

Def.3.5 - Uma norma | . | em um espaço vetorial E dá origem a uma noção de distância em E . Dados $x, y \in$ .a distância de x a y é definida por : d(x,y) = | x-y |

Def.3.6 - Bola Aberta $B_r(a)$ de centro a $\in$ R, e raio r >0 é o conjunto dos pontos x $\in \mathbb{R}$ cuja distância é menor que r.

Def.3.7 - Dado um conjunto $X \subset \mathbb{R}^n$, um ponto x chama-se ponto interior de X quando existe uma Bola Aberta de centro em x e raio r tal que $B_r(x) \subset X$.

Dcf.3.8 - Um subconjunto $\boldsymbol{A} \subset \mathbb{R}^n$ chama-se conjunto aberto quando todos os seus pontos são interiores.

Def.3.9 - Um ponto a é aderente a um conjunto $X \subset \mathbb{R}^n$ quando para todo conjunto aberto que contém a interseta o conjunto X (Isto é, A aberto $a \in A — \boldsymbol{A} \cap X \neq \phi$.

Def.3.10 - Um conjunto é fechado quando contém todos os seus pontos aderentes.

Def.3.11 - Um conjunto $K \subset \mathbb{R}^n$ é compacto se somente se toda cobertura aberta de $K \subset \bigcup_{\lambda \in L} A_\lambda$ admite uma sub cobertura finita. K U . . . $\cup A_{\lambda_i}$.

3.3.2 Seqüências

Do ponto de vista intuitivo uma seqüência $(x_1, x_2, \ldots, x_n, \ldots)$ de números reais **é** como uma seqüência de pontos da reta e no seu limite como um ponto do qual os pontos ***x„*** tonam-se e permanecem arbitrariamente próximos, desde que n se torne arbitrariamente grande.

Def.3.12 - Uma seqüência de números reais é uma função $x : N \to \mathbb{R}$, definida no conjunto N={ 1,2,3,...} dos números naturais e tomando valores no conjunto *R* dos números reais. O valor $x(n)$, para todo $n \in \mathbb{N}$, será representada por x_n e chamado de termo de ordem n.

Def.3.13 - Diz-se que o número real a é limite de uma seqüência (x_n) de números reais e escreve-se $a=\lim_{n\to\infty} x_n$, quando para cada número real $\varepsilon > 0$, dado arbitrariamente, for possível obter um inteiro n tal que $|x_n - a| < \varepsilon$, sempre que $n > n_0$.

Def.3.14 - Uma seqüência em $\mathbb{R}^n$ é uma aplicação $x : \mathbb{N} \to$ R" , definida no conjunto dos números naturais .

Def.3.15 - Diz-se que o número real $a \in \mathbb{R}^n$, é limite de uma seqüência $(x_k \in \mathbb{R}^n)$ de números reais e escreve-se $a=\lim_{n\to\infty} x_n$, quando para cada número real $\varepsilon > 0$, dado arbitrariamente, for possível obter um inteiro tal que $|x_n - CI| < \varepsilon$, sempre que $k > k_0$.

3.3.3 Séries

Def.3.16 - Seja) uma seqüência de números reais. A partir dela formamos uma nova seqüência s_n cujos termos são :

$$s_1 = a_1; s_2 = a + a_2; \ldots; s_n \; a_1 + a_2 + \ldots + a_n$$

que chamamos reduzidas da série somatório de . A parcela a_n é chamado n-ésimo termo ou termo geral da série

Se existir o limite $s = \lim s_n = \lim_{n\to\infty} a_1 + a_2 + \ldots + a_n$ dizemos que Σa_n . é convergente e o limite s será chamado a soma da série. Caso contrário diremos que esta é divergente.

3.3.4 Seqüência de Funções

Def.3.17 - Seja X um conjunto de números reais .Uma seqüência de funções $f_n : X \to \mathbb{R}$ é uma correspondência que associa a cada número natural $n \in \mathbb{N}$ uma função, definida em X e tomando valores reais .

Def.3.18 - Diz-se que a seqüência de funções $fn : X \to \mathbb{R}$ converge simplesmente (pontualmente) para a função $f : X \to R$ quando, para cada $x \in X$, a seqüência de números $(f_1(x), f_2(x) \ldots f_n(x), \ldots)$ converge para $f(x)$. Ou seja, para todo x fixado, tem-se $\lim f_n(x) = f(x)$

Def.3.19 - Diz-se que a seqüência de funções converge uniformemente para a função $f : X \to \mathbb{R}$, quando para todo. $\varepsilon > 0$ dado, existe $n_0 \in \mathbb{N}$ tal que $n > n_0 \to |f_n(x) < \varepsilon$, seja qual for $x \in X$.

Para otimização (problemas de máximos, mínimos e minmax) o resultado fundamental é que toda função contínua definida em um conjunto compacto atinge o ínfimo e o supremo no compacto. Levando em consideração que um conjunto ser ou não compacto independe do espaço onde ele está embutido, notamos a importância do mesmo para garantir a validade do problema de otimização .

Outros relevantes resultados que podemos citar são:

Teo 3.1. Toda seqüência monótona limitada é convergente.

Prova: Lima (1976), pag. 86

Teo 3.2 (Bolzano-Weierstrass) Toda seqüência limitada em $\mathbb{R}^n$ possui uma sub seqüência convergente.

Prova: Lima (1981), pag. 16 .

3.3.5 Lei dos Grandes Números

Def.3.20 - Y_n converge para Y em probabilidade se para todo $\varepsilon > 0$,

$$P(|Y_n - Y| \geq \varepsilon) \to 0 \text{ quando}$$

Notação : $Y_n \to Y$

Def.3.21 - Y_n converge para Y quase certamente se $P(Y_n \to Y \text{ quando } n \to \infty) = 1$. i.e. o evento $= \{w : Y_n(w) \to Y(w)\}$ é de probabilidade 1.

Def.3.22 Sejam $X, X_1, X_2 \ldots$ variáveis aleatórias com. respectivamente, funções de distribuições $F, F_1, F_2, \ldots$ Dizemos que X_n converge em distribuição para X ,quando para $n \to \infty$, se $F_n(x) \to F(x)$ para todo x ponto de continuidade de F.

Notação : $X_n \to X$ ou $X_n \to F$.

Def.3.23 - Dizemos que X_n converge para X em média r-ésima ,e escrevemos $X_n \xrightarrow{r} X$, se $E|X_n - X|^r \to 0$.

Tco 3.4 - (Lei fraca de Tchebychev) . Sejam $X_1, X_2, \ldots$ variáveis aleatórias independentes 2 a 2 com variâncias finitas e uniformemente limitadas (i.e. existe c finito tal que para todo n $\mathrm{Var}\, X_n < c$). Então $X_1, X_2, \ldots$ satisfazem à Lei fraca dos grandes números :

$$\frac{S_n - ES_n}{n} \xrightarrow{p} 0.$$

Prova : James(1996) pag. 197

Corolário (Lei dos Grandes Números de Bernoulli, publicada em *Ars Conjectandi*, 1713) Considere-se uma seqüência de ensaios binomiais independentes, tendo a mesma probabilidade p de "sucesso" em cada ensaio. Se S_n é o número de sucessos nos n primeiros ensaios ,então $\frac{S_n}{n} \to p$ *em probabilidad*

Prova : James(1996), pag 198.

Teo3.5 (A Lei Forte de Kolmogorov). Sejam $X_1, X_2, \ldots$ variáveis aleatórias independentes ,identicamente distribuídas e integráveis .com $EX_n = \mu$. Então

$$\frac{X_1 + X_2 + \ldots + X_n}{n} \xrightarrow{} p \ \textit{quase certamente}$$

Jacob Bernoulli ilustrará seu próprio teorema com um exemplo dado em seu livro *Ars Conjectandi*, 1713.

