UNIVERSIDADE ESTADUAL PAULISTA
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA
DEPARTAMENTO DE ENGENHARIA MECÂNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

Avaliação de Operadores de Algoritmos Genéticos em Otimização Multidimensional

Alexandre Beletti Ferreira

Orientador: Prof. Dr. João Batista Aparecido

Ilha Solteira, Setembro de 2007.

UNIVERSIDADE ESTADUAL PAULISTA
FACULDADE DE ENGENHARIA DE ILHA SOLTEIRA
DEPARTAMENTO DE ENGENHARIA MECÂNICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA

Avaliação de Operadores de Algoritmos Genéticos em Otimização Multidimensional

Alexandre Beletti Ferreira

Dissertação apresentada à Faculdade de Engenharia de Ilha Solteira da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos exigidos para a obtenção do título de **Mestre em Engenharia Mecânica**.

Orientador: Prof. Dr. João Batista Aparecido

Ilha Solteira, Setembro de 2007.

FICHA CATALOGRÁFICA

Elaborada pela Seção Técnica de Aquisição e Tratamento da Informação
Serviço Técnico de Biblioteca e Documentação da UNESP - Ilha Solteira.

F383a

Ferreira, Alexandre Beletti
Avaliação de operadores de algoritmos genéticos em otimização multidimensional / Alexandre Beletti Ferreira. -- Ilha Solteira : [s.n.], 2007
199 p. : il.

Dissertação (mestrado) - Universidade Estadual Paulista. Faculdade de Engenharia de Ilha Solteira, 2007

Orientador: João Batista Aparecido
Bibliografia: p. 180-182

1. Algoritmos genéticos. 2. Otimização. 3. Problema do caixeiro viajante.

unesp UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Campus de Ilha Solteira

CERTIFICADO DE APROVAÇÃO

TÍTULO: Avaliação de Operadores de Algoritmos Genéticos em Otimização Multidimensional.

AUTOR: ALEXANDRE BELETTI FERREIRA
ORIENTADOR: Prof. Dr. JOÃO BATISTA APARECIDO
DATA DA REALIZAÇÃO: 06 DE SETEMBRO DE 2007

Aprovada com parte das exigências para obtenção do Título de MESTRE em ENGENHARIA MECÂNICA pela Comissão Examinadora:

Prof. Dr. JOAO BATISTA APARECIDO (Presidente)
Departamento de Engenharia Mecânica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. EMANUEL ROCHA WOISKI
Departamento de Engenharia Mecânica / Faculdade de Engenharia de Ilha Solteira

Prof. Dr. LUIS CARLOS DE CASTRO SANTOS
Departamento de Matemática Aplicada / Universidade de São Paulo

Dedicado

à Célia Maria Beletti Ferreira.

A G R A D E C I M E N T O S

Primeiramente à Deus e o exemplo de Jesus Cristo, que me deram forças para continuar até o fim de mais uma etapa.

Ao professor João Batista Aparecido por me motivar, compreender, ajudar e por seu rigor, entre outros fatores que ajudaram a modificar meu caráter de maneira positiva.

Aos familiares e amigos Célia Maria Beletti Ferreira, Silvia Maria Beletti, Alice Vendramini Ferreira, Alessandra Vieira Basseti, Luiz Henrique Gazeta de Souza, Jussara Malia Zachi, Rangel Ferreira do Nascimento, Valdeci Reis do Amaral, Thiago Henrique Veronezi Beppu, Rui Silvestrin, Dorotea Vilanova Garcia, Guilherme Sampaio Roxo, Marcos Arias da Silva, Luís Fernando Costa Compiani e Luiz Alexandre da Costa Araújo.

Muito obrigado a todos.

R E S U M O

Ferreira, Alexandre Beletti, Avaliação de Operadores de Algoritmos Genéticos em Otimização Multidimensional, Ilha Solteira, Faculdade de Engenharia de Ilha Solteira – UNESP, 2007, 199p., Dissertação (Mestrado em Engenharia Mecânica).

Desenvolveu-se neste trabalho a implementação computacional de um algoritmo genético. Este se constituiu de uma população inicial sobre a qual agem quatro operadores fundamentais: *seleção*, *"crossover"*, *substituição* e *mutação*, e produz uma nova população. Sobre a qual agem novamente os operadores genéticos, e assim sucessivamente produzindo uma seqüência de populações. O operador *seleção* foi implementado em três algoritmos básicos: *roda da roleta, amostragem estatística universal* e *torneio*. O *"crossover"* também foi desenvolvido em algumas opções: *um ponto, dois pontos, múltiplos pontos,* e *uniforme*. A *substituição* de indivíduos da população pelos filhos ocorre de três maneiras básicas: *dos pais, dos menos aptos,* e *dos indivíduos sorteados aleatoriamente*. A *mutação* ocorre de apenas uma maneira. Inicialmente, o algoritmo genético foi executado em computador de maneira seqüencial. Resolveu-se um conjunto de problemas de otimização multidimensional e também o Problema do Caixeiro Viajante (*TSP – Traveler Salesman Problem*). Fez-se um estudo paramétrico dos vários parâmetros que aparecem no algoritmo genético, tais como: *tamanho da população, número de gerações, taxa de seleção, probabilidade de mutação,* e *taxa de elitismo*. No caso de problemas de otimização multidimensional a representação do cromossomo de cada indivíduo é binária, já no caso do TSP a representação é inteira decimal. Em ambos os casos da otimização multidimensional e do TSP também foi utilizada a técnica de *hill-climbing* visando aumentar a taxa de convergência da solução. A técnica de *janelamento* foi utilizada somente no caso de otimização multidimensional, também visando

aumentar a taxa de convergência. Posteriormente, o algoritmo genético foi executado também em processamento computacional paralelo, usando vários computadores ligados em rede. Em cada computador executou-se um algoritmo genético sobre uma população local. A interação entre as várias populações foi realizada através da migração de clones de indivíduos mais aptos de uma população para outra. Computacionalmente, neste caso utilizou-se a biblioteca MPI (*Message Passing Interface*) para transferir informações de um computador ao outro.

Palavras-chave: Algoritmo genético, otimização, seleção, *crossover*, mutação, substituição, problema do caixeiro viajante, *hill-climbing*.

A B S T R A C T

Ferreira, Alexandre Beletti, Evaluation of Operators of Genetic Algorithm in Multidimensional Optimization, Ilha Solteira, Faculdade de Engenharia de Ilha Solteira – UNESP, 2007, 199p., Dissertação (Master in Mechanical Engineering).

It was developed in this work the computational implementation of a genetic algorithm. That is constituted of an initial population upon which act four basic operators: *selection, crossover, substitution* and *mutation*, producing a new population. Upon which act again the genetic operators, and thus, successively, producing a sequence of populations. The operator *selection* was implemented in three basic algorithms: *roulette wheel, stochastic universal sampling*, and *tournament*. The *crossover* also was developed in some options: *one point, two points, several points*, and *uniform*. *Substitution* of individuals from the population by the newborns happens in three basic ways: *the fathers, the less apt*, and *the individuals sorted randomly*. *Mutation* happens in only one manner. Initially, the genetic algorithm was processed sequentially in the computer. It was solved a set of multidimensional optimization problems and also the Traveler Salesman Problem - TSP. It was done a parametric study of the several parameters that appear in the genetic algorithm, such as: *population size, number of generations, selection rate, mutation probability,* and *elitism rate*. In the case of multidimensional optimization problems the chromosome representation of each individual is binary, but in the case of TSP the representation is integer decimal. In both cases of multidimensional optimization and TSP also it were used the *hill-climbing* technique aiming to increase the solution convergence rate. The *windowing* technique was used just for the multidimensional optimization case, also aiming to increase the convergence rate. Lately, the genetic algorithm was also performed in a computational parallel processing mode, using

several computers linked by a net. In each computer it was executed one genetic algorithm upon a local population. The interaction among several populations was done through the migration of the more apt individual clone, from one population to another. Computationally, in this case it was used a library called MPI (Message Passing Interface) to transfer information from one computer to another.

Keywords: Genetic algorithm, optimization, selection, crossover, mutation, substitution, traveler salesman problem, hill-climbing.

SUMÁRIO

LISTA DE SÍMBOLOS

n_{p}	Tamanho da população
n_{G}	Número de Gerações
n	Número de variáveis
n_{gv}	Número de genes por variável
ε	Taxa de elitismo
λ	Taxa de seleção
p_{m}	Probabilidade de mutação
n_{C}	Número de cidades
f_1	f_1 de De Jong
f_2	f_2 de De Jong
f_3	f_3 de De Jong
f_4	f_4 de De Jong

LISTA DE FIGURAS

LISTA DE TABELAS

1. INTRODUÇÃO

Em razão da quantidade de cálculos utilizados em engenharia atualmente, mesmo com a utilização de computadores com grande capacidade de armazenamento e alta velocidade de processamento, muitas vezes isso não basta.

Com o desenvolvimento de novas técnicas para solucionar esse problema de excesso de tempo consumido para realizar cálculos extensos e consumo de memória cada vez mais alto, procura-se cada vez mais diminuir a quantidade de recursos consumidos.

Dentro da problemática da solução de problemas da área de Ciências Térmicas, surge um mecanismo de otimização utilizando métodos não determinísticos, que possibilitam a busca do melhor resultado de maneira mais eficaz e robusta. Tal mecanismo recebe o nome de algoritmo genético.

Quando abordado o aspecto dos algoritmos inspirados em modelos biológicos, no caso especifico o algoritmo genético, diversos fatores serão analisados no processo de otimização de funções matemáticas.

Primeiramente abordar-se-á a maneira pela qual os códigos computacionais podem se inspirar em modelos biológicos, observando o limite de representação, onde a analogia a ser seguida deve respeitar certos limites, principalmente em se tratando da resolução de problemas não abordados na área de biologia.

A implementação de operadores genéticos, como a mutação e o *crossover*, evidenciarão a eficiência desse tipo de algoritmo, proporcionando diversidade genética para os indivíduos e gerando filhos que contem características dos pais geradores.

O aspecto da seleção dos mais aptos, processo natural proposto por Darwin (2006) adaptado ao modelo computacional, mostrou que bons resultados tendem a se manter ao longo das gerações, fazendo com que o melhor resultado não seja perdido facilmente.

Além disso, serão abordados mecanismos que ajudam no processamento desse tipo de algoritmo, como por exemplo o elitismo, que juntamente com o processo de seleção, ajuda a manter intactos indivíduos bons ao longo das gerações, até que outros melhores sejam, ou não, obtidos pelos operadores genéticos comentados no capítulo 2 desse trabalho.

Para comprovar a eficiência de tal implementação, várias funções serão propostas ao longo do capítulo 3, e com os testes realizados serão obtidos valores de parâmetros adequados para uma boa execução do processo do algoritmo genético e então tais parâmetros serão mantidos por quase todos os outros testes, excetuando-se o fato de ajustes que precisarão ser feitos para casos específicos de alguns problemas.

Em um outro ponto da abordagem bio inspirada, procura-se codificar o conhecido problema do caixeiro viajante, no qual procura-se obter o caminhos ótimo de percorrer um dados número de cidades.

Esse tipo de problema tende a possuir um alto tempo de execução conforme ocorre o crescimento da quantidade de cidades envolvidas na solução, porém a codificação do tipo algoritmo genético favoreceu muito a solução desse tipo de problema em um tempo razoavelmente pequeno.

Nessa abordagem a codificação do problema precisa ser ajustada, bem como alguns operadores genéticos, o que leva a construção de um modelo de algoritmo genético modificado para TSP (Travel Salesman Problem).

Paralelamente aos algoritmos genéticos, existe a computação paralela, que é atualmente uma necessidade fundamental em muitas aplicações que envolvem processamento de algoritmos muito complexos e/ou grande volume de dados. No entanto, sistemas especialistas de processamento paralelo, com vários processadores interligados, são ainda muito caros e complexos, tanto em sua construção (*hardware*) como na programação (*software*).

Conectando estações comuns de trabalho de forma que elas se comportem como um único computador pode-se atingir desempenho compatível com o dos grandes sistemas paralelos especialistas com custos e complexidade bem inferiores. Este é o princípio básico do *clustering*. *Cluster* são, em uma definição mais simples, redes de estações coordenadas por um sistema operacional que possua funcionalidades para processamento paralelo, realizando operações distribuídas de forma paralela. Também são conhecidos informalmente como PoPCs – Pipe Of PCs (Pitanga, 2003).

Utilizando a estrutura paralela descrita acima, pode-se implementar um método de otimização como o Algoritmo Genético, inspirado na teoria de evolução de Darwin, que implementa um código de programação que busca melhores resultados ao longo de gerações.

A combinação de programação paralela com Algoritmos Genéticos num paradigma de arquipélago tende a gerar melhores resultados em um espaço de tempo menor, o que será visto ao final deste trabalho.

1.2 Objetivo

Desenvolver um código de programação para otimização multidimensional utilizando-se de Algoritmos Genéticos e recursos da programação paralela para microcomputadores.

Avaliar a eficiência dos operadores genéticos que fazem parte do algoritmo em questão, medindo a eficácia da alteração de determinadas variáveis e quais são os melhores parâmetros na processo de execução de tal algoritmo.

Com esse código espera-se obter os melhores resultados possíveis utilizando essa estrutura de programação, baseado em trabalhos existentes na área.

O tempo e a credibilidade dos dados são pontos cruciais nesse trabalho, que irá mostrar como é possível trabalhar com essa codificação bio-inspirada, ou seja, um código de programação escrito que faz uma analogia com uma teoria de evolução, no caso a teoria de Darwin.

No que tange ao aspecto paralelo poderá ser observado a eficiência do paralelismo em se tratando da maneira como o código se comporta, ou seja, quando escrito da maneira como a programação distribuída exige, tende a produzir melhoras na velocidade que os resultados são obtidos.

Por outro lado, fatores envolvendo a maneira como as redes de computadores atuam, além a capacidade computacional de cada componente envolvido no processamento distribuído, podem influenciar negativamente, fazendo com que um bom código paralelo perca seu desempenho por conta de fatores externos aos aspectos da programação propriamente dita.

Ao final do trabalho ocorrerá a junção do aspecto paralelo com a questão de algoritmos genéticos, ou seja, o algoritmo será reescrito para o ambiente de programação distribuída, obedecendo limites e regras computacionais adotadas quando aborda-se o aspecto paralelo.

Deverão ser adotadas regras para determinar o final de execução de um algoritmo genético em paralelo, ainda mais quando adotada uma implementação do tipo não sincronizada na qual os nós, máquinas da rede, tendem a terminar em um tempo variante seu processamento.

Por fim, o algoritmo genético deverá ser testado quanto à sua robustez e utilidade em diversos tipos de problemas, mesmo quando sua codificação precisou ser alterada para abordar o problema do caixeiro viajante, adicionando a isso o fato de que o paralelismo, se implementado de forma correta como um todo, pode influenciar positivamente na solução de alguns problemas, atuando como uma ferramenta de auxilio na otimização.

2. ALGORITMOS GENÉTICOS – ASPECTOS CONCEITUAIS

Neste capítulo são abordados aspectos dos chamados Algoritmos Genéticos, os quais possibilitam aplicações nos campos de Otimização, Busca, e Inteligência Artificial. Apresentam-se os conceitos fundamentais, bem como alguma experimentação numérica envolvendo problemas de otimização bidimensional, n = 2. Problemas de maior dimensionalidade, n > 2, são enfocados em capítulos subseqüentes.

2.1 – Definição dos Algoritmos Genéticos

Genericamente, **Algoritmos Genéticos** são procedimentos de busca, em espaços adequados, baseados nos mecanismos da seleção natural (Darwin, 2006) e da genética (Griffihs et al., 2000). Eles combinam a mais provável sobrevivência dos indivíduos mais aptos de uma população com os mecanismos da genética. Nas diferentes variantes em que se pode encontrar, o algoritmo genético é uma metáfora da evolução biológica vertida para o campo da computação. Anteriormente, ao uso dos algoritmos genéticos para solução de problemas em sistemas artificiais, biologistas utilizaram computadores digitais para realizar a simulação de sistemas genéticos (Baricelli, 1957, 1962; Fraser, 1960, 1962; Martin & Cockerham, 1960). Embora estes estudos fossem dirigidos ao entendimento de fenômenos naturais, o trabalho de Fraser(1960, 1962) não era muito distante da noção atual de algoritmo genético (Goldberg & Sastry, 2007). Os resultados de Fraser assemelham-se à otimização de funções, no entanto, não havia em seu trabalho o reconhecimento de que o algoritmo de busca em processos naturais pudesse ser utilizado em processos artificiais. O desenvolvimento dos algoritmos genéticos para sistemas artificiais foi efetuado por Holland (1962a-c) e seus estudantes utilizando operadores semelhantes aos operadores genéticos: reprodução, *crossover* e mutação. Em seu livro *Adaptation in Natural and Artificial Systems*, Holland (1975) apresentou os algoritmos genéticos como uma abstração da evolução biológica e forneceu uma fundamentação teórica para eles (Mitchell, 1999). A primeira menção ao termo ***"algoritmo genético"*** e a primeira aplicação publicada de um algoritmo genético foi feita por Bagley (1967).

Na literatura correlata existem os termos **Algoritmos Evolutivos** e **Algoritmos Genéticos**. É interessante mencionar que todo algoritmo genético é também evolutivo, no entanto, nem todo algoritmo evolutivo é genético. Isto se deve à possibilidade de se criar

algoritmos nos quais não haja a menção ou uso de operadores genéticos, apenas de operadores evolutivos. Este fato tem inclusive fundamentação histórica, pois, o livro de Darwin(2006) sobre evolução não contém aspectos genéticos, uma vez que estes conceitos só apareceram após os trabalhos de Mendel (1865,1866) sobre a genética de ervilhas. O trabalho de Mendel foi posterior ao de Darwin.

Neste capítulo a apresentação e aplicação dos algoritmos genéticos será direcionada à solução de problemas de otimização, os quais podem ser transformados em problemas de busca, definidos sobre espaços-de-busca adequados. Neste contexto, o processo de otimização consiste em buscar a melhor solução para um dado problema, tentando várias soluções e escolhendo a melhor ou as melhores entre todas.

2.2 – Otimização

Os problemas de otimização aparecem em duas categorias principais: **Maximização** ou **Minimização**. A teoria da evolução considera que a probabilidade de sobrevivência em uma população dos indivíduos mais aptos é maior do que a dos menos aptos. Assim, parece mais razoável considerar como problemas para solução, via algoritmo genético, apenas aqueles de maximização. Embora, seja possível, no bojo da metáfora computacional, considerar em problemas de minimização, os indivíduos mais aptos, como sendo os que minimizam o problema de otimização. Mais adiante, nesta seção mostra-se como contornar a questão da minimização, considerando apenas os problemas de maximização.

Seja uma função objetivo unidimensional,

$$f(x) \in \Re, \;\; x \in \mathrm{I}_x = [a,b] \subset \Re \,, \tag{2.1}$$

definida sobre o intervalo I_x, para a qual pretende-se determinar o ponto $x^* \in \mathrm{I}_x$ em que a função $f(x^*)$ é máxima. Adicionalmente, considere-se o fato, verdadeiro, de que $f(x)$ possui no mínimo um ponto de máximo, local ou global, no intervalo de definição. A função $f(x)$ pode ter vários pontos de máximos locais. Por simplicidade, trata-se, inicialmente, de problemas de otimização unidimensionais e, posteriormente, estende-se a metodologia para otimização multidimensional.

Todo problema de minimização pode ser convertido em um problema de maximização, e vice-versa, bastando multiplicar a função objetivo do problema de

minimização por (-1). Assim, de um modo geral, neste trabalho, trata-se "apenas" de problemas de maximização. A minimização fica parametrizada como se fosse maximização. A modalidade de otimização, maximização ou minimização, será então tratada como um parâmetro multiplicador da função objetivo: (+1) para maximização e (-1) para minimização.

2.3 – Indivíduo

Tal como nas populações naturais, as populações abstratas dos algoritmos genéticos possuem indivíduos. Uma dada população é constituída de indivíduos. Cada indivíduo é constituído de um **genótipo** e de um **fenótipo**. O termo genótipo refere-se a um particular conjunto de **genes** contido em um **genoma**. Este conjunto particular de genes é denominado **cromossomo**. Os seres vivos, em geral, possuem vários cromossomos. Em algoritmos genéticos para solução de problemas unidimensionais necessita-se de indivíduos com apenas um cromossomo. Neste trabalho, mesmo para problemas multidimensionais optou-se por utilizar apenas um cromossomo, como se verá mais adiante neste capítulo. Os **genes** são blocos funcionais de informação (em seres vivos é o DNA). Um dado estado (condição existencial) de um gen é chamado de **alelo**. Cada gen está alocado em uma dada posição – **lócus** – no cromossomo. O genótipo propicia ao indivíduo, sob um dado contexto populacional, um conjunto de características (físicas, químicas e mentais em organismos vivos), conhecido como **fenótipo**.

Para problemas multidimensionais (n – dimensões, $n > 1$) pode-se utilizar um cromossomo para cada dimensão, tendo-se então n – cromossomos, ou apenas um cromossomo composto com as informações de todas as n-direções. Neste trabalho adota-se a última possibilidade, assim, todos os indivíduos, para problemas unidimensionais ou multidimensionais, possuem apenas um cromossomo. A diferença está em como o cromossomo é composto. Quando, mais adiante, se realiza a extensão da metodologia de unidimensional para multidimensional, retornar-se-á ao tema.

Um dos paradigmas computacionais modernos é **encapsulamento**. Este mecanismo encapsula determinadas informações em um repositório (*container*) de dados e só permite seu acesso sob determinadas condições. Este mecanismo de acesso restrito ajuda a evitar, parcialmente, a corrupção de dados que tende a existir em maior freqüência quando o tamanho e a complexidade dos sistemas de informação aumentam. Computacionalmente, o aplicativo para o algoritmo genético foi desenvolvido utilizando a linguagem computacional C. No caso da linguagem C o recurso disponível para encapsulamento é a chamada ***struct***. Em

outras linguagens funcionais existem mecanismos semelhantes para produção, pelo usuário, de tipos derivados de dados (UDT), também chamados de tipos de dados abstratos (ADT). Em linguagens orientadas a objetos também é possível, até desejável, o uso do conceito de classes, que além de prover encapsulamento, provê também os mecanismos de herança e polimorfismo. A seguir, Figura 2.1, apresenta-se o código fonte com a definição de indivíduo utilizada no aplicativo destinada à otimização unidimensional, bem como multidimensional.

```
typedef struct individuo
{
        char                cromoss[tam_cromo];
        double              funcao;
        double              aptidao;
        double              soma_aptidao;
        unsigned short int  flagpenalty;
        unsigned short int  flagelite;
        unsigned short int  flagfunc;
} ind;
```

Figura 2.1 - Código fonte da definição de indivíduo - genótipo e fenótipo

O código fonte na Figura 2.1 contém a "receita" para criação de todos os indivíduos da população. O genótipo do indivíduo é representado pelo *array* de caracteres **cromoss** com tamanho igual a **tam_cromo**. Para a variável computacional **tam_cromo** utiliza-se a simbologia n_c, de tal forma que **tam_cromo** $\equiv n_c$. O *array* **cromoss** contém a representação do genótipo do indivíduo. Nos algoritmos genéticos a representação é dependente do problema, e pode ser binária, ternária, decimal, lista encadeada, árvore binária, ou outras formas mais abstratas de representação. No caso de otimização unidimensional ou multidimensional utilizou-se representação binária. Os símbolos binários utilizados foram: 0 ou 1. Ou seja os **alelos** de um dado gen são **0** ou **1**. Para se criar o cromossomo de um dado indivíduo gera-se, mediante o uso de algum algoritmo, um conjunto de alelos em quantidade igual a **tam_cromo**, e atribui-se estes alelos aos respectivos *loci* dos genes correspondentes. O fenótipo do indivíduo é constituído pelo restante das variáveis-membro da *struct*: **funcao**, **aptidao**, **soma_aptidao**, **flagpenalty**, **flagelite**, e **flagfunc**. As variáveis-membro que constituem o fenótipo têm as seguintes definições e/ou utilidades:

- **funcao** – variável do tipo *floating-point* que armazena o valor da função objetivo correspondente ao genótipo do indivíduo;

- **aptidao** – variável do tipo *floating-point* que armazena a aptidão do indivíduo. A aptidão do indivíduo será discutida mais detalhadamente na seção sobre **seleção** para reprodução, mas, resumidamente, representa a **posição** do indivíduo na **classificação** dos indivíduos da população. Indivíduos mais aptos possuem melhor posição, indivíduos menos aptos, pior posição, e assim por diante;
- **soma_aptidao** – é a soma acumulada das aptidões de todos os indivíduos que precedem, numa dada ordem, um dado indivíduo, inclusive. Em geral é utilizada como informação no processo decisório sobre quais indivíduos irão produzir descendentes e/ou quais indivíduos serão removidos da população;
- **flagpenalty** – é uma variável "do tipo lógico" que sinaliza se um dado indivíduo está marcado ou não para ser eliminado, baseado em algum critério lógico, da população. Pode ser utilizado para auxiliar no processo decisório de remover indivíduos inviáveis ou indesejáveis, da população, ou mesmo abrir espaço para indivíduos da nova geração;
- **flagelite** – é uma variável do "tipo lógico" que informa se um dado indivíduo pertence ou não à elite da população. O elitismo será tratado em detalhes mais adiante, neste capítulo. Quando utilizada serve, em geral, para impedir que indivíduos, enquanto, de elite sejam descartados da população;
- **flagfunc** - é uma variável do "tipo lógico" que informa se a função objetivo para um dado indivíduo já foi ou não calculada. Serve para evitar o cômputo, repetidas vezes, da função objetivo para indivíduos cujo genótipo já foi avaliado. Também serve para indicar a necessidade do cômputo da função objetivo para os indivíduos cujo genótipo ainda não foi avaliado.

2.4 População

Neste trabalho considerou-se a existência, em algum momento do processo evolutivo, de apenas uma população em um dado processo computacional. Seria possível a criação de subpopulações utilizando o conceito computacional de *threading* dentro de um mesmo processo. Em capítulos posteriores apresenta-se o processamento distribuído do algoritmo genético utilizando vários processos computacionais sendo executados em vários nós processadores. Considerou-se também que o tamanho da população é constante e igual a n_p que corresponde à variável computacional **tam_pop**, ou seja $n_p \equiv$ **tam_pop**. Seria possível a utilização de população com tamanho variável. Devido ao fato de haver apenas uma

população não há recobrimento de populações, embora, parte dos indivíduos da população anterior permaneçam na população posterior, como se verá em detalhes quando da apresentação dos operadores genéticos.

2.4.1 População inicial

Conforme conceituado anteriormente os algoritmos genéticos tratam da evolução de indivíduos em populações. Então para haver sucessão de populações é necessário haver uma população inicial, Figura 2.2.

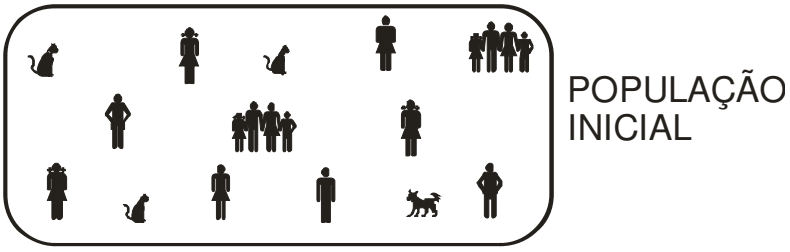


Figura 2.2 - Representação simbólica de uma população inicial.

Utilizou-se dois tipos de algoritmos para geração da população inicial: **aleatório** e **semi-aleatório.** No algoritmo **aleatório** todos os indivíduos da população inicial são gerados aleatoriamente, significando que todos os alelos de todos os genes de todos os cromossomos da população inicial são gerados aleatoriamente. No algoritmo **semi-aleatório** parte dos indivíduos são gerados aleatoriamente, como mencionado na frase anterior, posteriormente, o restante dos indivíduos faltantes para completar a população inicial são gerados utilizando algum mecanismo não aleatório.

Embora exista na natureza processos aleatórios, computacionalmente, em geral, os processos aleatórios são em realidade processos pseudo-aleatórios (Alves, 2005). Neste trabalho os alelos binários foram gerados utilizando-se funcionalidades disponíveis na linguagem computacional C. Metaforicamente, o algoritmo corresponde ao giro de uma roleta, Figura 2.3, seguidas vezes e a produção de um fluxo de alelos que são utilizados para preencher, seqüencialmente, os *loci* nos cromossomos dos indivíduos. As probabilidades de a roleta parar no **0** ou no **1** são iguais.

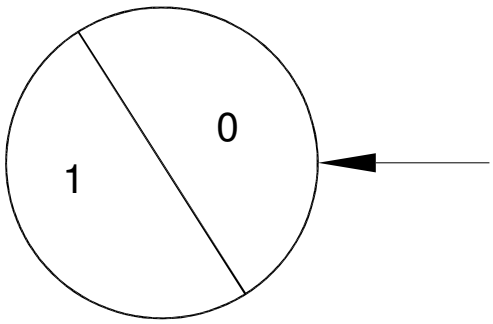


Figura 2.3 - Roleta para geração de alelos binários: 0 ou 1.

Assim, para gerar um indivíduo com 16 genes, aciona-se a roleta 16 vezes, produzindo um indivíduo, por exemplo, como mostrado na Tabela 2.1.

Tabela 2.1 - Cromossomo de indivíduo com 16 alelos.

Indivíduo	Genótipo															
1	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**

Então, o genótipo de um indivíduo, x_g, pode ser representado por um *array* de genes da seguinte forma

$$x_g \equiv [g_1, g_2, ..., g_{n_g-1}, g_{n_g}], \quad g_k \in \{0,1\}, k = 1,...,n_g ; \tag{2.2}$$

na qual os *g*'s são os genes do indivíduo, que podem assumir os valores {**0**} ou {**1**} que são os alelos.

Quando se trata de otimização unidimensional, $x \in \Re^1$, todos os genes do indivíduo são utilizados para armazenar os alelos correspondentes ao valor de *x*. Porém quando a função objetivo é definida sobre domínio multidimensional,

$$f(\mathbf{x}) \in \Re^n, \quad n = 2,3,4..., \quad \mathbf{x} \equiv (x_1, x_2, ..., x_n) \in \mathrm{I}_\mathbf{x} \equiv \{[a_1, b_1], [a_2, b_2], ..., [a_n, b_n]\} \subset \Re^n , \tag{2.3}$$

para o qual pretende-se determinar o ponto $\mathbf{x}^* \in \mathrm{I}_\mathbf{x}$ em que a função $f(\mathbf{x}^*)$ é máxima, então os n_g alelos serão utilizados para conter as informações sobre os valores de x_1, x_2,..., x_n. Portanto, n_g deve ser igual ao número de variáveis, *n*, do espaço de definição do problema de

otimização multidimensional, multiplicada pelo número de genes que armazenará as informações de cada variável x_k, $k = 1, 2,..., n$. Denominando o número de genes destinados a representar cada variável por n_{gv}, então tem-se a seguinte relação

$$n_g \equiv n \times n_{gv}. \tag{2.4}$$

Assim, o genótipo de um indivíduo, $\boldsymbol{x}_g \in \Re^n$, pode ser representado por um *array* de genes da seguinte forma

$$\mathbf{x}_g \equiv [g_1, g_2, ..., g_{n_g-1}, g_{n_g}], \quad g_k \in \{0,1\}, k = 1,...,n_g; \quad n_g \equiv n \times n_{gv}; \tag{2.5}$$

na qual cada variável x_k, $k = 1, 2,..., n$ é expressa, seqüencialmente, em n_{gv} genes.

Conforme descrito, pode-se gerar, aleatoriamente, quantidade finita de indivíduos formando a população inicial **aleatória**. Na Tabela 2.2 apresenta-se uma população inicial com 10 indivíduos gerados aleatoriamente. Portanto, toda a população inicial é gerada aleatoriamente. A geração aleatória de um indivíduo é usada para criar a população inicial, mas pode ser usada também para suprir algum indivíduo faltante na população, uma vez que o tamanho da população é constante, neste trabalho. A ocorrência de indivíduos faltantes na população pode ocorrer quando algum indivíduo sofre penalidade e é removido da população. No próximo capítulo trata-se em detalhes a penalização de indivíduos.

Assim, ao final deste processo aleatório tem-se população inicial, $\boldsymbol{X}_{g,0}$, como segue

$$\mathbf{X}_{g,0} \equiv [\mathbf{x}_{g,1}, \mathbf{x}_{g,2}, \mathbf{x}_{g,3}, ..., \mathbf{x}_{g,n_p}]^T. \tag{2.6}$$

Tabela 2.2 – Cromossomos de indivíduos, com 16 alelos: geração aleatória.

Indivíduo	Genótipos															
1	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
2	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**
3	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
4	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**1**
5	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**
6	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**1**
7	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**0**
8	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
9	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**
10	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**

No caso da geração **semi-aleatória** da população inicial gera-se, aleatoriamente, indivíduos em quantidade igual à metade da população. Se o tamanho da população for impar o resultado da divisão da população por dois é arredondado para cima. O restante da população é gerada através de algoritmo não aleatório. Neste trabalho utilizou-se tomar a primeira metade dos genes do cromossomo do **Indivíduo 1** combinado com a segunda metade dos genes do cromossomo do **Indivíduo 2** para produzir o primeiro dos indivíduos com geração **não aleatória**. E assim, sucessivamente, até gerar-se o penúltimo dos indivíduos gerados não aleatoriamente. Na geração do último indivíduo, não aleatoriamente, toma-se a primeira parte do último indivíduo gerado, aleatoriamente, combinado com a segunda parte do primeiro indivíduo gerado aleatoriamente. Na Tabela 2.3 apresenta-se uma população de dez indivíduos gerados usando o algoritmo **semi-aleatório**. Conforme descrito anteriormente, os cinco primeiros indivíduos foram gerados aleatoriamente. O sexto indivíduo, o primeiro dos gerados não aleatoriamente, está constituído pela primeira parte do **Indivíduo 1** e pela segunda parte do **Indivíduo 2**. No caso, os respectivos alelos estão sublinhados. E assim, sucessivamente, até gerar o **Indivíduo 9**. Para a geração do **indivíduo 10** tomou-se a primeira parte do **Indivíduo 5** com a segunda parte do **Indivíduo 1**. Neste caso, os alelos estão com sublinhado duplo.

Tabela 2.3 – Cromossomos de indivíduos, com 16 alelos: geração semi aleatória.

Indivíduo	Genótipos															
1	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
2	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**
3	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
4	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**1**
5	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**
6	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**
7	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**1**	**1**	**0**	**1**
8	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**1**
9	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**
10	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**

Uma técnica denominada *seeding* (Santana, 2004) consiste em colocar, na população inicial, soluções encontradas através de outros métodos de otimização. Isto, juntamente com a preservação dos melhores indivíduos geração após geração, garantiria que a solução gerada pelo algoritmo genético não seria pior que as soluções geradas por aqueles métodos. Esta técnica não foi usada neste trabalho.

2.4.2 População final

Após a geração da população inicial, esta população sofre operações seqüenciais, geração após geração, e ao final de um determinado número de gerações total, n_G, o processo evolutivo é parado, e tem-se então a população final. Metaforicamente, apresentada na Figura 2.4. O critério de parada pode ser o número total de gerações ou algum outro mecanismo de parada, baseado em indicadores populacionais e/ou individuais.

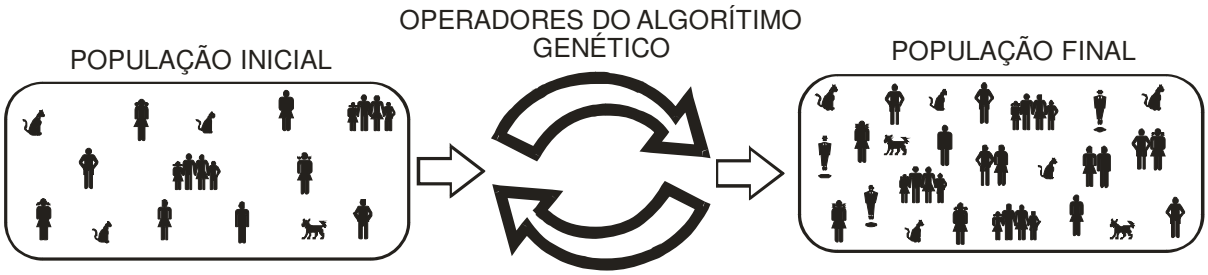


Figura 2.4 – Representação simbólica de uma população final.

2.5 Operadores fundamentais do Algoritmo Genético

Os operadores fundamentais do algoritmo genético que transformam uma população inicial, geração após geração, em uma população final, Figuras 2.4 e 2.5, são:

- **Seleção;**
- ***Crossover*;**
- **Substituição;**
- **Mutação.**

Alguns autores, tal como Goldberg (1989), utiliza a palavra **reprodução** (***reproduction***) ao invés da palavra **seleção**. Entende-se neste trabalho que a palavra **seleção** é mais adequada, pois, trata-se, efetivamente de **seleção** para a reprodução. Na metáfora computacional a reprodução em si é realizada no ***crossover***. Assim, em Goldberg **reprodução** significa: **seleção para a reprodução**; e neste trabalho **seleção** significa: **seleção para a reprodução**. Ou seja, o mesmo significado, porém utilizando significantes diferentes. Goldberg (1989) considera apenas três operadores: reprodução, *crossover* e mutação. Mitchell(1999) utiliza também três operadores, porém os denomina: **seleção**, ***crossover*** e **mutação**. Neste trabalho preferiu-se utilizar os quatro operadores (Alves, 2005), mencionados

anteriormente. Entende-se que a opção feita aqui é computacionalmente de granulometria mais fina e permite tratamento mais amplo do desenvolvimento, codificação e aplicação do algoritmo genético. Naturalmente, no modelamento apresentado em Goldberg e Sastry (2007) e Mitchell(1999), com apenas três operadores, o operador **substituição** também está presente, porém imerso nas três funcionalidades usadas.

Após a geração da população inicial, os quatros operadores fundamentais do algoritmo genético são aplicados, seqüencialmente, e sucessivamente, geração após geração: **seleção, crossover, substituição** e **mutação**. Após cada aplicação completa dos quatro operadores produz-se uma nova geração com n_p indivíduos.

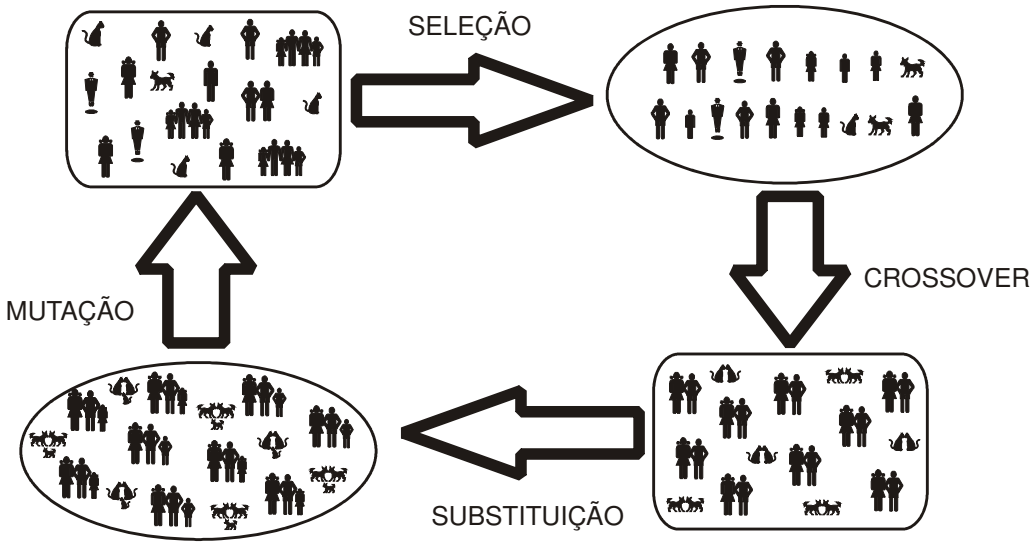


Figura 2.5 - Representação simbólica da aplicação dos operadores genéticos fundamentais sobre uma dada população.

Dos quatro operadores, dois deles, ***crossover*** e **mutação,** operam diretamente sobre o genótipo do indivíduo, ou seja, cromossomo, genes e alelos, e destinam-se a produzir recombinação genética, no caso do *crossover* e alteração genética no caso da mutação. Os outros dois operadores atuam sobre o fenótipo do indivíduo e destinam-se a responder duas questões fundamentais: quais indivíduos da população serão selecionados para reprodução, no caso do operador **seleção**; e quais indivíduos da população serão descartados para abrir espaço para os filhos, no caso do operador substituição. Adicionalmente, menciona-se que os operadores ***crossover*** e **mutação** têm ação direta sobre o genótipo dos indivíduos, e os operadores **seleção** e **substituição** exercem ação indireta sobre o genótipo dos indivíduos.

Na Figura 2.6 apresenta-se parte do código fonte, em linguagem C, o qual realiza a evolução da população através dos operadores do algoritmo genético: seleção, *crossover*, substituição e mutação. A geração da população inicial é realizada apenas uma vez, e, portanto, não aparece na codificação mostrada. Também estão omitidas funcionalidades de entrada e saída de dados, que embora úteis, não influenciam as operações fundamentais. Neste caso o código mostrado parte da população inicial e a evolui para a geração **1**, e assim sucessivamente até produzir a quantidade de gerações sucessoras iguais a **geracoes**. Neste caso o **geracoes** computacional corresponde ao n_G textual, isto é **geracoes** ≡ n_G.

```
for (cg = UM; cg <= geracoes; cg++)
    {
      selecao();              // Seleciona indivíduos para reprodução
      crossover();            // Realiza crossover dos indivíduos selecionados
      substituicao();         // Substitui parte da população pelos filhos
      mutacao();              // Realiza mutação nos genes da população
    }
```

Figura 2.6 – Código fonte da evolução da população, utilizando os operadores: seleção, crossover, substituição e mutação.

2.5.1 – Seleção

O mecanismo de **seleção**, baseado em características fenotípicas dos indivíduos, destina-se a selecionar em uma população quais são os indivíduos que poderão gerar descendentes através de *crossover*. Em resumo: quais serão os permitidos a reproduzir. Pelo princípio da seleção natural é mais provável que os indivíduos selecionados sejam os mais aptos.

O resultado final do processo de seleção é uma lista contendo os indivíduos selecionados para participar da fase de reprodução. Nem todos os indivíduos da população poderão se reproduzir. Assim, o conjunto, hipoteticamente não-vazio, dos indivíduos selecionados deve ser menor que a população, portanto, a intersecção dos dois conjuntos deve produzir um conjunto também não-vazio. A quantidade n_s de indivíduos selecionados deverá ser um valor inteiro positivo e par. *Inteiro* porque os indivíduos pertencem ao conjunto das entidades contáveis e, portanto são mensurados utilizando-se valores inteiros positivos. *Par* porque o processo de acasalamento para o *crossover* é feito aos pares. Por hipótese, no conjunto dos indivíduos selecionados não deve haver indivíduos repetidos, ou seja, clones.

Em geral, conforme mencionado anteriormente, o conjunto dos indivíduos selecionados é não-vazio, no entanto demonstra-se mais adiante neste trabalho que é possível utilizar taxa de seleção nula e ainda obter bons resultados, sob certas circunstâncias. Para que haja seleção é importante que se evite taxas de seleção próximas da unidade o que significaria que a maioria dos indivíduos seria selecionada para a reprodução, e, portanto, não haveria seleção. Em outras palavras, para haver seleção é necessário que parte significativa da população não produza descendentes.

Conhecidos o tamanho da população, n_p, e a quantidade de indivíduos selecionados, n_s, é possível calcular a **taxa de seleção**, que é a fração entre a quantidade de indivíduos selecionados pela quantidade de indivíduos na população, como segue

$$\sigma \equiv \frac{n_s}{n_p}; \quad n_s, n_p \in \Im; \quad 0 \le n_s \le n_p; n_p > 0; \quad \sigma \in \Re; \quad 0 \le \sigma \le 1. \tag{2.7}$$

Caso se conheça o tamanho da população, n_p, e a taxa de seleção, σ, pode-se calcular a quantidade de indivíduos a selecionar, n_s, da seguinte maneira

$$n_s \equiv \text{int}(\sigma \times n_p); \quad n_s, n_p \in \Im; \quad 0 \le n_s \le n_p; \quad n_p > 0; \quad \sigma \in \Re; \quad 0 \le \sigma \le 1; \tag{2.8}$$

na qual n_s deve ser par. Na equação (2.8) **int(...)** é a função que arredonda o argumento para o número inteiro e par mais próximo, no intervalo de validade.

Conversão binário-decimal

Conforme mencionado anteriormente o genótipo dos indivíduos pertence ao conjunto dos números binários e o argumento da função objetivo pertence ao conjunto dos números reais. Assim, faz-se necessária a conversão de números binários para números decimais reais. Esta transformação faz-se em duas partes: na primeira converte-se números binários para decimais inteiros, e na segunda decimais inteiros para decimais reais.

Para um indivíduo com representação binária, conforme a equação (2.2), pode ser associado, biunivocamente, um número inteiro decimal, $x_N \in \Im$, mediante o equacionamento a seguir

$$x_g \equiv (g_1, g_2, ..., g_{n_{gv}-1}, g_{n_{gv}}), \quad g_k \in \{0,1\}, k = 1,...,n_{gv} \quad \Rightarrow$$
$$\Rightarrow x_N = g_1 \times 2^{n_{gv}-1} + g_2 \times 2^{n_{gv}-2} + ... + g_{n_{gv}-1} \times 2^1 + g_{n_{gv}} \times 2^0. \quad (2.9)$$

Por exemplo, os dois primeiros indivíduos mostrados na Tabela 2.4 são de uma variável, $n = 1$, de 16 alelos, $n_{gv} = 16$, portanto, $n_g = 16$. O valor inteiro, x_N, correspondente aos dois primeiros indivíduos, segundo a equação (2.2), são

$$x_N = 0\times 2^{15} + 0\times 2^{14} + 0\times 2^{13} + 0\times 2^{12} + 0\times 2^{11} + 0\times 2^{10} + 0\times 2^9 + 0\times 2^8 + \\ + 0\times 2^7 + 0\times 2^6 + 0\times 2^5 + 0\times 2^4 + 0\times 2^3 + 0\times 2^2 + 0\times 2^1 + 0\times 2^0 = 0, \quad (2.10)$$

$$x_N = 1\times 2^{15} + 1\times 2^{14} + 1\times 2^{13} + 1\times 2^{12} + 1\times 2^{11} + 1\times 2^{10} + 1\times 2^9 + 1\times 2^8 + \\ + 1\times 2^7 + 1\times 2^6 + 1\times 2^5 + 1\times 2^4 + 1\times 2^3 + 1\times 2^2 + 1\times 2^1 + 1\times 2^0 = \\ = 32768+16384+8192+4096+2048+1024+512+256+128+64+32+16+8+4+2+1 = \\ = 65535 = 2^{16} - 1. \quad (2.11)$$

Tabela 2.4 – Indivíduos com um cromossomo de 16 alelos.

Indivíduo	Genótipo															
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
3	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**

Portanto, os indivíduos quando representados utilizando base binária estão no intervalo binário, $I_{x,g}$, definido a seguir

$$I_{x,g} \equiv [a_{x,g}; b_{x,g}] \equiv [(\underbrace{000\ldots000}_{n_{gv}vezes}); (\underbrace{111\ldots111}_{n_{gv}vezes})] \subset B^1. \quad (2.12)$$

Na equação (2.12), B^1 representa o conjunto dos números binários.

O intervalo $I_{x,g}$ quando convertido para a base decimal se transformará em um intervalo, $I_{x,N}$, contido no conjunto dos números inteiros, como segue

$$I_{x,g} \equiv [a_{x,g}; b_{x,g}] \equiv [(\underbrace{000\ldots000}_{n_{gv}vezes}), (\underbrace{111\ldots111}_{n_{gv}vezes})] \subset B^1 \Leftrightarrow I_{x,N} \equiv [a_{x,N}; b_{x,N}] \equiv [0, 2^{n_{gv}} - 1] \subset N^1. \quad (2.13)$$

Para o caso de n_{gv} = 16 a equação (2.13) torna-se

$$I_{x,g} \equiv [a_{x,g}; b_{x,g}] \equiv [(0000000000000000), (1111111111111111)] \subset B^1$$
$$\Updownarrow \tag{2.14}$$
$$I_{x,N} \equiv [a_{x,N}; b_{x,N}] \equiv [0, 65535] \subset N^1.$$

Em conseqüência, o terceiro indivíduo da Tabela 2.4, que em realidade é o indivíduo da Tabela 2.1, quando convertido para inteiro decimal corresponde ao número 57956 ⊂ [0, 65535].

Na Tabela 2.5 apresenta-se três indivíduos com o mesmo genótipo de 16 alelos dos indivíduos da Tabela 2.4, porém representando cada um deles duas variáveis, x_1 e x_2. Portanto, os primeiros 8 alelos representam a variável x_1 e os últimos 8 alelos representam a variável x_2.

Tabela 2.5 – Indivíduos de 16 alelos, com duas variáveis.

Indivíduo	Genótipo															
1	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**
3	**1**	**1**	**1**	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**

Para este caso o intervalo de representação, binário e inteiro decimal, para as variáveis x_1 e x_2 são:

$$I_{x_1,g} \equiv [a_{x_1,g}; b_{x_1,g}] \equiv [(00000000), (11111111)] \subset B^1 \Leftrightarrow I_{x_1,N} \equiv [a_{x_1,N}; b_{x_1,N}] \equiv [0, 255] \subset N^1, \tag{2.15}$$

e

$$I_{x_2,g} \equiv [a_{x_2,g}; b_{x_2,g}] \equiv [(00000000), (11111111)] \subset B^1 \Leftrightarrow I_{x_2,N} \equiv [a_{x_2,N}; b_{x_2,N}] \equiv [0, 255] \subset N^1. \tag{2.16}$$

Então, para o terceiro indivíduo da Tabela 2.5 os primeiros 8 alelos (11100010) são a representação binária da variável x_1 vertida para inteiro decimal $x_{1,N} = 226 \subset [0, 255]$; e os segundos oito alelos (01100100) são a representação da variável x_2 em inteiro decimal $x_{2,N} = 100 \subset [0, 255]$.

Os indivíduos da Tabela 2.5 poderiam ainda estar representando quatro variáveis com 4 alelos cada. Neste caso, o intervalo de variação de cada variável seria [0, 15]. Claramente, quando a quantidade de alelos por variável diminui a extensão do intervalo de variação inteiro decimal cai rapidamente. Vice-versa, quando a quantidade de alelos aumenta, a extensão do intervalo de definição inteiro decimal aumenta também rapidamente. Na Tabela 2.6 apresenta-se os intervalos inteiro decimal, para uma variável, em função do número de alelos.

Trata-se agora da transformação das variáveis representadas em inteiro decimal, $x_{i,N}; i = 1,2,...,n$ para as variáveis representadas no conjunto dos reais decimais, $x_i; i = 1,2,...,n$.

Conforme estabelecido na equação (2.3) cada variável real está definida em um dado intervalo, a seguir

$$x_i \in I_{x_i} \equiv [a_i, b_i], \quad i = 1,2,...,n, \tag{2.17}$$

por outro lado, as variáveis quando representadas em números inteiros decimais estão definidas da seguinte forma

$$x_{i,N} \in I_{x_{i,N}} \equiv [a_{i,N}, b_{i,N}] \equiv [0, 2^{n_{gv}} - 1], \quad i = 1,2,...,n. \tag{2.18}$$

A maneira mais simples e efetiva de transformar uma variável com representação inteiro decimal $x_{i,N} \in [0, 2^{n_{gv}} - 1], \ i \in [1,n] \subset N$ para a representação real $x_i \in [a_i, b_i], \ i \in [1,n] \subset N$ é através de transformação linear única e inversível, mostrada a seguir

$$x_i = a_i + (x_{i,N} - a_{i,N}) \frac{b_i - a_i}{b_{i,N} - a_{i,N}} = a_i + \frac{b_i - a_i}{2^{n_{gv}} - 1} x_{i,N}; \quad i = 1,2,..,n. \tag{2.19}$$

As variáveis x_1 e x_2 apresentadas nas Figuras 2.4 a 2.8 foram convertidas da representação binária para a representação real decimal utilizando as equações (2.9) e (2.19), nesta ordem.

Conversão decimal-binário

Embora, em otimização multidimensional, a conversão binário-decimal seja a mais utilizada, também se faz necessário às vezes fazer conversão decimal-binário. Para converter de um valor real decimal para inteiro decimal, pode-se reestruturar a equação (2.19), como segue

$$x_{i,N} = \text{int}[\frac{x_i - a_i}{b_i - a_i}(2^{n_{gv}} - 1)]; \quad i = 1,2,..,n\,, \tag{2.20}$$

na qual **int(...)** é função que arredonda seu argumento real para o inteiro mais próximo.

Para converter um valor inteiro decimal para a respectiva representação binária, basta dividir o número por 2 e os quocientes sucessivas vezes, também por 2, e construir o número binário correspondente combinando os restos de **"0"** e **"1"** das respectivas divisões, dos últimos para os primeiros, agrupados da esquerda para a direita.

Seja por exemplo, a conversão do número primo inteiro decimal **"13"** para a representação binária

$$\begin{aligned} &\frac{13}{2} = 6 \Rightarrow \text{Resto} = 1, \\ &\frac{6}{2} = 3 \Rightarrow \text{Resto} = 0, \\ &\frac{3}{2} = 1 \Rightarrow \text{Resto} = 1, \\ &\frac{1}{2} = 0 \Rightarrow \text{Resto} = 1, \end{aligned} \tag{2.21}$$

portanto, a representação binária de **13** é **1101**. De fato ao se aplicar a equação (2.9) sobre o número binário **1101** obtém-se inteiro decimal **13**.

Precisão da representação binária

O menor número binário não nulo, para qualquer quantidade de alelos, que pode ser representado é (000...0001) que corresponde ao inteiro decimal $2^0 = 1$. Assim, a precisão absoluta, $\varepsilon^a_{x_{i,N}}$, em inteiros decimais é constante e igual a uma unidade,

$$\varepsilon^{a}_{x_{i,N}} = 1, \tag{2.22}$$

e a precisão relativa, $\varepsilon^{r}_{x_{i,N}}$, é igual a

$$\varepsilon^{r}_{x_{i,N}} \equiv \frac{1}{2^{n_{gv}} - 1}; \quad i = 1,2,...,n; \quad n_{gv} \geq 1 \tag{2.23}$$

Valores típicos para a precisão relativa, associada ao intervalo inteiro decimal, podem ser vistos na Tabela 2.6.

Tabela 2.6 – Intervalo inteiro decimal e acurácia relativa para diferentes quantidades de alelos.

n_{gv}	Intervalo	Acurácia relativa
8	255	3,9215686274E-03
16	65535	1,5259021896E-05
32	4294967295	2,3283064370E-10
64	18446744073709551615	5,4210108624E-20

Para a transformação linear das variáveis com representação em inteiros para as respectivas representações em reais, também haverá relação linear entre a acurácia para a representação em inteiros e a acurácia para representação em reais. Assim, tem-se a acurácia absoluta em reais para as variáveis do espaço de busca

$$\frac{\varepsilon^{a}_{i}}{|b_i - a_i|} \equiv \frac{\varepsilon^{a}_{i,N}}{2^{n_{gv}} - 1} \Leftrightarrow \varepsilon^{a}_{i} = \frac{|b_i - a_i|}{2^{n_{gv}} - 1}; \quad i = 1,2,...,n. \tag{2.24}$$

A correspondente acurácia relativa é obtida da seguinte relação

$$\varepsilon^{r}_{i} \equiv \frac{\varepsilon^{a}_{i}}{|b_i - a_i|} = \frac{1}{2^{n_{gv}} - 1}; \quad i = 1,2,...,n, \tag{2.25}$$

a qual é igual à acurácia relativa da representação em inteiros decimais.

Pode-se definir os vetores, absoluto e relativo, de acurácia global da seguinte forma

$$\boldsymbol{\varepsilon}_a \equiv [\varepsilon_1^a, \varepsilon_2^a, ..., \varepsilon_n^a]^T \quad \text{e} \quad \boldsymbol{\varepsilon}_r \equiv [\varepsilon_1^r, \varepsilon_2^r, ..., \varepsilon_n^r]^T . \tag{2.26a,b}$$

Para mensurar as precisões globais, absoluta e relativa, pode-se utilizar, por exemplo, a norma L_2, mostrada a seguir

$$\varepsilon_a \equiv \left\| \boldsymbol{\varepsilon}_a \right\|_2 \equiv \sqrt{(\boldsymbol{\varepsilon}^a)^T \bullet \boldsymbol{\varepsilon}^a} = \sqrt{\sum_{i=1}^{n} (\varepsilon_i^a)^2} \quad \text{e} \quad \varepsilon_r \equiv \left\| \boldsymbol{\varepsilon}_r \right\|_2 \equiv \sqrt{(\boldsymbol{\varepsilon}^r)^T \bullet \boldsymbol{\varepsilon}^r} = \sqrt{\sum_{i=1}^{n} (\varepsilon_i^r)^2} . \tag{2.27a,b}$$

O símbolo (•) que aparece nas equações (2.27a,b) é o operador vetorial produto interno também conhecido como produto escalar.

A norma L_2 foi a utilizada neste trabalho, mas pode-se utilizar qualquer norma que faça sentido para o espaço de busca da definição do problema de otimização multidimensional. Outra norma bastante utilizada é a L_∞. Neste caso, as equações (2.27a,b) tomam o seguinte aspecto

$$\varepsilon_a \equiv \left\| \boldsymbol{\varepsilon}_a \right\|_\infty \equiv \underset{i=1,2,...,n}{máx} \{ | \varepsilon_i^a | \} \quad \text{e} \quad \varepsilon_r \equiv \left\| \boldsymbol{\varepsilon}_r \right\|_\infty \equiv \underset{i=1,2,...,n}{máx} \{ | \varepsilon_i^r | \} . \tag{2.28a,b}$$

Avaliação

No item anterior mostrou-se como transformar a representação do genoma de um indivíduo, definido no espaço B^n, para o espaço dos inteiros decimais, N^n, utilizando a equação (2.5), e por último a transformação de inteiros decimais para o campo dos números reais, R^n, usando a equação (2.15). Com esta metodologia obtém-se uma segunda representação para os indivíduos agora no espaço dos números reais. Esta transformação é necessária, pois, para realizar a **avaliação** da função objetivo o argumento da função deve pertencer ao campo dos números *floating-point,* os quais são a representação, computacional, dos números reais. Haveria uma segunda possibilidade na qual a função objetivo seria capaz de operar com argumentos pertencentes ao campo dos binários.

Então uma dada população com genótipo representado em binários pode e deve ser transformada para a representação em números reais, como mostrado, simbolicamente, a seguir

$$\mathbf{X}_g \equiv [\mathbf{x}_{g,1}, \mathbf{x}_{g,2}, \mathbf{x}_{g,3}, ..., \mathbf{x}_{g,n_p}]^T \xrightarrow{T_{B \to N}} \mathbf{X}_N \equiv [\mathbf{x}_{N,1}, \mathbf{x}_{N,2}, \mathbf{x}_{N,3}, ..., \mathbf{x}_{N,n_p}]^T \tag{2.29}$$

e

$$\mathbf{X}_N \equiv [\mathbf{x}_{N,1}, \mathbf{x}_{N,2}, \mathbf{x}_{N,3}, ..., \mathbf{x}_{N,n_p}]^T \xrightarrow{T_{N\to\Re}} \mathbf{X}_\Re \equiv [\mathbf{x}_{\Re,1}, \mathbf{x}_{\Re,2}, \mathbf{x}_{\Re,3}, ..., \mathbf{x}_{\Re,n_p}]^T . \quad (2.30)$$

Um dado $\mathbf{x}_{g,1}$ poderá ser representado também em inteiro decimal, $\mathbf{x}_{N,1}$, e também em números reais, $\mathbf{x}_{\Re,1}$. Para efeito de avaliação e mesmo de aptidão um dado indivíduo será então representado por $\mathbf{x} \in \Re^n$.

Portanto, **avaliação** é a ação de atribuir um valor imagem, $f(\mathbf{x})$, a cada indivíduo, $\mathbf{x}$, da população, isto é, o cômputo da função objetivo para um dado indivíduo

$$f : \mathbf{x} \to f(\mathbf{x}), \quad \mathbf{x} \in \Re^n, \quad f(\mathbf{x}) \in \Re . \quad (2.31)$$

Como $\mathbf{x}$ é a representação em números reais do genótipo de um indivíduo e a função objetivo é a expressão deste genótipo, então a função $f(\mathbf{x})$ é o fenótipo do indivíduo ou no mínimo é parte dele. Na definição da estrutura de dados para os indivíduos, a variável membro **funcao** representa o valor avaliado da função objetivo. Quando o valor de **funcao**, para um determinado indivíduo, é atribuído, logo em seguida a variável membro **flagfunc** é alterado de "**false**" para "**true**". Como **flagfunc** foi definida do tipo inteiro então **false** corresponde a **zero** e **true** corresponde a **um**. Conforme estabelecido na definição da estrutura de dado dos indivíduos, o fenótipo é constituído de algumas variáveis membro entre elas **funcao**, **flagfunc** e **aptidao**. Assim, sempre que o sub-operador **avaliação** pertencente ao operador **seleção** é ativado ele analisa a variável membro **flagfunc** de todos os indivíduos da população e efetua a **avaliação** para todos os indivíduos que ainda não haviam sido avaliados. Os que já haviam sido avaliados não são re-avaliados, desde que seu material genético não tenha sido alterado. Caso o material genético do indivíduo tenha sido alterado por algum tipo de ação então a variável membro já teria sido modificada para **false**. O uso das demais variáveis membro, constituintes do fenótipo, tais como **flagelite** e **flagpenalty**, será estabelecido nas próximas seções e capítulos.

Adicionalmente, há um procedimento para considerar o modo de otimização, se maximização ou minimização.

Devido ao princípio evolutivo da mais provável reprodução dos indivíduos mais aptos optou-se neste trabalho por tratar "apenas" do modo maximização na otimização. Sendo que o modo minimização ficou parametrizado. Para tal definiu-se um parâmetro, ζ_o, de modo de otimização da seguinte forma:

Se o modo de otimização é maximização, então $\zeta_o = +1$;

Se o modo de otimização é minimização, então $\zeta_o = -1$.

Os procedimentos basicamente consistem em produzir uma transformação multiplicativa nos valores da avaliação do indivíduo através da seguinte operação

$$\hat{f} : \hat{f}(\mathbf{x}) = \zeta_o f(\mathbf{x}), \quad \mathbf{x} \in \Re^n, \quad f(\mathbf{x}), \hat{f}(\mathbf{x}) \in \Re. \tag{2.32}$$

Quando o modo de otimização é maximização a transformação (2.32) da função objetivo produz uma identidade, isto é $\hat{f}(\mathbf{x}) = f(\mathbf{x})$, deixando inalterada os valores da função objetivo. No caso do modo de otimização ser igual à minimização, então a transformação (2.32) torna-se $\hat{f}(\mathbf{x}) = -f(\mathbf{x})$, invertendo os valores da função objetivo e "transformando" a otimização de minimização para maximização. Desta forma, os dois modos, minimização e maximização, podem ser tratados apenas como maximização.

Aptidão

Aptidão é uma ação ou regra que associa a cada indivíduo a sua posição relativa em uma população. A aptidão de um indivíduo depende de seu fenótipo e do fenótipo dos outros indivíduos da população. A influência do ambiente está implícita na avaliação do indivíduo através da função objetivo. Fica claro que a aptidão de um indivíduo é um valor coletivo que depende da população e do ambiente ao qual pertence.

Existem inúmeras maneiras de definir regras para a aptidão. Neste trabalho considera-se três regras básicas: **translação**, **escalonamento** e **potenciação**.

Como a função objetivo, $\hat{f}(\mathbf{x}) \in \Re$, é genérica e pode assumir variados valores, positivos, negativos ou nulos, então realiza-se sobre todos os indivíduos uma **translação** coletiva nos valores da avaliação, produzindo a primeira forma de aptidão, $\tilde{f}(\mathbf{x})$. Esta forma visa atribuir aptidão nula ao indivíduo menos apto da população, no caso aquele com menor

avaliação, e aptidões com valores positivos para todos os demais indivíduos da população. A **translação** processa-se como mostrada a seguir

$$\tilde{f}_k(\mathbf{x}_k) \equiv \hat{f}_k(\mathbf{x}_k) - \hat{f}_{\min}; \quad k = 1,2,...,n_p; \quad \hat{f}_{\min} \equiv \min_{k=1,2,...,n_p} \{\hat{f}_k(\mathbf{x}_k)\}. \tag{2.33}$$

A operação de **escalonamento** opera seqüencialmente à operação de **translação**, e visa normalizar em um dado intervalo conveniente, os valores das aptidões dos indivíduos, produzindo novas valorações, $\hat{f}(\mathbf{x})$, para a aptidão. No caso, optou-se pelo intervalo finito e fechado [0,1]. A seguir apresenta-se a expressão usada para o cômputo do **escalonamento** das aptidões dos indivíduos

$$\hat{f}_k(\mathbf{x}_k) \equiv \frac{\tilde{f}_k(\mathbf{x}_k)}{\tilde{f}_{\text{máx}}}; \quad k = 1,2,...,n_p; \quad \tilde{f}_{\text{máx}} \equiv \max_{k=1,2,...,n_p} \{\tilde{f}_k(\mathbf{x}_k)\}. \tag{2.34}$$

O **escalonamento** é uma transformação linear e, portanto, altera proporcionalmente os valores da aptidão. A **potenciação** é uma transformação não linear e que, portanto, altera de forma não proporcional os valores das aptidões dos indivíduos. Dependendo dos valores do parâmetro de potenciação, $\gamma > 0$, os indivíduos menos aptos da população poderão ser, relativamente, beneficiados em detrimento dos mais aptos, e vice-versa. A **potenciação** para obtenção da aptidão, $\breve{f}(\mathbf{x})$, é definida como segue

$$\breve{f}_k(\mathbf{x}_k) \equiv [\hat{f}_k(\mathbf{x}_k)]^{\gamma}; \quad k = 1,2,...,n_p; \quad \gamma > 0. \tag{2.35}$$

É possível reestruturar as equações 2.34 e 2.35, para que as definições das transformações de escalonamento e potenciação, ocorram diretamente sobre os valores da função objetivo, $\hat{f}(\mathbf{x})$. Assim, aquelas equações tornam-se

$$\hat{f}_k(\mathbf{x}_k) \equiv \frac{\hat{f}_k(\mathbf{x}_k) - \hat{f}_{\min}}{\hat{f}_{\text{máx}} - \hat{f}_{\min}}; k = 1,2,...,n_p; \hat{f}_{\min} \equiv \min_{k=1,2,...,n_p} \{\hat{f}_k(\mathbf{x}_k)\}; \hat{f}_{\text{máx}} \equiv \max_{k=1,2,...,n_p} \{\hat{f}_k(\mathbf{x}_k)\}, \tag{2.36a}$$

$$\breve{f}_k(\mathbf{x}_k) \equiv \left[\frac{\hat{f}_k(\mathbf{x}_k) - \hat{f}_{\min}}{\hat{f}_{\text{máx}} - \hat{f}_{\min}}\right]^{\gamma}; k = 1,2,...,n_p; \gamma > 0, \quad \hat{f}_{\min} \equiv \min_{k=1,2,...,n_p} \{\hat{f}_k(\mathbf{x}_k)\}; \hat{f}_{\text{máx}} \equiv \max_{k=1,2,...,n_p} \{\hat{f}_k(\mathbf{x}_k)\}$$

. (2.36b)

A partir da opção de regra para aptidão, $\tilde{f}(\mathbf{x})$, $\hat{f}(\mathbf{x})$ ou $\breve{f}(\mathbf{x})$, pode-se realizar o cômputo das aptidões de todos os indivíduos da população e a partir destes valores estabelecer, via comparação de valores, uma classificação dos indivíduos, do **mais apto**, ao **menos apto**, sendo o mais apto aquele que possuir o maior valor numérico da aptidão, e o menos apto aquele que possuir o menor valor numérico da aptidão.

Eventualmente, caso necessário, pode-se inserir no processo de obtenção da aptidão outros aspectos do fenótipo dos indivíduos.

Como a **aptidão** de um indivíduo depende não somente de sua **avaliação**, mas também das avaliações dos outros indivíduos da população, toda vez que se realiza o processo de seleção faz-se necessário calcular as aptidões de todos os indivíduos porque, necessariamente, surgiram indivíduos novos na população via *crossover* ou algum indivíduo foi mutado e, portanto, teve seu genótipo alterado, e em conseqüência também teve o fenótipo mudado.

A aptidão de cada indivíduo, obtida conforme descrito anteriormente, é constituinte de seu fenótipo, e seu valor é atribuído à variável membro **aptidao**, da estrutura de dados do indivíduo.

Elitismo

Neste trabalho, **elitismo** é um modo de execução do algoritmo genético no qual se considera ou não a necessidade de se ter na população indivíduos de elite. **Indivíduos de elite** constituem um subconjunto da população e possuem as maiores aptidões. Caso se considere necessário o uso do elitismo deve haver na população no mínimo um indivíduo de elite. O indivíduo de elite adquire "imortalidade" momentânea, significando que não pode ser descartado da população enquanto permanecer indivíduo de elite. Um indivíduo de elite deixa de sê-lo quando aparece na população algum indivíduo mais apto que ele e o desloca para fora do grupo de elite.

Da mesma forma que a aptidão de um indivíduo depende de suas características individuais bem como das características do restante da população, o elitismo individual também depende de características individuais e de características da população.

A característica fenotípica de elitismo é atribuída à variável membro **flagelite**. Todos indivíduos quando são criados têm atribuídos às variáveis membro o valor "**false"**. Nenhum indivíduo é criado pertencendo ao grupo de elite, quando ele existe. Quando o grupo de elite

não existe todos os indivíduos permanecem como de não-elite por todas as gerações. Quando o grupo de elite existe, os indivíduos mais aptos da população têm o valor **"true"** atribuído às suas respectivas variáveis membro. Quando um indivíduo de elite é deslocado para fora do grupo de elite o valor da variável membro flagelite é mudado de **"true"** para **"false"**. Esta atribuição ocorre logo após todas as aptidões serem calculadas e atribuídas.

Quando se utiliza a estratégia do elitismo, pode-se definir a **taxa de elitismo**, λ, como sendo a relação entre a quantidade de indivíduos de elite na população, n_e; e a quantidade de membros da população, n_p, como segue

$$\lambda \equiv \frac{n_e}{n_p}; \quad n_e < n_p; \quad n_e, n_p \in \Im; \quad \lambda \in \Re . \tag{2.37}$$

Caso tenha-se a taxa de elitismo e o tamanho da população pode-se obter a quantidade de indivíduos de elite

$$n_e = \text{int}(\lambda \times n_p); \quad n_e < n_p; \quad n_e, n_p \in \Im; \quad \lambda \in \Re , \tag{2.38}$$

na qual **int(...)** é uma função que transforma seu argumento real em um número inteiro mais próximo e dentro do intervalo de definição.

Quando utilizado o elitismo, a taxa de elitismo não deve ser próxima da unidade, pois inviabilizaria o algoritmo genético não permitindo que os indivíduos evoluam, uma vez que a maioria da população seria de elite. Em realidade, o principal papel positivo do elitismo é não permitir que o melhor indivíduo da população seja descartado. O lado negativo do elitismo ocorre quando um indivíduo de elite chega a um ponto maximizante local e por ser não-descartável promove o travamento da evolução. Por exemplo, para uma população com cem indivíduos bastaria uma taxa de elitismo de 1% para reter sempre o melhor indivíduo. Com 5% reter-se-ia os cinco melhores.

Pênalti

Pênalti é um modo de execução do algoritmo genético no qual os indivíduos da população são, ou não, marcados para serem descartados, oportunamente. Caso não haja opção pelo **pênalti** todos os indivíduos não serão marcados para descarte, via pênalti. Mesmo

assim, indivíduos serão descartados por outros mecanismos de substituição a serem discutidos nas próximas seções e capítulos. Caso haja a opção pelo **pênalti** faz-se necessária uma regra de pênalti para que se decida quais indivíduos serão marcados para descarte e quais não serão.

A **regra de pênalti** é um algoritmo decisório associado ao domínio de definição do problema de otimização, com a avaliação e com a aptidão do indivíduo.

Por exemplo, seja o seguinte problema de minimização: Minimizar a função $f(x) = \sqrt{x}; x, f \in \Re$, sobre o intervalo $-1 \leq x \leq +1$. Como o intervalo de definição do problema admite valores negativos, mas a função pertence ao reais e não admite valores negativos como argumento, então todos os indivíduos cujo genótipo corresponda a valores reais no intervalo $-1 \leq x < 0$ devem ser descartados como inviáveis. Neste caso a regra de pênalti é: se o indivíduo pertence ao intervalo $-1 \leq x < 0$, então deve ser marcado para descarte. Caso contrário, não é marcado. A regra de pênalti para este exemplo baseou-se em informações sobre o domínio de definição do problema e sobre características da função objetivo.

Para o exemplo apresentado poder-se-ia também, simplesmente, redefinir o intervalo de definição do problema e não haveria a necessidade de se recorrer ao mecanismo de pênalti. No entanto, este foi um exemplo didático. Podem ocorrer casos onde a regra de pênalti seja complexa e difícil de vislumbrar uma solução simples. Este mecanismo também pode ser útil para resolver problemas de otimização com restrições.

Quando todos os indivíduos da população são criados, suas respectivas variáveis membro **flagpenalty** recebem a atribuição do valor **"false"**. Quando das operações de avaliação e aptidão todos os indivíduos que atendem à regra de pênalti são marcados para descarte e a respectiva variável membro **flagpenalty** recebe o valor **"true"**. No caso do exemplo, quando um dado indivíduo é marcado para descarte, não é necessário calcular a função objetivo.

Processo de seleção

A ação de **seleção**, basicamente, responde à questão de quais são os indivíduos, em cada geração, que poderão produzir descendentes. Em geral a seleção baseia-se em dois mecanismos: a prevalência do mais apto sobre o menos apto e, simultaneamente, a aleatoriedade. Embora seja mais provável que o mais apto deixe descentes, porém nem sempre isto ocorre. Na natureza nem sempre o mais apto está no local certo, no momento

certo, para a reprodução. Às vezes o mais apto sofre algum acidente e fica inviável. Na metáfora computacional do algoritmo genético os acontecimentos fortuitos da vida real são modelados por processos aleatórios. Importante dizer que os processos aleatórios numéricos, não são exatamente aleatórios, mas apenas pseudo-aleatórios (Langdon; Poli, 2002). Em geral, a maioria dos algoritmos de seleção utilizam alguma combinação dos dois mecanismos, prevalência do mais apto e aleatoriedade, para produzir o conjunto dos indivíduos selecionados. Seria possível produzir dois algoritmos de seleção que se baseasse somente na prevalência do mais apto ou somente na aleatoriedade. Neste trabalho optou-se por utilizar apenas algoritmos que fossem baseados, simultaneamente, nos dois mecanismos. Os algoritmos aqui utilizados foram: **roda da roleta**, **amostragem estatística universal**, e **torneio**.

Conforme descrito e definido, anteriormente, são selecionados, n_s, indivíduos da população, valor este que corresponde a uma taxa de seleção, σ.

Roda da roleta

Neste algoritmo, todos os indivíduos são colocados, simbolicamente, sobre uma roda de roleta à semelhança do mostrado na Figura 2.7. Neste caso, são hipoteticamente, os dez indivíduos constantes da Tabela 2.2. A fatia que cada indivíduo ocupa da roda da roleta é diretamente proporcional à sua aptidão. Assim, pode-se calcular o ângulo, α, da fatia de cada indivíduo sobre a roda da roleta, como segue

$$\alpha_k \equiv 2\pi \frac{f^*(\mathbf{x}_k)}{\sum_{k=1}^{n_p} f^*(\mathbf{x}_k)}; \quad k = 1,2,...,n_p; \quad f^* \in \{\tilde{f}, \hat{f}, \breve{f}\}. \tag{2.39}$$

Na equação (2.39) a aptidão *f**, refere-se a qualquer uma das três formas de se obter a aptidão, apresentadas anteriormente. O somatório que aparece no denominador da equação (2.39) é a chamada **aptidão global acumulada** da população. A idéia de **aptidão acumulada** pode ser definida também para indivíduos, como mostrado a seguir

$$S_k^*(\mathbf{x}_k) \equiv \sum_{k^*=1}^{k} f^*(\mathbf{x}_{k^*}); \quad k = 1,2,...,n_p; \quad f^* \in \{\tilde{f}, \hat{f}, \breve{f}\}. \tag{2.40}$$

A aptidão acumulada, $S^*(\mathbf{x})$, de cada indivíduo foi também considerada como parte de seu fenótipo e seu valor é atribuído à variável membro **soma_aptidao**, da estrutura de dados do respectivo indivíduo.

Seguindo o descrito anteriormente tem-se então a roda da roleta constituída com todos os indivíduos da população posicionados e ocupando cada um a sua quota-parte angular da roda. Então se gira a roda da roleta n_s vezes, e obtém-se uma lista com os n_s indivíduos selecionados. Em seguida, verifica-se se não existem indivíduos selecionados mais de uma vez. Caso exista, a roda da roleta é girada, novamente e novamente, até produzir um conjunto suficiente de n_s indivíduos sem nenhum repetição.

Uma vez obtida a lista de n_s indivíduos selecionados, sem repetição, o processo de seleção está terminado.

Neste trabalho optou-se por não permitir a seleção de indivíduos repetidos, em realidade clones perfeitos, porque implicaria em menor diversidade genética, e conseqüente menor robustez do algoritmo genético.

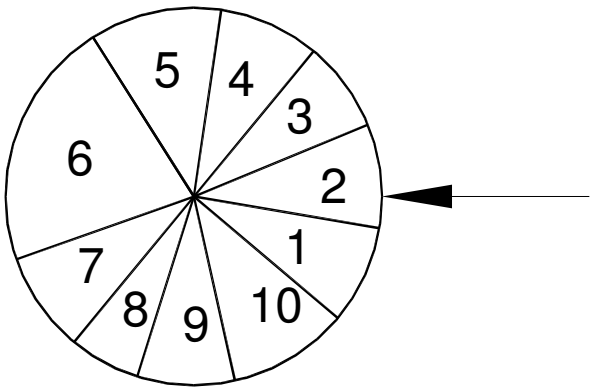


Figura 2.7 – Roda da roleta

Nota-se que na roda da roleta existem os dois mecanismos, prevalência do mais apto e aleatoriedade. A prevalência do mais apto está presente no fato de que os indivíduos mais aptos ocupam quotas-parte angulares maiores do que aquelas ocupadas por indivíduos menos aptos. A aleatoriedade está presente no fato de que embora seja mais provável que os mais aptos sejam selecionados pode ocorrer, aleatoriamente, de um indivíduo pouco apto ser o selecionado.

Note-se que também seria computacionalmente possível uma **"roda da roleta dinâmica"** na qual após o sorteio de um determinado indivíduo, ele sairia da roda, esta se

redefiniria dinamicamente e a seleção continuaria. Neste caso não haveria a possibilidade de ocorrer indivíduos selecionados repetidos.

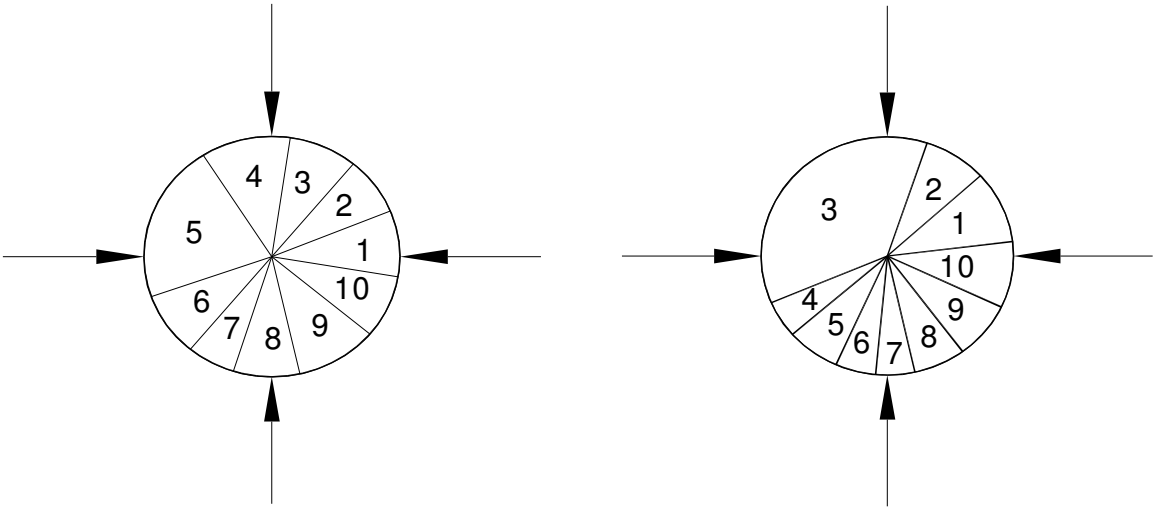
Amostragem estocástica universal

No algoritmo da roda da roleta tem-se um ponteiro e gira-se a roleta n_s vezes obtendo-se um indivíduo selecionado de cada vez. Caso hajam indivíduos repetidos entre os selecionados faz-se necessário girar a roleta mais algumas vezes até produzir os n_s indivíduos, sem repetição conforme descrito anteriormente.

O caso da **amostragem estocástica universal** corresponde ao uso de uma roleta definida nas suas porções angulares de maneira igual ao algoritmo da roda da roleta. A diferença está que ao invés de possuir apenas um ponteiro têm-se n_s ponteiros, igualmente espaçados angularmente, e a roda é girada apenas uma vez. Possuindo n_s ponteiros a roda apontará para n_s indivíduos em uma única rodada.

Na Figura 2.13(a) apresenta-se o algoritmo da amostragem estocástica universal para uma população com dez indivíduos dos quais pretende-se selecionar quatro deles. No caso, considerando que a roleta foi girada uma vez e parou, os indivíduos selecionados são: um, quatro, cinco e oito.

Neste algoritmo também pode ocorrer a seleção de indivíduos repetidos, conforme apresentado na Figura 2.8(b), na qual o indivíduo três está sendo apontado por dois ponteiros.