"Suponha que, sem seu conhecimento. 3000 bolas brancas e 2000 bolas negras sejam ocultas dentro de uma urna. c que você tenta descobrir seus números sorteando uma pedra atrás de outra (recolocando a anterior antes de tirar a próxima, de modo a não alterar o número de pedras na urna) e notando o quão freqüentemente uma pedra branca ou negra aparece. A questão é . pode você realizar tantos sorteios de forma a fazer com que seja 10 ou 100 ou 1000 ,etc,

vezes mais provável (isto é, moralmente certo) que a razão entre as freqüências das pedras brancas e negras será 3 para 2, como é o caso do número de pedras brancas e negras na urna , do que qualquer outra fração ? Se isso não fosse verdade, eu confesso que nada restaria da nossa tentativa de explorar o número de casos através de experimentos. Mas se isso pode ser obtido e a certeza moral pode finalmente ser adquirida, teremos casos enumerados *a posteriori* com quase a mesma confiança de se eles fossem conhecidos *a priori*. E para fins práticos, onde "moralmente certo" é tomado como "absolutamente certo" pelo Axioma 9, capítulo II, é mais do que suficiente para direcionar as nossas conjecturas em qualquer matéria contingente de forma não menos científica do que em jogos de azar".

Para se mostrar as implicações da Lei dos Grandes números nos algoritmos probabilisticos tome-se o mais simples algoritmo deste gênero, o de busca aleatória pura.

Este algoritmo visa resolver o seguinte problema de otimização global

$$Max\ f(x)$$
$$sujeito\ a\ \ x \in S$$

onde S é um conjunto compacto.

Este pode ser mostrado como segue:

Passo 0 : Faça n = 1 , $y_0 = -\infty$;

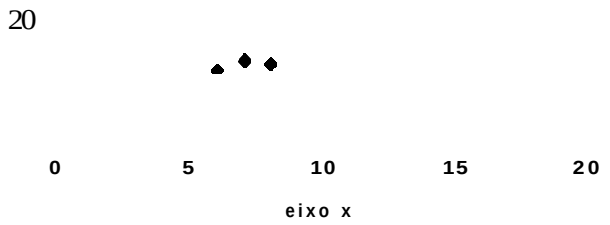
Passo 1: Gere um ponto x da distribuição uniforme em S

Passo 2: Se $f(x) > y_{n-1}$ então faça $y_n = J(x)$ e $x_n = x$. Caso contrario, faça $y_n = y_{n-1}$ e $x_n = x_{n-1}$.

Passo 3 : Incremente n e retorne ao passo 1.

liste algoritmo foi implementado sendo a função que se deseja maximizar descrita como segue, função 1:

Função 1



$$x \in \{0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15\}$$

$$\text{função 1} - f(x) = \begin{cases} 2 \cdot x + 1 & se\ x \in \{0,1,2,3,4,5,6,7\} \\ 30 - 2 \cdot x & se\ x \in \{8,9,10,11,12,13,14,15\} \end{cases}$$

Note que o conjunto S é compacto.

Tem-se que a probabilidade de se achar o ponto de máximo em uma iteração do algoritmo é 1/16. Esta distribuição é Binomial, com probabilidade de sucesso é 1/16. Tomando-se n iterações vai-se obter que a probabilidade de achar-se k vezes o ponto de máximo é dada por : $P(Y_n = k) = \binom{n}{k} p^k \cdot (1-p)^{n-k}$. Como o evento de interesse é que surja pelo menos um vez o máximo ter-se-á $P(Y_n \geq 1) = \sum_{k=1}^{n} \binom{n}{k} p^k \cdot (1-p)^{n-k}$. Vê-se que a certeza

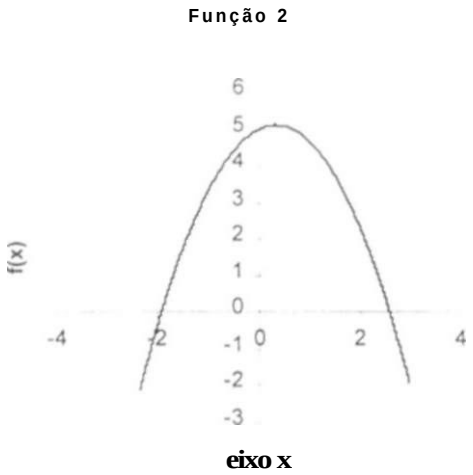
absoluta de achar-se o máximo, mesmo com uma função tão simples quanto a apresentada, só será conseguida quando

Neste exemplo, como no exemplo das urnas de Bernoulli, ao se estabelecer a probabilidade desejada para a ocorrência do evento, pode-se imediatamente estabelecer, pela fórmula acima mostrada, o número de iterações de forma a garantir esta probabilidade. Este cálculo que acaba de ser mencionado, a escolha do número de iterações como função da probabilidade desejada, é não trivial, pois, envolve somatórios de combinações, mas, é factível de ser efetuado através da aproximação da distribuição binomial pela distribuição normal.

Nota-se também que como as variáveis são independentes e identicamente distribuídas é equivalente realizar n iterações deste algoritmo ou se gerar uma única vez n elementos aleatórios e depois escolher dentre estes qual resultou no maior valor da função. É possível observar este fato através do evento sair um 6 num jogo de dados. A probabilidade de ocorrência deste evento se jogar n vezes o mesmo dado é igual à de n dados idênticos jogados uma única vez. A característica de independência de eventos revela o caráter inerentemente paralelo do algoritmo de busca aleatória pura e que mais tarde será salientada também no algoritmo Talus.

Nestes algoritmos a convergência é garantida com probabilidade 1, se o conjunto de busca for compacto, (Teo 3.1).

Agora pode-se salientar também características relevantes da aplicação do algoritmo de busca aleatória pura em funções contínuas. Ao estudar-se a aplicação do mesmo na função $f : [-2,2] \rightarrow \mathbb{R}$ descrita por $f(x) = -x \cdot (x - \frac{4}{3}) + 5$, por exemplo, observa-se, o seguinte :.



O ponto de máximo desta função é $\frac{4}{3}$, que não pode ser representado no computador. Neste caso mesmo com um número infinito de iterações será impossível obter exatidão na obtenção do ponto de máximo. Se está diante, neste caso, de um erro de arredondamento que surge quando se trabalha com máquinas digitais para representar os números reais. Dada esta limitação computacional, a representação dos números reais por um número finito de pontos, para fins práticos o problema de otimização de uma função contínua torna-se idêntico ao caso de se ter uma função discreta como é o caso da função 1. Sendo assim, todas as afirmações feitas para o caso da função discreta tornam-se válidas para as funções contínuas .

Suponha que em uma urna existem 10 bolas numeradas de 0 a 9. Ao escolher aleatoriamente uma bola e anotar o algarismo referente a esta e novamente colocar a mesma na urna, cria-se uma procedimento experimental que ao ser repetido gera uma seqüência de números aleatórios uniformes. Tipicamente trabalha-se com uma seqüência de números aleatórios entre 0 e 1. Assim, por exemplo, um grupo de tais números com quatro casas decimais pode ser formado tirando-se quatro número de uma vez desta seqüência registrada e colocando-se zero e virgula na frente de cada grupo de quatro algarismos.

Processos matemáticos são muitas vezes usados para gerar dígitos que dão seqüências que satisfazem propriedades estatísticas de um processo verdadeiramente aleatório. Uma vez que tal processo não é verdadeiramente aleatório este é batizado como um gerador de números pseudo-aleatórios.

3.4.1 Requisitos Exigidos dos Métodos

Naylor et al.(1971) e Graybeal et al. (1980) salientam os seguintes requisitos para um método de geração de números aleatórios ser considerado aceitável.

1. Seqüência de números que sejam uniformemente distribuídos;
2. Estatisticamente independentes;
3. Reprodutíveis;
4. Não repetidos em qualquer extensão considerada ;
5. Gerar números em alta velocidade;
6. Utilizar o mínimo de memória.