(a) (b)

Figura 2.8 – Amostragem estocástica universal

Torneio

No algoritmo do **torneio** define-se uma roda da roleta com frações angulares iguais para cada indivíduo, Figura 2.9. Roda-se a roleta duas vezes obtendo-se dois indivíduos distintos. Caso sejam iguais gira-se a roleta até obter um segundo indivíduo distinto do primeiro. Estes dois indivíduos participam de um torneio no qual o vencedor é o que tem maior aptidão.

O procedimento descrito no parágrafo anterior é repetido n_s vezes obtendo assim n_s indivíduos selecionados.

Na lista final de indivíduos selecionados não pode haver nenhum repetido, se houver faz-se novamente o processo de torneio até obter os indivíduos adicionais e completar a lista de n_s indivíduos selecionados e distintos.

Tendo executado os sub-operadores **avaliação**, **aptidão** e **processo de seleção** tem-se ao final o resultado do operador **seleção** que é uma lista com n_s indivíduos distintos, os selecionados para a reprodução.

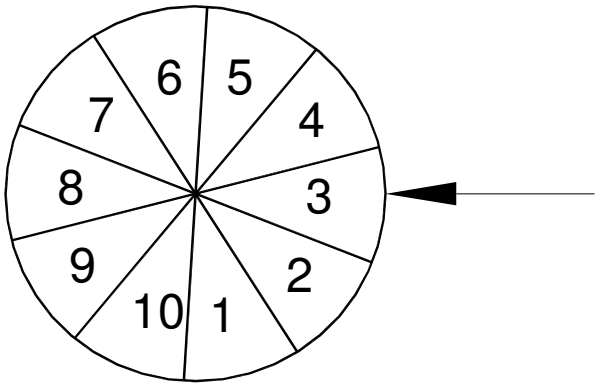


Figura 2.9 – Torneio

2.5.2 – Crossover

Crossover é a recombinação de material genético de dois pais para produzir um ou mais descendentes. Sendo um processo de recombinação o *crossover* não produz material genético novo apenas recombina o já existente.

Foram utilizados cinco algoritmos para realização de *crossover*: *crossover* de **um ponto**, *crossover* de **dois pontos**, *crossover* de **múltiplos pontos** e *crossover* **uniforme**.

Destas modalidades de *crossover* todas geram dois filhos. Assim, tem-se para todos os casos n_s filhos. Como n_s é inteiro par, então $n_s/2$ é inteiro.

Para o pareamento dos pais sorteia-se aleatoriamente dois indivíduos, não repetidos, da lista de selecionados para constituir um par. Um indivíduo também não pode participar de mais de um par. Não pode haver pares iguais.

Executando este procedimento $n_s/2$ vezes tem-se igual quantidade de pares. Os pares são utilizados no processo de *crossover* para gerar os descendentes. Seja qual for o algoritmo de *crossover* ao final terão sido gerados certa quantidade de filhos, n_f, $0 \le n_f \le n_p$.

Os n_f filhos gerados são, momentaneamente, colocados em uma "maternidade" e imediatamente após o término do *crossover* eles são introduzidos na população através do uso do operador **substituição**.

***Crossover* de um ponto**

No ***crossover* de um ponto** sorteia-se um valor inteiro, *k*, entre 1 e (n_g – 1). Este procedimento corresponde a definir e girar uma vez uma roda de roleta com (n_g – 1) parcelas angulares iguais. Como n_g é a quantidade de genes no cromossomo do indivíduo, (n_g – 1) corresponde às posições intergenes. Após a definição aleatória da posição intergenes, *k*, procede-se à troca de material genético. Seja um par de pais, com os cromossomos $(g_1, g_2, ..., g_k, ..., g_{n_g-1}, g_{n_g})$ e $(h_1, h_2, ..., h_k, ..., h_{n_g-1}, h_{n_g})$, o *crossover* entre ocorre pela troca de material genético da posição intergenes, *k*, para diante, como mostrado a seguir

$$\begin{matrix} (g_1, g_2, ..., g_k, ..., g_{n_g-1}, g_{n_g}) \\ \\ (h_1, h_2, ..., h_k, ..., h_{n_g-1}, h_{n_g}) \end{matrix} \quad Crossover \Rightarrow \quad \begin{matrix} (g_1, g_2, ..., g_k, h_{k+1} ..., h_{n_g-1}, h_{n_g}) \\ \\ (h_1, h_2, ..., h_k, g_{k+1} ..., g_{n_g-1}, g_{n_g}) \end{matrix} \tag{2.41}$$

gerando dois descentes com cromossomos $(g_1, g_2, ..., g_k, h_{k+1} ..., h_{n_g-1}, h_{n_g})$ e $(h_1, h_2, ..., h_k, g_{k+1} ..., g_{n_g-1}, g_{n_g})$.

Seja, como exemplo, um par de pais, Tabela 2.7, cada um com cromossomo de 16 genes. Assim, um dada posição 5 corresponde ao intergenes entre os genes 5 e 6. Para apreciação visual o pai 1 está grafado em negrito enquanto o pai 2 não.

Tabela 2.7 – Par de pais selecionados para realizar crossover.

Pais	Genótipos															
1	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
2	0	1	0	0	0	1	1	0	1	0	1	1	1	0	0	1

Suponha que a posição intergenes sorteada seja $k = 12$, em conseqüência os descendentes gerados usando o *crossover* de um ponto são apresentados na Tabela 2.8. Estão grafadas em fundo cinza as partes recombinadas dos cromossomos.

Tabela 2.8 – Par de filhos gerados usando crossover de um ponto.

Filhos	Genótipos															
1	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	1	0	0	1
2	0	1	0	0	0	1	1	0	1	0	1	1	**1**	**0**	**1**	**1**

Se os cromossomos de pais e filhos nas Tabelas 2.7 e 2.8 representam indivíduos pertencentes ao $\Re^1$, então os pais 1 e 2 correspondem aos valores inteiros iguais a 38971 e 18105, respectivamente. Os filhos 1 e 2 correspondem, respectivamente, aos inteiros 38969 e 18107.

Suponha agora que a posição intergenes sorteada seja $k = 1$, para os mesmos pais, os descendentes gerados usando o *crossover* de um ponto são aqueles apresentados na Tabela 2.9.

Tabela 2.9 – Par de filhos gerados usando crossover de um ponto.

Filhos	Genótipos															
3	**1**	1	0	0	0	1	1	0	1	0	1	1	1	0	0	1
4	0	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**

Se os cromossomos de pais e filhos nas Tabelas 2.7 e 2.8 representam também indivíduos pertencentes ao $\Re^1$, então os filhos 3 e 4 correspondem, respectivamente, aos inteiros 50873 e 6203.

Nota-se que, embora dependa do valor dos alelos, quanto mais à esquerda estiver a posição intergenes sorteada maior será a variação numérica inteira, e conseqüentemente também no mapeamento em números reais, entre os dois filhos, e entres os filhos e os pais.

Se os cromossomos de pais e filhos nas Tabelas 2.7 e 2.9 representam indivíduos pertencentes ao $\Re^2$, então os pais 1 e 2 correspondem aos valores inteiros para as duas variáveis iguais a (152, 59) e (70, 185), respectivamente. Os filhos 3 e 4 correspondem, respectivamente, aos inteiros (397, 185) e (24, 59).

***Crossover* de dois pontos**

O ***crossover* de dois pontos** é semelhante ao *crossover* de um ponto, porém neste algoritmo sorteiam-se, aleatoriamente, dois valores inteiros não repetidos, ou seja, duas posições intergenes, k_1 e k_2, entre 1 e (n_g – 1). Depois da definição das posições intergenes, k_1 e k_2, procede-se à troca de material genético conforme seqüência adiante

$$\begin{array}{ll} (g_1, g_2, ..., g_k, ..., g_{n_g-1}, g_{n_g}) & (g_1, ..., g_{k_1}, h_{k_1+1} ..., h_{k_2}, g_{k_2+1} ..., g_{n_g}) \\ \qquad\qquad\qquad Crossover \Rightarrow & \\ (h_1, h_2, ..., h_k, ..., h_{n_g-1}, h_{n_g}) & (h_1, ..., h_{k_1}, g_{k_1+1} ..., g_{k_2}, h_{k_2+1} ..., h_{n_g}) \end{array} \qquad (2.42)$$

gerando dois descentes com cromossomos $(g_1, ..., g_{k_1}, h_{k_1+1} ..., h_{k_2}, g_{k_2+1} ..., g_{n_g})$ e $(h_1, ..., h_{k_1}, g_{k_1+1} ..., g_{k_2}, h_{k_2+1} ..., h_{n_g})$.

Considere, como exemplo, o mesmo par de pais da Tabela 2.7. Suponha que as posições intergenes sorteadas sejam $k_1 = 3$ e $k_2 = 9$, seguindo o algoritmo do *crossover* de dois pontos tem-se os filhos apresentados na Tabela 2.10.

Tabela 2.10 – Par de filhos gerados usando crossover de dois pontos.

Filhos	Genótipos															
5	**1**	**0**	**0**	0	0	1	1	0	1	**0**	**1**	**1**	**1**	**0**	**1**	**1**
6	0	1	0	**1**	**1**	**0**	**0**	**0**	**0**	0	1	1	1	0	0	1

Caso os cromossomos de pais e filhos nas Tabelas 2.7 e 2.10 representem indivíduos pertencentes ao $\Re^1$, então os filhos 5 e 6 correspondem, respectivamente, aos inteiros 34491 e 22585.

Como são sorteadas duas posições intergenes no *crossover* de dois pontos, aumenta a probabilidade de que uma destas posições esteja mais à esquerda no cromossomo e, portanto tenha maior variação numérica no valor inteiro correspondente.

Crossover de múltiplos pontos

O *crossover* de dois pontos é a extensão do *crossover* de um ponto. O ***crossover* de múltiplos pontos** é a generalização dos dois. Seja $n_m < n_g$ a quantidade dos pontos múltiplos. Neste algoritmo sorteiam-se, aleatoriamente, n_m valores inteiros não repetidos, ou seja, n_m posições intergenes, k_i, $i = 1,...,n_m$ entre 1 e (n_g – 1). Depois da definição das posições intergenes, k_i, $i = 1,...,n_m$, tem-se para cada indivíduo um conjunto de genes de 1 a k_1, outro de (k_1+1) a k_2, e assim, sucessivamente até o conjunto de k_{n_m} a n_g. O material genético trocado, à semelhança do *crossover* de dois pontos corresponde aos conjuntos de genes pares. Ou seja, do segundo, do quarto, até o conjunto $(n_m+1)/2$. Quando a conta anterior for fracionária arredonda-se para o menor inteiro mais próximo.

Considere, novamente como exemplo, o par de pais da Tabela 2.7. Considere também que a quantidade de pontos múltiplos seja cinco, $n_m = 5$.

Suponha que as posições intergenes sorteadas sejam $k_1 = 3$, $k_2 = 5$, $k_3 = 7$, $k_4 = 8$, e $k_5 = 11$, pelo algoritmo do *crossover* de pontos múltiplos tem-se os filhos apresentados na Tabela 2.11.

Tabela 2.11 – Par de filhos gerados usando crossover de múltiplos pontos.

Filhos	Genótipos															
7	**1**	**0**	**0**	0	0	**0**	**0**	0	**0**	**0**	**1**	1	1	0	0	1
8	0	1	0	**1**	**1**	1	1	**0**	1	0	1	**1**	**1**	**0**	**1**	**1**

O caso limite para o *crossover* de pontos múltiplos é quando $n_m = (n_g - 1)$, neste caso todos os genes pares dos cromossomos são trocados.

Crossover uniforme

Para o *crossover* uniforme gera-se, aleatoriamente, um conjunto constituído de **"true" ≡ "1"** e **"false" ≡ "0"** em quantidade igual à quantidade de genes em cada cromossomo, n_g, e correspondentes às posições dos respectivos genes dos cromossomos. A geração aleatória dos **"1"** ocorre com probabilidade p_u, $0 < p_u \leq 1$, e a geração dos **"0"**

ocorre com probabilidade $(1 - p_u)$. Na metáfora da roleta corresponderia a uma roleta com duas partes angulares, Figura 2.10, uma diretamente proporcional a p_u e a outra diretamente proporcional a $(1 - p_u)$. Neste trabalho utilizou-se iguais probabilidades para geração de **"0"** e **"1"** ou seja 50% para cada, logo $p_u = 0{,}5$. No caso de probabilidades iguais, $p_u = 0{,}5$ e $(1\text{-}p_u) = 0{,}5$, a roleta fica igual àquela apresenta na Figura 2.3.

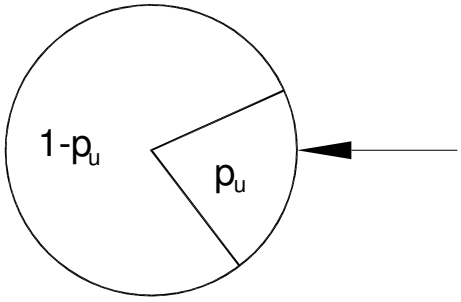


Figura 2.10 – Roleta de probabilidade para o crossover uniforme.

Após a geração da lista com os n_g **"0"** e **"1"** realiza-se então a recombinação dos genes dos pais. Inspeciona-se então a lista de **"0"** e **"1"** e para as posições nas quais o valor é **"1"** troca-se os genes correspondentes dos cromossomos dos pais para aquela posição.

Como exemplo faz-se o *crossover* dos dois pais apresentados na Tabela 2.7, para tal gera-se a lista com 16 **"0"** ou **"1"** prevista no algoritmo. Na Tabela 2.12, em cinza estão marcados os valores iguais a **"1",** os quais definem as posições onde serão trocados os genes dos cromossomos.

Tabela 2.12 – Lista com 16 "0" e "1" para o crossover uniforme.

Lista	0	1	1	0	0	0	1	0	0	1	0	1	0	0	1	0

Trocando-se os genes dos pais nas respectivas posições marcadas na lista de **"0"** e **"1"** resultam os dois filhos mostrados na Tabela 2.13.

Tabela 2.13 – Par de filhos gerados usando crossover uniforme.

Filhos	Genótipos															
7	**1**	1	0	**1**	**1**	**0**	1	**0**	**0**	0	**1**	1	**1**	**0**	0	**1**
8	0	**0**	**0**	0	0	1	**0**	0	1	**0**	1	**1**	1	0	**1**	1

2.5.3 – Substituição

Conforme estabelecido anteriormente os filhos são gerados através do uso do operador *crossover* e colocados em uma "maternidade" para posterior introdução na população. Computacionalmente, os filhos são colocados em contêineres de indivíduos, logo após o *crossover*, e daí substituídos imediatamente na população.

O operador **substituição**, basicamente, é o responsável por definir quais são os indivíduos da população que serão descartados, para que os filhos sejam inseridos na população.

Eventualmente, o operador substituição é o responsável por inserir na população outros subgrupos de indivíduos que não sejam filhos. Por exemplo: indivíduos migrantes de outras populações. Este conceito é útil na implementação do algoritmo genético com processamento paralelo.

O operador substituição, neste trabalho, pode ser realizado via três algoritmos: **substituição dos pais**, **substituição dos menos aptos**, e **substituição aleatória**.

Após a execução da substituição dos filhos na população, a "maternidade" ou contêiner de indivíduos é totalmente esvaziado.

Quando se executa o algoritmo genético no modo **elitismo** os indivíduos de elite não podem ser descartados, enquanto permanecerem de elite, assim serão descartados outros indivíduos, não de elite, sorteados aleatoriamente.

Caso exista na população indivíduos penalizados, isto é, com a variável membro **flagpenalty** com valor igual a **"true"**, eles são descartados e substituídos por outros indivíduos gerados aleatoriamente.

Finalmente, o operador substituição inspeciona a nova população atualizada à procura de clones perfeitos. Os que eventualmente existam são descartados e substituídos por novos indivíduos, gerados aleatoriamente.

Substituição dos pais

No algoritmo de **substituição dos pais** pelos filhos os pais são descartados e os n_f filhos colocados em seus lugares. Caso algum dos pais seja do grupo de elite então não será descartado. Um outro indivíduo, não de elite, será sorteado aleatoriamente para ser

descartado. Neste algoritmo esta possibilidade vai ocorrer com certa freqüência, pois, os indivíduos de elite possuem alta aptidão e muitas vezes serão os selecionados para serem pais.

Substituição dos menos aptos

No algoritmo de **substituição dos menos aptos** pelos filhos, os indivíduos da população são classificados do mais apto ao menos apto, e os n_f menos aptos são selecionados para descarte. Em seguida, os n_f indivíduos menos aptos são então substituídos pelos filhos.

Caso haja algum indivíduo de elite entre os menos aptos o indivíduo é preservado e sorteia-se aleatoriamente um outro indivíduo para o descarte. Esta possibilidade é rara porque, em geral, os indivíduos de elite possuem alta aptidão e estão portanto entre os mais aptos.

Substituição aleatória

No algoritmo de **substituição aleatória** seleciona-se, aleatoriamente, n_f indivíduos para descarte. Os indivíduos selecionados são substituídos pelos filhos. Neste algoritmo também é necessária a verificação se entre os selecionados para descarte não existe algum indivíduo de elite. Caso exista deve ser preservado e outro indivíduo da população deve ser sorteado, aleatoriamente, para descarte.

2.5.4 – Mutação

O operador **mutação** altera, aleatoriamente, o valor dos alelos de alguns genes localizados em algumas posições do cromossomo. Devido a esta alteração no código genético do indivíduo o processo de mutação é um produtor ou destruidor de "boa genética" no sentido de que pode transformar indivíduos menos aptos em mais aptos e vice-versa. Ao contrário do *crossover* que apenas recombina material genético, a mutação produz material genético novo. Os novos indivíduos mutantes se forem mais aptos terão maior probabilidade de passar adiante, aos seus filhos, os seus genes. Se forem menos aptos terão menor probabilidade de propagar na população os seus genes.

Goldberg e Sastry (2007) e Linden (2006) consideram a mutação associada à reprodução, ou seja, a mutação ocorreria basicamente associada ao processo de *crossover* e, computacionalmente, em geral é implementada como cópia imperfeita de material genético

quando do movimento de blocos de genes. Neste trabalho utilizou-se a idéia de que todo indivíduo da população está sujeito à mutação. Seria um tipo de mutação associada ao ambiente no qual todos indivíduos estariam sujeitos e não apenas aqueles indivíduos que estariam expostos à mutação associada à reprodução. Assim, computacionalmente, todos os $(n_p \times n_g)$ genes da população estão sujeitos ao processo de mutação.

Na natureza a probabilidade de mutação, p_m, é baixa, algo em torno de um para mil. Computacionalmente, este valor pode ser maior, até algo em torno de 5%.

Ao se executar o algoritmo genético no modo elitismo pode-se aumentar a taxa de mutação, pois, os indivíduos mais aptos estarão protegidos de possível alteração prejudicial de seu material genético.

Utilizou-se neste trabalho apenas um algoritmo de mutação que consiste em vistoriar todos os genes de todos os indivíduos da população e para cada um deles rodar a roleta de probabilidade de mutação, Figura 2.11. Se a roleta parar na quota-parte angular correspondente à mutação, p_m, o alelo do gen é mutado, isto é seu valor é alterado para algum outro alelo possível e distinto. Por exemplo, no caso binário se o gene que foi selecionado para ser mutado for o alelo **"0"** então será modificado para **"1"**, e se for **"1"** será modificado para **"0"**. Caso a roleta pare na parte correspondente à não mutação $(1 - p_m)$ o alelo do gen permanecerá inalterado.

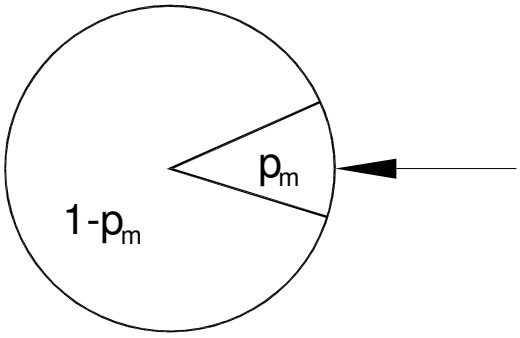


Figura 2.11 – Roleta de probabilidade para a mutação.

2.6 – Critérios de parada

A evolução de uma população geração após geração produz uma seqüência de populações. Como toda seqüência, tem início e deve ter fim.

Para finalizar a seqüência de populações faz-se necessário um ou mais critérios de parada considerados separadamente ou combinados.

A seguir analisa-se, brevemente, três grupos possíveis de para implementar critérios de parada para a evolução da população.

2.6.1 – Número máximo de gerações

No mínimo o processo evolutivo da população deve ser parado quando se atingir um número máximo pré-estabelecido de gerações, n_G. Mesmo que se utilize outro critério de parada, este critério é interessante de ser usado, combinado com algum outro critério, pois, garante a parada quando se atinge um número máximo de iterações, independente se os outros critérios foram atendidos. É a garantia de que o processo vai terminar em tempo finito.

2.6.2 – Variação da norma sobre população ou indivíduos

Pode-se utilizar a variação de informações sobre a população para terminar o processo evolutivo. Se a população estagnar e não mais evoluir significa que se chegou a algum local que pode ser ou não a solução. De qualquer forma nestes casos é interessante parar-se. Como exemplo de indicador populacional abstrato poderia-se usar a variação ao longo das gerações da distância, medida por alguma norma adequada, entre o indivíduo mais apto e o menos apto. Poderia-se também comparar a evolução da posição do indivíduo mais apto.

Em problemas de teste, para os quais se conhece a solução, pode-se utilizar como critério de parada a distância entre o indivíduo mais apto e o indivíduo solução. Quando a distância do indivíduo mais apto até a solução do problema tender a zero, dentro de certa vizinhança, o processo pode ser parado.

2.6.3 – Variação da função objetivo

A evolução do valor da função objetivo ao longo das gerações pode ser usado como critério de parada. Caso o valor da função objetivo fique estagnado e não evolua, ou evolua muito pouco, dentro de certa vizinhança, pode-se terminar o processo.

Se o valor da função objetivo na solução do problema for conhecida, pode-se terminar o processo iterativo, quando a função objetivo para o indivíduo mais apto, estiver dentro de vizinhança adequada do valor ótimo da função objetivo. O valor da função objetivo na solução do problema pode ser conhecido em problemas de teste nos quais se conhece a solução e, portanto o valor da função objetivo no ponto de solução do problema; ou em casos nos quais não se conhece a solução do problema, mas que, por razões conceituais ou empíricas, se conhece o valor da função objetivo no ponto de ótimo.

3. ALGORITMOS GENÉTICOS – OTIMIZAÇÃO MULTIDIMENSIONAL

No Capítulo 2 apresentou-se os principais aspectos conceituais dos algoritmos genéticos. Neste capítulo, apresenta-se a aplicação daquela teoria na solução de problemas de otimização multidimensionais.

Inicialmente, aplica-se os algoritmos genéticos desenvolvidos neste trabalho na solução de problema de otimização bidimensionais, visando experimentar numericamente com os parâmetros (tamanho da população, dimensão do espaço de busca, número de genes para cada variável independente, número de gerações, probabilidade de mutação, probabilidade de *crossover* uniforme, taxa de elitismo, taxa de seleção, entre outros); tipos de algoritmos dos operadores fundamentais (seleção: roda da roleta, amostragem estocástica universal, e torneio; *crossover*: de um ponto, de dois pontos, de múltiplos pontos, e uniforme; substituição: dos pais, dos menos aptos, e aleatória; e mutação); e também dos modos de execução: tipo de otimização (minimização ou maximização); e modo de elitismo (com ou sem elitismo). Ao final desta exploração numérica ter-se-á alguma noção empírica de como os algoritmos genéticos operam e quais os conjuntos de opções são as mais adequadas.

Posteriormente, estende-se a solução de problemas de otimização para problemas multidimensionais, com dimensionalidade mais alta que $n = 2$.

3.1 Conjunto de problemas teste bidimensionais

Diferentes algoritmos, em geral, operam de modo distinto na solução de problemas variados. Em outras palavras, a performance de um dado algoritmo depende das características do problema que está sendo resolvido. Assim, para testar a eficiência e a eficácia de uma da metodologia faz-se necessário realizar um conjunto amplo de testes.

No que tange aos problemas de otimização bidimensionais, estabeleceu-se um conjunto de quatro funções definidas sobre domínios bidimensionais. Estas funções são as quatro primeiras funções teste de De Jong(1975), adaptadas para os domínios aqui considerados. A seguir apresenta-se as funções teste

$$f_1(x_1, x_2) \equiv x_1^2 + x_2^2, \tag{3.1a}$$

$$f_2(x_1, x_2) \equiv 100(x_1^2 - x_2)^2 + (1 - x_1)^2, \tag{3.1b}$$

$$f_3(x_1, x_2) \equiv \text{int}(x_1) + \text{int}(x_2), \tag{3.1c}$$

e

$$f_4(x_1, x_2) \equiv x_1^4 + 2x_2^4 + Gauss(0,1). \tag{3.1d}$$

Na Tabela 3.1 apresenta-se os domínios, as soluções e os valores das funções objetivo para os pontos de solução dos problemas de minimização das funções bidimensionais f_1, f_2, f_3 e f_4.

Tabela 3.1 – Dados sobre as funções teste bidimensionais.

Função	Domínio	Solução	Função na solução
f_1	$-5{,}12 \le x_1, x_2 \le +5{,}12$	$(x_1^*, x_2^*) \equiv (0,0)$	$f_1(x_1^*, x_2^*) = 0$
f_2	$-2{,}048 \le x_1, x_2 \le +2{,}048$	$(x_1^*, x_2^*) \equiv (1,1)$	$f_2(x_1^*, x_2^*) = 0$
f_3	$-5{,}12 \le x_1, x_2 \le +5{,}12$	$(x_1^*, x_2^*) \equiv viz(-5{,}12; -5{,}12)$	$f_3(x_1^*, x_2^*) = -10$
f_4	$-1{,}28 \le x_1, x_2 \le +1{,}28$	$(x_1^*, x_2^*) \equiv viz(0,0)$	$f_1(x_1^*, x_2^*) = viz[Gauss(0,1)]$

A função teste f_1 é quadrática e simétrica, e como é bem-sabido na teoria de otimização, trata-se da função mais fácil de ser otimizada. Mesmo os métodos mais simples, em geral, são capazes de resolver a minimização desta função. É função infinitamente derivável e, portanto de classe C^{∞}.

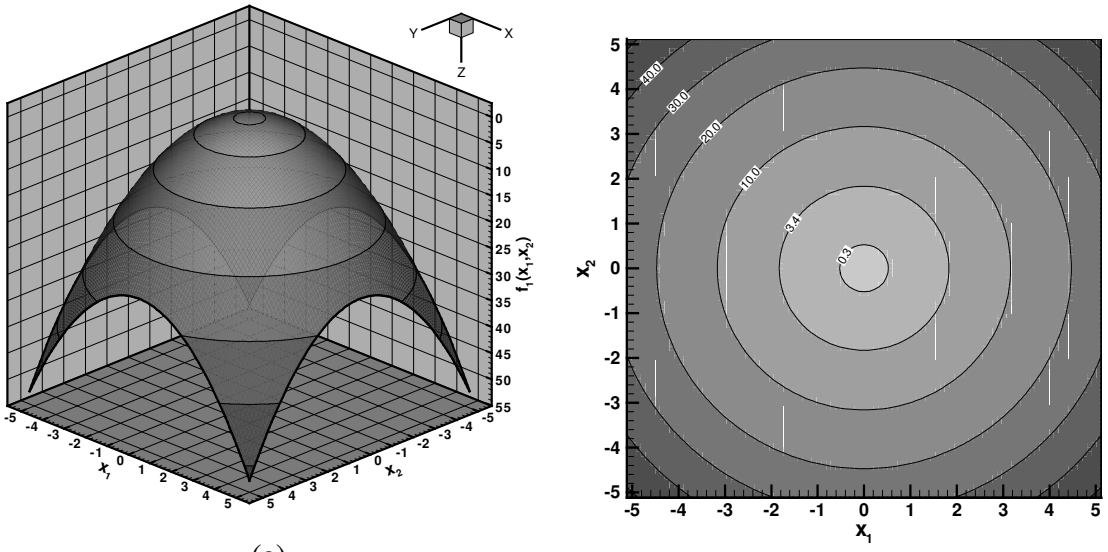
A função f_2 é em realidade a função de Rosenbrock (1960), caso clássico na teoria de otimização, pois, apresenta um longo e estreito vale curvo. Embora seja também função de classe C^{∞}, a obtenção de seu ponto de mínimo já é bem mais difícil do que no caso da função f_1.

A função f_3 não admite nenhuma derivada contínua sobre todo seu domínio de definição e, portanto, é de classe C^0. Conseqüentemente, sua otimização não pode ser realizada por nenhum método baseado em gradiente. Apenas métodos de busca direta como os algoritmos genéticos, entre outros, são capazes de resolver este problema.

Do lado direito da igualdade na função f_4 existem três termos aditivos. Os dois primeiros são, relativamente, assemelhados aos dois termos à direita da igualdade na função f_1, embora os dois referidos termos na função f_4 tenham expoentes iguais a quatro, e não dois, e também não são simétricos. Entretanto, a maior diferença está no terceiro termo à direita da

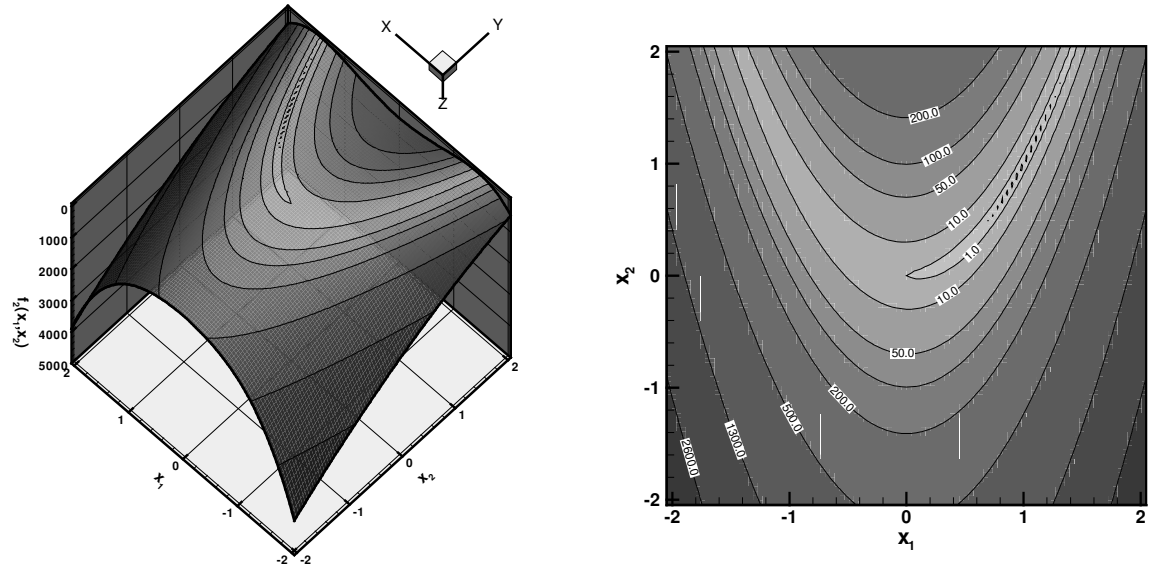
igualdade na função f_4, o termo "*Gauss*(0,1)" que representa um valor estocástico com média zero e variância igual à unidade. Assim, a função f_4 também é descontínua e, portanto de classe C^0. A principal diferença entre as funções f_3 e f_4 é que na f_3 as descontinuidades são fixas sobre o domínio e na f_4 são móveis, estocasticamente.

O grau de dificuldade na otimização das funções teste apresentadas aumenta da função f_1 para f_4.



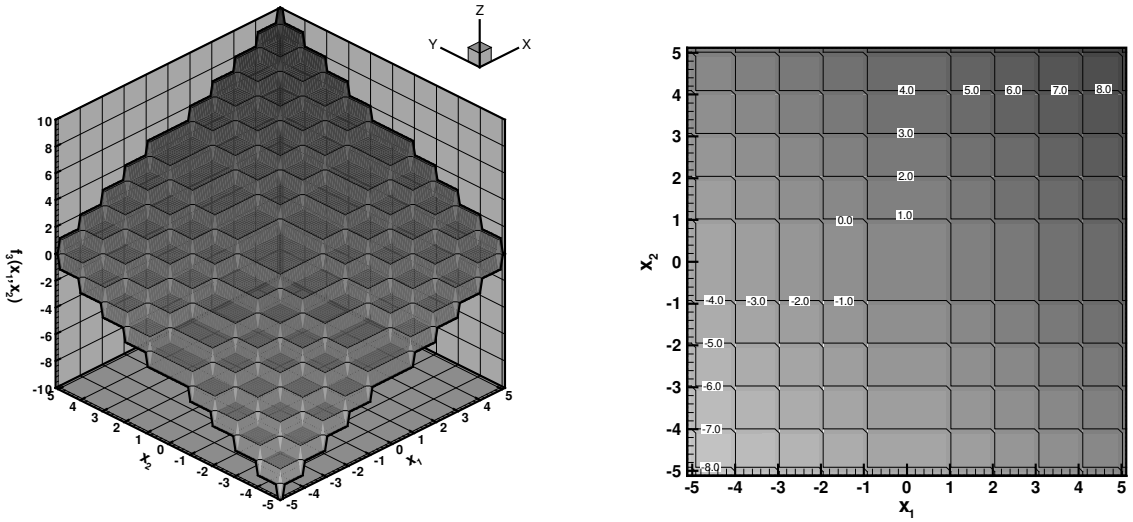
(a) (b)

Figura 3.1 – Superfície (a) e isovalores (b) da função $f_1(x_1, x_2)$



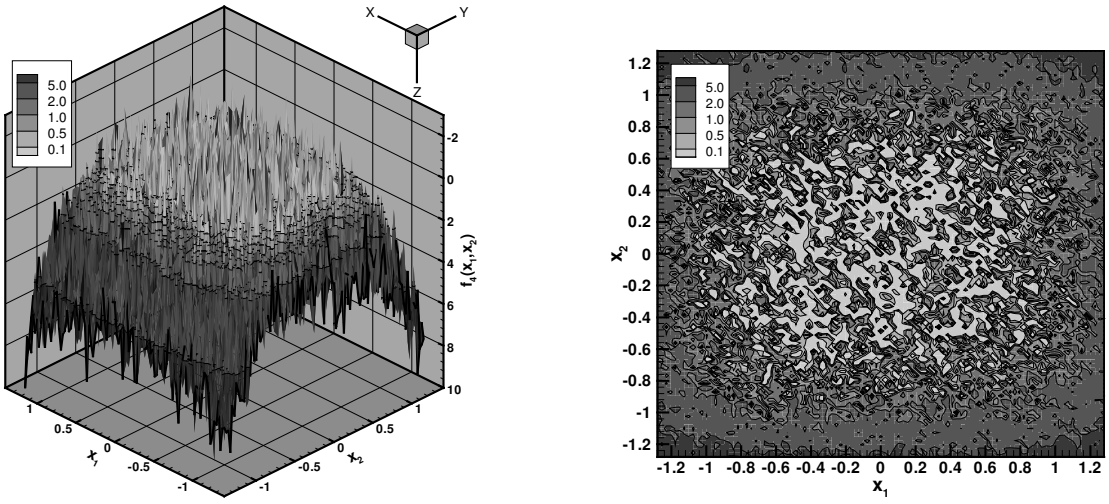
(a) (b)

Figura 3.2 – Superfície (a) e isovalores (b) da função $f_2(x_1, x_2)$



(a) (b)

Figura 3.3 – Superfície (a) e isovalores (b) da função $f_3(x_1, x_2)$.



(a) (b)

Figura 3.4 – Superfície (a) e isovalores (b) da função $f_4(x_1, x_2)$.

3.2 População inicial para domínio bidimensional

Na Figura 3.5 apresenta-se quatro sub figuras, todas para populações iniciais com 100 indivíduos, cada. Estas populações foram geradas utilizando-se o algoritmo **aleatório**. Para confecção das Figuras 3.4 a 3.8 utilizou-se cromossomos com 22 genes, e, portanto, 11 genes para cada variável. Neste caso trata-se de domínio bidimensional, $I_{\mathbf{x}} \equiv (x_1, x_2) \in \Re^2 = \{-1 \le x_1 \le +1, -1 \le x_2 \le +1\}$.

Por inspeção, percebe-se que os indivíduos das populações iniciais estão, no senso estatístico, uniformemente distribuídos sobre todo o domínio, respectivamente. Efetuou-se testes estatísticos, não apresentados aqui, mas que também comprovam a adequada distribuição inicial dos indivíduos da população. Cada população apresenta o ponto médio, correspondente às médias da população, em relação a cada variável, bastante próximo do centro do domínio. As variâncias correspondentes são pequenas.

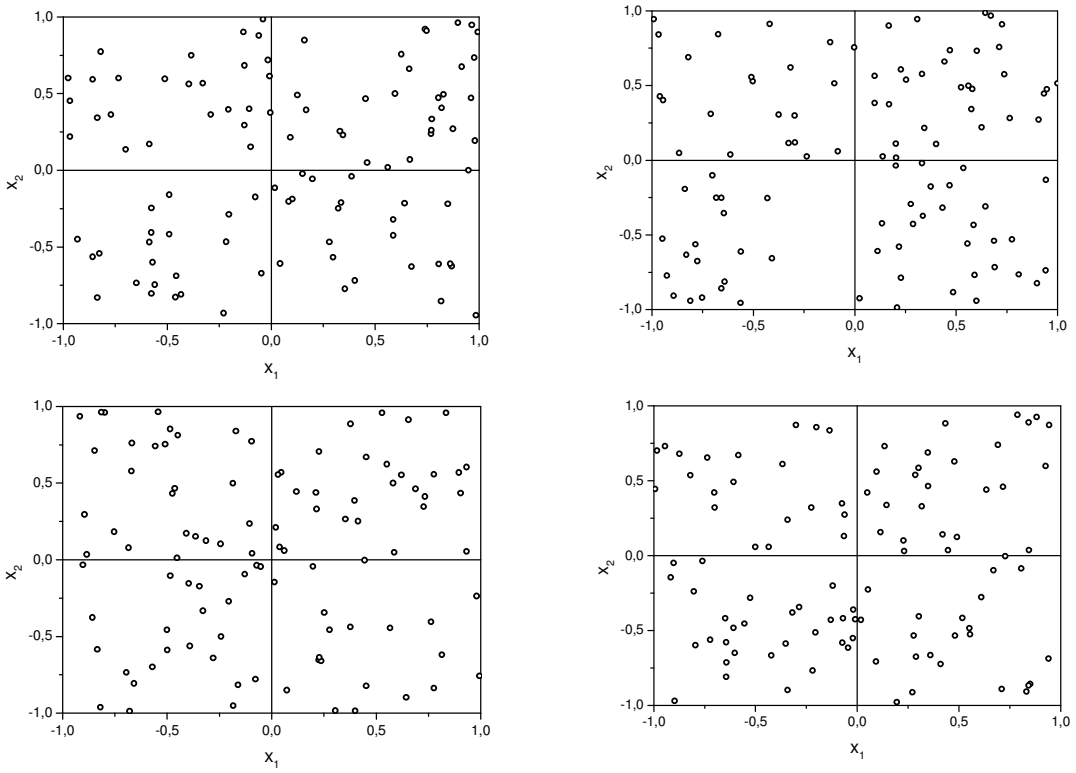


Figura 3.5- Populações iniciais geradas utilizando algoritmo aleatório. Todas quatro populações possuem 100 indivíduos, cujos cromossomos possuem 22 genes.

Na Figura 3.6 apresenta-se resultados semelhantes àqueles apresentados na Figura 3.5, porém, neste caso, os indivíduos foram gerados utilizando-se o algoritmo **semi-aleatório**. Para este tipo de algoritmo a metade dos indivíduos foram gerados aleatoriamente e o restante foi gerado fazendo um dado tipo de combinação entre a primeira metade dos indivíduos para produzir a segunda metade. Neste caso os primeiros 50 indivíduos foram criados aleatoriamente e os últimos 50 foram gerados semi aleatoriamente. Os primeiros 50 garantem a diversidade estatística e os 50 últimos são resultados de combinação determinística. Como pode se observar e também testar estatisticamente, os indivíduos assim gerados também apresentam boa distribuição sobre o domínio de definição.

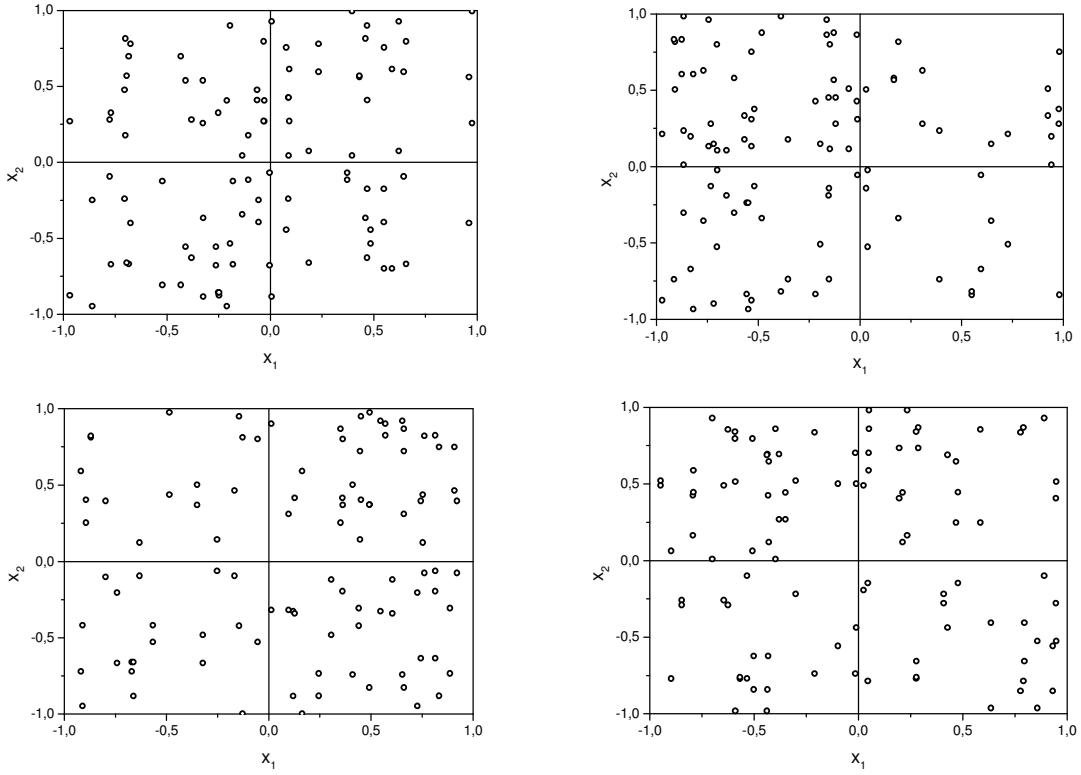


Figura 3.6 - Populações iniciais geradas utilizando algoritmo semi-aleatório. As quatro populações possuem 100 indivíduos, cujos cromossomos possuem 22 genes.

Nas Figuras 3.5 e 3.6 comparou-se populações com aleatoriedade distinta, porém com a mesma quantidade de indivíduos na população. Nas Figuras 3.7 e 3.8 procura-se analisar a influência do número de indivíduos na população inicial. As sub-figuras possuem: (a) 10; (b) 50; (c) 250 e (d) 1250 indivíduos com cromossomos de 22 genes.

Os indivíduos da Figura 3.7 foram gerados utilizando-se o algoritmo **aleatório**. Nota-se que para a população com 10 indivíduos, a distribuição é menos uniforme do que para a população com 50; esta por sua vez é menos uniforme do que a de 250; e esta é também menos uniforme do que a de 1250 indivíduos. Também a distância média entre os indivíduos decai quando o tamanho da população aumenta.

Os resultados apresentados na Figura 3.8, gerados utilizando-se o algoritmo **semi-aleatório**, são qualitativamente semelhantes àqueles apresentados na Figura 3.7. Quando o tamanho da população diminui a uniformidade da distribuição dos indivíduos também diminui e a distância média entre os indivíduos aumenta.

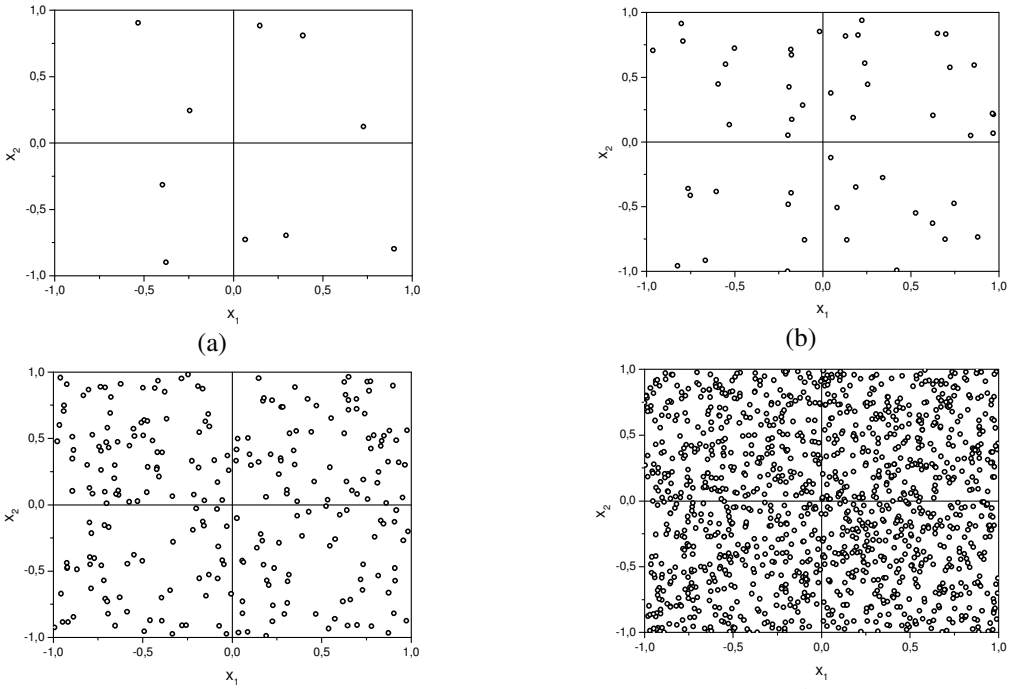


Figura 3.7 - Populações iniciais geradas utilizando algoritmo aleatório. As populações (a), (b), (c) e (d) possuem, respectivamente, 10, 50, 250 e 1250 indivíduos, cujos cromossomos possuem 22 genes.

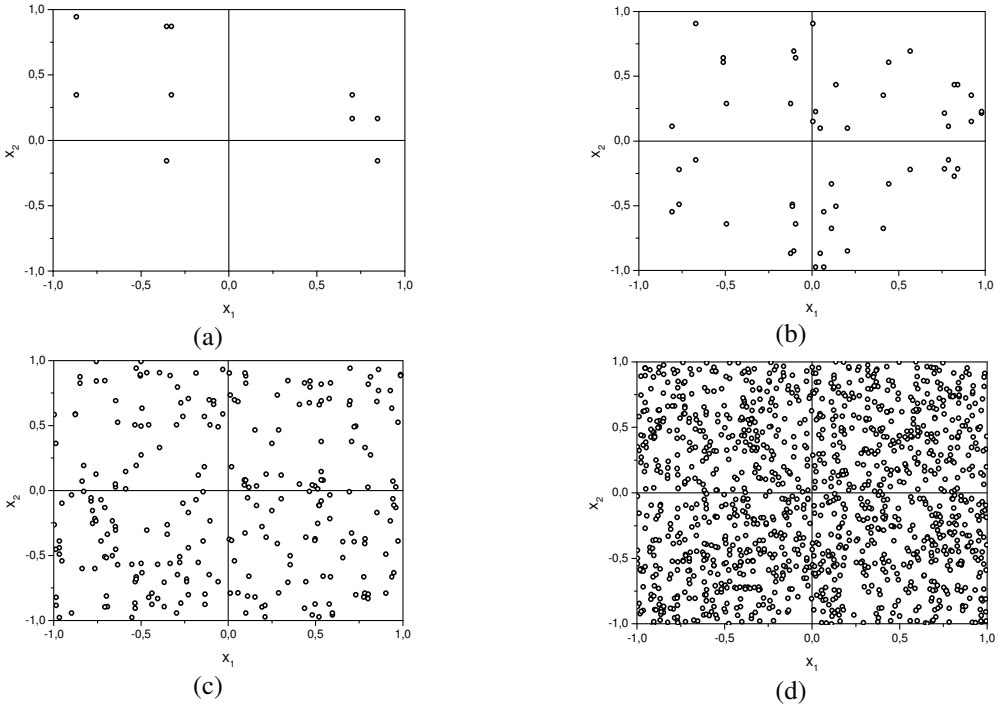
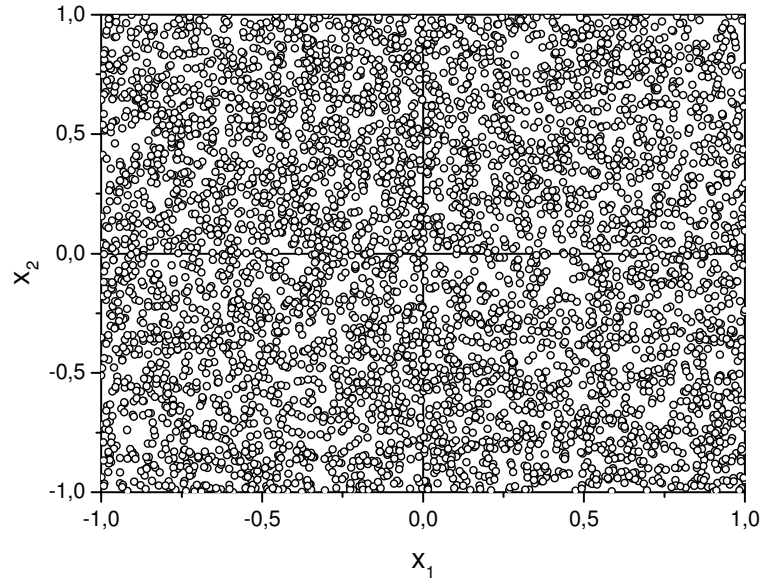


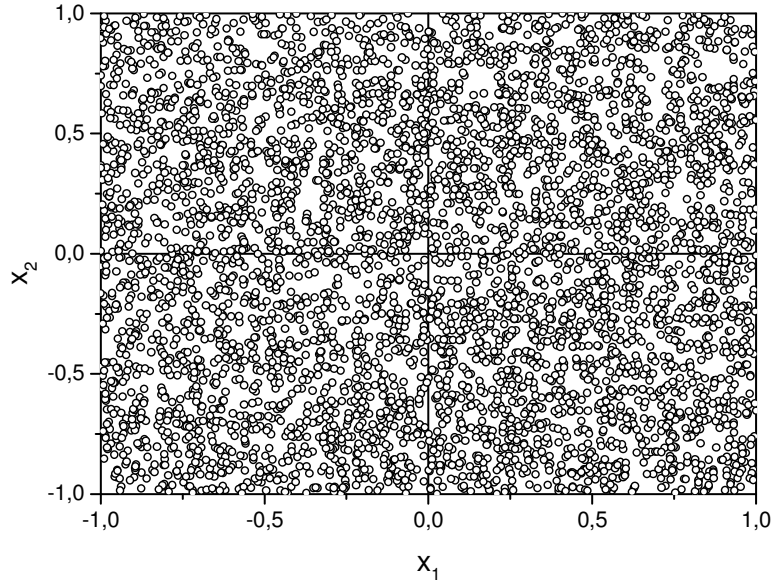
Figura 3.8 - Populações iniciais geradas utilizando algoritmo semi-aleatório. As populações (a), (b), (c) e (d) possuem, respectivamente, 10, 50, 250 e 1250 indivíduos, cujos cromossomos possuem 22 genes.

Adicionalmente, na Figura 3.9 apresenta-se duas populações iniciais, ambas com 5000 indivíduos. A sub figura (a) foi gerada utilizando-se o algoritmo aleatório e a (b) o

semi-aleatório. Visualmente, e também estatisticamente as duas são muito semelhantes. Nota-se distribuição bem uniforme dos indivíduos sobre todo o domínio de definição.



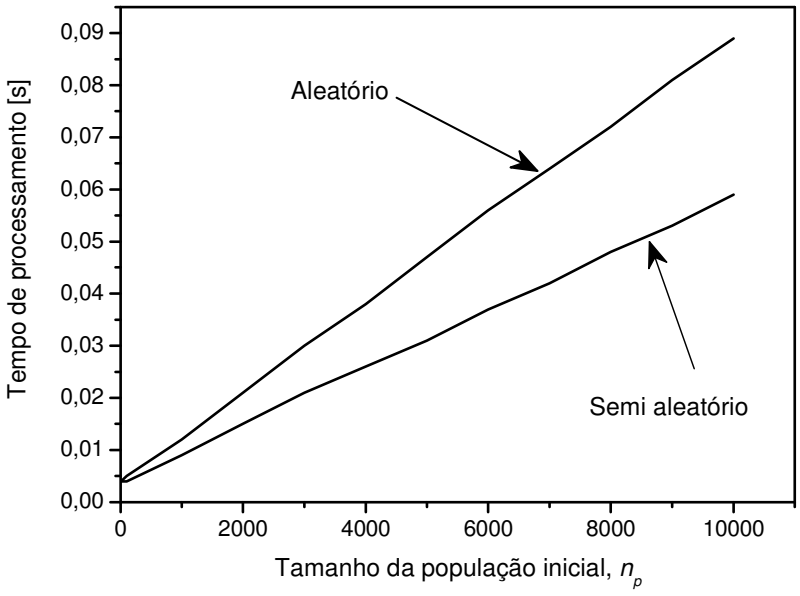
(a)



(b)

Figura 3.9 – Populações iniciais, geradas utilizando os algoritmos: aleatório (a) e semi-aleatório (b). Ambas com 5000 indivíduos com cromossomos que possuem 22 genes.

Também a distância média entre os indivíduos diminui se comparada com aquelas apresentadas nas figuras anteriores para populações menores.



(b)

Figura 3.10 – Tempo para geração de populações iniciais, utilizando os algoritmos aleatório e semi-aleatório. Todos indivíduos com cromossomos de 22 genes.

3.3 Modo de otimização

Nesta seção mostra-se que o algoritmo genético, conforme apresentado no capítulo anterior, pode resolver problemas de maximização e também, via a definição adequada do operador aptidão, pode resolver problemas de minimização. Como todas os problemas teste apresentados anteriormente, neste capítulo, são de minimização, então quando as funções objetivo $f_1(x_1,x_2)$, $f_2(x_1,x_2)$, $f_3(x_1,x_2)$, e $f_4(x_1,x_2)$ são multiplicadas por (-1) as funções objetivo resultantes constituem problemas de maximização.

Adicionalmente, aproveitando os resultados apresentados, comenta-se alguns aspectos característicos dos algoritmos genéticos. Também são comentados os diferentes tipos de figuras utilizadas para apresentar os resultados.

Na Figura 3.11 apresenta-se os resultados dos valores mínimos, médios e máximos, para cinco rodadas, correspondentes à minimização da função objetivo $f_1(x_1,x_2)$. Os valores mínimos são aqueles atingidos pelo indivíduo mais apto a cada geração, não necessariamente o mesmo indivíduo. Os valores máximos correspondem ao indivíduo menos apto a cada geração. Os valores médios são aqueles obtidos efetuando-se a média aritmética da função objetivo para todos os indivíduos da população. Observando-se a Figura 3.11 nota-se que as curvas correspondentes aos valores médios e máximos apresentam aspecto estocástico. Isto se deve ao fato de que na definição dos operadores do algoritmo genético existem muitas

operações estocásticas. Os valores mínimos apresentam-se como uma função monótona devido ao fato de que as cinco rodadas apresentadas foram realizadas com elitismo, sendo a taxa de elitismo igual a 1%. Como o elitismo não permite, neste caso, que o indivíduo mais apto seja mutado ou descartado o valor mínimo da função terá um decréscimo monótono à medida que se aumenta o número de gerações. Nota-se também nesta figura que os valores médios estão mais próximos do mínimo do que dos valores máximos, significando que boa parte dos indivíduos da população estão, razoavelmente, próximos do valor mínimo, ou seja, da solução.

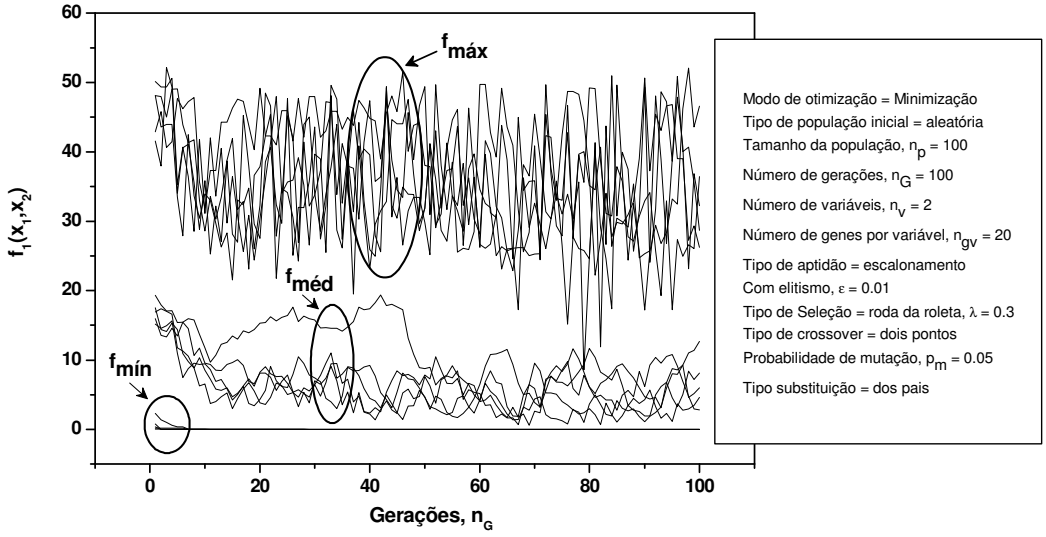


Figura 3.11 – Valores mínimo, médio e máximo da função objetivo, $f_1(x_1, x_2)$, em função do número de gerações, para cinco rodadas do algoritmo genético.

A Figura 3.12 representa os mesmos dados apresentados na Figura 3.11, porém o eixo das ordenadas está em escala logarítmica. Neste caso o principal efeito é ressaltar o comportamento dos valores mínimos atingidos pelo algoritmo genético. Nota-se que para diferentes rodadas, há sempre o decréscimo monótono dos valores mínimos da função objetivo. Como o algoritmo genético tem natureza estocástica, então para cada rodada têm-se curvas distintas umas das outras, porém com padrões de comportamento semelhantes.

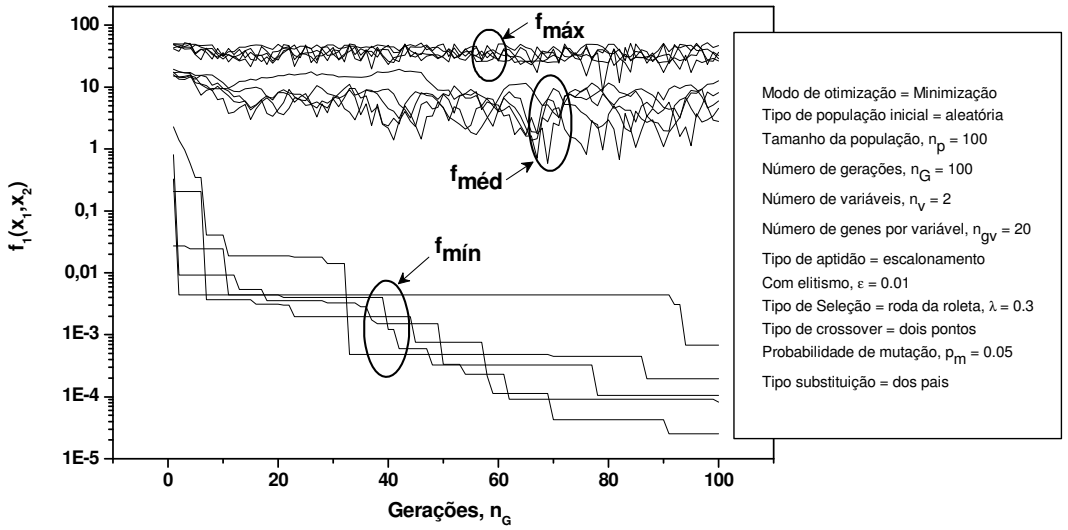


Figura 3.12 - Valores mínimo, médio e máximo da função objetivo, $f_1(x_1, x_2)$, com escala logarítmica, em função do número de gerações, para cinco rodadas do algoritmo genético.

Apresenta-se na Figura 3.13 os resultados correspondentes à rodada que produziu o menor valor mínimo para a função objetivo, dentre a cinco rodadas apresentadas na Figura 3.12. Neste caso, na centésima geração, o valor mínimo da função objetivo foi de $f_1(x_1, x_2) = 0{,}0000255$, correspondente ao indivíduo (0111111111000100011; 0111111111100110111) ou seja $(x_1, x_2) = (-0{,}0046533; 0{,}0019580)$. Note que este indivíduo está a uma distância Euclidiana igual a 0,00505 da solução exata. Ou seja, o algoritmo genético foi capaz de em cem gerações, a partir de uma população inicial de cem indivíduos produzir uma solução aproximante localizada em vizinhança próxima da solução exata. Esclareça-se que todos estes resultados foram obtidos utilizando-se os parâmetros que estão listados na tabela constante no lado direito das figuras. Os parâmetros lá existentes foram definidos com base na experiência adquirida neste trabalho e também de resultados relatados na literatura correlata. Mais adiante neste capítulo serão discutidos todos os parâmetros envolvidos e seus valores serão alterados. No caso das Figuras 3.11 a 3.13 os parâmetros utilizados foram "**modo de otimização**" igual a "**minimização**", indicando que se trata um problema de minimização; o "**tipo de população inicial**" foi "**aleatória**"; o "**tamanho da população**" inicial foi igual cem; o "**número de gerações**" igual a cem; o "**número de variáveis**" foi definido igual a dois porque se trata de um problema definido sobre um domínio bidimensional; o "**número de genes por variável**" foi utilizado igual a vinte; o operador "**aptidão**" foi escolhido como "**escalonamento**"; todas a rodadas desta seção foram realizadas no modo "**elitismo**" com "**taxa de elitismo**" igual a 0,01 ou 1%, significando que havia apenas um indivíduo de elite, ou seja, 1% de 100; o "**tipo de seleção**"

usado foi a "**roda da roleta**" com taxa de seleção de 30%; o "**tipo de *crossover***" foi de "**dois pontos**"; a "**probabilidade de mutação**" igual a 5%; e o "**tipo de substituição**" foi "**dos pais**" pelos filhos.

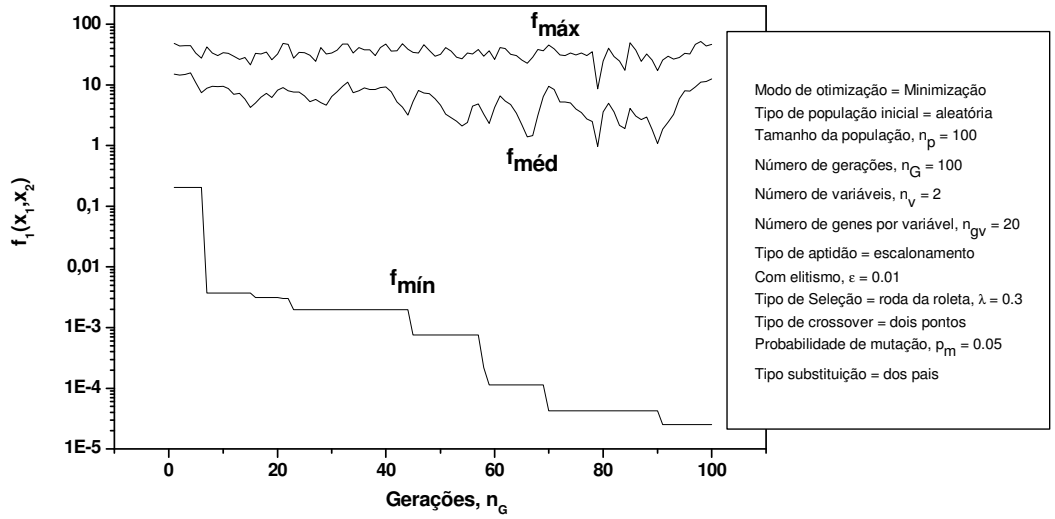


Figura 3.13 – Valores mínimo, médio e máximo da função objetivo, $f_1(x_1,x_2)$, em função do número de gerações, para a melhor rodada das cinco realizadas.

Na Figura 3.14 apresenta-se os resultados para a maximização da função objetivo, $f_1^*(x_1,x_2) = -f_1(x_1,x_2)$. Note que o "**modo otimização**" está com a opção de "**maximização**". Neste caso o indivíduo mais apto é aquele que apresenta os maiores valores para a função objetivo, conseqüentemente, devido ao elitismo, a curva dos valores máximos apresenta aspecto de acréscimo monótono à medida que o número de gerações aumenta. Note que o eixo das ordenadas está invertido para possibilitar a construção da figura com eixo do tipo logarítmico. Os resultados apresentados correspondem à rodada que produziu o maior valor para a função objetivo $f_1^*(x_1,x_2) = -0{,}0000042$ correspondente ao indivíduo (0111111111100101110; 1000000000000000101) ou seja $(x_1,x_2) = (-0{,}0020459;\ 0{,}0000537)$. A distância Euclidiana desta solução aproximante até a solução exata é de 0,00205. Note-se ainda que o comportamento quantitativo e qualitativo da minimização de $f_1(x_1,x_2)$, Figura 3.13, e da maximização de $f_1^*(x_1,x_2) = -f_1(x_1,x_2)$, Figura 3.14, são muito semelhantes, e levaram ambos a soluções aproximantes que estão na vizinhança próxima da solução exata.

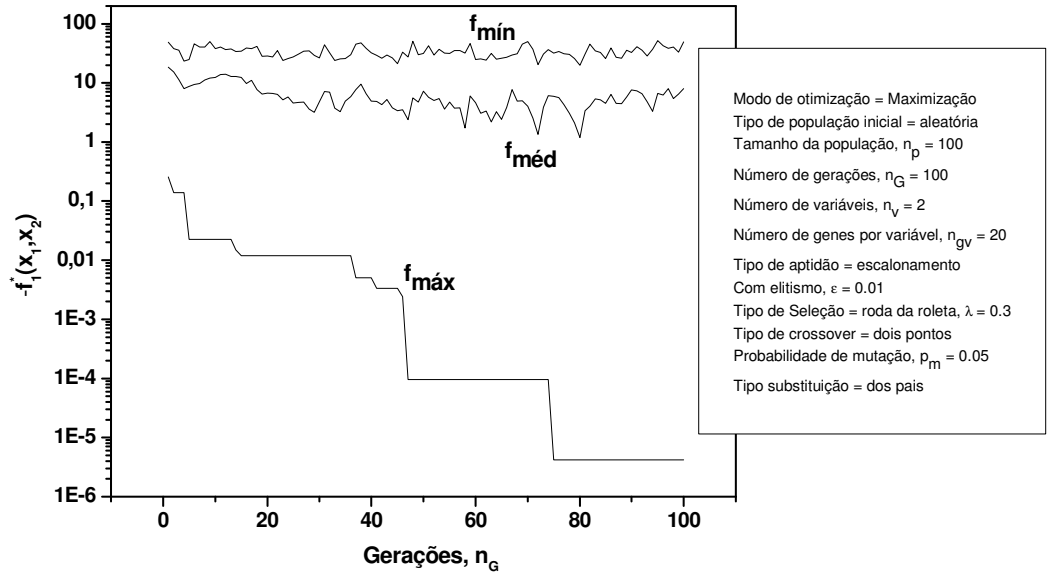


Figura 3.14 – Valores mínimo, médio e máximo da função objetivo, $f_1^*(x_1, x_2) = -f_1(x_1, x_2)$, em função do número de gerações, para a melhor rodada das cinco realizadas.

São apresentados, Figura 3.15, os resultados correspondentes ao indivíduo mais apto, para a minimização da função objetivo, $f_2(x_1, x_2)$. Estes resultados correspondem a cinco rodadas. Novamente, devido ao uso do "**modo elitismo**" as curvas apresentadas, correspondentes ao indivíduo mais apto, têm comportamento monótono com o aumento do número de gerações. Na rodada que levou ao melhor resultado o valor da função objetivo foi $f_2(x_1, x_2) = 0{,}0001404$, associado ao indivíduo (10111110110010001111; 10111111001000111100), ou seja, $(x_1, x_2) = (1{,}0045616; 1{,}0102373)$. Esta solução aproximante encontrada está à distância Euclidiana de 0,0112 da solução exata.

Na Figura 3.16 estão resultados qualitativamente semelhantes aos da Figura 3.15, porém obtidos via a maximização da função objetivo $f_2^*(x_1, x_2) = -f_2(x_1, x_2)$. Também se confirma, neste caso, que o algoritmo genético apresentado mostra-se apto a resolver problemas de otimização multidimensionais de minimização, bem como de maximização.

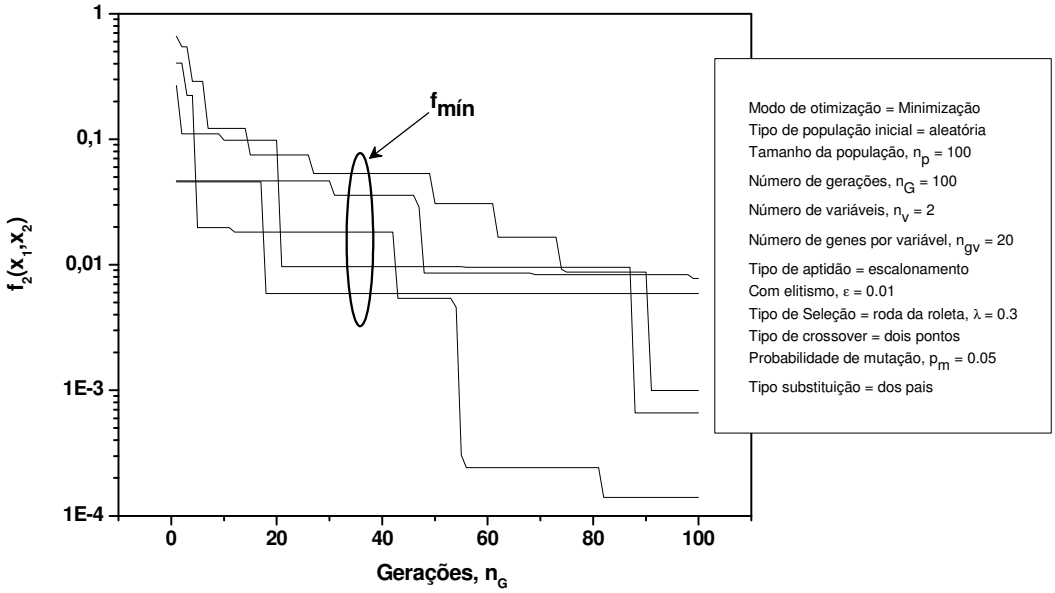


Figura 3.15 – Valores mínimos da função objetivo, $f_2(x_1,x_2)$, correspondentes a cinco rodadas em função do número de gerações.

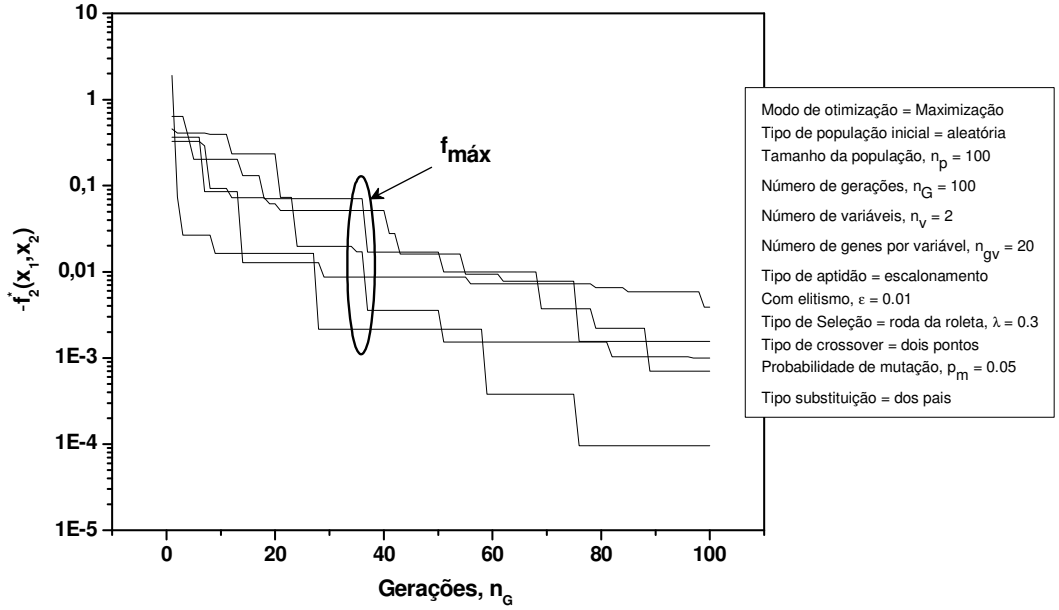


Figura 3.16 – Valores máximos da função objetivo, $f_2^*(x_1,x_2) = -f_2(x_1,x_2)$, correspondentes a cinco rodadas, em função do número de gerações.

Nas Figuras 3.17 e 3.18 são apresentados os resultados correspondentes à minimização da função $f_3(x_1,x_2)$ e à maximização da função $f_3^*(x_1,x_2) = -f_3(x_1,x_2)$. Neste caso também obtêm-se valores aproximantes para a solução da otimização bidimensional tanto via minimização quanto via maximização. Note que as funções objetivo $f_3(x_1,x_2)$ e $f_3^*(x_1,x_2)$ são funções degrau bidimensional e portanto de classe C^0. Esta classe de problema apresenta solução difícil ou quase-impossível via métodos do gradiente, porém utilizando-se algoritmo

genético a solução convergiu para o valor exato da função objetivo em, relativamente, poucas iterações. Isso é um indicativo da robustez da metodologia.

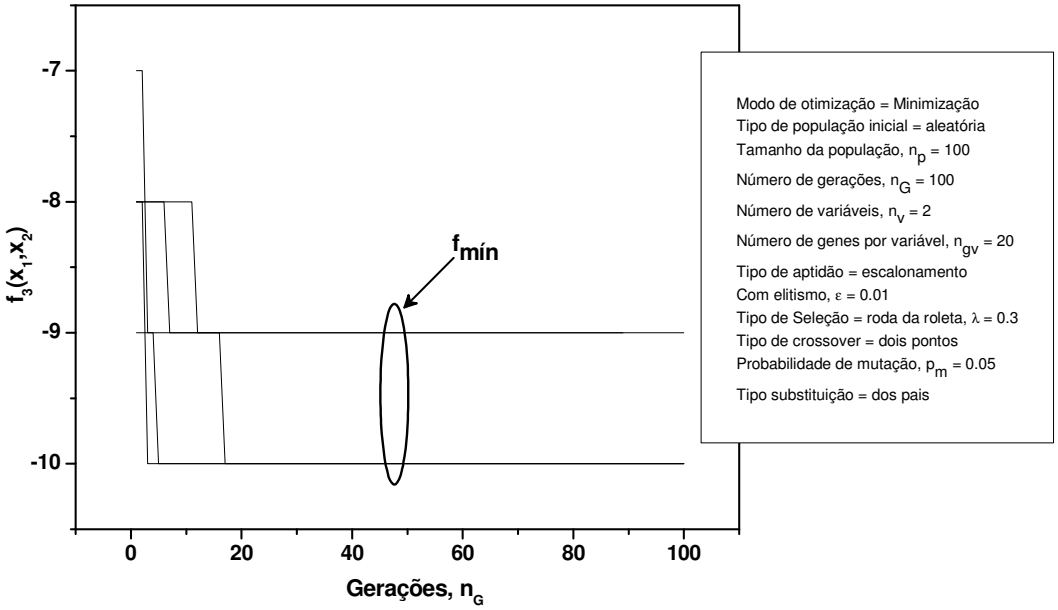


Figura 3.17 – Valores mínimos da função objetivo, $f_3(x_1,x_2)$, correspondentes a cinco rodadas em função do número de gerações.

Devido à natureza destas funções objetivo seus valores vão decrescendo monotonicamente, em degrau, quando a otimização é do tipo minimização e vão crescendo monotonicamente, também em degrau, quando a otimização é do tipo maximização.

A função objetivo $f_4(x_1,x_2)$ tem em sua definição uma parte determinística $(x_1^4+2x_2^4)$ e uma parte estocástica $(Gauss(0,1))$. Como as duas partes são adicionadas, então a função $f_4(x_1,x_2)$ é estocástica e portanto seu valor varia, estocasticamente, para um mesmo ponto toda vez que a função é calculada. Isto implica que esta função não possui um ponto estacionário fixo. A minimização da parte determinística da função objetivo produz a solução exata (0;0), assim a solução da função estocástica será um conjunto de pontos que estarão na vizinhança de (0;0) e os valores da função serão os menores valores obtidos para a distribuição gaussiana, $Gauss(0,1)$.

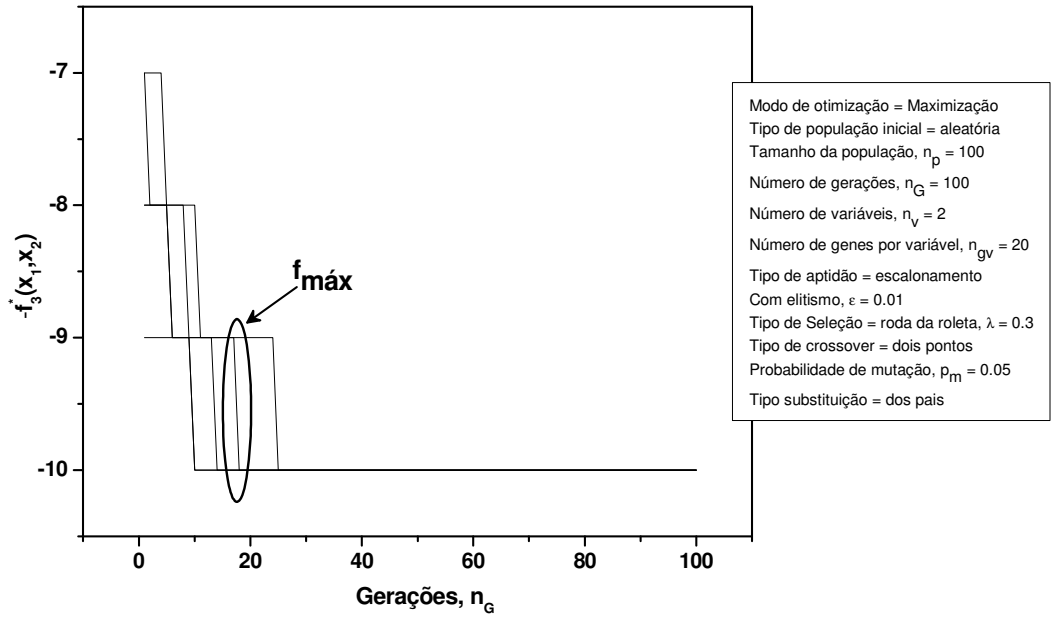


Figura 3.18 - Valores máximos da função objetivo, $f_3^*(x_1,x_2) = -f_3(x_1,x_2)$, correspondentes a cinco rodadas, em função do número de gerações.

Na Figura 3.19 apresenta-se os valores mínimos da função objetivo $f_4(x_1,x_2)$. Note que mesmo utilizando elitismo os valores não apresentam o comportamento de decréscimo monótono que houve na solução da minimização das funções objetivo $f_1(x_1,x_2)$, $f_2(x_1,x_2)$ e $f_3(x_1,x_2)$. No caso da função $f_4(x_1,x_2)$, o comportamento apresentado pelo indivíduo mais apto deve-se ao fato de que a função é estocástica e portanto está sempre variando. Neste caso o uso do elitismo não produz efeito benéfico perceptível. Pode-se afirmar, baseado nesta evidência, que provavelmente, quando a função objetivo apresenta caráter estocástico ou mesmo determinístico transiente o mecanismo de elitismo não traz benefício perceptível. Na próxima seção discute-se um pouco a questão do elitismo.

Na Figura 3.21(a) são apresentados os últimos vinte e cinco indivíduos de uma dada rodada. Nota-se que formam um conjunto de pontos distribuídos na vizinhança da solução exata (0;0) da parte determinística do problema de otimização. Percebe-se que o conjunto de pontos (indivíduos) está achatada na direção do eixo x_2. Isso deve-se ao fato de que o coeficiente, da parte determinística, que multiplica x_2^4 é 2 e o que multiplica x_1^4 é 1. Ou seja, os indivíduos estão, estocasticamente, concentrados no vale alongado que cerca o ponto estacionário da solução da parte determinística.

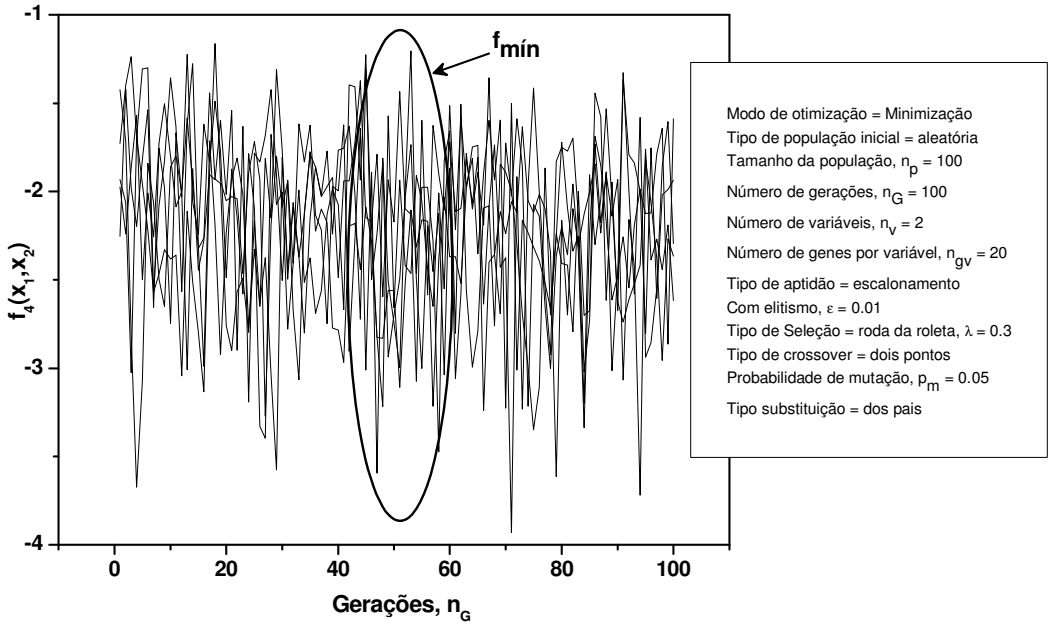


Figura 3.19 - Valores mínimos da função objetivo, $f_4(x_1,x_2)$, correspondentes a cinco rodadas em função do número de gerações.