Como mostrado em Cavalcante (1996). pode-se classificar os métodos em três categorias, são ela:

Processo físico aleatório - Por exemplo, interpretar a emissão de elétrons em um cátodo aquecido como uma fase de algum ciclo de relógio. O viés muitas vezes é introduzido através da sincronização do mesmo.

Gerar números *off-line* e armazenar em um arquivo em disco - Esta técnica foi muito difundida na primeira fase da computação, pois, foram aproveitados os livros com as tabelas de números aleatórios. Só que o acesso aos dados muitas vezes é lento e produz um viés, além da necessidade de armazenamento de toda a tabela na memória .

Usar um algoritmo implementado em computador - Com esta técnica foram eliminados os problemas quanto ao acesso aos dados e de capacidade de memória. Estes algoritmos são normalmente de fácil implementação. Tendo em vista que os números gerados são resultados de um processo determinístico estes são chamados de números pseudo-aleatórios.

3.4.3 Algoritmos de Geração de Números Pseudo-Aleatórios

0 método do *Middle-Squared*, proposto por Jonh von Neumann em (1951), foi o primeiro algoritmo de geração de números pseudo-aleatórios em computador digital. Este método, apesar da vantagem de ser de fácil implementação, é considerado como um método com desempenho inferior, em relação aos outros métodos. Seu principal defeito é a facilidade deste de

repetir, depois de um certo número de passos do algoritmo, a mesma seqüência numérica, isto é, a facilidade de ocorrência de ciclos.

O gerador *Middle-Square* de números pseudo-aleatórios

Entrada : N- número de dígitos do número aleatório;

- uma semente com N dígitos positivos.

1. $j = 1$
2. $X = M^2$
3. $M = \begin{cases} \text{os } N \text{ dígitos do meio de } X, \text{ se } X \text{ é par} \\ \text{os } N \text{ dígitos do meio de } 10 \cdot X, \text{ caso contrário} \end{cases}$
4. $U(j) = 0.M$
5. $j = j + 1$
6. Vá para o passo 2.

Saída : $\{U(j)\}$, a seqüências de números pseudo-aleatórios.

Agora serão apresentados os métodos desenvolvidos por Lehmer e alguns variantes deste. Torna-se necessário inicialmente apresentar conceitos básicos da aritmética modular .

Dizemos que dois número x e y são congruentes, módulo m se a quantidade $(x - y)$ for um múltiplo inteiro de m. Por exemplo, fazendo $m = 10$

$3 \equiv 3 \ (módulo\ 10)$ $\quad$ $4 \equiv 4 \ (módulo\ 10)$
$13 \equiv 3 \ (módulo\ 10)$ $\quad$ $84 \equiv 4 \ (módulo\ 10)$
$513 \equiv 3 \ (módulo\ 10)$ $\quad$ $124 \equiv 4 \ (módulo\ 10)$

Pode-se agora apresentar o Método da Congruência Multiplicativa. A fórmula geral de geração de números aleatórios é $r_n = a \cdot r_{n-1} (módulo\ m)$ onde a e m e a semente r_0 são especificados de forma a fornecer propriedades estatísticas desejáveis à seqüência resultante.

porem fazer-se esta escolha de forma a fornecer ciclos longos é não trivial . Pode-se observar que graças à aritmética modular r_n e $\{0,1,2,\ldots,m-1\}$ V $n \in \mathbb{N}$.

Outro método também baseado no mesmo principio é o da Congruência Aditiva. Este envolve K valores iniciais onde K é um inteiro positivo, e gera uma seqüência através da expressão $r_{i+1} = r_i + r_{i-k}$ $(módulo\ m)$. Esse tem a virtude de ser o único método que produz períodos maiores que *m* .

Um gerador equivalente ao *middle-square* de dupla precisão é apresentado em Graybeal et al. (1980), este método só pode ser usado quando m é potência de 2. Sua forma recursiva é dada por $= (x_n \cdot (x_n + 1))$ *módulo nu* $n \geq 0$, a semente x_0 deve satisfazer a relação $x_0\ MOD\ 4 = 2$.

Também podemos citar os geradores de números pseudo-aleatórios baseados em teoria do caos. Um exemplo de geradores deste gênero é o de Feigenbrum baseado na forma recursiva dada por $x_{n+1} = r \cdot x_n \cdot (1 - x_n)$ principalmente para $r = 4$. A densidade de probabilidade desta variável aleatória é dada por $h(x) = \frac{1}{\pi \cdot \sqrt{x(1-x)}}$ $x\ e\ (0,1)$ (normalizada). Estes geradores tem a virtude de com pequenas variações na semente x_0 resultar séries diferentes (ver Ruelle).

3.4.4 Testes estatísticos dos números pseudo-aleatórios

As seqüências $\{x_i\}$ por métodos como os descritos no item anterior não é de fato uma seqüência de números aleatórios, sendo esta tão boa quanto melhor aproxime o

comportamento estatístico de variáveis aleatórias uniformemente distribuídas. Em muitos casos esta aproximação é pobre, comprometendo assim o desempenho dos algoritmos nelas baseados.

Para termos um critério eficiente de comparação entre os métodos .foram desenvolvidos os testes de aleatoriedade que são algoritmos que computam uma estatística para uma seqüência de números. Em Naylor ct.al.(1971) está apresentados os testes de forma pormenorizada.

Os principais testes são :

1. Teste de freqüência ;
2. Teste de série;
3. Teste do produto intervalado ;
4. Teste de encadeamento ;
5. Teste do intervalo;
6. Teste do máximo ;
7. Teste do poker ;
8. Teste de Kolmogorov-Smirnov;
9. Teste da distância.

Existem vários métodos e pesquisas em andamento na questão de geração de números pseudo-aleatórios. Vários algoritmos baseados nos métodos da congruência, têm sido desenvolvidos; bem como algoritmos para geração de números pseudo-aleatório em computadores de arquitetura paralela. Referências mais recentes incluem Knuth (1981);

Marsaglia (1985); Park & Miller (1988): Ferrenberg, Landau & Wong (1992); Coddington (1994) ; Vattulainen. Ala-Nissila & Kankaala (1995).

3.5 Algoritmos Probabilísticos de Otimização

Um ponto extremo é um ponto $x^* \in D$ tal que $f(x^*)$ é o valor máximo ou mínimo da função vetorial multidimensional $f : D \subset \mathbb{R}^n \rightarrow \mathbb{R}$, sendo D um subconjunto compacto do $\mathbb{R}^n$. O máximo é local se $f(x^*)$ é máximo em $|x^* - x| < e$ para algum , e $f(x^*)$ é máximo global se $f(x^*) > f(x)$, $\forall\, x \in D \subset \mathbb{R}^n$.

3.5.1 Integração por Monte Carlo

Existem dois esquemas básicos de integração por Monte Carlo : o método do sucesso e fracasso e o método da média amostrai. Pelo fato do método do sucesso ou fracasso ter sido muito popular nos primórdios de utilização da integração por Monte Carlo, e por sua ineficiência, os métodos de Monie Carlo foram considerados grosseiramente inferiores aos métodos clássicos.

3.5.1.1 Método de sucesso / fracasso

O problema geral é o de estimar a integral múltipla $\Phi = \int \phi(x) \cdot dv$ onde x representa um vetor no espaço de coordenadas $\xi_1, \xi_2, \ldots, \xi_n$, dv representa um volume

elementar do espaço n-dimensional e a integral tem limites fixos e finitos. Desse modo assume-se que ϕ é finita em toda parte e limitada a essa região.

Para o caso monodimensional, seja a função $g(x)$ a ser integrada no intervalo $a \le x \le b$ seja $c = \max\{g(x)\}$ no mesmo intervalo. Tem-se então um retângulo de lados $(b-a)$ e c, que contêm, e é idêntico ao espaço de probabilidades $\Omega = \{(x, g(x)) a < x < b, 0 < g(x) \le c\}$.