Nas Figuras 3.20 e 3.21(b) são apresentados resultados para a maximização da função objetivo $f_4^*(x_1,x_2) = -f_4(x_1,x_2)$, qualitativamente semelhantes àqueles apresentados nas Figuras 3.19 e 3.21(a) para a minimização de $f_4(x_1,x_2)$. Os valores máximos da função objetivo $f_4^*(x_1,x_2)$, Figura 3.20 também apresentam caráter estocástico, e os vinte e cinco últimos pontos (indivíduos) de uma dada rodada, Figura 3.21(b), também estão na vizinhança da solução exata da parte determinística. Semelhante ao caso da minimização, no caso da maximização também o elitismo não apresentou melhoria considerável nos resultados. Os indivíduos obtidos via maximização da função $f_4^*(x_1,x_2)$ também estão localizados no vale achatado na direção x_2 ao redor a solução exata da parte determinística da função.

Conclui-se também, neste caso de uma função objetivo estocástica, que a otimização pode ser realizada tanto via minimização quanto via maximização, dependendo naturalmente da definição adequada e correta das funções objetivo a serem usadas em um caso e no outro.

Finalmente, nesta seção, para os quatro tipos de funções analisadas foi possível a obtenção de resultados coerentes e corretos, dentro de certa acurácia, da otimização das funções bidimensionais apresentadas tanto via minimização quanto via maximização. Ambas maneiras funcionam adequadamente e podem ser usadas indistintamente.

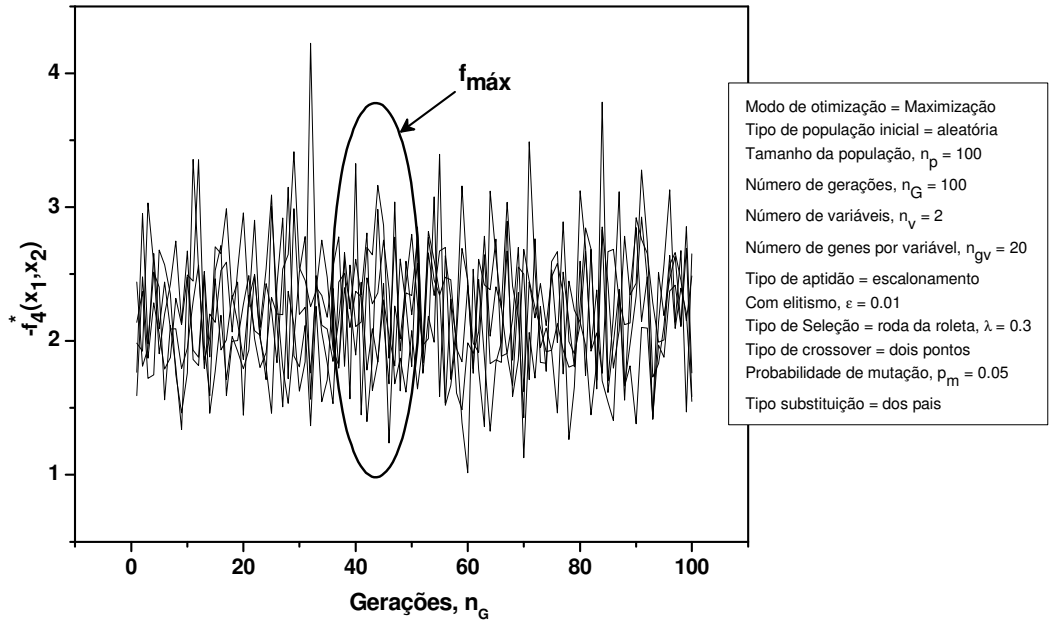
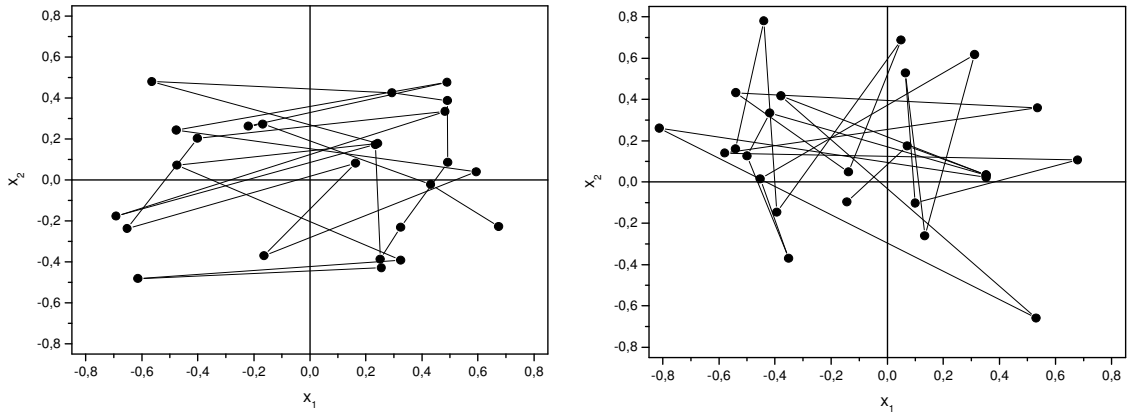


Figura 3.20 - Valores máximos da função objetivo, $f_4^*(x_1,x_2) = -f_4(x_1,x_2)$, correspondentes a cinco rodadas, em função do número de gerações.



(a) (b)

Figura 3.21 – Últimos vinte e cinco indivíduos de uma determinada rodada, correspondentes às funções objetivo: (a) $f_4(x_1,x_2)$ e (b) $f_4^*(x_1,x_2) = -f_4(x_1,x_2)$.

3.4 Modo de elitismo

Na seção anterior sobre o **modo otimização** tangenciou-se a questão do elitismo. Naquela seção todos os resultados apresentados foram obtidos utilizando-se elitismo. Na otimização das funções objetivo $f_1(x_1,x_2)$, $f_1^*(x_1,x_2)$, $f_2(x_1,x_2)$, $f_2^*(x_1,x_2)$, $f_3(x_1,x_2)$ e $f_3^*(x_1,x_2)$ o elitismo funcionou adequadamente. Porém na otimização das funções

$f_4(x_1, x_2)$, e $f_4^*(x_1, x_2)$ o elitismo não foi eficiente na obtenção de solução estável, devido à natureza estocástica do problema.

Nesta seção analisa-se mais detidamente aspectos da utilização de elitismo em otimização bidimensional. Na Figura 3.22 são apresentados os resultados dos valores mínimos da função $f_1(x_1, x_2)$ obtidos executando dois casos: um com elitismo, com taxa de 1%; e outro sem elitismo. Nota-se claramente da figura que a solução com elitismo apresenta decréscimo monótono da função objetivo, com o aumento das gerações, enquanto que a solução sem elitismo apresenta comportamento oscilante dos valores mínimos da função. Adicionalmente, o resultado com elitismo converge mais rapidamente para a solução do que a solução sem elitismo.

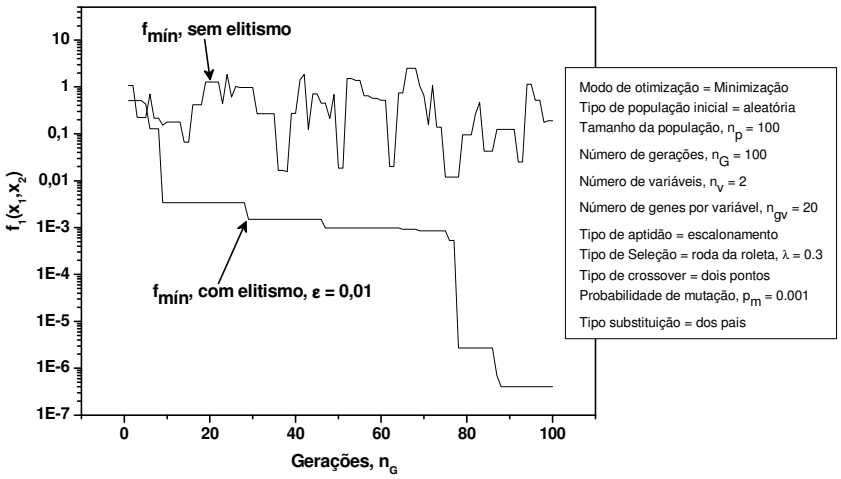


Figura 3.22 – Valores mínimos da função objetivo $f_1(x_1, x_2)$ para duas rodadas, uma com elitismo e outra sem.

Nas Figuras 3.23 e 3.24 apresenta-se resultados para a minimização das funções objetivo $f_2(x_1, x_2)$ e $f_3(x_1, x_2)$ semelhantes àqueles apresentados na Figura 3.22 correspondentes à minimização de $f_1(x_1, x_2)$. Nos casos das funções $f_2(x_1, x_2)$ e $f_3(x_1, x_2)$ também o uso do elitismo leva a resultados claramente melhores do que os obtidos sem o uso de elitismo: solução não oscilante e maior taxa de convergência.

Nos casos aqui apresentados o elitismo provê melhores resultados porque o indivíduo mais apto fica protegido da modificação genética das mutações e também não pode ser eliminado no processo de substituição; enquanto os indivíduos menos aptos obtidos sem o uso de elitismo têm seus alelos alterados pela mutação ou são removidos da população pelo operador de substituição.

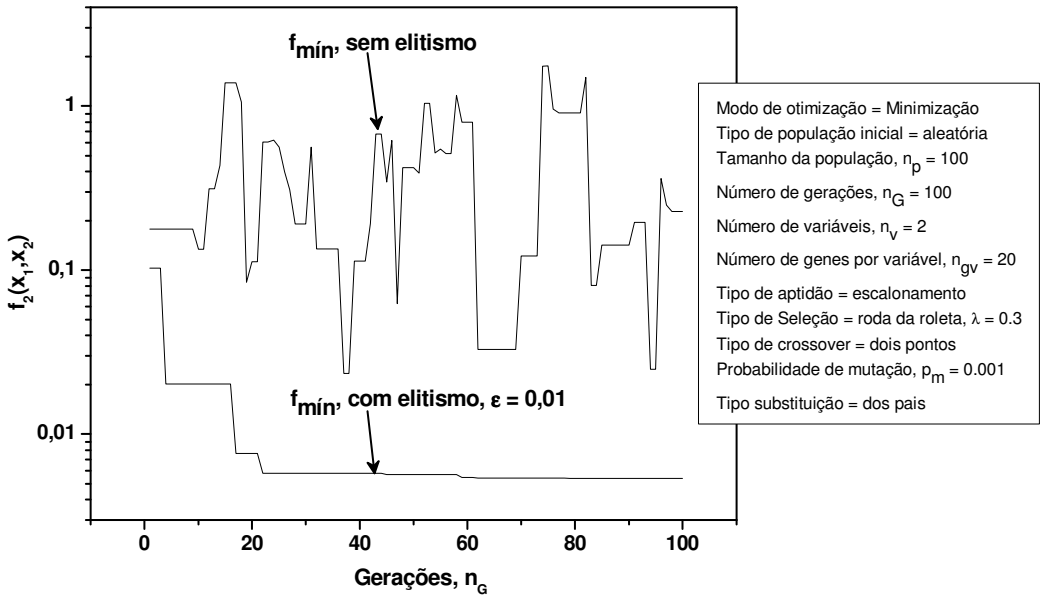


Figura 3.23 – Valores mínimos da função objetivo $f_2(x_1, x_2)$ para duas rodadas, uma com elitismo e outra sem.

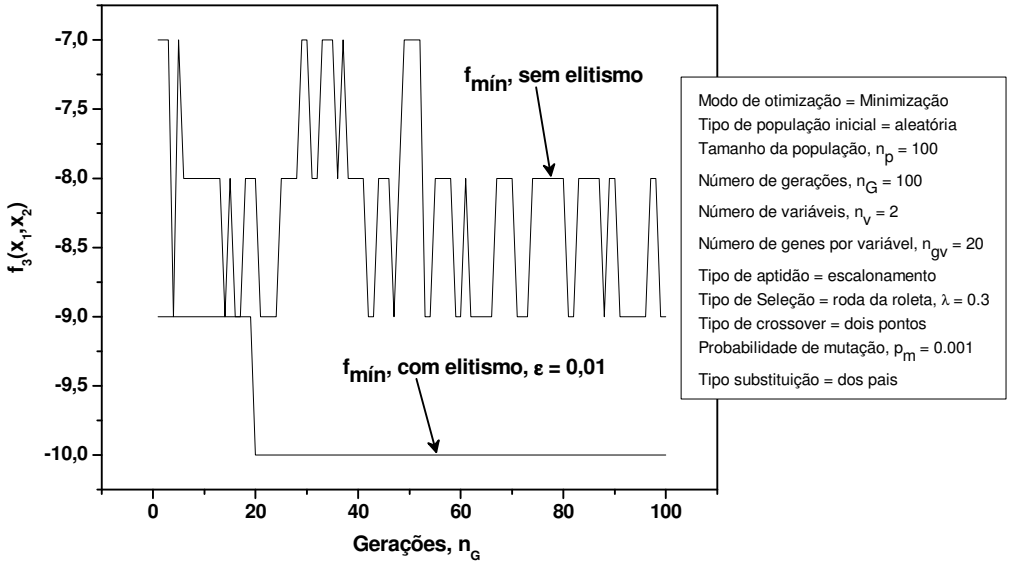


Figura 3.24 – Valores mínimos da função objetivo $f_3(x_1, x_2)$ para duas rodadas, uma com elitismo e outra sem.

Na minimização da função $f_4(x_1, x_2)$, no entanto, o elitismo não opera tão bem. Em realidade os resultados obtidos com e sem elitismo são quase-equivalentes, Figura 3.25, embora mesmo neste caso os resultados obtidos com elitismo são na média ligeiramente melhores do que os obtidos sem elitismo. Este fato pode ser observado na Figura 3.25 na qual a curva contínua, correspondente ao uso de elitismo, está na maioria das vezes abaixo da curva pontilhada correspondente à ausência de elitismo.

Dos resultados aqui apresentados conclui-se que para os problemas teste o uso de elitismo leva a melhores resultados para problemas de otimização que sejam bem definidos e

determinísticos; enquanto não apresenta vantagem clara sobre problema de otimização na qual as funções objetivo é estocástica.

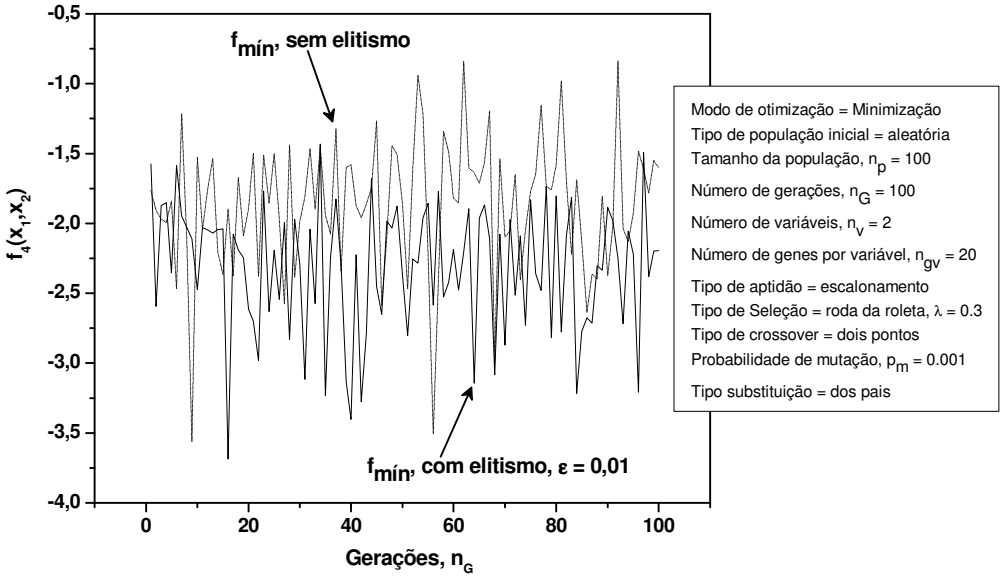


Figura 3.25 – Valores mínimos da função objetivo $f_4(x_1,x_2)$ para duas rodadas, uma com elitismo e outra sem.

3.5 Taxa de elitismo

Nas seções sobre **modo de otimização** e **modo elitismo** concluiu-se que para funções objetivo determinísticas o uso de elitismo favorecia a obtenção de melhores resultados quando comparados com os obtidos sem elitismo. Todos os resultados, com elitismo, apresentados até o momento, eram para taxa de elitismo $\varepsilon = 0{,}01$ (1%). Então, uma questão que surge é: Como as soluções se comportam para diferentes taxa de elitismo? Procura-se, nesta seção, responder, parcialmente, a esta questão.

Como as funções $f_1(x_1,x_2)$, $f_2(x_1,x_2)$ são de classe C^{∞}, e os resultados apresentados nas duas seções anteriores são qualitativamente semelhantes, apresenta-se nesta seção apenas os resultados correspondentes às funções $f_2(x_1,x_2)$, $f_3(x_1,x_2)$ e $f_4(x_1,x_2)$.

Na Figura 3.26 apresenta-se os resultados da minimização da função objetivo $f_2(x_1,x_2)$ com cinco rodadas para diferentes taxas de elitismo: 1%, 5%, 10% e 50%. Nota-se que o melhor resultado foi obtido para a taxa de elitismo de 1%. O segundo melhor resultado para a taxa de 5%. Para maiores valores da taxa de elitismo os resultados foram piores. Para a função $f_1(x_1,x_2)$ os resultados, não apresentados aqui, foram qualitativamente semelhantes.

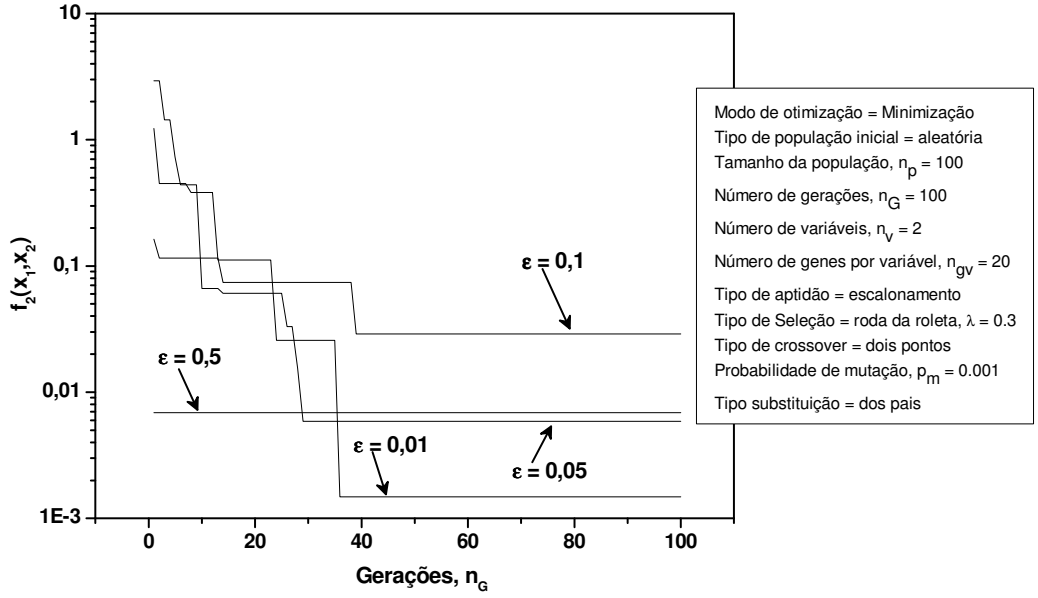


Figura 3.26 – Valores mínimos da função objetivo $f_2(x_1,x_2)$ para diferentes taxas de elitismo.

Para a função degrau $f_3(x_1,x_2)$ as taxas de elitismo, Figura 3.27, que apresentaram os melhores resultados foram 1% e 5%. A taxa de elitismo de 10% também levou ao mesmo resultado final que as de 1% e 5%, porém com menor taxa de convergência. Os piores resultados foram obtidos utilizando-se a taxa de elitismo de 50%.

Conclui-se do exposto que para as funções $f_1(x_1,x_2)$, $f_2(x_1,x_2)$ e $f_3(x_1,x_2)$ a taxa de elitismo ideal é razoavelmente pequena, menor que 10%. Para problemas de otimização com características semelhantes àquelas das referidas funções as taxa de elitismo ótimas serão semelhantes. Provavelmente, os valores ótimos para a taxa de elitismo sejam dependentes do problema em foco, mas serão relativamente pequenos. Quando se procura apenas uma solução, basta que haja na população apenas um indivíduo de elite. Também se pode afirmar que altos valores da taxa de elitismo são inúteis e até prejudiciais, pois, a função do elitismo é proteger de alteração os indivíduos mais aptos. Com alta taxa de elitismo serão protegidas parcelas de indivíduos que não são tão aptos. Adicionalmente, a existência de muitos indivíduos de elite irá inibir, em excesso, a ação dos operadores **mutação** e **substituição**, impedindo uma evolução mais adequada da população e implicando em baixa convergência do algoritmo genético.

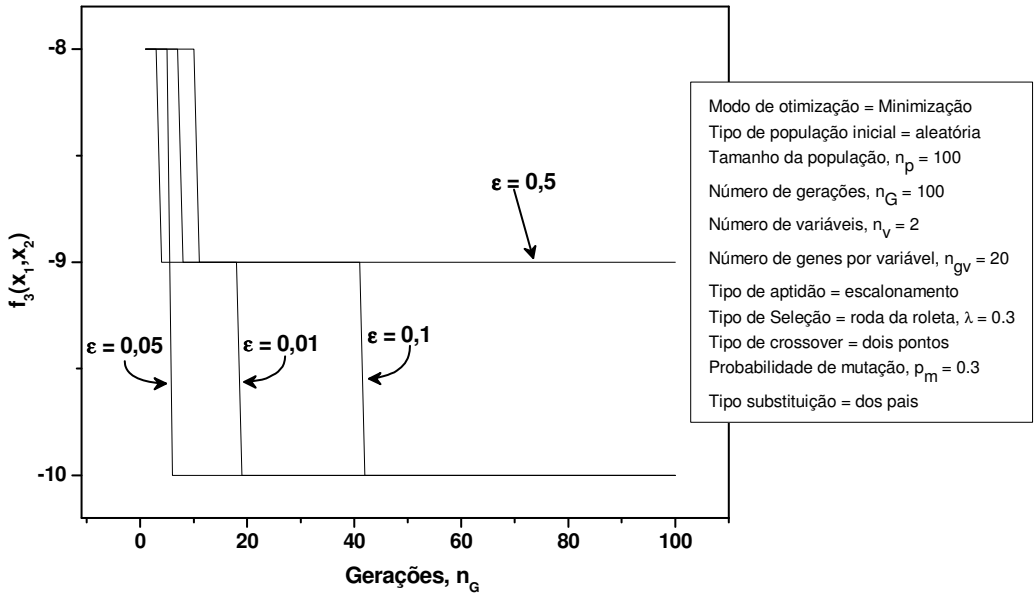


Figura 3.27 – Valores mínimos da função objetivo $f_3(x_1,x_2)$ para diferentes taxas de elitismo.

Conforme mencionado anteriormente o elitismo não melhora os resultados da minimização da função $f_4(x_1,x_2)$, em relação aos obtidos sem a utilização de elitismo. Percebe-se também, Figura 3.28, que o algoritmo genético quando aplicado a funções estocásticas é insensível quanto à taxa de elitismo. Estatisticamente, os resultados são semelhantes para as diferentes taxas: 1%, 5%, 10% e 50%. Neste caso um dado indivíduo de elite deixará de sê-lo em poucas gerações, ou mesmo em uma geração e adicionalmente os valores da função objetivo irão variar estocasticamente. Muito provavelmente, problemas determinísticos cujas respectivas funções objetivo sejam variáveis ao longo das gerações também apresentarão certa insensibilidade ao elitismo.

Finalmente, conclui-se que o elitismo quando usado com moderação, ou seja, com baixas taxas de elitismo, será benéfico na solução de problemas de otimização determinísticos e será neutro na solução de problemas estocásticos.

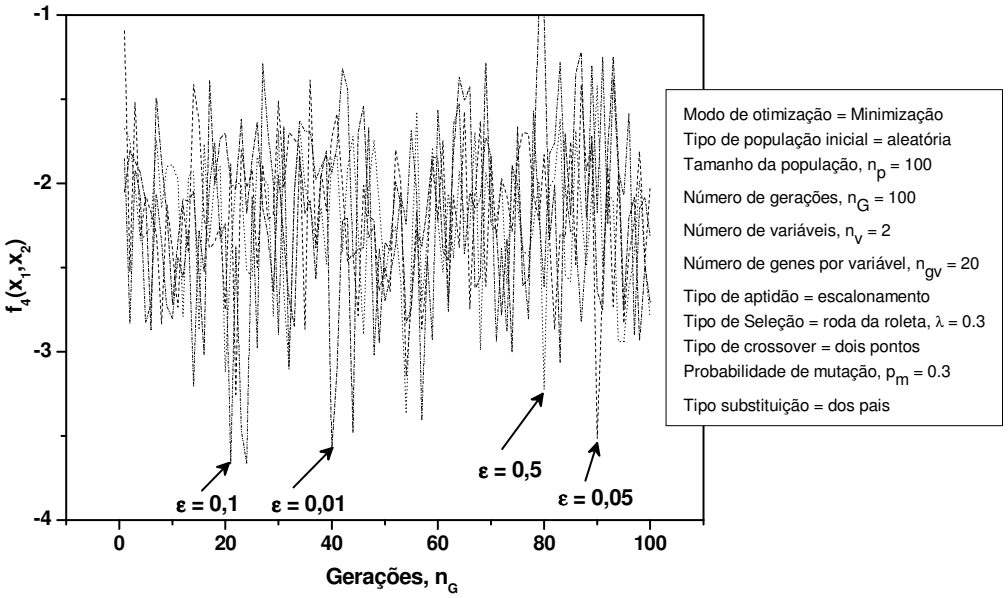


Figura 3.28 – Valores mínimos da função objetivo $f_4(x_1, x_2)$ para diferentes taxas de elitismo.

3.6 Tamanho da população inicial

Da teoria e também da prática de técnicas clássicas de otimização, principalmente aquelas baseadas no gradiente da função, sabe-se que o ponto inicial de busca é muito importante. A otimização de uma dada função objetivo pode convergir rapidamente a partir de um determinado ponto inicial, bem como poderá convergir lentamente a partir de um outro ponto inicial.

Nos algoritmos genéticos ao invés de um ponto inicial (um indivíduo) existe uma população de indivíduos candidatos à solução, assim, por analogia, especula-se que para populações iniciais maiores e mais bem postas a solução convergirá mais rapidamente, em termos de iterações.

Percebe-se na Figura 3.29, correspondente à função objetivo $f_2(x_1, x_2)$, que a convergência da solução aumenta à medida que o tamanho para população inicial, n_p, também aumenta. Ao se variar o tamanho da população de 10 para 50 e depois para 250 indivíduos, a solução sempre melhorou. Quando aumentou-se o tamanho da população de 250 para 1250 a vantagem já não foi tão significativa, pois, as respectivas curvas cruzaram-se quatro vezes, embora ao final os resultados para 1250 indivíduos foi melhor. No entanto, se continuasse o processamento para mais gerações, talvez, os resultados obtidos com 250 indivíduos pudessem se tornar melhores novamente, uma vez que as duas curvas já haviam se cruzado

algumas vezes.

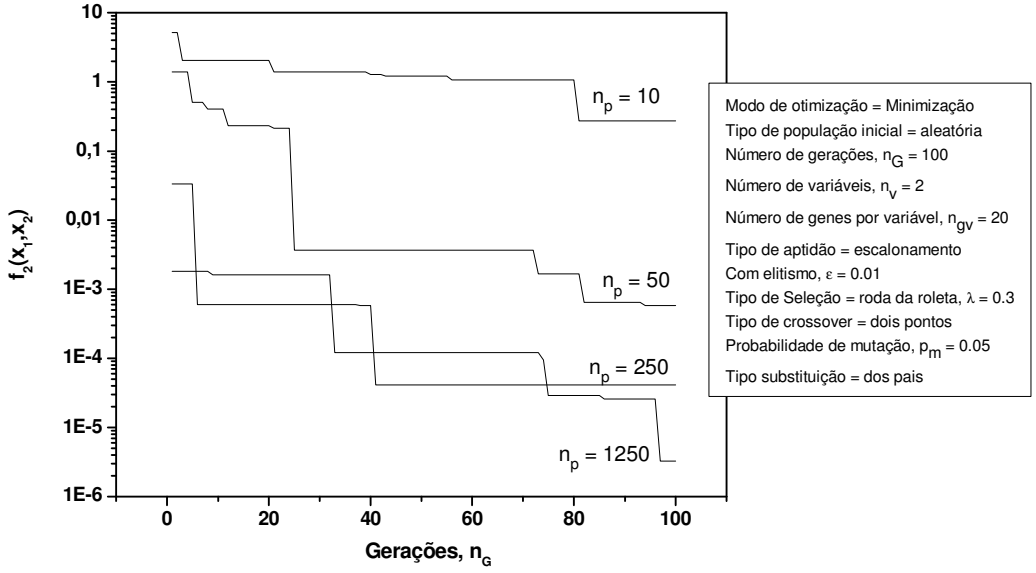


Figura 3.29 – Valores mínimos da função objetivo $f_2(x_1,x_2)$ para diferentes tamanhos da população.

Dos dados já analisados neste capítulo apenas o aumento do tamanho da população foi capaz de produzir um discernimento claro de resultados para a função $f_4(x_1,x_2)$, conforme mostrado na Figura 3.30.

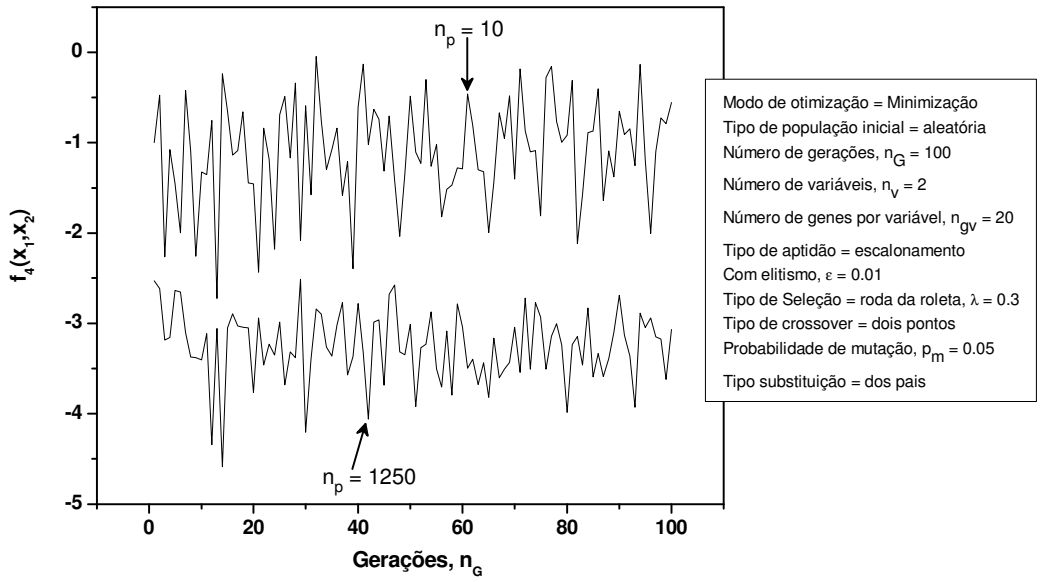
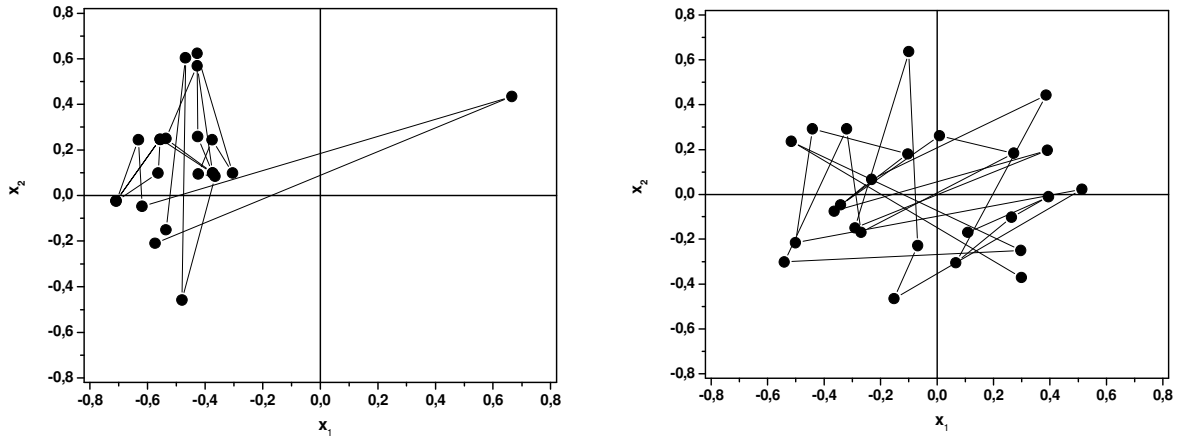


Figura 3.30 – Valores mínimos da função objetivo $f_4(x_1,x_2)$ para diferentes tamanhos da população.

Neste caso a curva correspondente à população com 10 indivíduos está bem separada daquela correspondente a 1250. As duas curvas não se cruzam em nenhuma ocasião. As

curvas correspondentes a 50 e 250 indivíduos foram omitidas para melhor clareza na visualização dos resultados.

As "órbitas" dos vinte e cinco últimos indivíduos mais aptos obtidos para a função objetivo $f_4(x_1, x_2)$ utilizando diferentes quantidades de indivíduos na população: 10 e 1250 indivíduos, são mostradas na Figura 3.31, (a) e (b), respectivamente. A otimização da parte determinística de $f_4(x_1, x_2)$, que é $\{x_1^4 + 2x_2^4\}$, produziria solução de mínimo no ponto (0,0) cuja contribuição para o valor da função objetivo seria nulo. Por outro lado a função gaussiana {Gauss(0,1)} é estocástica e possui média nula. Assim, as "órbitas" dos indivíduos mais aptos deveriam ocorrer na vizinhança do ponto (0,0). Das Figuras 3.31(a) e 3.31(b) a que claramente provê melhores resultados é aquela correspondente à população com 1250 indivíduos, concordando também com os correspondentes resultados na Figura 3.30.



(a) (b)

Figura 3.31 – Últimos vinte e cinco indivíduos mais aptos de cada uma de duas determinadas rodadas, correspondentes à função objetivo $f_4(x_1, x_2)$ e diferentes tamanhos da população: (a) 10 e (b) 1250 indivíduos.

3.7 Tipos de população inicial

Conforme descrito anteriormente, neste capítulo, utilizou-se dois tipos de algoritmos para geração da população inicial: o **aleatório** no qual todos os alelos dos genes dos indivíduos são gerados aleatoriamente; e o **semi-aleatório** no qual metade, ou aproximadamente metade, dos indivíduos são gerados aleatoriamente e os restantes são gerados através de operação aritmética definida adequadamente sobre o conjunto dos números binários. Na Figura 3.32 são apresentados os resultados, referentes à função $f_2(x_1, x_2)$, para

dez corridas do algoritmo genético, sendo cinco utilizando populações iniciais aleatórias e outras cinco utilizando populações iniciais semi-aleatórias. Observa-se que há muitos cruzamentos entre as curvas indicando que os resultados são quase-equivalentes, porém os dois melhores resultados foram obtidos utilizando-se populações iniciais aleatórias e os dois piores foram obtidos usando populações iniciais semi-aleatórias. Parece haver ligeira vantagem no uso de populações inicias **aleatórias**. Conceitualmente, as populações iniciais aleatórias possuem maior diversidade genética, pois, seus alelos são todos gerados aleatoriamente, enquanto no caso das populações iniciais **semi-aleatórias** haverá certo nível de correlação entre parte dos alelos, devido ao algoritmo algébrico usado na geração de parte dos indivíduos da população.

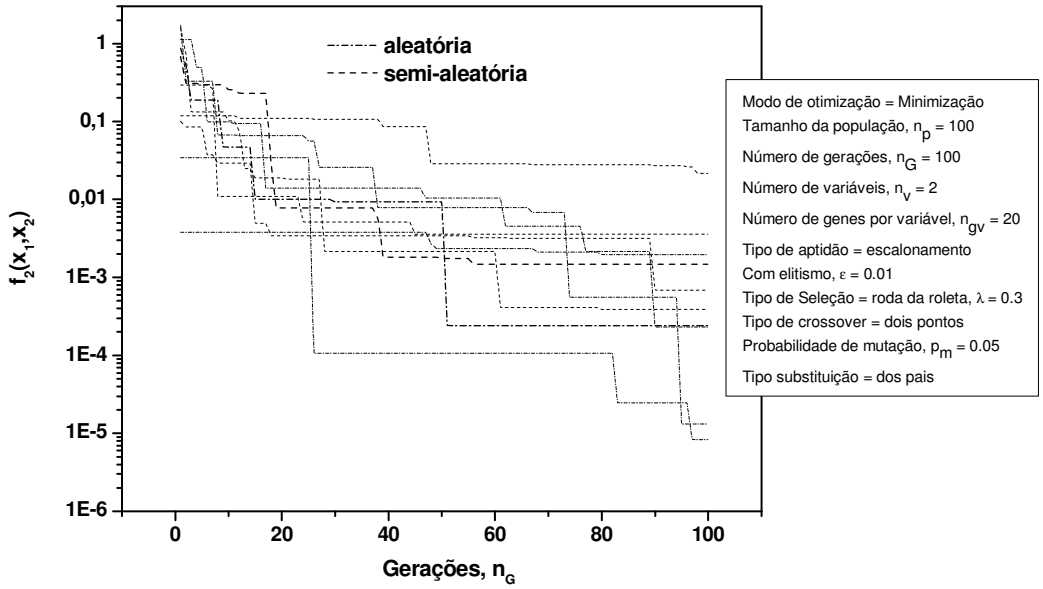


Figura 3.32 – Valores mínimos da função objetivo $f_2(x_1, x_2)$ para diferentes tipos de população inicial.

No caso da minimização da função objetivo $f_4(x_1, x_2)$ o modo de geração da população inicial não produz qual diferenciação observável, como pode ser visto na Figura 3.33. As duas curvas correspondentes aos dois tipos de populações iniciais cruzam-se freqüentemente e adicionalmente também não apresentam decréscimo continuado no valor da função objetivo com o aumento do número de gerações, ao contrário apresentam comportamento oscilatório estocástico.

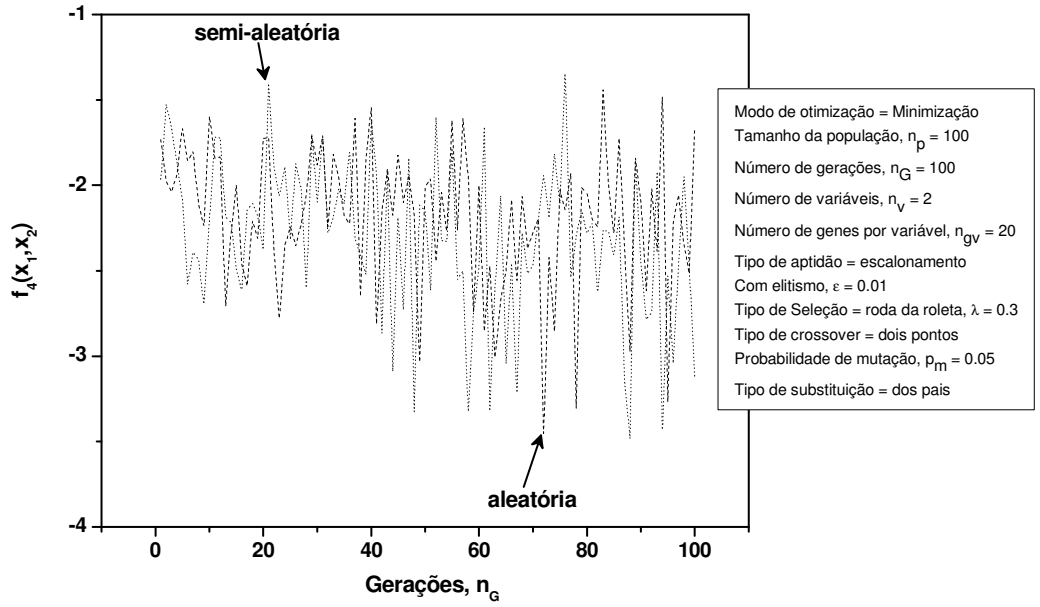


Figura 3.33 – Valores mínimos da função objetivo $f_4(x_1,x_2)$ para diferentes tipos de população inicial.

3.8 Número de genes por variável

Conforme apresentado no Capítulo 2 a melhor acurácia passível de ser obtida utilizando-se algoritmo genético em otimização multidimensional depende da quantidade de genes utilizados para cada variável, n_{gv}. Na Figura 3.34 são apresentados resultados para a função $f_2(x_1,x_2)$ usando-se 2, 4, 8, 16 e 31 genes por variável. Note-se que para 2 genes o maior inteiro sem sinal representável é o número 3 e para 4, 8, 16 e 31 genes são os número 15, 255, 65537 e 2147483647, respectivamente. Assim, a representação utilizando-se 2 genes é muito pobre de possibilidades e este fato reflete-se na pior convergência apresentada na Figura 3.34. Neste caso o indivíduo mais apto apresentou inicialmente um valor da função objetivo próximo do valor 5, o qual não mais se alterou ao longo de todas as gerações. Depois, para 4 genes as possibilidades de representação aumentam e o valor mínimo final da função objetivo é de, aproximadamente, 5×10^{-2}, e assim sucessivamente para maiores quantidades de genes. Note-se que para 16 e 31 genes seriam necessárias muito mais gerações para se atingir toda a potencialidade de representação. A razão de se utilizar 31 genes ao invés de 32 é a de que para inteiros de 32 bits, sem sinal, não é possível representar 2^{32} e sim $\{2^{32} - 1\}$.

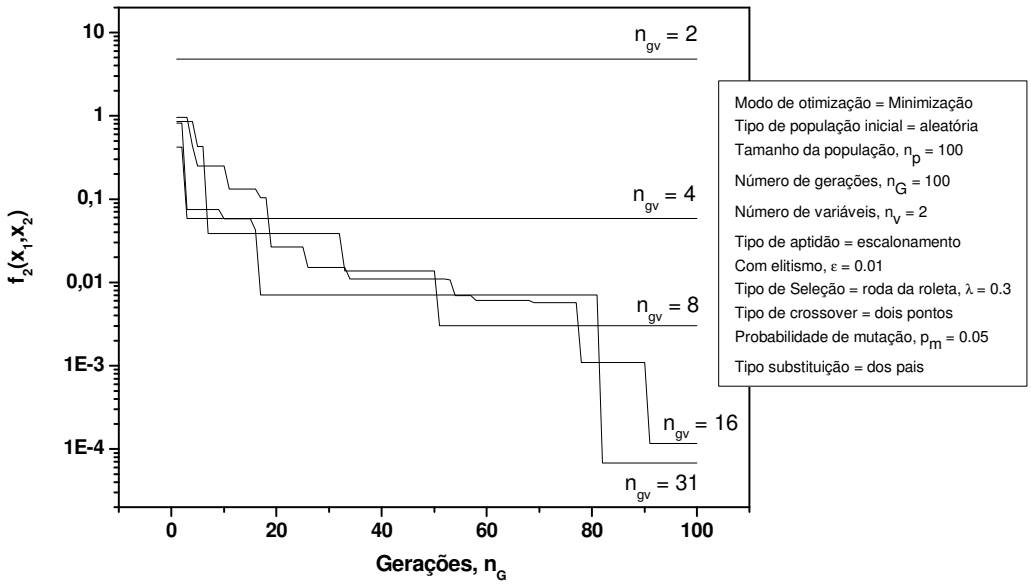


Figura 3.34 - Valores mínimos da função objetivo $f_2(x_1,x_2)$ para diferentes números de genes por variável.

Tal como para a maioria dos indicadores ou algoritmos apresentados anteriormente, o número de genes por variável também não apresenta diferença notável nos resultados obtidos na minimização da função $f_4(x_1,x_2)$, conforme Figura 3.35. Os resultados correspondentes aos números de genes por variável 4, 8 e 16 não são mostrados para facilitar a visualização dos resultados.

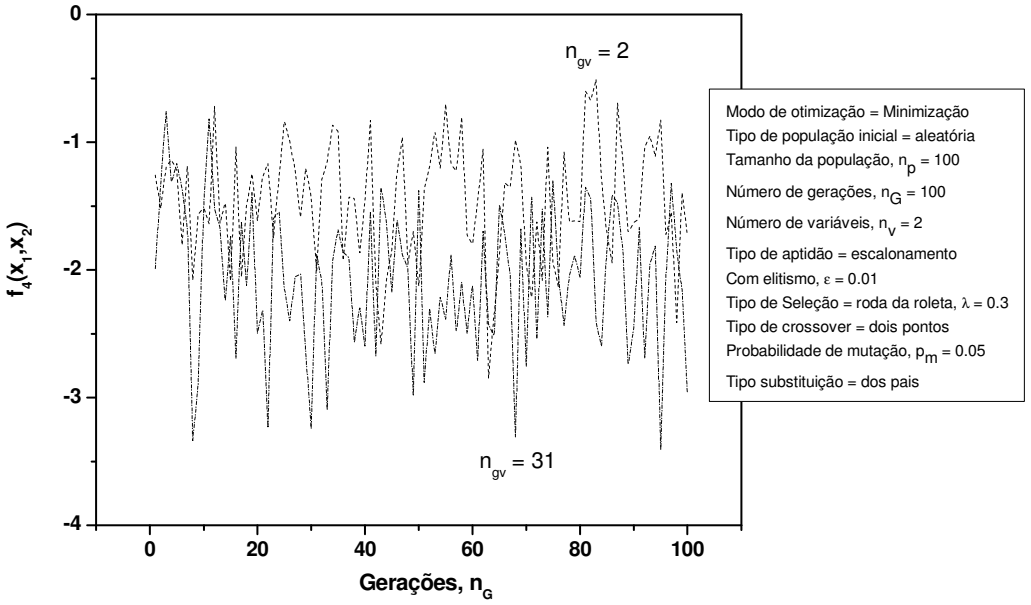


Figura 3.35 - Valores mínimos da função objetivo $f_4(x_1,x_2)$ para diferentes números de genes por variável.

3.9 Tipo de aptidão

Os algoritmos de aptidão utilizados neste trabalho foram: **translação**, **escalonamento** e **potência** ($\gamma = 0{,}5$; 1 e 2). O algoritmo do escalonamento é um algoritmo de translação normalizado, ou seja, uma combinação linear da translação; o algoritmo de potência, para $\gamma = 1$, é exatamente o mesmo do escalonamento. Assim, estes algoritmos devem prover resultados estocasticamente semelhantes. Diferente conceitualmente apenas o algoritmo de potência, para $\gamma = 0{,}5$ e 2, que representam transformações não lineares sobre o algoritmo do escalonamento.

Percebe-se na Figura 3.36, para a função $f_2(x_1, x_2)$, que todos os resultados são quase-equivalentes, pois, as curvas cruzam-se várias vezes com o passar das gerações. As curvas para os algoritmos de translação, escalonamento e potência ($\gamma = 1$) ficaram efetivamente bastante próximas. Já as curvas correspondentes ao algoritmo da potência, casos $\gamma = 0{,}5$ e 2, para a maioria das gerações produziram os piores resultados, embora não muito distantes dos demais.

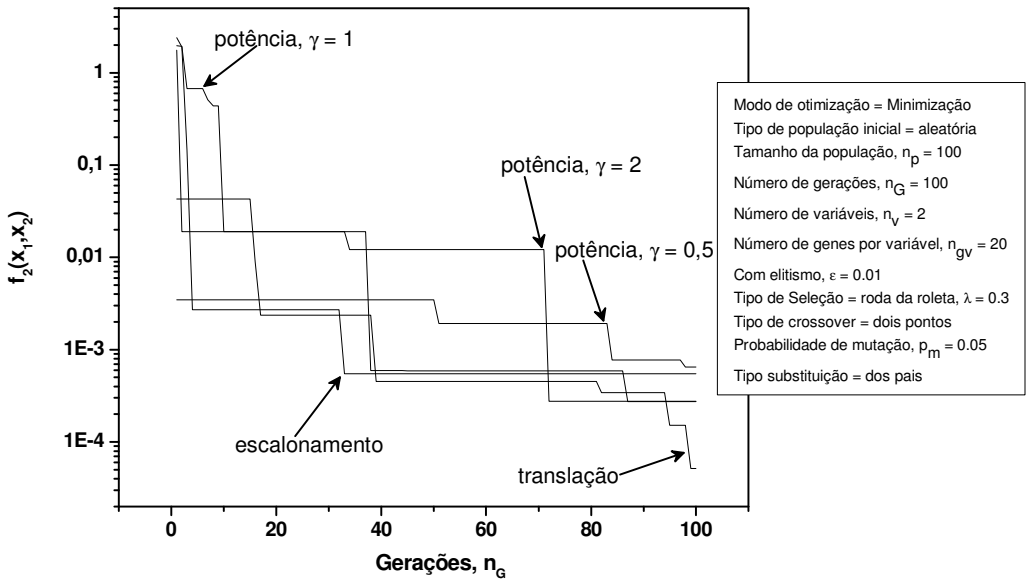


Figura 3.36 - Valores mínimos da função objetivo $f_2(x_1, x_2)$ para diferentes tipos de aptidão: escalonamento, potência e escalonamento.

Quando um determinado parâmetro ou funcionalidade não provê resultados que possam ser discriminados entre si para a minimização da função $f_2(x_1, x_2)$, então muito dificilmente o conseguirá para a função $f_4(x_1, x_2)$, devido ao seu caráter estocástico. De fato, nota-se na Figura 3.37 que todas curvas produzidas utilizando-se os diversos algoritmos de

aptidão não apresentam diferenciação sensível nos valores da função objetivo, nem apresentam decréscimo continuado e notável nos seus valores em função do aumento do número de gerações.

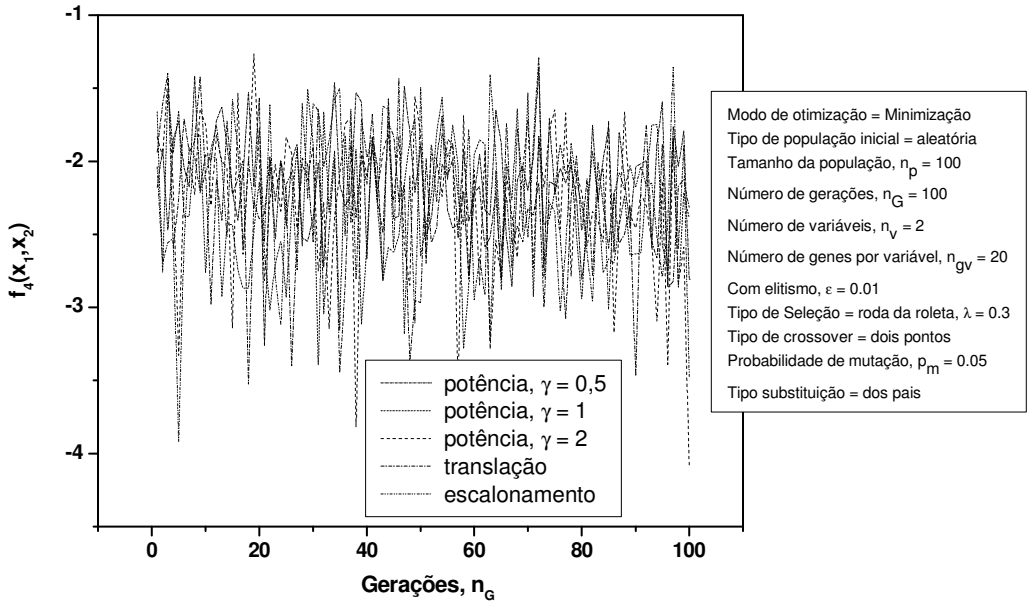


Figura 3.37 - Valores mínimos da função objetivo $f_4(x_1,x_2)$ para diferentes tipos de aptidão: escalonamento, potência e escalonamento.

3.10 Tipo de seleção

Utilizou-se neste trabalho três tipos de algoritmos de seleção dos indivíduos que realizaram *crossover* e deixarão descendentes: **roda da roleta**, **torneio** e **SUS** (*Stochastic Universal Sampling*). Surpreendentemente, o algoritmo roda da roleta apresentou o piores resultados na maioria das gerações e o SUS os melhores, embora ao final seus resultados eram bastante próximos. Por outro lado, o algoritmo do torneio que para a maioria das gerações tenha ficado em posição intermediária entre os outros dois, ao final, apresentou o melhor resultado. Como o SUS é uma espécie de roda da roleta simplificada, pode ser que apresente resultados estocasticamente semelhantes aos da roda da roleta para populações que não sejam muito pequenas. De qualquer forma ao final das 100 gerações os resultados eram qualitativamente semelhantes. Também as respectivas curvas cruzaram-se algumas vezes indicando sua proximidade.

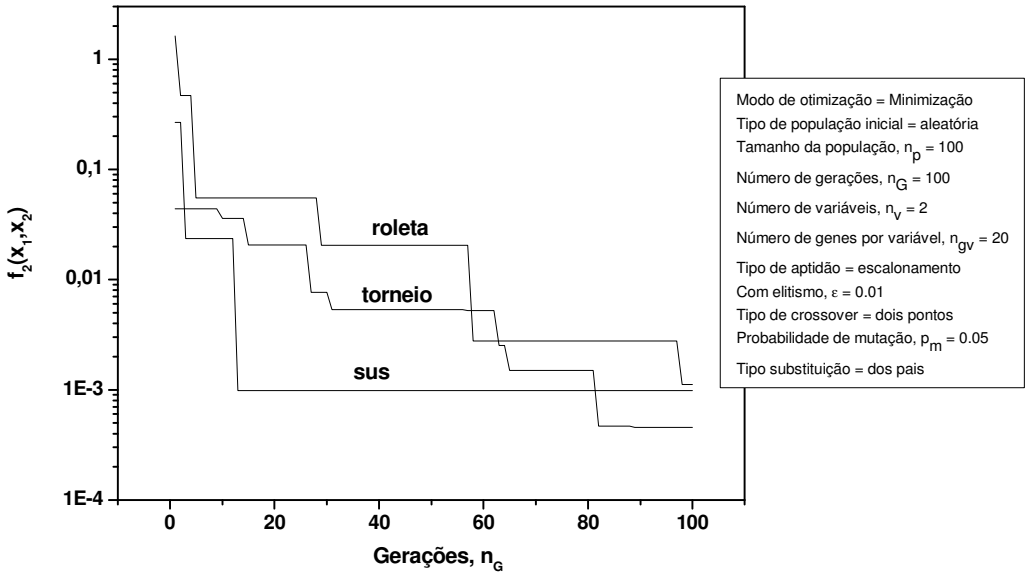


Figura 3.38 - Valores mínimos da função objetivo $f_2(x_1,x_2)$ para diferentes algoritmos de seleção: roda da roleta, torneio e SUS (Stochastic Universal Sampling).

Os resultados para a minimização da função objetivo $f_4(x_1,x_2)$ não apresentaram distinção perceptível entre os três algoritmos de seleção para o *crossover*, como pode ser notado na Figura 3.39.

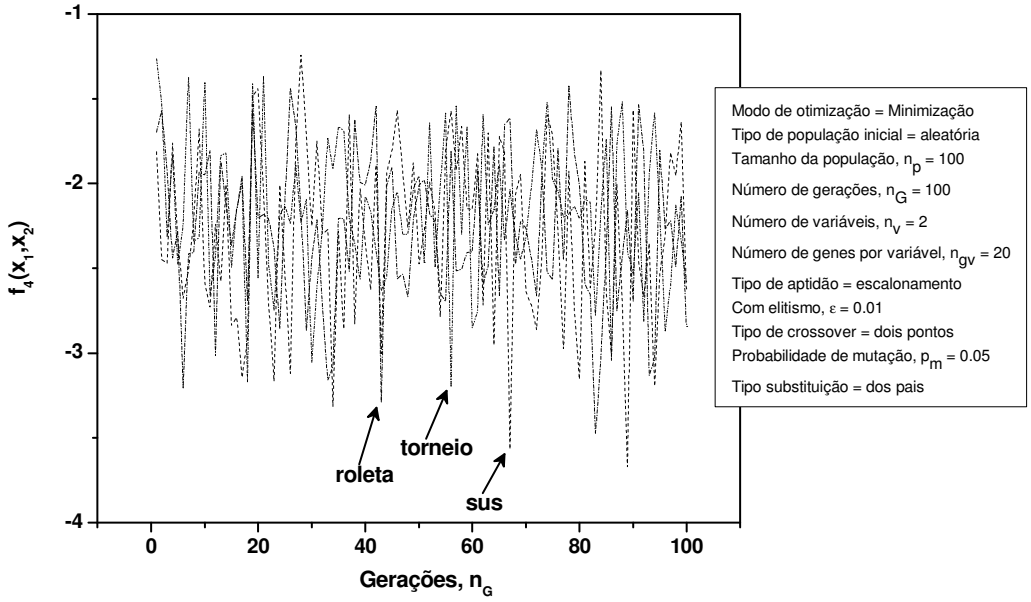


Figura 3.39 - Valores mínimos da função objetivo $f_4(x_1,x_2)$ para diferentes algoritmos de seleção: roda da roleta, torneio e SUS (Stochastic Universal Sampling).

3.11 Taxa de seleção

Um dos parâmetros que se descobriu possuir influência não linear sobre os resultados é a taxa de seleção. Conforme definido no capítulo anterior a taxa varia de zero a um. A taxa nula corresponde a não haver *crossover* e a taxa igual à unidade corresponde a que todos os indivíduos se reproduzem. Estes dois casos extremos quando usados podem gerar resultados, porém o uso de taxas de seleção muito pequenas (próximas de zero) ou muito grandes (próximas de um) não são recomendáveis. Taxas muito pequenas ou nulas corresponderiam a não utilização do *crossover* que é um operador fundamental na natureza. Como o algoritmo genético é inspirado nos mecanismos de evolução das espécies seria um contra-senso a não utilização de *crossover*. Por outro lado, para taxas de *crossover* altas ou mesmo igual à unidade implicariam que naquela população a maioria dos indivíduos se reproduziriam o que em geral não ocorre na natureza, pois, os mais jovens e os mais velhos não se reproduzem, mesmo alguns de idade intermediária, por motivos variados, também não se reproduzem. Assim, estima-se, conceitualmente que as taxas de seleção devem ser menores que 50%. Os testes realizadas para a taxa de seleção podem ser vistos na Figura 3.40, na qual nota-se que os melhores resultados foram obtidos para uma taxa de seleção de 30% seguido pelos resultados obtidos com a taxa de 40%.

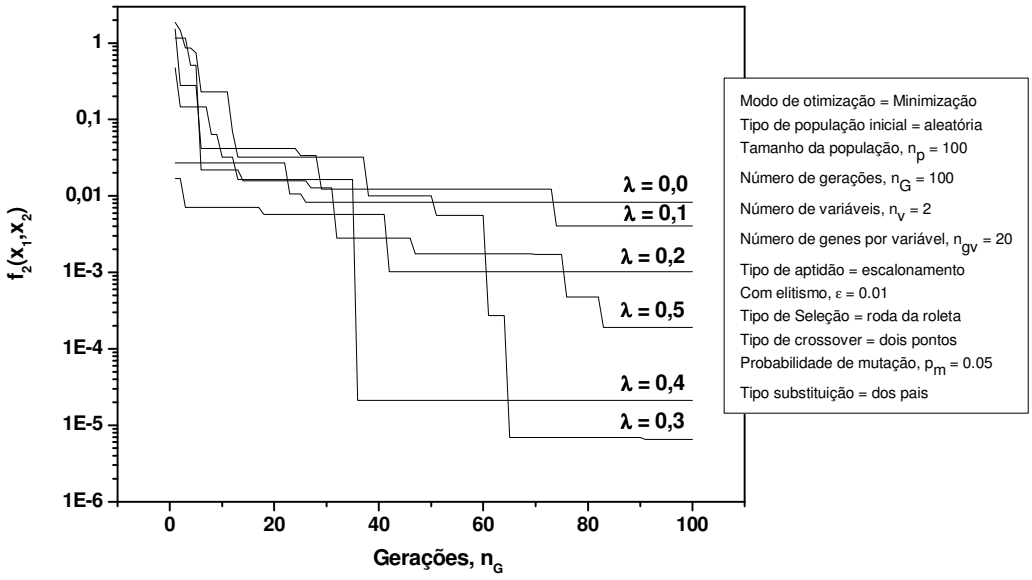


Figura 3.40 - Valores mínimos da função objetivo $f_2(x_1, x_2)$ para diferentes taxas de seleção.

Os piores resultados foram para a inexistência de *crossover*, ou seja, taxa de 0%. Taxas de 10% e 20% produziram resultados piores do que aqueles obtidos utilizando-se a taxa de 50%. Fica claro dos resultados que há uma influência não linear da taxa de seleção sobre a convergência da solução da minimização de $f_2(x_1,x_2)$. O ponto ótimo da taxa de *crossover* para esta função objetivo é na vizinhança de 30%. Este ponto ótimo é dependente da função objetivo.

A taxa de seleção que apresentou boa capacidade de discriminação nos resultados de otimização de $f_2(x_1,x_2)$, que é função determinística, não conseguiu realizar a discriminação de resultados na otimização de $f_4(x_1,x_2)$, nem de apresentar comportamento convergente. Apenas comportamentos oscilantes, estocásticos, Figura 3.41.

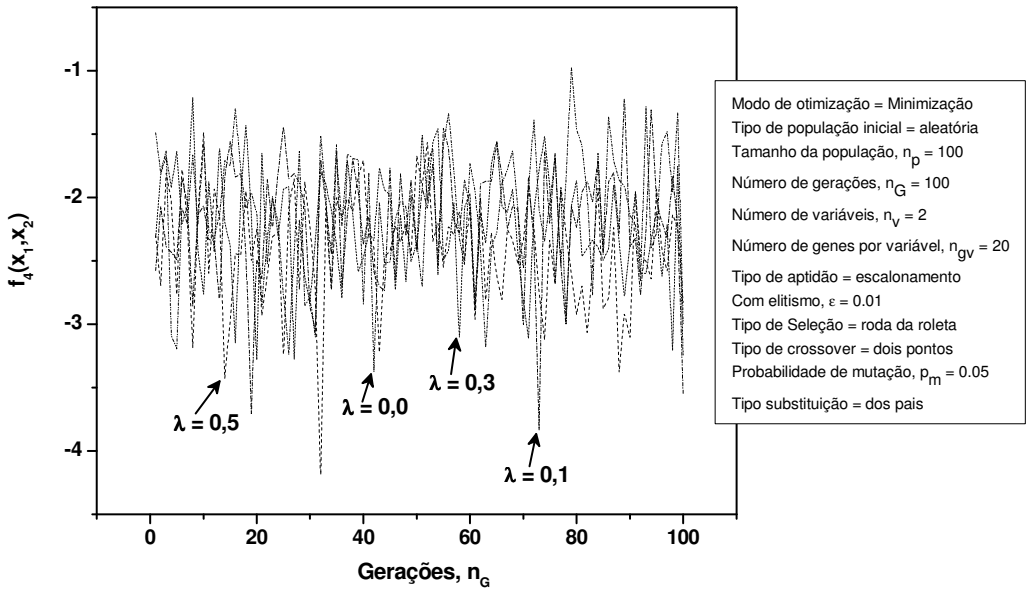


Figura 3.41 - Valores mínimos da função objetivo $f_4(x_1,x_2)$ para diferentes taxas de seleção.

3.12 Tipo de crossover

Desenvolveu-se neste trabalho quatro tipo de algoritmos de *crossover*: **1 ponto**, **2 pontos**, **múltiplos pontos** e **uniforme**. O de múltiplos pontos é a generalização dos de um ponto e os de dois pontos. Na Figura 3.42 apresenta-se os resultados da minimização da função objetivo $f_2(x_1,x_2)$ para os algoritmos: **1 ponto**, **2 pontos**, e **uniforme**. Como os parâmetros e funcionalidades estão sendo testados para problemas bidimensionais

considerou-se desnecessário realizar-se o teste para o caso de **múltiplos pontos**, mesmo porque o algoritmo **uniforme** consiste na realização de *crossover* "em múltiplos pontos", porém com algoritmo distinto. Percebe-se na figura que o pior resultado é obtido para o algoritmo **uniforme**. A probabilidade utilizada no *crossover* uniforme é de 50%, ou seja, na média um em cada dois alelos são intercambiados no *crossover*. Para um cromossomo com 40 genes isto corresponde, na média, em 20 pontos de *crossover*. O *crossover* não destrua nem crie alelos, como a mutação faz, ele apenas recombina os alelos, porém a recombinação pode destruir "blocos de alelos" que representavam indivíduos mais aptos, produzindo indivíduos menos aptos. Assim, algoritmos de *crossover* com muitos pontos tendem a destruir os "blocos" constituintes dos indivíduos mais aptos transformando-os em menos aptos. Uma alternativa para melhorar a performance do *crossover* uniforme, neste caso, provavelmente seria diminuir a probabilidade de *crossover* para, por exemplo, 10%, o que daria uns 4 pontos de *crossover*. Neste trabalho não se realizou o teste da variação da probabilidade de *crossover*. No caso apresentado na figura os algoritmos de *crossover* de 1 e 2 pontos apresentaram resultados finais praticamente iguais. Pode ser que para problemas de maior dimensionalidade talvez seja melhor utilizar algoritmos com maior número de pontos de *crossover*. Esta conjectura não foi testada.

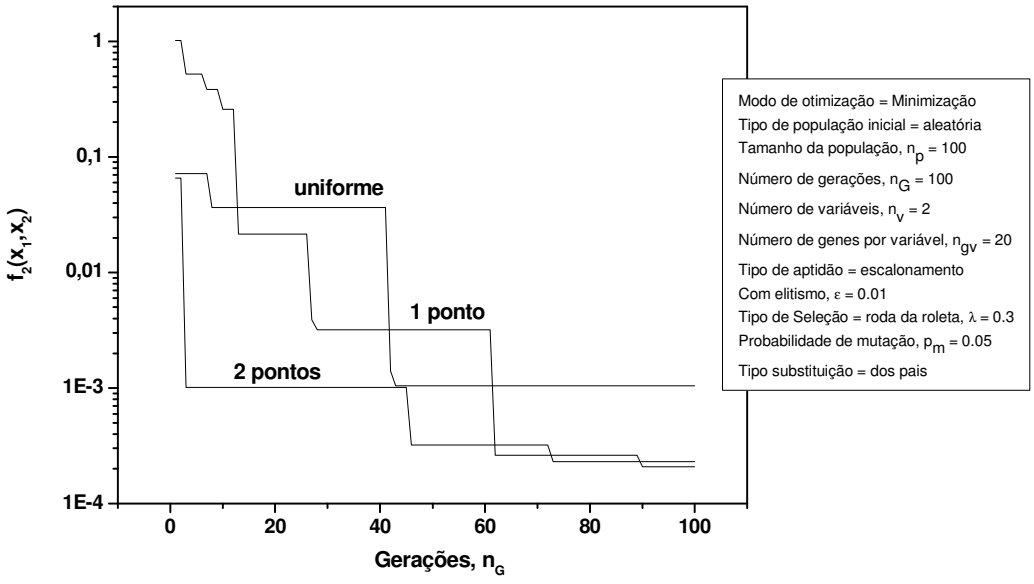


Figura 3.42 - Valores mínimos da função objetivo $f_2(x_1,x_2)$ para diferentes algoritmos de crossover: um ponto, dois pontos e uniforme.

Os diferentes tipos de *crossover* não foram capazes de causar discriminação entre os resultados da minimização da função $f_4(x_1,x_2)$, Figura 3.43. Também não se observa

diminuição consistente no valor da função objetivo.

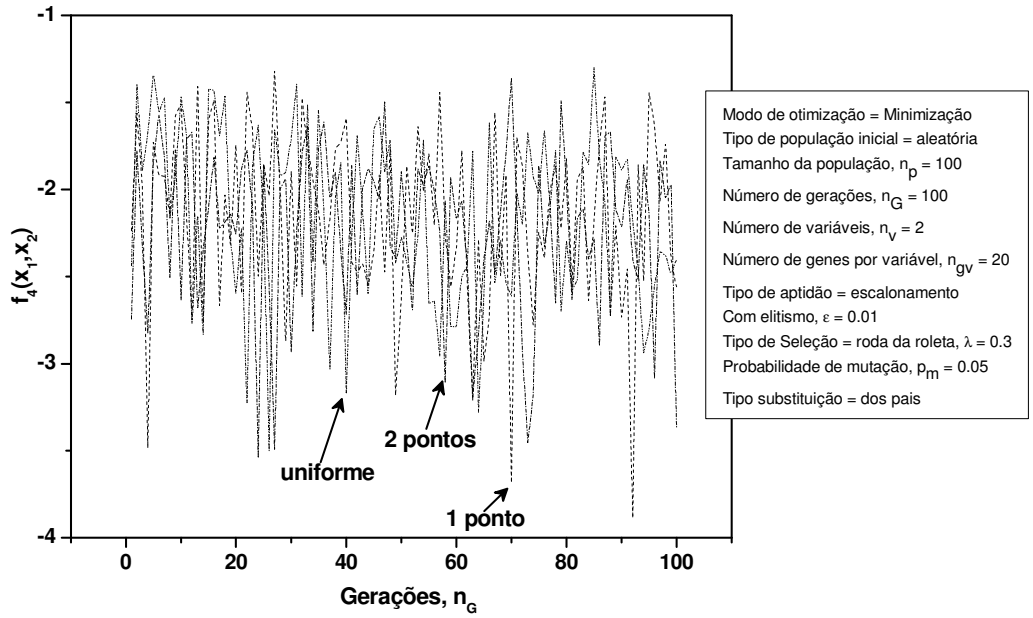


Figura 3.43 - Valores mínimos da função objetivo $f_4(x_1,x_2)$ para diferentes algoritmos de crossover: um ponto, dois pontos e uniforme.

3.13 Probabilidade de mutação

O operador mutação cria ou destrói informação à medida em que altera valores de alelos. Por exemplo, se um indivíduo com 40 genes tinha 25 alelos {**0**} e 15 alelos {**1**} e um dos alelos foi mutado de {**0**} para {**1**}, então ao final da mutação o indivíduo terá 24 alelos {**0**} e 16 alelos {**1**}. Ou seja, a quantidade total de cada valor de alelo alterou-se no indivíduo. Esta alteração poderá ser "boa" se o indivíduo mutado ficar mais apto do que era, e será "ruim" se o indivíduo ficar menos apto. Por esta razão probabilidades muito altas de mutação em geral são ineficientes, pois, tendem a degradar o material genético dos indivíduos mais aptos e a melhoria no material genético dos menos aptos, se houver, não é suficiente para contrabalançar o declínio dos que eram os mais aptos. Portanto, deve-se evitar probabilidades de mutação próximas ou iguais a 100%, em realidade as probabilidades de mutação são menores do que 1% na natureza. Note-se que uma das diferenças entre *crossover* e mutação é que o *crossover* preserva a quantidade total de cada valor de alelo da população enquanto a mutação não preserva. A mutação tem papel fundamental no processo de adaptação, pois, é o operador que produz novas informações que poderão prover os "blocos" responsáveis para a

formação de indivíduos mais aptos. Por esta razão também a probabilidade de mutação não deve ser nula. Com elitismo pode-se utilizar taxas de mutação ligeiramente mais altas, pois, o elitismo preserva os indivíduos mais aptos contra a destruição de material genético promovido pela mutação. Sem elitismo deve-se utilizar taxas de mutação bastante baixas, menores que 1%, para que a probabilidade de indivíduos mais aptos serem degradados geneticamente pela mutação seja baixa. Na Figura 3.44 pode-se ver o comportamento não linear da probabilidade de mutação. Note que os piores resultados são para as probabilidades de mutação de 0% e de 50%. Enquanto as probabilidades de 10% e 20% produzem resultados bem melhores. Neste caso para a função $f_2(x_1, x_2)$ e para os valores mostrados a probabilidade de 20% produziu o melhor resultado.

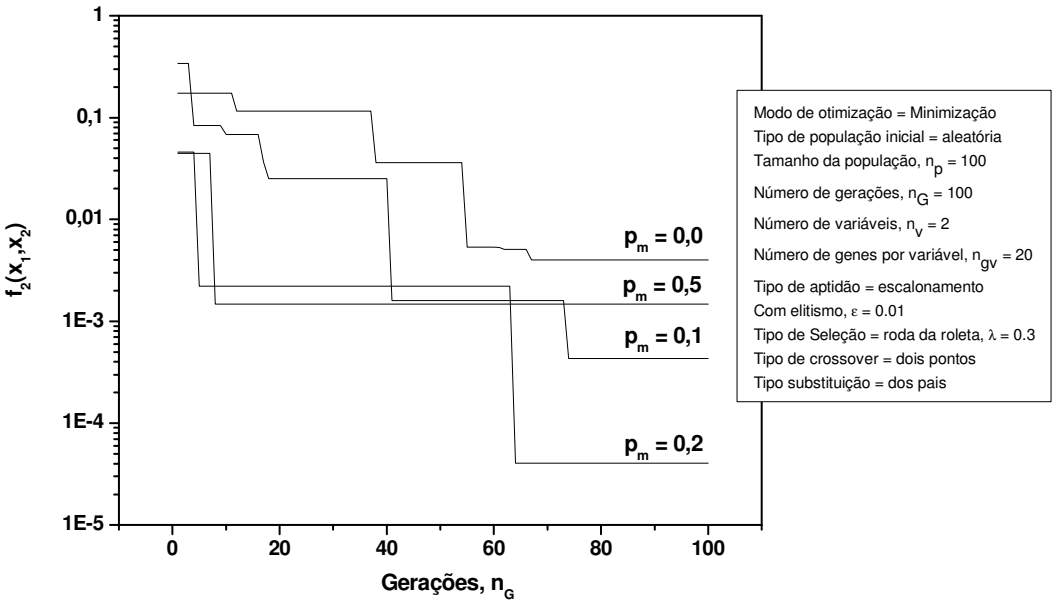


Figura 3.44 - Valores mínimos da função objetivo $f_2(x_1, x_2)$ para diferentes probabilidades de mutação.

Para a função $f_4(x_1, x_2)$, Figura 3.45, a probabilidade de mutação também não é capaz de produzir resultados que discriminem quais são os melhores resultados.

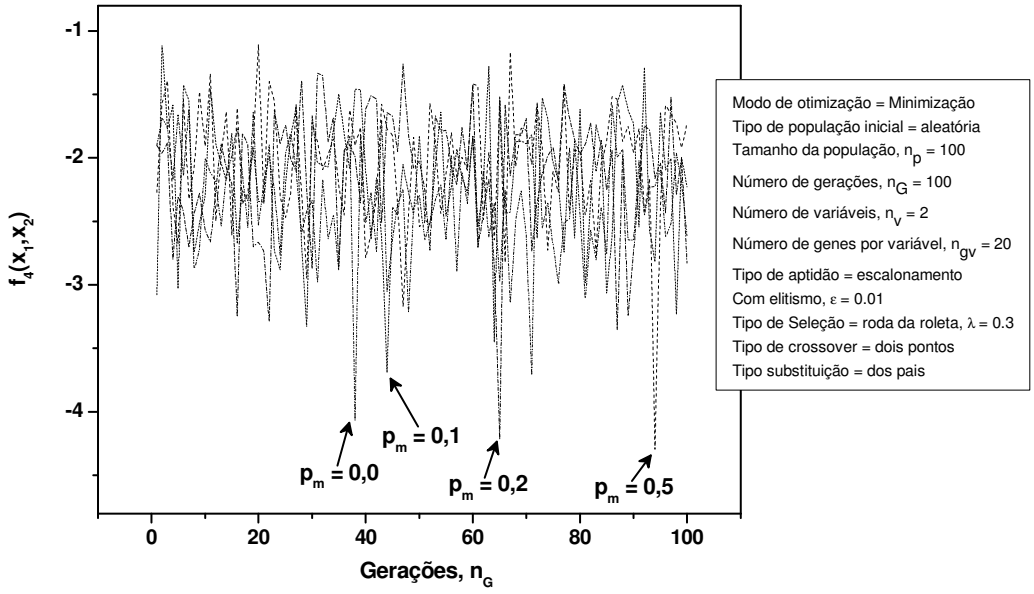


Figura 3.45 - Valores mínimos da função objetivo $f_4(x_1,x_2)$ para diferentes probabilidades de mutação.

3.14 Tipo de substituição

O operador seleção define quais indivíduos da população serão utilizados no processo de *crossover*, ou seja, quais irão gerar descendentes. Para abrigar os descendentes na população faz-se necessário decidir quais indivíduos serão descartados para abrir espaço para os filhos. O operador que faz este processo decisório e por fim realiza a inserção dos filhos na população é o operador **substituição**. Foram usados neste trabalho três operadores de substituição: **dos pais**, **dos menos aptos** e **aleatório**.

No caso da minimização da função $f_2(x_1,x_2)$, Figura 3.46, nota-se que o pior resultado é obtido utilizando-se a substituição aleatória, provavelmente, este comportamento deve-se ao fato de que neste tipo de substituição muitos indivíduos relativamente aptos são descartados e substituídos por filhos, na média, não tão aptos. Esta opção pode ainda ser mais ineficiente quando não houver elitismo, pois, neste caso até o indivíduo mais apto da população poderia ser substituído. No caso do uso de elitismo os indivíduos mais aptos são preservados quando da ação do operador substituição.

A substituição dos pais gera resultados relativamente melhores porque os filhos herdam o material genético dos pais, embora recombinado, assim não haverá muita perda de diversidade genética na população. Quando se usa elitismo os pais de elite também são preservados.

Por fim, a substituição dos menos aptos foi a que gerou os melhores resultados, para este caso.

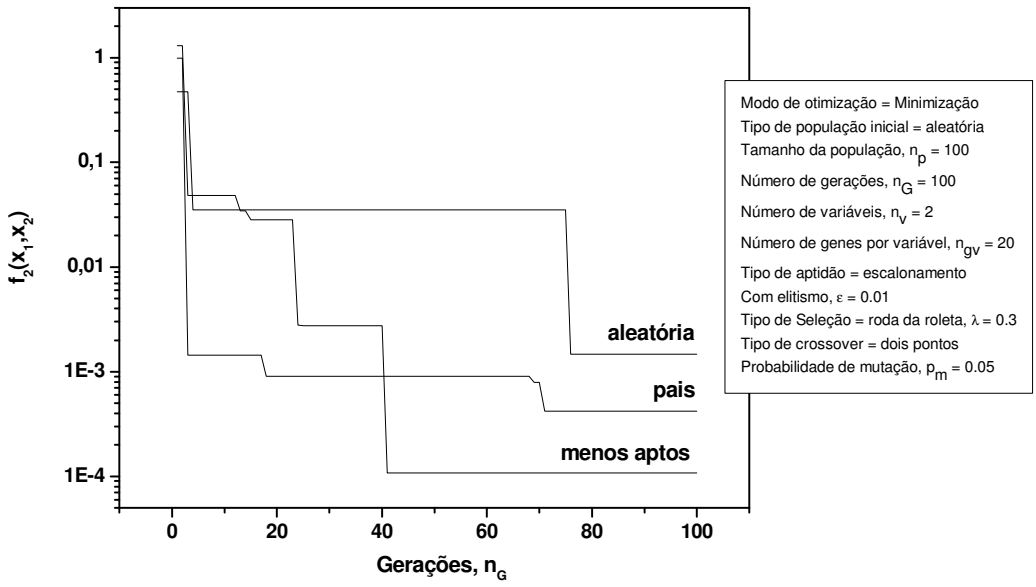


Figura 3.46 - Valores mínimos da função objetivo $f_2(x_1,x_2)$ para diferentes tipos de algoritmos de substituição: aleatória; dos pais; e dos menos aptos.

Para todos os parâmetros e funcionalidades apresentados anteriormente, exceto quanto ao tamanho da população, nenhum deles foi capaz causar discriminação entre os resultados obtidos na minimização da função $f_4(x_1,x_2)$, nem diminuir de modo consistente o valor da função objetivo. Devido à natureza estocástica da função os resultados mostram-se oscilantes e não convergem.

No caso do tamanho da população, mostrou-se anteriormente, neste capítulo, que seu aumento melhora os resultados, diminuindo os valores para a função objetivo.

No caso do operador substituição, Figura 3.47, também não há discriminação de resultados entre os três algoritmos de substituição: dos pais, dos menos aptos e aleatório.

Em resumo, quanto à função $f_4(x_1,x_2)$ o máximo que se obteve foi uma "órbita" convergida de indivíduos que estão em torno do ponto de minimização da parte determinística da função. Quando se aumenta a quantidade de indivíduos na população o tamanho da órbita diminui. Mais adiante, neste capítulo, retorna-se à questão de funções estocásticas.