O conjunto de todos os pontos $(x, y) \in \Omega$ tais que $y < g(x)$ forma então a área sob a curva $g(x)$. Assim $A = \int_a^b g(x) \cdot dx =$ I como a área de $\Omega = c \cdot (b - a)$, tem-se que dado um ponto de coordenadas $(x,y) \in \Omega$, a probalidade de $(x, y) \in A$ é :

$$P = \frac{\text{area de } \Omega}{c \cdot (b-a)} \quad \frac{c \cdot (b-a)}{c \cdot (b-a)}$$

O método de Monte Carlo de sucesso e fracasso se apóia nesse fato simples para gerar um estimador para I, a partir de uma seqüência aleatória de pares (x_i, y_i) tais que $y_i \le g(x_i)$, contabilizados como n_A. Então a probabilidade de. em n pontos, $(x_i, y_i) \in A$ é estimada como:

$p = \frac{n_a}{n}$, e então pode-se estimar o valor de I como

$$\hat{I} = c \cdot (b-a) \cdot \frac{n_a}{n}$$

Pode-se observar que n_a é o número de acertos da hipótese $(x_i, y_i) \in A$, e $n_E = n - n_A$ é o número de pontos que não atende a hipótese, ou seja , o número de erros.

Seja $f_X(x)$ uma função de densidade de probabilidade qualquer tal que $f_X(x) > 0$ e $g(x) \neq 0$. Se $X \sim U(a,b)$ então $f_X(x) = \frac{1}{(b-a)}$ se $a < x < b$ e $f_X(x) = 0$ em outros casos, e portanto

$$I = \int_a f_X(x) \cdot f_X(x) \cdot dx = E\left[\frac{g(x)}{f_X(x)}\right] = (b-a) \cdot E[g(x)],$$ e daí para uma seqüência $X =$ foi,#2. . . x_n) de variáveis aleatórias uniformente distribuída no intervalo $a \leq x_i \leq b$ pode-se obter:

$$E[g(x)] = \frac{1}{n} \cdot \sum g(x_i).$$ e então o estimador da integral será dado por

$$\hat{I} = (b-a) \cdot \frac{1}{n} \cdot \sum_{i=1}^{n} g(x_i)$$

3.5.2 Otimização baseada cm Integração

Assumindo que o conjunto ***S*** de restrições é fecho de um conjunto aberto limitado não vazio e $f : D \to \mathbb{R}$ uma função contínua, pode-se utilizar o seguinte resultado

$$Max\ f(x) = \lim_{n} \frac{1}{n} \cdot \log \int_D e^{k \cdot f(x)} dx$$ (Equação 3.5.1) descoberto por Laplace (como citado cm Reiner & Pardalos (1995)) e que hoje é utilizado no contexto dos grandes desvios em estatística. A integral pode ser aproximada rapidamente usando o método de Monte Carlo da média amostrai. Um estudo detalhado pode ser encontrado em Hiriart-Urruty (1995).

() teorema a seguir citado em Hiriart-Urruty (1995) também ilustra como é possível encontrar o ponto de ótimo através de integração.

Teorema - Assuma que D é o fecho de um conjunto aberto limitado não vazio e $f : D \to \mathbb{R}$ é contínua. Suponha ainda que se f églobalmente maximizada em $I)$ em um único ponto x^*. Então a seqüência

$$x_k = \frac{\int x \cdot e^{k \cdot f(x)} dx}{\quad}, k = 1, 2 \_\_\_$$ converge para x^* quando $k \to \infty$

3.5.3 Métodos de duas fases

i. Busca aleatória pura

Esse algoritmo consiste na geração de uma seqüência de pontos uniformemente independentes e identicamente distribuídos numa região S. c conseqüentemente seleção dos melhores pontos encontrados

Passo 0 : Faça n = 1 , $y_0 = -\infty$;

Passo 1: Gere um ponto x da distribuição uniforme em S

Passo 2: Se $f(x) > y_{n-1}$, então faça $y_n = f(x)$ e $x_n = x$. Caso contrario, faça $y_n = y_{n-1}$ e $x_n = x_{n-1}$.

Passo 3 : Incremente n e retorne ao passo 1.

ii. *Multistart*

E um método mais eficiente que o anterior, com a desvantagem de inevitavelmente encontrar o mesmo máximo local mais de uma vez. Consiste em aplicar um procedimento local antes do global.

Passo 1: Gere um ponto x da distribuição uniforme sobre S e aplique a x o procedimento local I. obtendo

Passo 2: Se $f(x) > y_{n-1}$ então faça $y_n = f(x)$ e $x_n = x'$. Caso contrario, faça $y_n = y_{n-1}$ e $x'_n = x_{n-1}$.

Passo 3 : Incremente n e retorne ao passo 1.

3.5.4 Métodos de grupamento

A idéia básica que dá suporte a este tipo de método é iniciar a partir de amostras uniformes sobre ***S*** e criar grupos fechados de pontos, e aí aplicar L (procedimento local) àqueles grupos.

i. *Density clustering*

Neste caso a função é localmente aproximada por uma função quadrática. O êxito em otimizar globalmente uma função depende, segundo Boender & Romeijn, do quanto esta aproximação é boa. No esquema de grupamento esse método utiliza uma distância crítica que faz uso do cálculo da matriz Hessiana.

ii. Grupamento com ligação simples

Aqui os grupos são formados seqüencialmente. Cada grupo novamente é iniciado a partir de um ponto procurado, depois que um grupo **C** é iniciado, verifica-se os pontos não agrupados através do critério $d(x,y) = \min \|x - y\|$, seja mínima. Este ponto então é adicionado a

C, depois que o procedimento é repetido até $d(x, C)$ exceder um valor crítico r_k. Boender & Romeijn afirmam que o grupamento com ligação simples aproxima o conjunto de níveis com mais precisão que o *density clustering*.

iii. Ligação simples multiníveis

Esse método, segundo Boender & Romeijn, combina a eficiência computacional do método de grupamento com as virtudes teóricas do *multistart*. O procedimento de busca local L é aplicado a todo ponto da amostra, exceto se existir outro ponto com a mesma distância crítica que têm grande valor de função.

3.5.5 Técnicas de busca aleatória dirigida

Esse procedimento é de aplicabilidade restrita, uma vez que, dependendo do problema, exige a avaliação de um número proibitivamente grande de elementos até que um elemento próximo ao máximo seja identificado. Entretanto, desenvolveram-se diversas técnicas que, embora utilizando um processo aleatório para busca do elemento máximo, incorporam mecanismos altamente eficientes para direcionar esta busca aos subconjuntos do domínio que apresentam características propícias à ocorrência do máximo. Assim estas técnicas foram denominadas técnicas de busca aleatória dirigida.

Dentre as técnicas pertencentes a esta classe, incluem-se Algoritmos Genéticos, Estratégias Evolutivas e *Simulated Annealing*. Algoritmos Genéticos serão descritos com detalhes no capítulo 4; a seguir, descrevem-se, sucintamente, as outras duas técnicas .

Esta é uma técnica de otimização inspirada em fenômenos físicos estudados pela área de mecânica estatística, como a solidificação e cristalização de diversos materiais. No caso de solidificação de um material fundido, por exemplo, as condições em que é efetuado seu resfriamento têm influência marcante sobre a estrutura cristalina obtida para o material; se o resfriamento é suficientemente lento, a energia do estado final dos átomos do material é baixa e a estrutura cristalina bastante regular; se o material é resfriado muito rapidamente, a energia do estado final é alta e não é formada uma estrutura cristalina regular.

Tomando por base uma técnica para simulação de comportamento de materiais nestas condições, desenvolvido por Metropolis (ver Davis (1987), Kirkpatrick et al.(1983)) nestas condições propuseram um método iterativo para otimização, *Simulated Annealing*, descrito pelo algoritmo 3.3 .