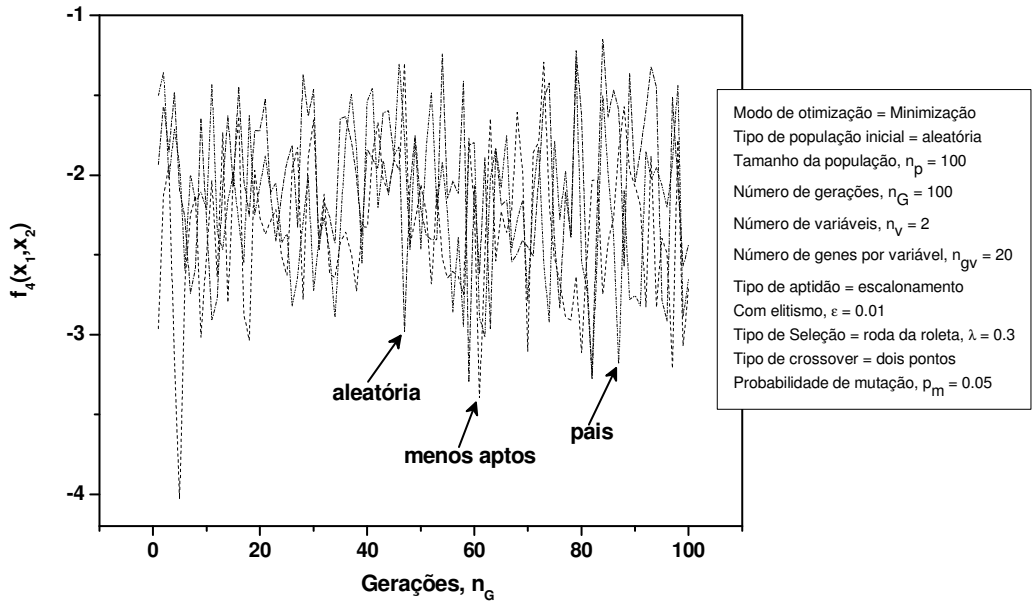


Figura 3.47 - Valores mínimos da função objetivo $f_4(x_1, x_2)$ para diferentes tipos de algoritmos de substituição: aleatória; dos pais; e dos menos aptos.

3.15 Conjunto estendido e ampliado de problemas teste multidimensionais

Nas seções anteriores deste capítulo realizou-se uma série de testes de parâmetros e funcionalidades na solução de um conjunto básico de problemas de otimização bidimensionais.

Uma vez estabelecido um conjunto de valores e opções para os parâmetros e funcionalidades pode-se então redefinir e ampliar aquele conjunto de funções teste, no sentido de tornar multidimensionais aqueles problemas. As funções $f_1(x_1, x_2)$ a $f_4(x_1, x_2)$ foram redefinidas agora para $f_5(\mathbf{x})$ a $f_8(\mathbf{x})$, adicionou-se as funções $f_9(\mathbf{x})$ e $f_{10}(\mathbf{x})$, cujas definições estão a seguir:

$$f_5(\mathbf{x}) \equiv \sum_{i=1}^{n} x_i^2, \quad \mathbf{x} \in \Re^n, \quad f(\mathbf{x}) \in \Re; \tag{3.2a}$$

$$f_6(\mathbf{x}) \equiv \sum_{i=1}^{n-1} [100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2], \quad \mathbf{x} \in \Re^n, \quad f(\mathbf{x}) \in \Re; \tag{3.2c}$$

$$f_7(\mathbf{x}) \equiv 5n + \sum_{i=1}^{n} \text{int}(x_i - \frac{12}{100}), \quad \mathbf{x} \in \Re^n, \quad f(\mathbf{x}) \in \Re; \tag{3.2d}$$

$$f_8(\mathbf{x}) \equiv Gauss(0,1) + \sum_{i=1}^{n}[ix_i^4], \quad \mathbf{x} \in \Re^n, \quad f(\mathbf{x}) \in \Re; \tag{3.2e}$$

$$f_9(\mathbf{x}) \equiv 10n + \sum_{i=1}^{n}[x_i^2 - 10\cos(2\pi x_i)], \quad \mathbf{x} \in \Re^n, \quad f(\mathbf{x}) \in \Re; \tag{3.2b}$$

e

$$f_{10}(\mathbf{x}) \equiv \begin{cases} \sqrt{\prod_{i=1}^{n} x_i}, & se \quad \prod_{i=1}^{n} x_i > 0 \\ 1, & se \quad \prod_{i=1}^{n} x_i = 0 \quad e \quad \sum_{i=1}^{n} abs(x_i) \neq 0 \\ 0, & se \quad \prod_{i=1}^{n} x_i = 0 \quad e \quad \sum_{i=1}^{n} abs(x_i) = 0 \\ \textit{não definida se} \prod_{i=1}^{n} x_i < 0 \end{cases}, \mathbf{x} \in \Re^n, \quad f(\mathbf{x}) \in \Re. \tag{3.2f}$$

A função $f_9(\mathbf{x})$ é conhecida na literatura como **Função de Rastrigin** e sua principal característica é possuir muitos pontos de mínimos e máximos locais, no espaço n - dimensional, devido à existência em sua definição de uma função trigonométrica no caso um co-seno.

A função objetivo $f_{10}(\mathbf{x})$ foi definida neste trabalho visando testar o conceito de pênalti aqui desenvolvido e utilizado. Note-se que sob certas circunstâncias a função não é definida, sendo que os indivíduos correspondentes a estas posições não são conhecidos a priori e nem as posições também são conhecidas.

A seguir, na Tabela 3.2 apresenta-se algumas informações sobre os dados associados às funções teste multidimensionais.

Tabela 3.2 – Dados sobre as funções teste multidimensionais.

Função teste	Domínio	Solução	Função, na solução
$f_5(\mathbf{x})$	$-5 \le x_1,...,x_n \le +5$	$\mathbf{x}^* \equiv (0,...,0)$	$f_5(\mathbf{x}^*) = 0$
$f_6(\mathbf{x})$	$-5 \le x_1,...,x_n \le +5$	$\mathbf{x}^* \equiv (1,...,1)$	$f_6(x_1^*, x_2^*) = 0$
$f_7(\mathbf{x})$	$-5 \le x_1,...,x_n \le +5$	$\mathbf{x}^* \equiv (-5,...,-5)$	$f_7(\mathbf{x}^*) = 0$
$f_8(\mathbf{x})$	$-5 \le x_1,...,x_n \le +5$	$\mathbf{x}^* \equiv viz(0,...,0)$	$f_8(\mathbf{x}^*) = viz[Gauss(0,1)]$
$f_9(\mathbf{x})$	$-5 \le x_1,...,x_n \le +5$	$\mathbf{x}^* \equiv (0,...,0)$	$f_9(\mathbf{x}^*) = 0$
$f_{10}(\mathbf{x})$	$-5 \le x_1,...,x_n \le +5$	$\mathbf{x}^* \equiv (0,...,0)$	$f_{10}(\mathbf{x}^*) = 0$

3.16 – Minimização das funções $f_5(\mathbf{x})$ a $f_{10}(\mathbf{x})$ para n = 10 variáveis

Uma vez estabelecido um conjunto básico de conhecimentos sobre o que é e como funciona o algoritmo genético, nesta seção e subseções apresenta-se a minimização das funções objetivo $f_5(\mathbf{x})$ a $f_{10}(\mathbf{x})$ para n = 10 variáveis. Note que a quantidade de variáveis aumentou de 2 para 10, assim haverá muito mais hiper-volume para a população realizar o processo evolutivo.

3.16.1 – Minimização da função $f_5(\mathbf{x})$ para n = 10 variáveis

Na minimização da função $f_5(\mathbf{x})$ e das demais funções mencionadas anteriormente utiliza-se todo o conjunto de conhecimento acumulado até o momento no desenvolvimento deste trabalho. No caso da função $f_5(\mathbf{x})$ os resultados são mostrados de um modo mais detalhado. Para as demais funções vai se resumindo e simplificando a apresentação dos resultados para evitar material muito volumoso. Os parâmetros e funcionalidades utilizados em cada caso são discriminados na legenda constante à direita das figuras principais. As figuras secundárias "herdam" as informações das principais.

Na Figura 3.48 apresenta-se os resultados dos valores mínimo, médio e máximo, da função objetivo $f_5(\mathbf{x})$, obtidos utilizando-se valores de parâmetros e opções de funcionalidades tal como discutido nas seções anteriores. Note-se que neste caso o valor da função objetivo para o indivíduo mais apto reduziu-se apenas a algo próximo da unidade. Naturalmente, um problema de dimensão 10 é mais difícil de resolver do que um de dimensão 2.

A norma da distância Euclidiana do indivíduo mais apto e também do menos apto são apresentadas na Figuras 3.49. Percebe-se que o valor mínimo da norma é da ordem da unidade.

Necessita-se então melhorar estes resultados, para tal aumenta-se a quantidade de indivíduos na população inicial, de 100 para 1000, e aumenta-se o número de gerações de 100 para 20000. Outros valores podem ser visto nas legendas das figuras.

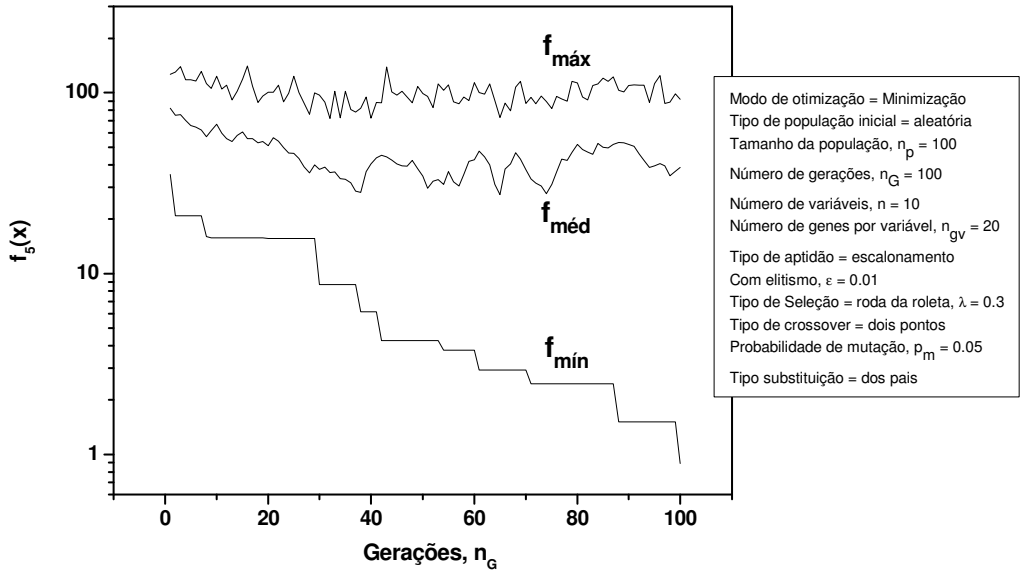


Figura 3.48 – Valores mínimo, médio e máximo dos valores da função objetivo $f_5(\mathbf{x})$, para $n = 10$ variáveis.

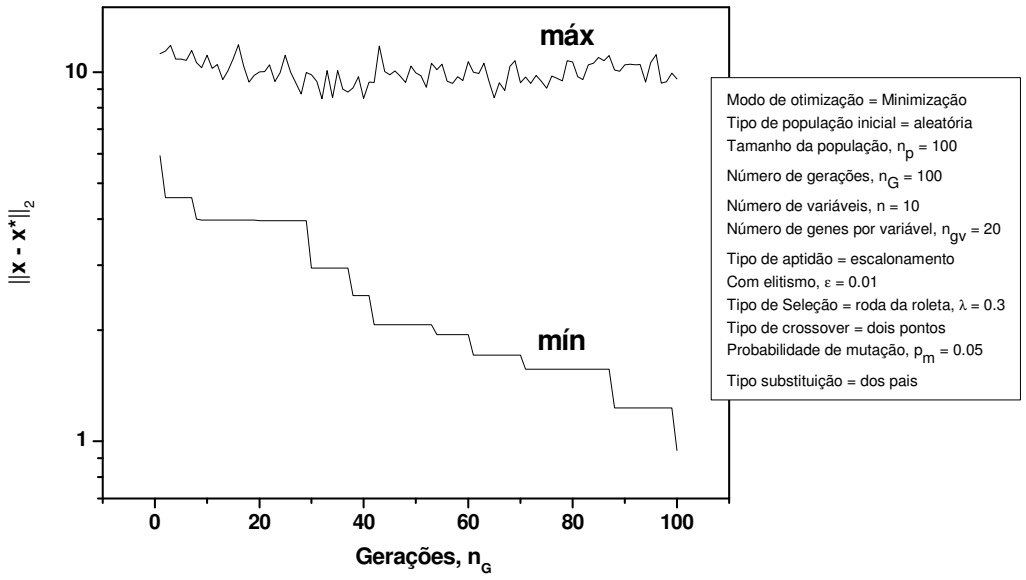


Figura 3.49 – Valores mínimo e máximo, da distância dos indivíduos mais apto e menos apto até a solução da função $f_5(\mathbf{x})$, para $n = 10$ variáveis.

Os resultados apresentados nas Figuras 3.50 e 3.51 são semelhantes àqueles apresentados nas Figuras 3.48 e 3.49, para a função $f_5(\mathbf{x})$, porém obtidos utilizando-se outros parâmetros e funcionalidades. Nota-se na Figura 3.50 que o valor mínimo da função objetivo agora se reduziu para, aproximadamente, 10^{-5}. A norma da distância do indivíduo mais apto até a solução correspondente também diminuiu para algo em torno de 4×10^{-3}. Ou seja, este novo conjunto de parâmetros e funcionalidade foi capaz de propiciar solução com boa convergência, via algoritmo genético.

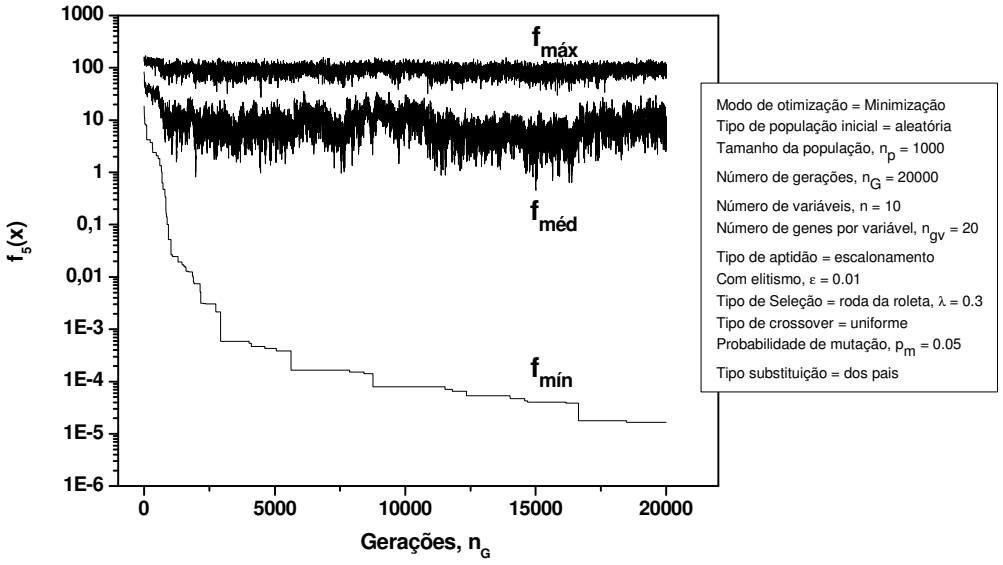


Figura 3.50 - Valores mínimo, médio e máximo dos valores da função objetivo $f_5(\mathbf{x})$, para $n = 10$ variáveis.

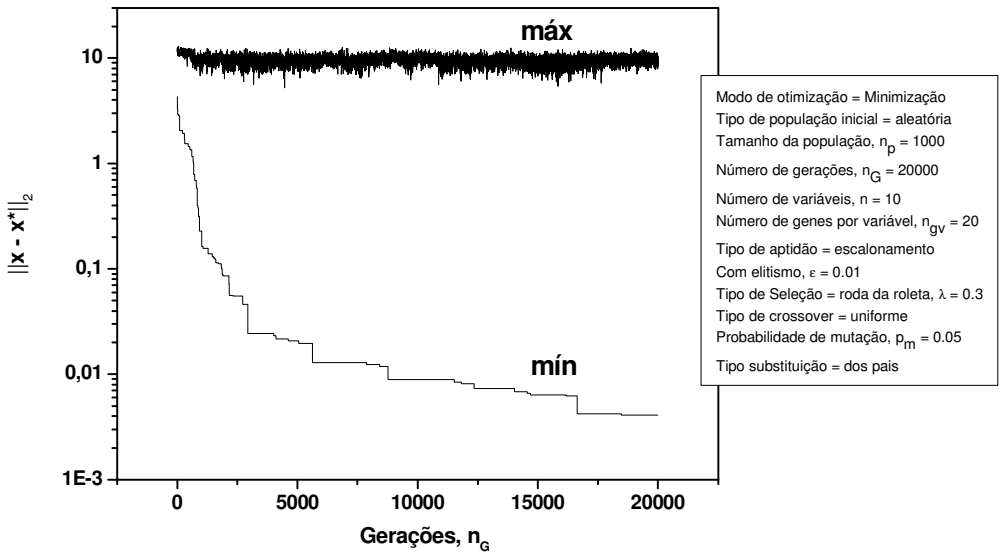


Figura 3.51 - Valores mínimo e máximo, da distância dos indivíduos mais apto e menos apto até a solução da função $f_5(\mathbf{x})$, para $n = 10$ variáveis.

3.16.2 – Minimização da função $f_6(\mathbf{x})$ para n = 10 variáveis

Os parâmetros e funcionalidades que foram utilizados para obter os resultados apresentados nas Figuras 3.50 e 3.51, foram então na seqüência usados para obtenção dos resultados para as funções $f_6(\mathbf{x})$ a $f_{10}(\mathbf{x})$. Estes resultados são relativos aos indivíduos mais

aptos da população para três probabilidades de mutação: 1, 5 e 30%. Para a função $f_6(\mathbf{x})$ os resultados são apresentados nas Figuras 3.52 e 3.53, e nota-se que o algoritmo genético conseguiu reduzir o valor da função objetivo de 10000 para, aproximadamente 6, nos casos das probabilidades de mutação iguais a 1 e 5%. No caso da probabilidade de mutação de 30% a função foi reduzida para 200, aproximadamente. Novamente, mostra-se que altas probabilidades de mutação levam a performance mais ineficiente do algoritmo genético.

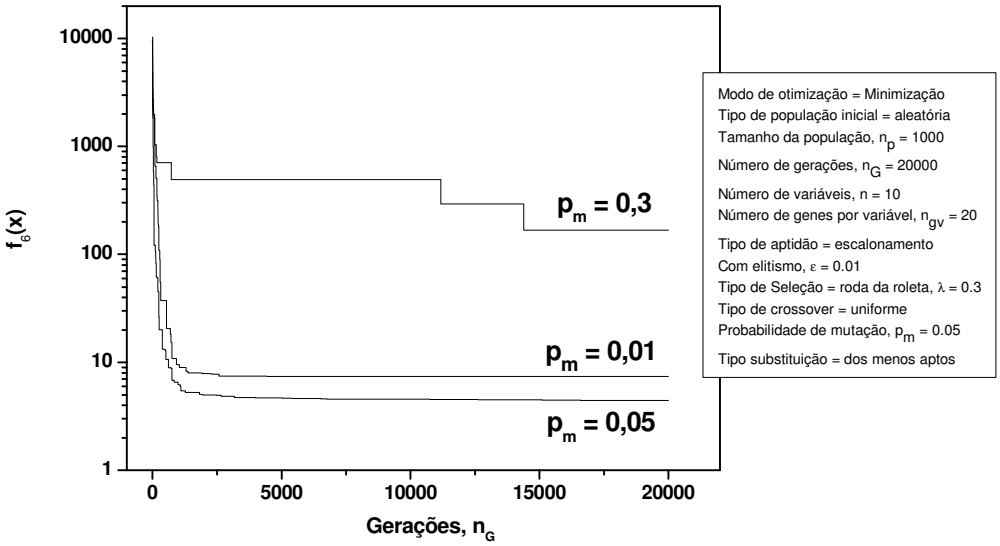


Figura 3.52 – Valores mínimos da função objetivo $f_6(\mathbf{x})$ para três probabilidades de mutação e 10 variáveis.

Quanto à distância do indivíduo mais apto até à solução do problema, Figura 3.53, os melhores resultados também são obtidos para as probabilidades de mutação de 1 e 5%, enquanto os resultados para 30% são relativamente piores. Para a norma da distância os resultados correspondentes a 1% são melhores do que os para 5%.

De qualquer forma os melhores resultados obtidos são ainda maiores do que a unidade. É desejável que possam ser diminuídos ainda mais. Retorna-se à questão de melhoria dos resultados relativos à função objetivo $f_6(\mathbf{x})$ mais adiante neste capítulo.

3.16.3 – Minimização da função $f_7(\mathbf{x})$ para n = 10 variáveis

Para o caso da função degrau multidimensional $f_7(\mathbf{x})$ o algoritmo genético tem eficiência muito boa conseguindo chegar a valor mínimo da função como pode ser visto na

Figura 3.54. É interessante notar que devido ao fato da função $f_7(\mathbf{x})$ ser de classe C^0 os métodos baseados no gradiente da função falham totalmente, uma vez que o gradiente de $f_7(\mathbf{x})$ é nulo em quase todos os pontos do domínio.

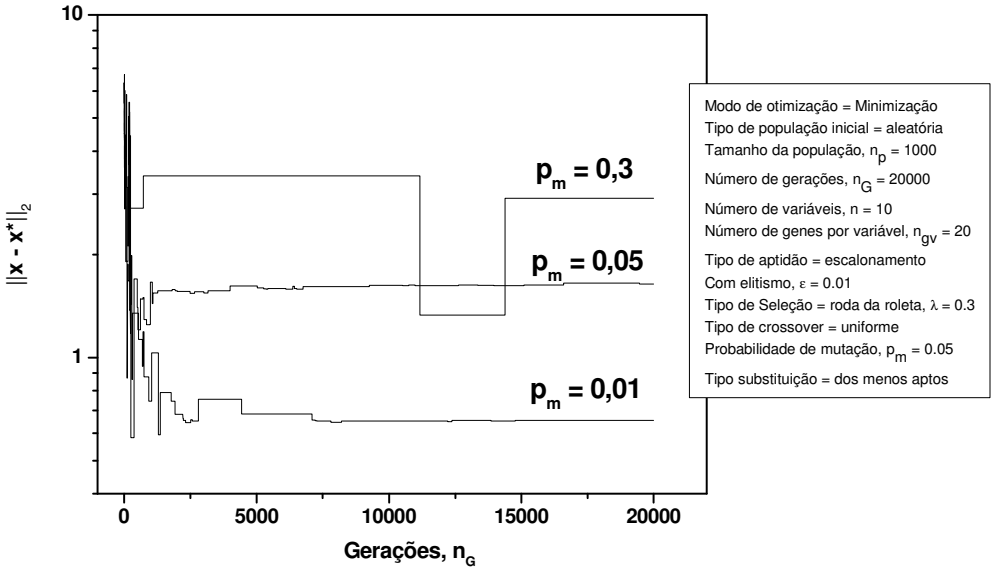


Figura 3.53 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_6(\mathbf{x})$, para três probabilidades de mutação e 10 variáveis.

Nota-se que os melhores resultados são obtidos para a probabilidade de mutação igual a 5%, enquanto os resultados para 1% e 30% são relativamente piores.

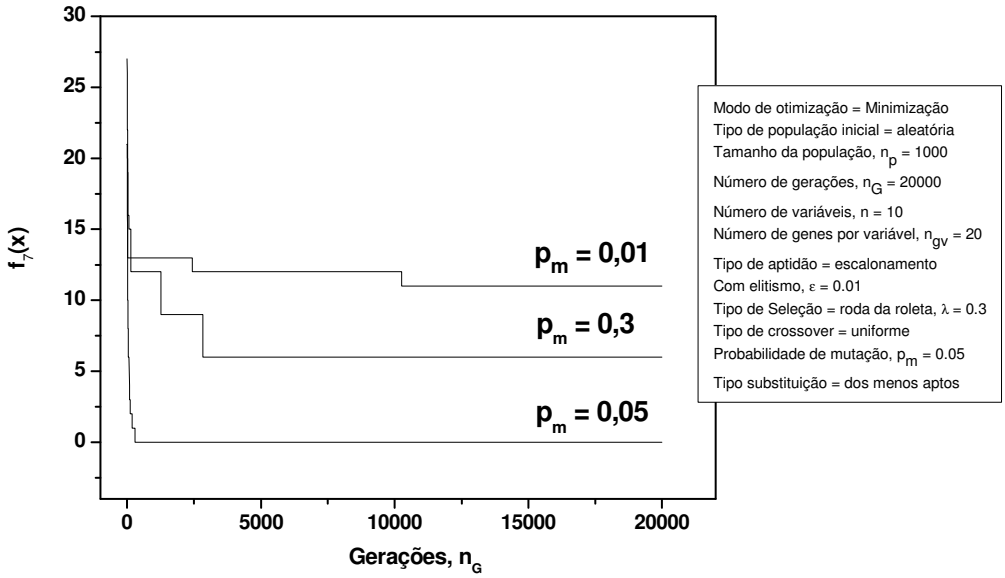


Figura 3.54 - Valores mínimos da função objetivo $f_7(\mathbf{x})$ para três probabilidades de mutação e 10 variáveis.

Para a distância do indivíduo mais apto até o conjunto de pontos que minimizam a função $f_7(\mathbf{x})$, Figura 3.55, também os melhores resultados são obtidos para a probabilidade de mutação de 5%.

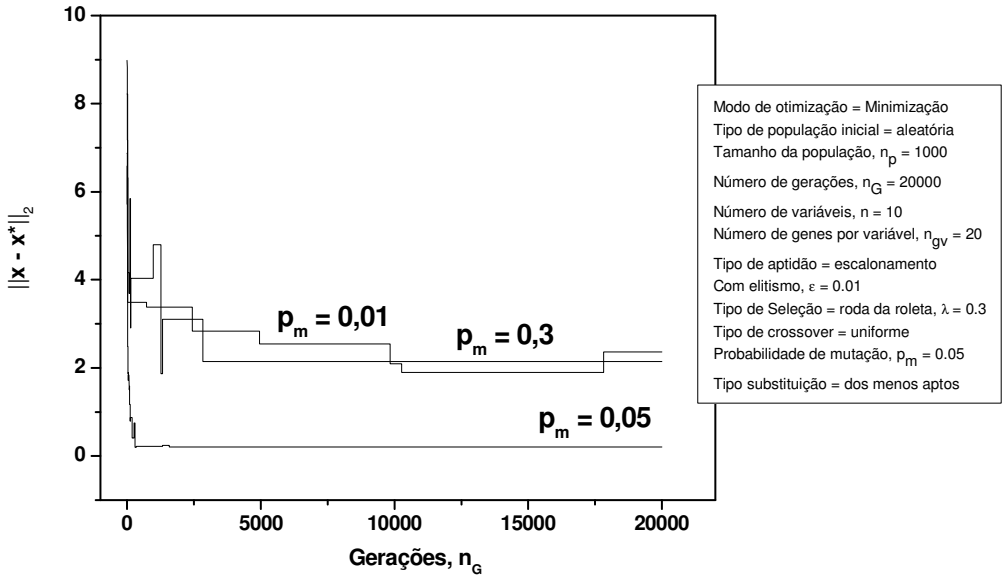


Figura 3.55 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_7(\mathbf{x})$, para três probabilidades de mutação e 10 variáveis.

3.16.4 – Minimização da função $f_8(\mathbf{x})$ para n = 10 variáveis

Mostrou-se anteriormente neste capítulo que para a função f_4 diferentes probabilidades de mutação não causavam discernimento perceptível de resultados. No caso da função $f_8(\mathbf{x})$ ocorre algum discernimento que pode ser notado nas Figura 3.56 e 3.57. Também o algoritmo genético foi capaz neste caso de reduzir significativamente o valor da função objetivo. A razão para tal diferenciação de resultados entre f_4 e $f_8(\mathbf{x})$, provavelmente está no fato de que a parte estocástica de $f_8(\mathbf{x})$ é muito maior do que a de f_4. Como a parte estocástica é igual nos dois casos, então, o problema de minimização de $f_8(\mathbf{x})$ é estocasticamente mais simples, e determinísticamente mais difícil porque a dimensão do problema passou de 2 para 10.

Novamente a probabilidade de mutação de 30% produz os piores resultados, sendo que os melhores são produzidos utilizando-se 1%, embora os correspondentes a 5% estejam bem próximos.

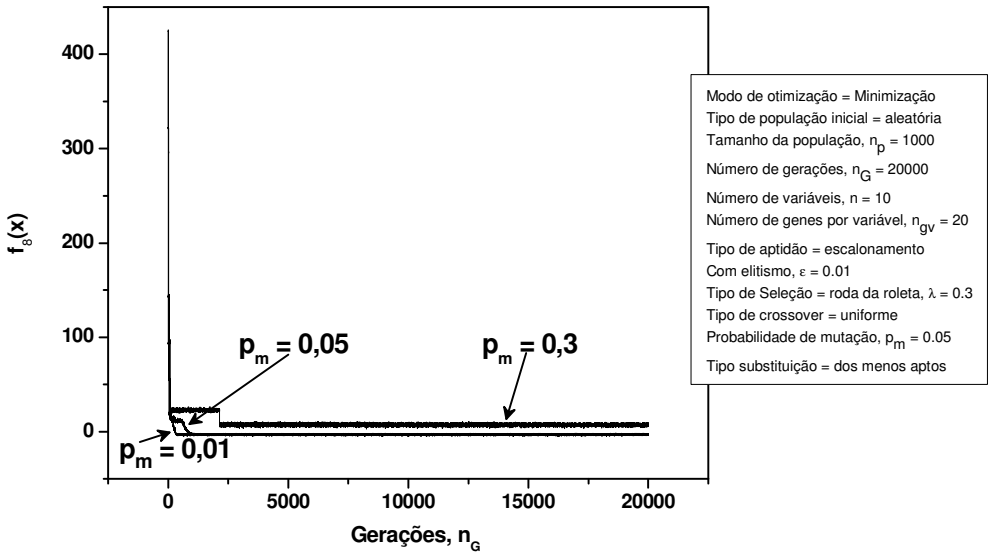


Figura 3.56 - Valores mínimos da função objetivo $f_8(\mathbf{x})$ para três probabilidades de mutação e 10 variáveis.

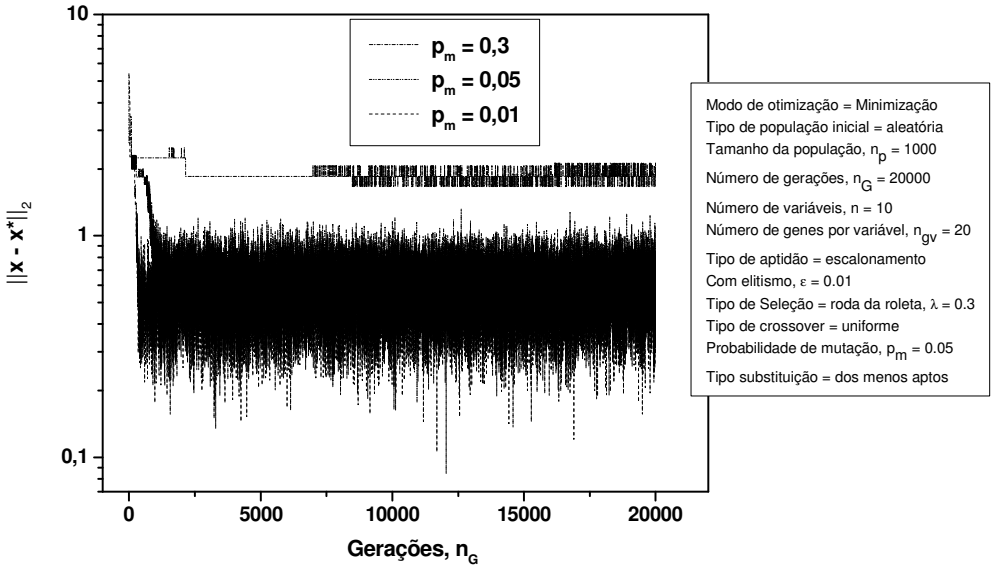


Figura 3.57 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_8(\mathbf{x})$, para três probabilidades de mutação e 10 variáveis.

3.16.5 – Minimização da função $f_9(\mathbf{x})$ para n = 10 variáveis

Embora a função de Rastringin, $f_9(\mathbf{x})$, seja considerada de difícil solução na literatura correlata, devido aos seus múltiplos pontos de mínimos e máximos locais, os resultados apresentados a seguir nas Figuras 3.58 e 3.59 mostram que o algoritmo genético desenvolvido neste trabalho operada bastante eficientemente na sua solução.

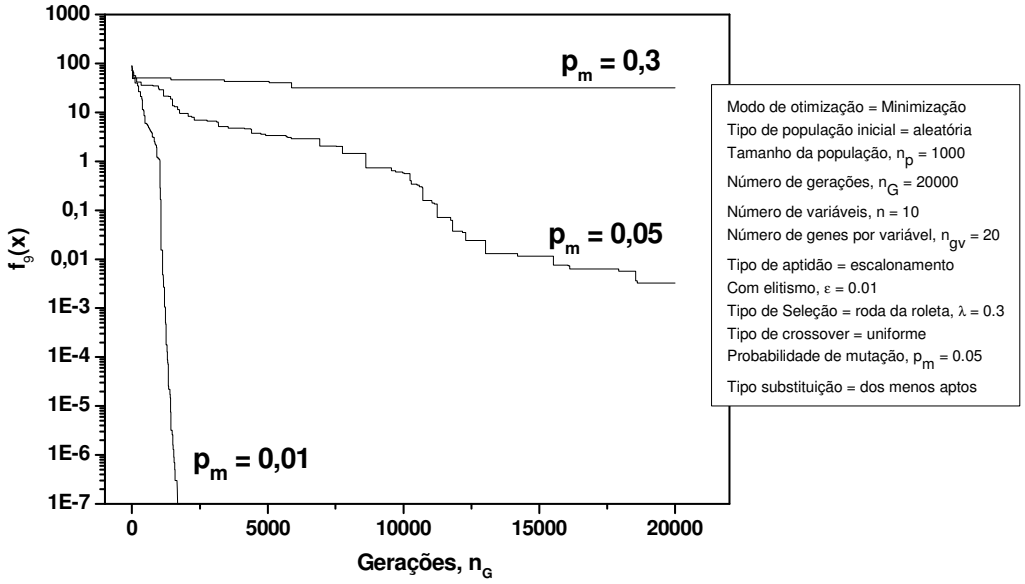


Figura 3.58 - Valores mínimos da função objetivo $f_9(\mathbf{x})$ para três probabilidades de mutação e 10 variáveis.

Também é notável que para as três probabilidades de mutação testadas a discriminação de resultados é claríssima. Os melhores resultados são obtidos usando a probabilidade de mutação de 1%, tanto para os valores da função objetivo quanto para a distância do indivíduo mais apto até à solução. Em posição intermediária aparecem os resultados, relativamente bons, correspondentes a 5% e por fim os resultados correspondentes a 30%, para os quais percebe-se que os valores tanto da função objetivo quanto da distância à solução, alteram-se muito pouco ao longo das gerações.

3.16.6 – Minimização da função $f_{10}(\mathbf{x})$ para n = 10 variáveis

A função $f_{10}(\mathbf{x})$ foi concebida para testar a idéia de pênalti desenvolvida no capítulo 2. Note que na sua definição parte do domínio levará à não definição da função para aquelas posições, as quais só serão bem conhecidas quando da avaliação da função objetivo para os indivíduos da população. Em conformidade com a estrutura de dados apresentada no capítulo 2, tem-se um indicador de pênalti associado a cada indivíduo. Quando houver a aplicação do operador substituição, todos os indivíduos que foram penalizados são removidos da população e substituídos por outros gerados estocasticamente.

Pelos resultados apresentados nas Figuras 3.60 e 3.61 o mecanismo de pênalti funcionou bem não permitindo que a existência de indivíduos inviáveis atrapalhassem o funcionamento do algoritmo genético.

Imagina-se que este mecanismo de pênalti pode ser utilizado para contornar uma infinidade de situações de restrição tanto sobre as variáveis independentes do domínio bem como sobre a função objetivo. Separadamente ou conjuntamente, basta que as restrições possam ser transformadas para a linguagem computacional que se esteja utilizando.

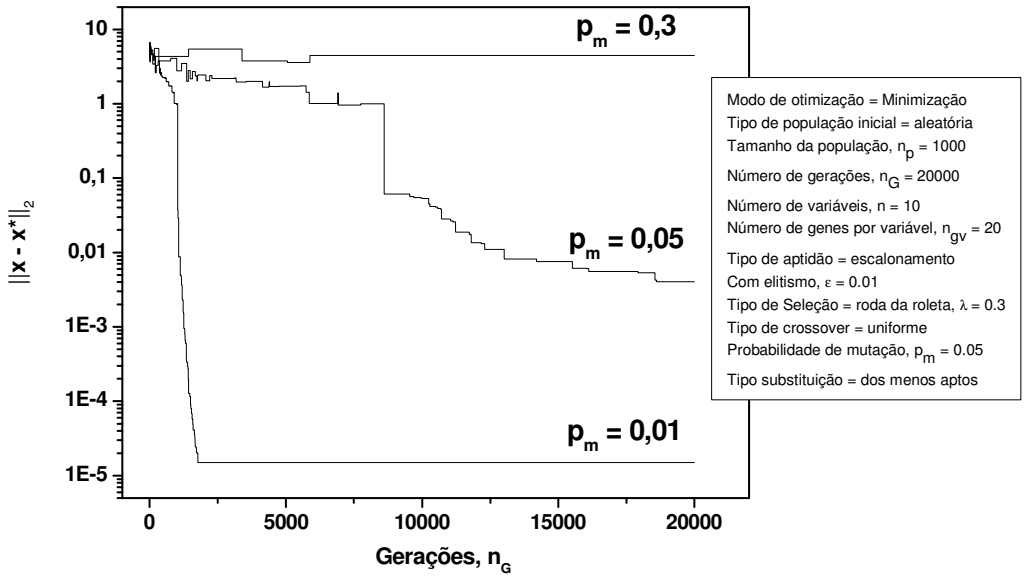


Figura 3.59 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_9(\mathbf{x})$, para três probabilidades de mutação e 10 variáveis.

3.17 – Minimização da função $f_6(\mathbf{x})$: Janelamento

Das funções determinísticas analisadas na seção 3.16 aquela para a qual o algoritmo genético apresentou os piores resultados foi a $f_6(\mathbf{x})$, conforme Figura 3.52 e 3.53. Então se concentra nesta seção à melhoria de sua solução. Para as outras, as mesmas técnicas podem ser aplicadas e certamente levará também a resultados assemelhados aos aqui apresentados.

Aplicou-se para a minimização da função $f_6(\mathbf{x})$ a idéia de janelamento que consiste em diminuir, à medida que a população evolui, a "janela" do domínio da função objetivo de tal forma a diminuir a dimensão do espaço de busca. O janelamento é realizado diminuindo 1% da extensão do domínio para cada variável independente. A diminuição do domínio, para cada variável, é sempre feita do lado mais distante dos dois extremos do domínio e a coordenada do indivíduo mais apto, para aquela variável. É necessário precaver-se para que o domínio, para uma dada variável, não se torne nulo, desta forma, o "lado esquerdo" do domínio não deve igualar nem suplantar o "lado direito".

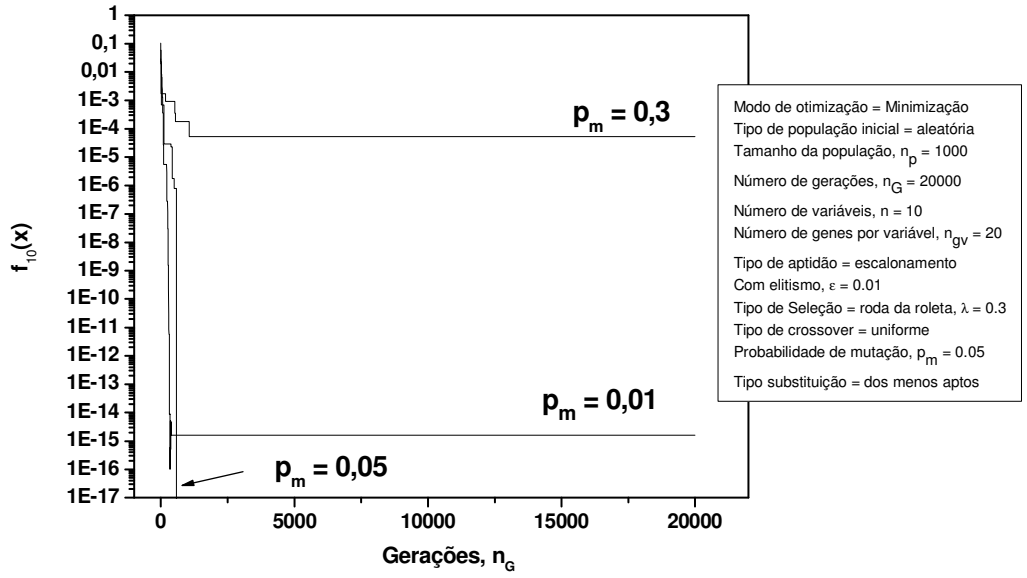


Figura 3.60 - Valores mínimos da função objetivo $f_{10}(\mathbf{x})$ para três probabilidades de mutação e 10 variáveis.