Um elemento central nesta técnica é o procedimento utilizado para redução gradativa do valor da temperatura, T . Como no caso do fenômeno físico que inspirou a técnica , se a temperatura é diminuída muito rapidamente, corre-se o risco de atingir uma situação de "alta energia" , ou seja um mínimo local. Diferentes procedimentos para efetuar esta operação têm sido propostos por pesquisadores da área (ver Kirkpatrick et al (1983)) .

dados: A = domínio de definição do problema

c = função de avaliação

- escolher aleatoriamente um elemento ***a*** de A
- avaliar a através da aplicação da função c
- inicializar o valor da variável T (temperatura)
- repetir, até que algum critério de término seja atingido:
  - aleatoriamente, escolher em A um elemento ***b***
  - se $c(a) > c(b)$, substituir ***a*** por ***b***
  - se $c(a) < c(b)$, substituir ***a*** por ***b*** com probabilidade dada por:

$$p = e^{\left(\frac{c(b)-c(a)}{k \cdot T}\right)}, \quad k = \text{constante}$$

  - reduzir o valor de ***T***, segundo algum critério pré-estabelecido.

Hanssoun (1995) comenta que "a efetividade do *simulated annealing* na localização do mínimo global e a sua velocidade de convergência dependem criticamente da escolha da agenda de resfriamento para **T.** Falando em geral, se a agenda de resfriamento é muito rápida, uma convergência prematura para um ótimo local pode ocorrer, e se ela é muito lenta, o algoritmo requererá uma quantidade excessiva de tempo de computação para convergir. Infelizmente foi encontrado teoricamente (Geman and German (1984)) que T precisa ser reduzido muito lentamente em proporção ao inverso do logaritmo do tempo (ciclo de processamento), para garantir que a busca através do *simulated annealing* convirja quase sempre para o mínimo global. O problema de acelerar a busca por *simulated annealing* tem recebido atenção cada vez maior e vários métodos tem sido propostos para acelerar a busca (ver Szu (1986), Salamon et. al. (1988))".

Devido à sua generalidade como método de otimização global, o *simulated annealing* tem sido aplicado a muitos problemas de otimização, mas tem essas limitações. Uma implementação mais recente pode ser encontrada em Goffe et. al.(1994), que utilizaram o programa SIMANN (implementado em Fortran) e verificaram que. em certos problemas de geofísica, a partir de 22 até 30 variáveis os algoritmos genéticos desempenham a contento com certa precisão, embora custosas. e o *simulatedanneling* teve apenas "utilidade" marginal.

3.5.5.2 Estratégias Evolutivas

Estratégias Evolutivas, similarmente aos Algoritmos Genéticos, constituem uma técnica de otimização que se inspira nos princípios de evolução de organismos. Foram desenvolvidos por I. Rechenberg e H.P. Schweffel, cm 1965, sendo voltadas originalmente à resolução de problemas com variáveis contínuas.

Estratégias Evolutivas. normalmente. empregam um operador para geração de novos vetores análogo ao operador de mutação de Algoritmos Genéticos. Algumas implementações desta técnica. entretanto. utilizam também operadores de recombinação (análogo ao operador de *crossover*). Uma implementação simples desta técnica é descrita pelo Algoritmo 3.4.

dados : k = tamanho da população

F = função a processar:

$F(x_1, x_2 \ldots x_n) : M \subseteq \mathbb{R}^n \rightarrow \mathbb{R},\ M \neq \phi$

criar aleatoriamente uma população - pai , constituída por um conjunto de k vetores pertencentes a M:

$\{X_i = (x_{i,1}, x_{i,2}, \ldots, x_{i,n})\}, i = 1 \ldots k$

repetir até que algum critério de término seja satisfeito :

- com base na população-pai, criar uma população filho :

  $\{X_i = (x'_{i,1}, x'_{i,2} \quad _n)\}, i = 1 \ldots k$ onde $x'_{i,j} = x_{i,j} + n_0(\sigma)$

  $n_0(\sigma)$ é um número escolhido aleatoriamente, com distribuição normal de média 0 e desvio padrão σ .

  criar uma população auxiliar contendo todos os elementos da população pai e da população filho

- aplicar a função F a todos os elementos da população auxiliar
- criar uma nova população-pai contendo k elementos da população auxiliar que acarretam maiores valores de F

3.6 Conclusões

Nota-se várias dificuldades inerentes ao problema de otimização global. Abaixo estão as mais relevantes .

1. Algumas dificuldades típicas surgem da própria construção do modelo. matemático, de acordo com Himmelblau (1972). algumas delas são:

- A função objetivo a ser otimizada pode ser insensível às mudanças nas variáveis de decisão independentes;

• A função objetivo ou uma ou mais restrições pode ser ilimitada no intervalo de busca, ou as derivadas parciais das funções do modelo pode vir a ser ilimitada ;

• Uma diferença de escala entre as variáveis, isso ocorre quando os termos da função objetivo são de ordens de magnitude muito diferentes (por exemplo $f(x) = 100 \cdot x_1^2 + 0.01 \cdot x_2^2)$;

• A iteração entre variáveis em um modelo pobre do ponto de vista de projeto (por exemplo, $f(x)=2 \cdot x_1 \cdot x_2 + 10$; neste caso os valores de x_1 e x_2 variam em intervalos muito grandes para um mesmo valor de $x_1 \cdot x_2$;

2. Como se pode obter suposições iniciais adequadas para as variáveis independentes? Como muitos problemas contêm funções não lineares, pode existir mais de um extremo. um aspecto ausente na análise linear. Consequentemente, se as suposições iniciais para as variáveis estão muito longe do extremo global. a otimização pode terminar em outro extremo que não o global ;

3. Como pode-se manusear os aspectos estocásticos da variável real?

4. Como se pode reduzir os erros numéricos computacionais? Erros de arredondamento reduzem a efetividade de muitos algoritmos. Aspectos relativos a estabilidade concernentes à resposta em questão : se a solução aproximada do problema de programação não linear converge no limite para a solução original do problema :

5. Não se pode encontrar condições necessárias e suficientes para otimização global para todas as classes de problemas de otimização, contudo, especialmente para os casos estruturados. eles podem ser manuseados para se obter resultados úteis ;

6. Boender & Romeijn afirmam que o problema de otimização global é inerentemente insolúvel em um número finito de passos .

As abordagem probabilísticas para a solução desse problema têm evoluído e vêm se tornando sofisticadas. Buscam-se também resultados teóricos em que possam assentar-se de forma mais efetiva.

Conceitualmente os algoritmos probabilísticos são algoritmos paralelos. A vasta maioria das implementações dos mesmos foram feitas em arquitetura seqüencial, já fornecendo resultados satisfatórios. A implementação destes em arquitetura paralela é uma abordagem extremamente interessante, reduzindo muito o tempo de processamento. Por concepção os algoritmos determinísticos não se prestam para este tipo de arquitetura, a não ser em casos muito específicos (métodos de busca em bloco baseados nos números generalizados de Fibonacci e seções áureas; (vide Aviei (1976) p.234 e p.275).

A atividade de pesquisa nesta área tem sido muito intensa, como pode ser visto, por exemplo, no "Optimization Web Site" na Internet, devido à grande demanda por algoritmos eficazes e eficientes de otimização em grande número de contextos. Os algoritmos probabilísticos são os que têm tido mais resultados e têm sido portanto os mais promissores.

Devido à maior complexidade, o desafio é maior ainda na área de Sistemas de Controle (Sistemas Dinâmicos). Referências recentes que discutem este assunto são Gopal & Biegler (1998) e Vidyasagar (1998), este último com ênfase nos algoritmos probabilísticos.

Capítulo 4

Algoritmos Genéticos

4.1 Introdução

Ao apresentar sua teoria de evolução através da seleção natural em 1858, Charles Darwin influencia não apenas o futuro da Biologia. Botânica e Zoologia, mas também teve grande influência no pensamento religioso, filosófico, político e econômico da época.

Segue-se, neste capítulo, as apresentações feitas em Hassoun (1995), Tang & Man et al (1996), Nonato (1997).

Simplificadamente, a evolução processa-se da seguinte forma, segundo a Teoria Darwiniana:

- o material genético determina as principais características dos organismos. Estas características, por sua vez, determinam o grau de adaptação do organismo ao meio que vive, ou seja, sua capacidade de sobreviver;
- os organismos mais adaptados ao meio têm maior probabilidade de sobrevivência que aqueles mal adaptados. Ao processo que favorece os primeiros em detrimento dos segundos dá-se o nome de *seleção natural*;
- uma conseqüência da seleção natural é a maior possibilidade dos organismos bem adaptados conseguirem se reproduzir (ou de se reproduzir mais freqüentemente);

- no processo de reprodução, o material genético do organismo-gerador é transferido a seus descendentes (em sua totalidade, no caso de reprodução assexuada, ou parcialmente no caso de reprodução sexuada);
- como um organismo bem adaptado tem maior probabilidade de se reproduzir (e conseqüentemente transferir o seu material genético) há uma tendência ao gradativo aumento do número de organismos que apresentam material genético que propiciou a melhor adaptação ao meio. Analogamente, há uma redução do número de indivíduos que apresentam material genético que não propiciou boa adaptação;
- ocasionalmente. parte do material genético de um organismo pode ser alterado de forma aleatória. Este fenômeno recebe o nome de *mutação*. Devido **à** mutação, surgem indivíduos com características genéticas novas, não apresentadas pelos organismos-geradores.

Por volta de 1900, o trabalho de Gregor Mendel, desenvolvido em 1865, sobre os princípios básicos de herança genética. foi redescoberto por cientistas e teve grande influência sobre os futuros trabalhos relacionados à evolução. A moderna teoria da evolução combina a genética e as idéias de Darwin sobre seleção natural, criando o princípio básico de genética populacional: a variabilidade entre indivíduos em uma população de organismos que se reproduzem sexualmente é produzida pela mutação e pela recombinação genética .

Nos anos 50 e 60 muitos biólogos começaram a desenvolver simulações computacionais de sistemas genéticos. Entretanto foi John Holland quem começou, seriamente. as primeiras pesquisas no tema. Holland foi gradualmente refinando suas idéias e em 1975 publicou o livro "*Adaptation in Natural and Artificial Systems*", hoje considerado a Bíblia de algoritmos genéticos.

Os algoritmos genéticos são métodos de otimização que simulam os processos naturais de evolução, aplicando as idéias darwinianas de seleção. De acordo com a aptidão e a combinação dos operadores genéticos, são produzidos métodos de grande robustez e aplicabilidade.

Esses algoritmos estão baseados nos processos genéticos dos organismos biológicos codificando uma possível solução a um problema de um "cromossomo" composto por cadeias de bits e caracteres. Estes cromossomos representam indivíduos que são levados ao longo de várias gerações, na forma similar aos processos naturais, evoluindo de acordo com os princípios de seleção natural dos mais aptos proposto por Darwin.

A combinação entre genes dos indivíduos que perduram na espécie, podem produzir um novo indivíduo muito melhor adaptado às características de seu meio ambiente.

Os algoritmos genéticos utilizam uma analogia direta deste fenômeno de evolução da natureza, onde cada indivíduo representa uma possível solução para um problema dado. A cada indivíduo é dada uma pontuação de adaptação, obtida pelo valor deste ao ser aplicado na função a que se pretende otimizar. Aos mais adaptados é dada a oportunidade de reproduzir-se mediante cruzamentos com outros indivíduos da população, produzindo descendentes com características de ambas as partes.

Os processos que mais contribuem para a evolução são o ***crossover*** e a adaptação baseada na seleção/reprodução. A mutação tem papel significativo, no entanto, seu grau de importância continua sendo debatido.

O campo de Algoritmos Genéticos incorpora muitos termos originários da área de biologia, uma vez que a genética dos organismos vivos serviu como ponto de partida para o estabelecimento da estrutura básica destes algoritmos. Como muitos destes termos são extremamente específicos, a seguir apresenta-se as definições principais (baseadas em Goldberg et. al. (1987)).

Um Algoritmo Genético procura obter a solução de um problema especifico através de um conjunto de soluções em potencial. Cada uma destas soluções constitui um indivíduo. O conjunto de soluções sob manipulação é denominado *população*. Um ciclo completo de processamento da população (seleção, *crossover*, mutação e substituição de um conjunto de soluções) define uma geração.

Se um problema pode ser representado por um conjunto de parâmetros (genes), estes podem ser unidos para formar uma cadeia de valores (cromossomos), este processo chama-se codificação. Em genética este conjunto representado por um cromossomo em particular **é** chamado de genótipo, contendo a informação necessária para construir um organismo, conhecido como fenótipo. Uma instância de um gene é denominada alelo. A "posição" de um gene dentro do cromossomo chama-se seu *locus*.

Por exemplo, para um problema de maximizar uma função de três variáveis , $f(x, y, z)$, poderia-se representar cada variável por um número binário de 10 bits. obtendo-se um cromossomo de 30 bits e 3 genes.

4.3 Estrutura básica de um Algoritmo Genético

Um elemento essencial de qualquer Algoritmo Genético é a existência de uma *função de adequação* (análoga ao grau de adaptação de um organismo vivo ao meio), que permite a determinação do valor da adequação de uma estrutura do conjunto, ou seja, a quantificação de quão bem sua estrutura correspondente satisfaz ao problema original.

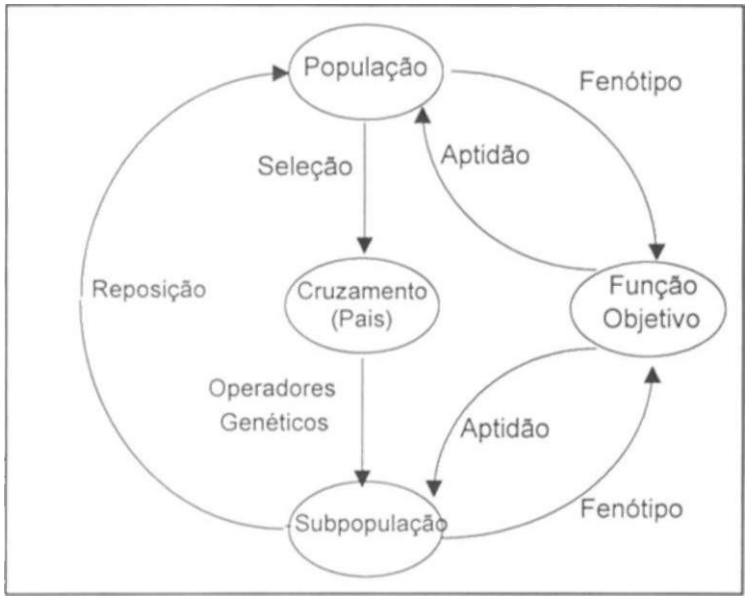
A função de adequação tem papel preponderante na reprodução das estruturas do conjunto, sendo um fator determinante durante a fase de seleção das estruturas. A probabilidade de uma estrutura ser selecionada para reprodução é função da sua adequação. Assim, estruturas com alta adequação têm possibilidade de se reproduzirem mais freqüentemente que aquelas com baixa adequação.

Uma vez selecionado um par de estruturas, estas podem ser combinadas através de um processo de *crossover* (análogo a uma fase específica do processo de recombinação genética biológica). Este processo gera duas novas estruturas, de alguma forma relacionadas com ambas as estruturas originais. Eventualmente, estas novas estruturas podem sofrer uma alteração aleatória, através de um processo de mutação (análogo à mutação biológica). Estas novas estruturas substituem duas antigas no conjunto.

As estruturas idealmente convergem para uma estrutura com alta adequação, através da repetição da seqüência de operações abaixo

[seleção - *crossover* - mutação - substituição de estruturas antigas]

O ciclo dos Algoritmos Genéticos pode ser visualizado na figura a seguir:



Este processo de evolução , que constitui o elemento central de qualquer Algoritmo Genético, é descrito de forma mais rigorosa pelo Algoritmo abaixo

- criar um conjunto inicial de estruturas.
- repetir até que algum critério de termino seja atingido:
  - calcular as adequações de todas as estruturas do conjunto.
  - com base nas adequações calculadas, selecionar alguns pares de estruturas do conjunto (uma mesma estrutura pode fazer parte de mais de um par).
  - em cada par. eventualmente combinar as estruturas através de um processo de *crossover,* gerando duas novas estruturas. Eventualmente, modificar as novas estruturas através de um processo de mutação.
  - calcular as adequações das novas estruturas criadas.
  - inserir as novas estruturas no conjunto, eliminando tantas estruturas antigas quantas forem necessárias para manter a cardinalidade do conjunto constante.

Algoritmo 4.1 - Estrutura Básica de um Algoritmo Genético

As características fundamentais dos Algoritmos Genéticos, as quais são invariantes em relação às diversas formas de implentação ,são as seguintes :

- operam sobre codificações, não sobre as soluções em si.
- manipulam conjuntos de soluções, não soluções individuais;
- para a resolução de um problema, valem-se exclusivamente da função de adequação e das estruturas que armazenam as soluções codificadas; nenhuma outra informação auxiliar sobre o problema é necessária;
- operam de forma probabilística.

Pode-se notar que muitas das características descritas acima são decorrentes das próprias características básicas do processo evolutivo na natureza, descritas no início do capítulo. Elas colaboram para que os Algoritmos Genéticos exibam alguns aspectos de grande interesse:

- são aplicáveis a uma ampla gama de problemas;
- constituem uma metodologia *robusta*, ou seja, mantêm sua eficiência mesmo na presença de alterações do problema em si;
- não são limitados por considerações restritivas sobre o problema, tais como unimodalidade, continuidade ou existência de derivadas da função de adequação;
- os algoritmos são em geral de implementação simples.

Desde a publicação do trabalho de John Holland, em meados da década de 70, inúmeras implementações de Algoritmos Genéticos foram desenvolvidas. O nível de complexidade varia enormemente. assim como seu grau de especificidade com relação às aplicações que se destinam.

Goldberg, seguindo fielmente a estrutura básica de Algoritmos Genéticos estabelecida por Holland. propôs uma implementação bastante simples a qual denominou *Simple Genetic Algorithm*, ou SGA (Algoritmo 4.2). O SGA é bastante inespecífico, sendo aplicável a uma ampla gama de problemas. Em sua versão mais elementar esta implementação não possui alta eficiência, ou seja. exige freqüentemente um número muito elevado de gerações até alcançar o resultado satisfatório.

entradas : N - número de indivíduos da população
n_{gen} - número de gerações
$f(\ )$- função de adequação
Pc - probabilidade de ocorrência de *crossover*
Pm - probabilidade de ocorrência de mutação

- ciar uma população inicial com N indivíduos
- repetir n_{gen} vezes :
  - calcular a adequação de todos os indivíduos da população
  - selecionar N/2 pares de indivíduos
  - criar uma nova população realizando para cada par de indivíduos:
    - com probabilidade p_c, efetuar *crossover* entre os indivíduos
    - se ocorreu *crossover* inserir os novos indivíduos na nova
    - população
    - senão, copiar os indivíduos na nova população para cada gene do cromossomo de cada indivíduo da nova
    - população com probabilidade p_m efetuar mutação
  - substituir a população original pela nova população

Algoritmo 4.2

Devido a sua simplicidade e fidelidade aos princípios de Holland, o SGA pode ser considerado um Algoritmo Genético canônico. Como tal. é freqüentemente utilizado como referência em estudos de desempenho de diferentes implementações de Algoritmo Genéticos ou como exemplo em trabalhos introdutórios sobre o campo de Algoritmos Genéticos.

As principais vantagens do SGA são as seguintes :

- genótipo codificado de forma binária;
- população inicial criada de forma aleatória ;
- crossover de um ponto;
- mutação *bit* a *bit*;
- substituição total da população a cada geração.

4.5.1 O operador seleção do SGA

O SGA pressupõe a utilização de uma função de adequação normalizada, o valor da adequação deve pertencer ao intervalo real [0,1], sendo este valor crescente com o nível de qualidade do indivíduo, ou seja, 0 indica o indivíduo completamente inadequado e 1, um indivíduo ótimo. Caso a função de adequação de um problema específico não satisfaça a estas condições é necessário adotarmos uma função de normalização, que mapeie o intervalo original no intervalo [0,1]. Sendo assim, a função de adequação utilizada será a composição da função de normalização com a função original.

No SGA, a probalidade de um indivíduo ser selecionado é dada por:

$$p_{sel}(individuo\ \ i) = \frac{f_i}{\sum_{i=1}^{N}}$$ onde: f_i =adequação do indivíduo i

N = números de indivíduos na população

A seleção de indivíduos é efetuada através de um método denominado seleção por roleta. A descrição deste método é mostrada abaixo

entradas: N - número de indivíduos na população

$\{f_i\}$ - conjunto de adequações dos indivíduos

- ordenar os indivíduos segundo um critério arbitrário
- associar a cada indivíduo um valor de adequação acumulada, igual à somatória das adequações de todos os indivíduos anteriores a ele (incluindo ele próprio)
- gerar aleatoriamente um número, com distribuição uniforme entre 0 e a somatória das adequações de todos os indivíduos
- o indivíduo selecionado será o primeiro cuja adequação acumulada for maior ou igual ao número gerado

Algoritmo 4.3

4.5.2 O operador de *crossover* do SGA

O SGA usa um método muito simples de *crossover*, o *crossover* de um ponto . Um ponto de cruzamento é escolhido e a partir deste ponto as informações genéticas dos pais serão trocadas. As informações anteriores a este ponto em um dos pais serão ligadas às informações posteriores à este ponto no outro pai, como é mostrado na figura 4.1. Uma característica importante deste método é que o genótipo dos novos indivíduos criados é formado por duas

seqüências contínuas de alelos, cada uma proveniente de um dos indivíduos originais. Este método é descrito pelo Algoritmo 4.4.

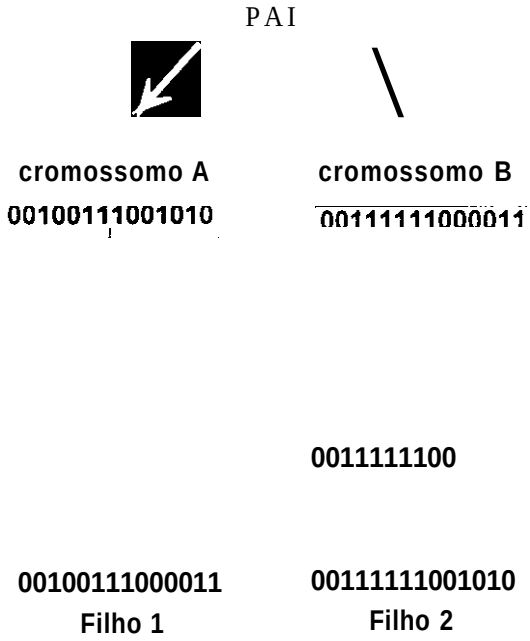


Figura 4.1

entrada: g_i e g_j - genótipos dos dois indivíduos

- aleatoriamente, selecionar um *locus* como ponto de *crossover*, restrito ao intervalo [1, *locus* do penúltimo gene]
- copiar os trechos de genótipo entre o *locus* 1 e o ponto de *crossoxer* dos dois indivíduos originais para dois novos indivíduos
- copiar os trechos restantes dos dois genótipos para novos indivíduos, de forma que cada um deles contenha um trecho de cada um dos indivíduos originais; a ordem relativa entre os trechos deve ser mantida

Algoritmo 4.4 - *crossover* de um ponto

Este operador é necessário para a introdução e manutenção da diversidade genética da população, alterando arbitrariamente um ou mais componentes de uma estrutura escolhida, como é ilustrado na figura abaixo, fornecendo assim meios de introdução de novos elementos na população. Desta forma. a mutação assegura que a probabilidade de se chegar a qualquer ponto do espaço de busca nunca é zero, além de controlar o problema dos mínimos locais, pois com este mecanismo, altera-se levemente a direção de busca. Vale salientar que se utiliza uma taxa de mutação pequena, pois este é um operador secundário.

Figura 4.2

Os Algoritmos Genéticos têm se mostrado bastante eficientes ao ser utilizados numa ampla gama de problemas. Porém, o embasamento teórico na qual esta metodologia se baseia ainda é precário. A importância deste embasamento pode ser vista, por exemplo, para garantir a convergência deste algoritmo ou, alternativamente, determinar condições para sua convergência. Neste item são mostrados os principais resultados teóricos relativo aos Algoritmos Genéticos.

Ao propor a estrutura geral dos algoritmos Genéticos, Holland apresentou um teorema que modela o comportamento de tais algoritmos. O Teorema do Esquema, como passou a ser conhecido o teorema proposto por Holland, procura explicar a capacidade de identificar e explorar as regularidades existentes no espaço de solução do problema em questão.

A metodologia proposta por Holland se baseia na idéia de Esquema. Esquema é uma entidade que descreve um conjunto de genótipos com algum padrão de similaridade. Especificamente, um esquema define o conjunto de todos os possíveis genótipos que apresentam um mesmo subconjunto de alelos. A construção de um esquema pode ser feita através da introdução do símbolo *don't care* , #, no alfabeto dos genes.

Como exemplo, considerem-se os seguintes genótipos codificados na forma binária:

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

Este conjunto de genótipos é membro de 16 diferentes esquemas, apresentados a seguir:

0 0 # # 1 1
0 0 # # **1** #
0 0 # # # **1**
0 # # # 1 1
#0 # # **1 1**
0 0 # # # # #
0 # # # 1 #
o # # # # 1
#0 # # 1 #
#0 # # # **1**
1 1
0 # # # # # #
0 # # # # #
1 #
#**1**
# # # # # # # #

Similarmente, o esquema **1** 1 **1 1 1** # # **1** define o seguinte conjunto de genótipos :

1 1 1 1 1 00 **1**
1 1 1 1 1 0 1 1
1 1 1 1 1 1 0 1
1 1 1 1 1 1 1 1

Pode-se notar que um mesmo indivíduo é membro de vários esquemas diferentes. Define-se, então que um esquema está presente em uma população se ao menos um de seus indivíduos é representante daquele esquema, isto é, possui um genótipo que é membro daquele esquema.

Adotando-se o conceito de esquema como ferramenta para modelar o comportamento de Algoritmos Genéticos, um primeiro passo para esta modelagem é a avaliação dinâmica do conjunto de esquemas representados numa dada população no caso de um algoritmo simplificado, cujo único operador aplicado é o de seleção.

Este algoritmo não produz novos indivíduos. Entretanto, dada uma população, aqueles indivíduos com adequação superior a média tem mais alta probabilidade de se reproduzir, e inversamente, os com adequação inferior tem mais baixa probabilidade. Assim, após uma geração, a população terá alta probabilidade de possuir um número de indivíduos idênticos aos pais de alta adequação e um número menor daqueles idênticos aos pais de baixa adequação.

Uma análise semelhante pode ser utilizada com relação ao comportamento dos esquemas presentes na população. Caso a adequação de um esquema específico seja baixa , após uma geração a população deverá possuir um número menor de representantes daquele esquema. Para analisar este fenômeno define-se uma grandeza análoga a adequação dos indivíduos, mas em relação aos esquemas. Esta grandeza, a adequação do esquema, define-se como a média das adequações de todos os representantes de um dado esquema ***S,*** representada por $f(S,t)$.

Como exemplo, considere-se a população formada pelos seguintes indivíduos:

indivíduo	genótipo	adequação
I_1	1 0 1	0.1
I_2	0 0 1	0.5
I_3	0 0 0	0.9

têm-se/(# # #) = (0,1 + 0,5 + 0,9) / 3 = 0,5
f (# #1) = (0,1 + 0,5) / 2 = 0,3
f (# # 0) = 0,9 / 1 = 0,9
f (0 0 #) = (0,9 +0,5)/2 = 0,7

Seja $\zeta(S,t)$ o número de elementos do esquema ***S*** na presente geração. A probabilidade deste esquema, isto é, de um dos elementos deste esquema, ser selecionado para

próxima geração é dada por $f(S, t)/F(t)$, onde $F(t)$ é a aptidão média da presente população. O número esperado de ocorrências de *S* para a próxima geração é dado por

$$\zeta(S, t+1) \quad \zeta(S, t) \cdot \frac{f(S,t)}{F(t)} \qquad (3.1)$$

Seja $\varepsilon = (f(S,t) - F(t))/$ (3.2)

Se $\varepsilon > 0$, isto significa que o esquema tem uma aptidão média superior e, vise-versa.

A substituição de (3.2) em (3.1) mostra que um esquema de aptidão média superior tem um aumento exponencial no número de elementos nas gerações seguintes:

$$\zeta(S, t) = \zeta(S, 0) \cdot (1 + \varepsilon)^t \quad (3.3)$$

Durante a evolução do Algoritmo Genético, os operadores genéticos destroem grande parle dos esquemas correntes: contudo, seus efeitos devem ser considerados. Assumindo que o comprimento do cromossomo é L c o *crossover* de um ponto é aplicado, em geral, o ponto de *crossover* é selecionado uniformemente dentre as L-1 possibilidades.

Isto implica que a probabilidade de destruição do esquema 5 é

$$\textbf{\textit{PÁS)}} = \frac{\sigma(S)}{L-1} (3.4)$$

ou a probabilidade do esquema sobreviver é

$$p_s(S) = 1 - \frac{\sigma(S)}{L-1} \text{ equação } 3.5$$

onde σ é o comprimento de definição do esquema *S* sendo este definido como a distância entre os dois alelos fixos mais extremos do esquema, ou seja a diferença entre os *loci* daqueles dois alelos. Por exemplo, o comprimento de definição do esquema # 0 0 0 # é dois enquanto o comprimento de definição do esquema 1 # 0 0 # é três.

Assumindo que a taxa de aplicação do operador *crossover* é p_c, a probabilidade de um esquema sobreviver é dada por

$$p_s \geq 1 - p_c \cdot \frac{}{L-1} \quad (3.6)$$

Agora pode-se analisar o efeito da mutação sobre os esquemas de uma dada população. Se a probabilidade de um bit sofrer mutação é p_m, então a probabilidade de um único bit sobreviver é $1 - p_m$ Definindo a ordem do esquema *S* (denotada por $o(S)$) como o número de posições fixas (i.e. posições com 0 ou 1) presentes no esquema, a probabilidade do esquema *S* sobreviver à mutação é dada por

$$\boldsymbol{Ps} = (1 - p_m)^{o(S)} \quad (3.7)$$

Desde que $p_m \ll 1$, esta probabilidade pode ser aproximada por

$$p_s = 1 - o(S) \cdot p_m \quad (3.8)$$

Combinando os efeitos de seleção, *crossover* e mutação, obtém-se uma nova equação que mostra o crescimento do número de esquemas dada por

$$\zeta(S, t+1) \geq \zeta(S, t) \cdot \frac{f(S,t)}{F(t)} \cdot \left[1 - p_c \cdot \frac{\sigma(S)}{L-1} - o(S) \cdot p_m \right] \quad (3.9)$$