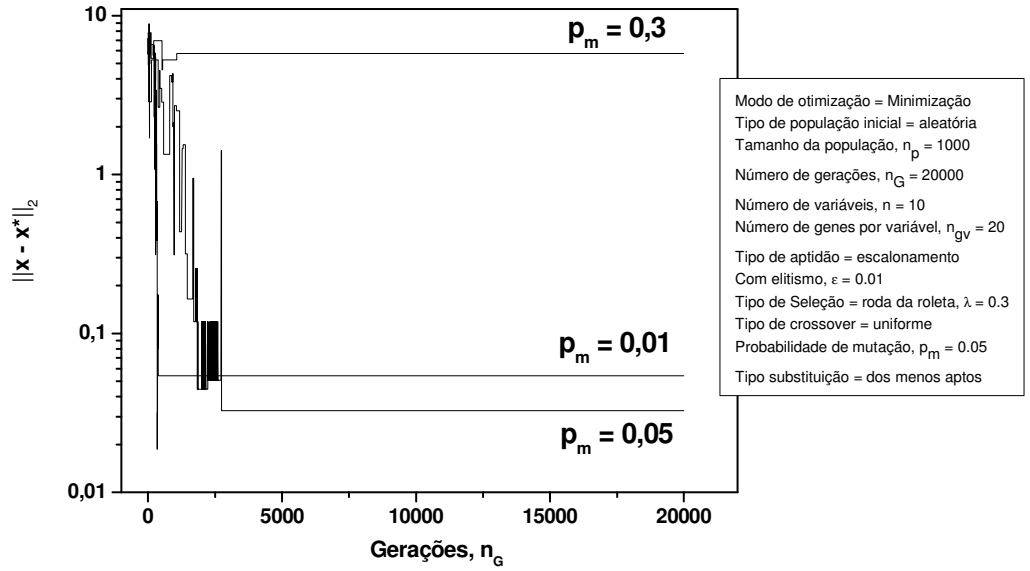


Figura 3.61 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_{10}(\mathbf{x})$, para três probabilidades de mutação e 10 variáveis.

3.17.1 – Minimização da função $f_6(\mathbf{x})$ para n = 10 variáveis, com janelamento

Aplicou-se o procedimento de janelamento à solução da minimização de $f_6(\mathbf{x})$ para o caso de 10 variáveis. Os resultados são apresentados nas Figuras 3.62 e 3.63. Também

constam das figuras os resultados obtidos sem o uso de janelamento, Figuras 3.52 e 3.53. Nota-se que os resultados melhoraram significativamente, pois, o melhor resultado obtido para a função objetivo sem janelamento era da ordem de 10 e o melhor resultado obtido com janelamento é da ordem de 10^{-10}.

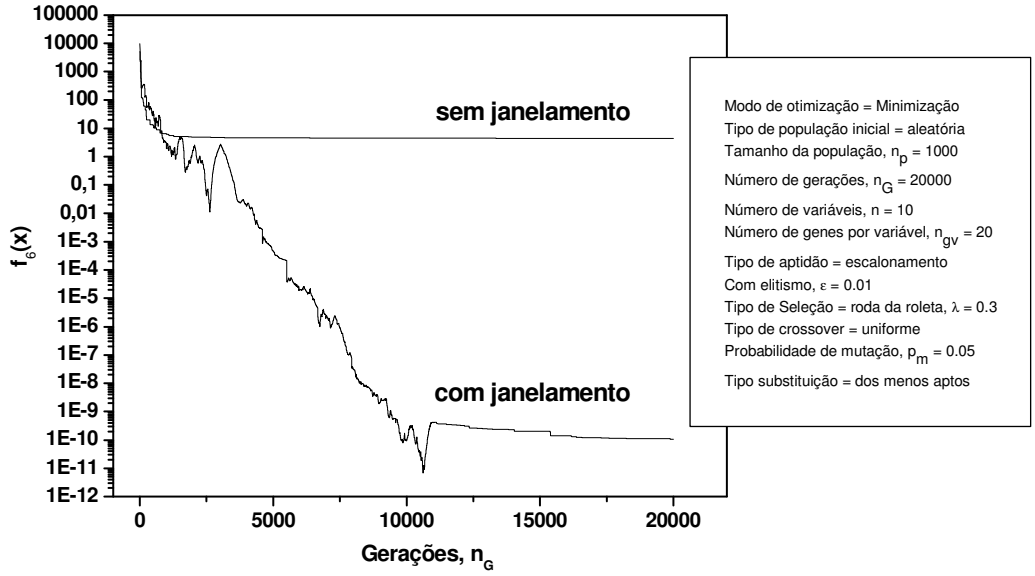


Figura 3.62 - Valores mínimos da função objetivo $f_6(\mathbf{x})$ para 10 variáveis, com e sem janelamento.

Também os resultados para a norma da distância do indivíduo mais apto até a solução do problema foram significativamente melhorados. Sem janelamento a distância era da ordem de 1 e com janelamento é da ordem de 10^{-5}.

Sem dúvida, o janelamento quando adequadamente utilizado, pode melhorar sensivelmente o resultado final bem como incrementar a taxa de convergência. Uma coisa está ligada com a outra. Tem-se que tomar cuidado para não deixar o ponto de otimização fora do espaço de busca. Isto pode ser obtido, na maioria das vezes, utilizando-se baixas taxas de janelamento. Nos resultados apresentados utilizou-se 1%, embora nestes casos talvez fosse possível usar valores ligeiramente maiores, tais como: 5% ou mesmo 10%. Também seria possível usar taxas de janelamento maiores no começo do processo evolutivo e depois, ir-se-ia, diminuindo seu valor.

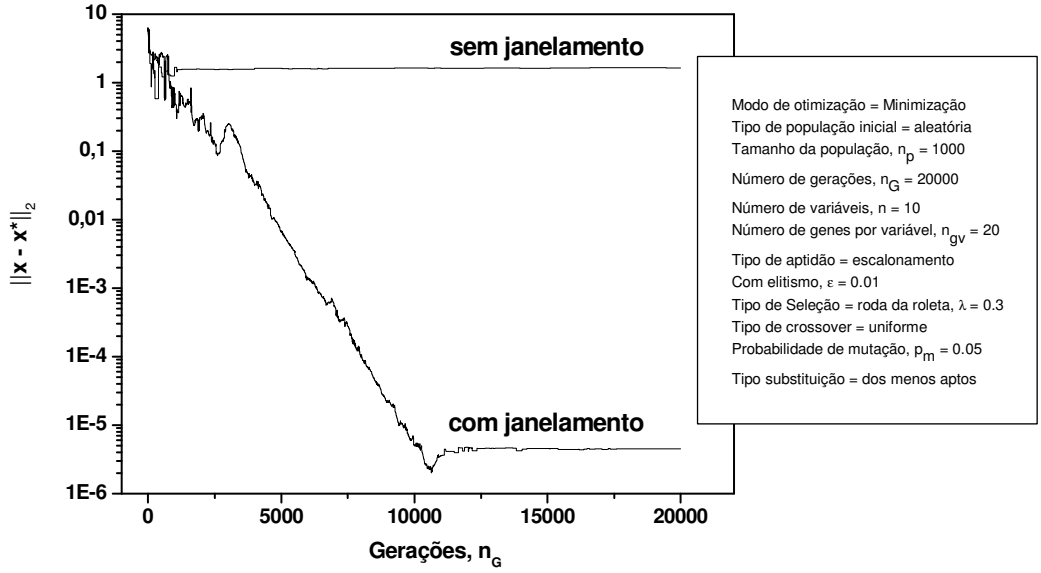


Figura 3.63 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_6(\mathbf{x})$, para 10 variáveis, com e sem janelamento.

3.17.2 – Minimização da função $f_6(\mathbf{x})$ para 100 e 1000 variáveis, com janelamento

Conforme apresentado na subseção anterior, a técnica de janelamento foi usada com sucesso para minimização da função objetivo $f_6(\mathbf{x})$, para 10 variáveis. Nesta subseção aumenta-se para 100 e 1000 o número de variáveis e aplica-se, então, a minimização com janelamento.

Na Figura 3.64, mostra-se os valores mínimos correspondentes aos indivíduos mais aptos das populações para minimização com janelamento da função objetivo com 10, 100 e 1000 variáveis. Para o caso com 10 indivíduos os resultados são os mesmos apresentados na Figura 3.62. O caso mais difícil é aquele para 1000 variáveis, pois, o espaço de busca aumenta muito a dimensão, quando comparado com o bidimensional ou mesmo com o de 10 dimensões. Nesta configuração o algoritmo genético reduziu o valor da função objetivo de, aproximadamente, 10^7 no início da evolução, para algo da ordem de 10^3, no final. Ou seja, o valor da função objetivo reduziu-se de um fator de 10^4.

A distância entre o indivíduo mais apto e a solução da otimização é apresentada na Figura 3.65, nota-se que foi reduzida de, aproximadamente 100 para 30, em problemas com 1000 variáveis, e de 30 para 10 em problemas com 100 variáveis.

Para 100 variáveis, os resultados são intermediários entre os correspondentes a 10 variáveis e os para 1000 variáveis.

Para melhorar os resultados para 100 e 1000 variáveis seria necessário aumentar o

tamanho da população, aumentar o número de gerações, aperfeiçoar o algoritmo de janelamento, e se necessário utilizar-se informações adicionais, se houver. No capítulo 5, aplica-se técnicas, conhecidas como *hill-climbing* para aumentar a convergência do algoritmo genético.

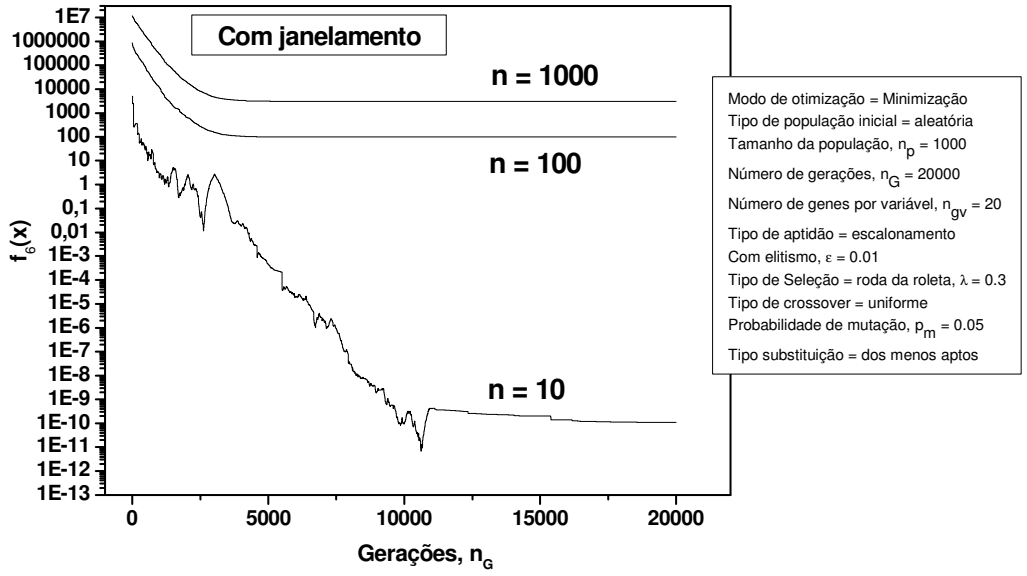


Figura 3.64 - Valores mínimos da função objetivo $f_6(\mathbf{x})$ para 10, 100 e 1000 variáveis, com janelamento.

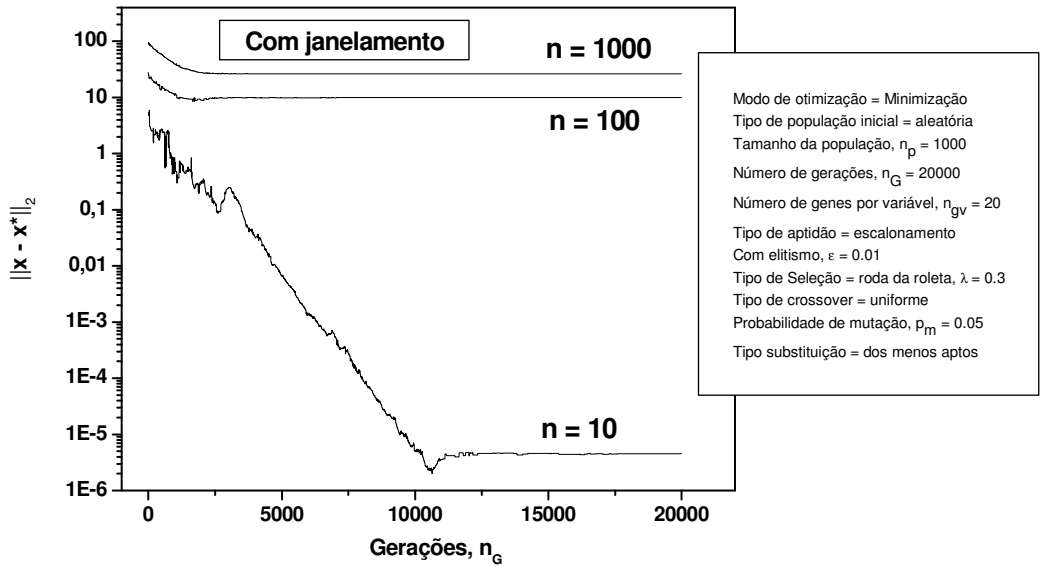


Figura 3.65 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_6(\mathbf{x})$, para 10, 100 e 1000 variáveis, com janelamento.

3.18 – Minimização da função $f_8(\mathbf{x})$: Janelamento e Filtragem

Seja um operador $\overline{(\ )}$ com as seguintes propriedades:

$$\overline{(Gauss(0,1))} \equiv 0\,, \tag{3.3a}$$

$$\overline{(A+B)} \equiv \overline{A} + \overline{B}\,, \tag{3.3b}$$

$$\overline{C} \equiv C\,, \tag{3.3c}$$

nas quais A e B são entidades estocásticas e C é uma entidade determinística ou não estocástica.

Lembrando que a função $f_8(\mathbf{x})$ foi definida como

$$f_8(\mathbf{x}) \equiv Gauss(0,1) + \sum_{i=1}^{n}[ix_i^4], \quad \mathbf{x} \in \Re^n, \quad f(\mathbf{x}) \in \Re\,,$$

então, aplicando-lhe o operador $\overline{(\ )}$, resulta

$$\overline{f_8(\mathbf{x})} \equiv \overline{Gauss(0,1) + \sum_{i=1}^{n}[ix_i^4]} = \overline{Gauss(0,1)} + \overline{\sum_{i=1}^{n}[ix_i^4]} = \sum_{i=1}^{n}[ix_i^4]\,. \tag{3.3d}$$

O exposto na equação (3.3d) indica que é possível encontrar uma solução para a minimização da função $f_8(\mathbf{x})$ através da utilização de algum filtro que elimine ou minimize os aspectos estocásticos associados com *Gauss(1,0)*. Para obtenção dos resultados apresentados nesta seção utilizou-se o seguinte filtro

$$\begin{aligned}\overline{f_8(\mathbf{x})} \equiv \frac{1}{K}\sum_{k=1}^{K} f_8(\mathbf{x}) &= \frac{1}{K}\sum_{k=1}^{K}[Gauss(0,1) + \sum_{i=1}^{n}[ix_i^4] = \\ &= \left\{\frac{1}{K}\sum_{k=1}^{K}[Gauss(0,1)]\right\} + \sum_{i=1}^{n}[ix_i^4], \quad K = 100\end{aligned} \tag{3.3e}$$

3.18.1 – Minimização da função $f_8(\mathbf{x})$, para duas variáveis com janelamento e filtragem

Os resultados obtidos para a minimização da função $f_8(\mathbf{x})$, com duas variáveis, utilizando-se de filtragem e de janelamento podem ser vistos na Figura 3.66 e 3.67, note-se que a amplitude de variação da função diminuíram bastante quando comparados com aqueles obtidos para a função f_4, e apresentados anteriormente neste capítulo. Lembre-se que a função f_4 é igual à função $f_8(\mathbf{x})$, para duas variáveis. Para a função f_4 os resultados obtidos sem filtragem e sem janelamento estavam em geral entre -4 e -1, enquanto aqueles apresentados na Figura 3.66 estão entre -0,5 e -0,2.

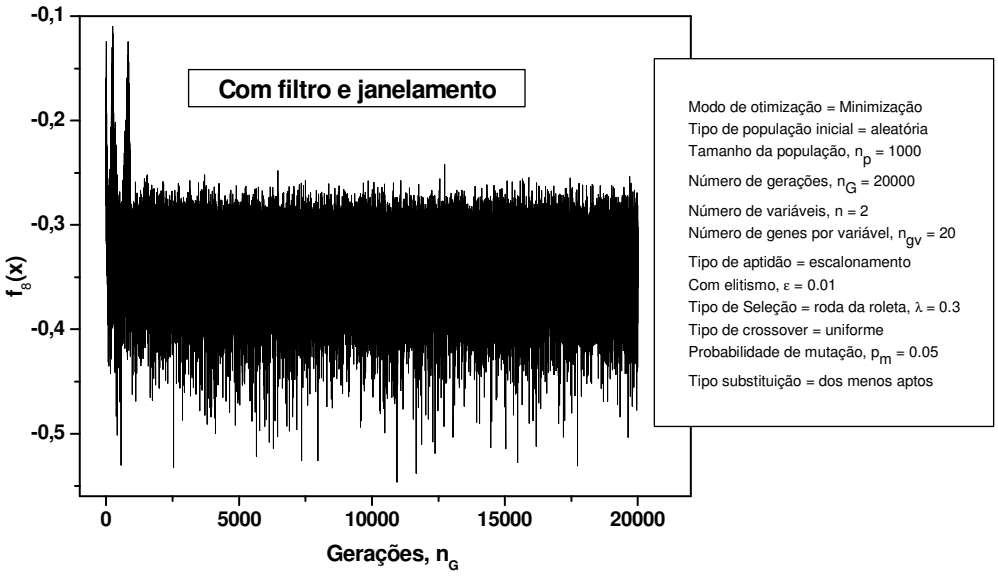


Figura 3.66 - Valores mínimos da função objetivo $f_8(\mathbf{x})$ para 2 variáveis, com filtragem e janelamento.

Quanto à distância entre o indivíduo mais apto e o ponto de minimização da função $f_8(\mathbf{x})$ o pior dos resultados indica valor da ordem de 0,3 sendo que também atinge-se valores até a ordem de 10^{-5}. O centro da "nuvem" de dados indica uma distância média da ordem de 10^{-1}.

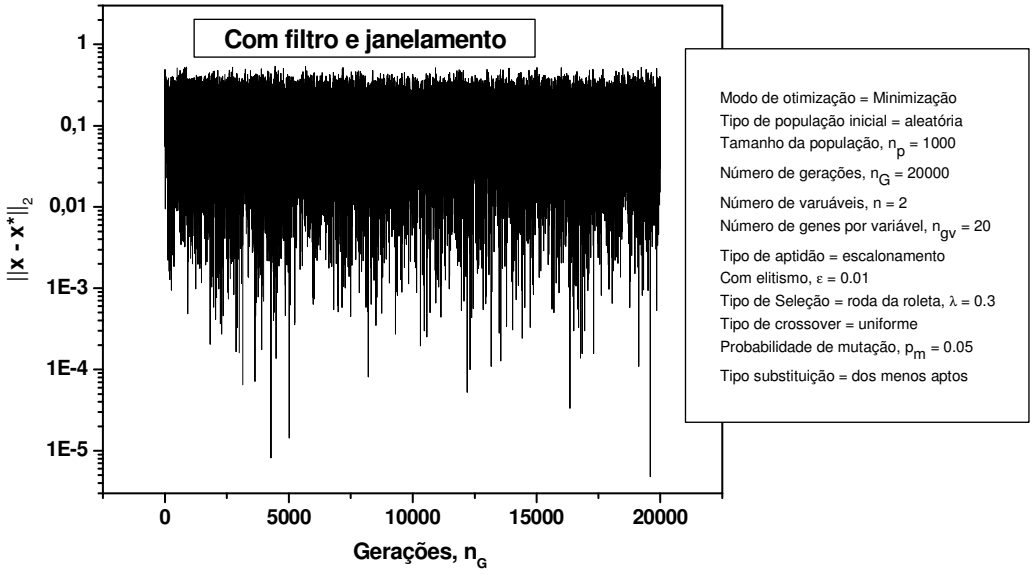


Figura 3.67 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_8(\mathbf{x})$, para 2 variáveis, com filtragem e janelamento.

3.18.2 – Minimização da função $f_8(\mathbf{x})$, para 10 variáveis com janelamento e filtragem

Nas Figuras 3.68 e 3.69 pode-se perceber que o uso de filtragem e de janelamento produz melhores resultados para a minimização de $f_8(\mathbf{x})$, 10 variáveis, do que quando se utiliza apenas a filtragem.

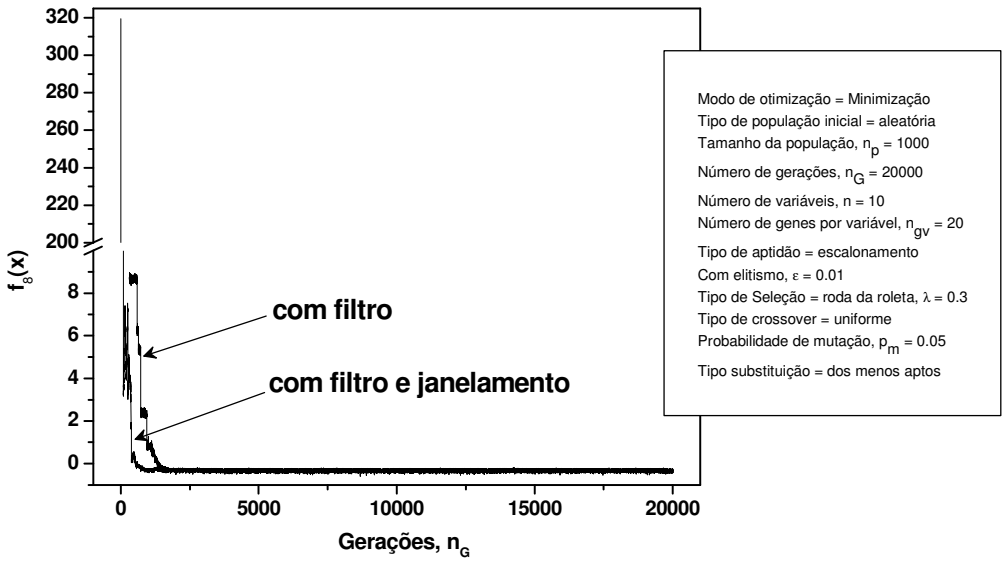


Figura 3.68 - Valores mínimos da função objetivo $f_8(\mathbf{x})$ para 10 variáveis, com filtragem e janelamento.

Por sua vez o uso apenas de filtragem também produz resultados melhores do que aqueles mostrados nas Figuras 3.56 e 3.57, e que foram obtidos sem o uso de filtragem ou de janelamento.

A distância do indivíduo mais apto até o ponto de minimização de $f_8(\mathbf{x})$ também diminuiu substancialmente com o uso de filtragem e janelamento, como pode ser vista na Figura 3.69.

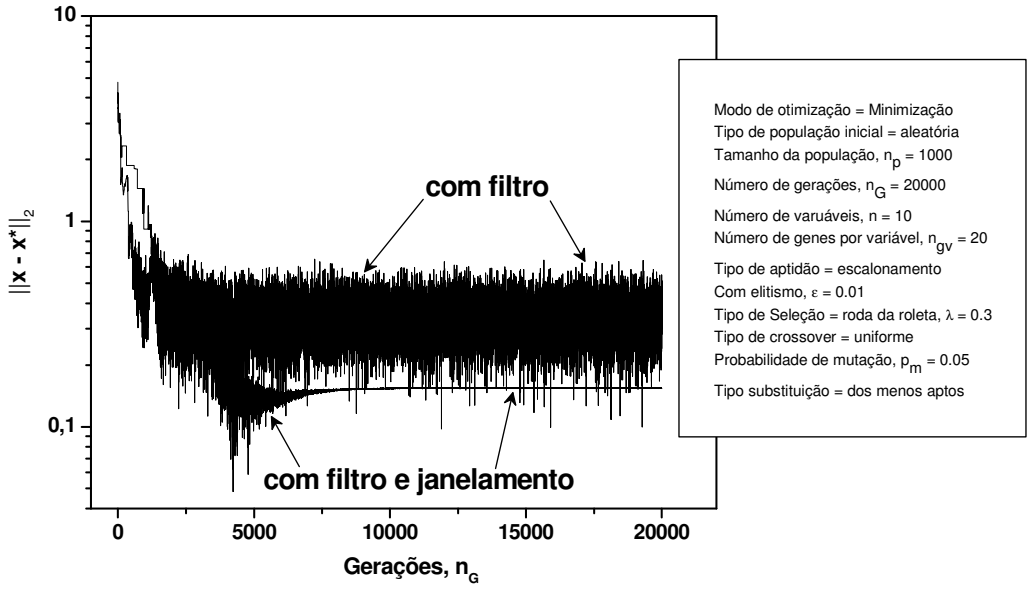


Figura 3.69 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_8(\mathbf{x})$, para 10 variáveis, com filtragem e janelamento.

3.18.3 – Minimização da função $f_8(\mathbf{x})$, para 100 variáveis com janelamento e filtragem

Para a minimização da função $f_8(\mathbf{x})$, com 100 variáveis, utilizando-se janelamento e filtragem, os resultados são promissores. Note-se que os valores da função objetivo foram reduzidos da ordem de 350000 para algo próximo de zero quando utilizando-se filtragem e janelamento, conforme pode ser visto na Figura 3.70. Também quando utilizando-se apenas filtragem o valor da função reduziu-se dos 350000 para algo da ordem de 150000.

Quanto à distância do melhor resultado obtido quando comparado ao valor médio exato a norma foi reduzida de, aproximadamente, 20 para 0,5, Figura 3.71.

Sem dúvida o uso de filtragem pode auxiliar na otimização de funções que tenham componentes estocásticos. Especula-se que talvez a filtragem possa também ser usada com

sucesso na otimização de funções determinísticas que tenham variação periódica temporal de alta freqüência e de baixa amplitude.

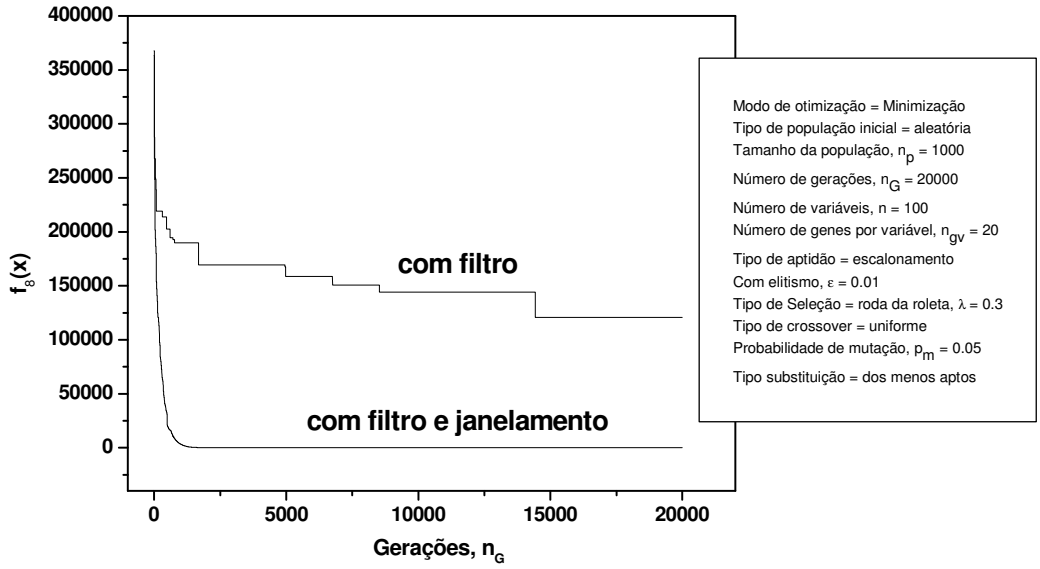


Figura 3.70 - Valores mínimos da função objetivo $f_8(\mathbf{x})$ para 100 variáveis, com filtragem e janelamento.

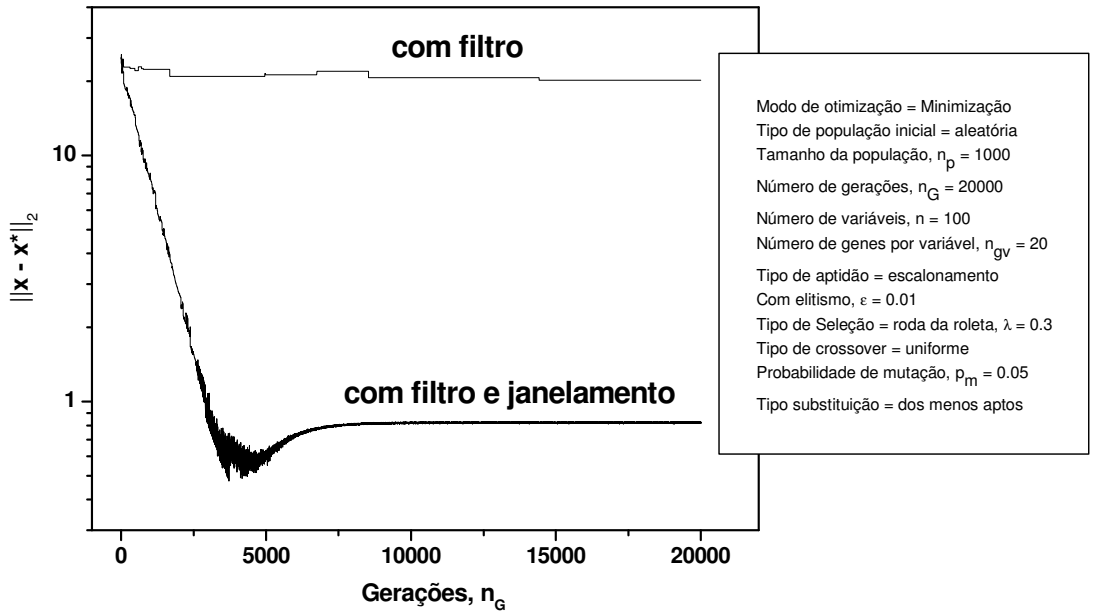


Figura 3.71 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_8(\mathbf{x})$, para 100 variáveis, com filtragem e janelamento.

4. ALGORITMO GENÉTICO –PROBLEMA DO CAIXEIRO VIAJANTE

O chamado problema do caixeiro viajante (TSP - *Traveler Salesman Problem*) aparece nas mais diversas áreas do conhecimento associado com otimização envolvendo entidades contáveis, ou seja, aquelas que podem ser associadas com números inteiros.

A metáfora de todos estes problemas é a do caixeiro viajante que precisa passar por um conjunto de cidades, interligadas por um sistema de rotas, seguindo um caminho ótimo que minimize uma dada função custo. A questão principal é: Qual a seqüência de cidades deve passar o caixeiro viajante para que sua viagem tenha custo mínimo?

Neste capítulo apresenta-se a aplicação de algoritmo genético a problema de combinatória, tendo como fio condutor a solução do TSP para 10, 20, 30 e 40 cidades.

Este problema atrai muita atenção devido a sua vasta aplicação nos mais variados campos do conhecimento, bem como pela dificuldade que apresenta para sua eficiente solução. Sabe-se das teorias de análise combinatorial que a quantidade de operações matemáticas para se resolver o TSP, via algoritmos combinatórios, é da ordem de $n_C!$, o fatorial do número de cidades. Suponha que se tenha disponível para resolver este problema um computador apto e disponível para executar 10^{12} FLOPS, então se gastaria os tempos apresentados na Tabela 4.1 para solução do TSP.

Tabela 4.1 – Tempo de processamento, estimado, para solução combinatorial do TSP

Número de cidades, n_C	**10**	**20**	**30**	**40**
Tempo de processamento	**$3{,}6\times10^{-6}$s**	**28,2dias**	**$8{,}4\times10^{12}$anos**	**$2{,}6\times10^{28}$anos**

Percebe-se a partir dos resultados estimados e apresentados na Tabela 4.1 que a solução do TSP via algoritmos puramente combinatoriais não é factível, exceto para pequena quantidade de cidades. O algoritmo genético surge como uma alternativa factível porque é capaz de resolver corretamente os TSPs de poucas cidades e resolver com certa aproximação os TSP de muitas cidades, em tempo compatível com as necessidades de uso dos resultados.

4.1 Definição do TSP

Para o TSP considerado neste capítulo, sejam as seguintes hipóteses:

- As cidades são dispostas em um plano e suas posições são conhecidas e fixas.
- A função custo associado com o trajeto de uma cidade A para uma cidade B é igual ao custo associado com o trajeto da cidade B para a cidade A. Seria simples introduzir assimetria no modelo da função custo, através da introdução de "pedágios" unidirecionais, que elevaria o custo do trajeto em uma direção de A para B, mas não elevaria de B para A.
- Todas as cidades podem ser atingidas a partir de qualquer outra cidade através de um trajeto em linha reta.
- O caixeiro viajante não passa duas vezes na mesma cidade.
- O custo do trajeto de uma cidade A para uma cidade B é igual à raiz quadrada da norma Euclidiana da distância entre as cidades.
- O custo total de um trajeto por várias cidades é igual à soma dos trajetos intercidades.

A função objetivo para o TSP é então modelada e toma a seguinte forma

$$f_{TSP}(c_1, c_2, ..., c_{n_C}) \equiv \sqrt{\sum_{i=1}^{(n_C-1)} \left\| \mathbf{x}_{c_i} - \mathbf{x}_{c_{i+1}} \right\|_2^2}, \tag{4.1}$$

na qual $\mathbf{x}_{c_i}$ é o par ordenado da posição da cidade c_i.

4.2 Representação genética do TSP

No algoritmo genético cada trajeto que o caixeiro viajante poderia realizar é um indivíduo da população, ou seja, a população é constituída de um conjunto de trajetos. Como as cidades são contáveis, isto é, pode-se contá-las, e também os números inteiros são contáveis, então, uma representação imediata e possível para uma caminho é uma seqüência de inteiros.

Na Figura 4.1 apresenta-se um conjunto de 15 cidades dispostas geograficamente em um plano. Estas cidades foram denominadas de **1, 2, 3, ..., 15**.

Na Tabela 4.2 são apresentados cinco indivíduos que podem ser gerados para o TSP envolvendo as 15 cidades dispostas na Figura 4.1. Para 15 cidades são possíveis 15! = 1307674368000 indivíduos.

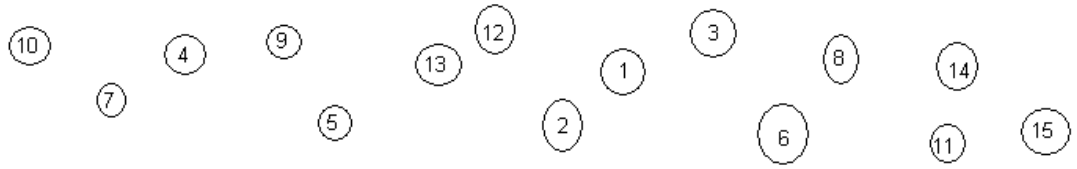


Figura 4.1 – Conjunto de 15 cidades dispostas geograficamente.

Tabela 4.2 – Cinco possíveis indivíduos, I1 a I5, associados do TSP da Figura 4.1

I_1	**2**	**9**	**14**	**3**	**4**	**5**	**6**	**11**	**13**	**10**	**15**	**7**	**1**	**12**	**8**
I_2	**15**	**11**	**3**	**6**	**13**	**12**	**8**	**7**	**14**	**10**	**1**	**5**	**2**	**4**	**9**
I_3	**9**	**1**	**6**	**12**	**7**	**8**	**4**	**15**	**5**	**11**	**10**	**3**	**2**	**13**	**14**
I_4	**14**	**7**	**15**	**3**	**2**	**9**	**4**	**10**	**13**	**6**	**8**	**1**	**5**	**11**	**12**
I_5	**8**	**14**	**2**	**4**	**7**	**13**	**6**	**11**	**15**	**12**	**1**	**10**	**5**	**9**	**3**

4.3 Crossover

Dos quatro operadores do algoritmo genético, **seleção**, ***crossover***, **mutação** e **substituição**, os operadores seleção e substituição operam sobre aspectos do fenótipo dos indivíduos e, portanto não dependem da representação do genótipo, assim, suas definições permanecem inalteradas. Os operadores que são afetados pela representação do tipo de problema que se pretende tratar são o *crossover* e a mutação.

4.3.1 Crossover de um ponto

Tal como no *crossover* com representação binária, apresentada no capítulo 2 e utilizada em otimização multidimensional no capítulo 3, o *crossover* de um ponto para o TSP parte da definição aleatória de um ponto de *crossover* que vai de 1 até ($n_C - 1$). Uma vez definido o ponto de *crossover* faz-se a troca de material genético entre os dois pais para gerar os dois filhos, de modo semelhante àquele descrito para a representação binária. Porém

devido à restrição de que o caixeiro viajante não passe duas vezes em uma mesma cidade, os alelos que representariam duplicidade, se trocados, são deixados nas suas posições originais. Troca-se apenas os alelos que não impliquem em duplicidade.

Por exemplo, considere os dois pais selecionados e apresentados na Tabela 4.3, considere também que o ponto de *crossover*, obtido aleatoriamente, seja a quarta posição intergenes, da esquerda para a direita.

Tabela 4.3 – Pais selecionados para realização de crossover

I_1	2	9	14	3	4	5	6	11	13	10	15	7	1	12	8
I_2	15	11	3	6	13	12	8	7	14	10	1	5	2	4	9

Na Tabela 4.4, apresenta-se nas primeiras quatro posições dos genes dos pais, em formato negrito-maiúsculo, os genes que permanecerão em suas posições. Nas últimas 11 posições estão os alelos candidatos a serem trocados. Destes existem alguns que estão no formato negrito-maiúsculo-sublinhado, os quais não poderão ser trocados porque implicaria em duplicidade do mesmo alelo no material genético dos filhos. Assim, estes alelos deverão permanecer também em suas posições, então, troca-se os alelos restantes que não estão em negrito.

Tabela 4.4 – Os alelos em negrito deverão permanecer em suas posições

I_1	**2**	**9**	**14**	**3**	4	5	**6**	**11**	13	10	**15**	7	1	12	8
I_2	**15**	**11**	**3**	**6**	**13**	12	8	7	**14**	10	1	5	**2**	4	**9**

Na Tabela 4.5 são apresentados os filhos F_1 e F_2 resultantes do crossover entre os pais I_1 e I_2. Note que ao final só foram trocados os pares de alelos em que ambos não estavam marcados em negrito.

Tabela 4.5 – Filhos resultantes do crossover

F_1	**2**	**9**	**14**	**3**	4	12	**6**	**11**	13	10	**15**	5	1	4	8
F_2	**15**	**11**	**3**	**6**	**13**	5	8	7	**14**	10	1	7	**2**	12	**9**

Para outros algoritmos de *crossover* basta proceder de maneira semelhantes àquela descrita no capítulo 2, porém tomando o cuidado para que não haja alelos duplicados no material genético dos filhos, conforme descrito nesta seção.

4.4 Mutação

A parte inicial do processo de mutação na representação de TSP é igual àquela apresentada no capítulo 2, ou seja, para cada alelo do cromossomo roda-se a roda da roleta com a probabilidade de mutação. Se a roda parar na posição "não mutar" progride-se para o próximo alelo e repete-se o giro da roda da roleta. Quando a roda parar na posição "mutar" sorteia-se aleatoriamente um segundo alelo distinto, e então os dois são trocados de posição entre si.

Por exemplo, considere o indivíduo apresentado na Tabela 4.6, para se realizar a mutação. Suponha que o segundo alelo, 7, passou no teste para mutação, então sorteia-se aleatoriamente um segundo alelo, por exemplo o quinto, 2. Uma vez definidos estes dois alelos eles são trocados, dentro do cromossomo do indivíduo.

Tabela 4.6 – Indivíduo e alelo selecionados para mutação

I_4	14	**7**	15	3	**2**	9	4	10	13	6	8	1	5	11	12

Na Tabela 4.7 apresenta-se o indivíduo mutado em relação ao segundo alelo. Completo o processo deste alelo, passe-se ao próximo e repete-se o procedimento até chegar no último alelo. Este procedimento elimina a possibilidade de se ter alelos duplicados no material genético.

Todo o restante das operações são muito semelhantes entre a representação binária usada em otimização multidimensional e a representação inteira do TSP.

Tabela 4.7 – Indivíduo com alelos mutados

I_4	14	**2**	15	3	**7**	9	4	10	13	6	8	1	5	11	12

4.5 TSP para 10 cidades

Aplicou-se o algoritmo genético desenvolvido para solução de TSPs para o caso simples de 10 cidades. De acordo com as estimativas apresentadas na Tabela 4.1 o TSP de 10 cidades pode ser resolvido muito rapidamente usando técnicas combinatoriais. De fato, percebe-se na Figura 4.2, que em poucas iterações, a função objetivo atingiu seu valor mínimo e estabilizou.

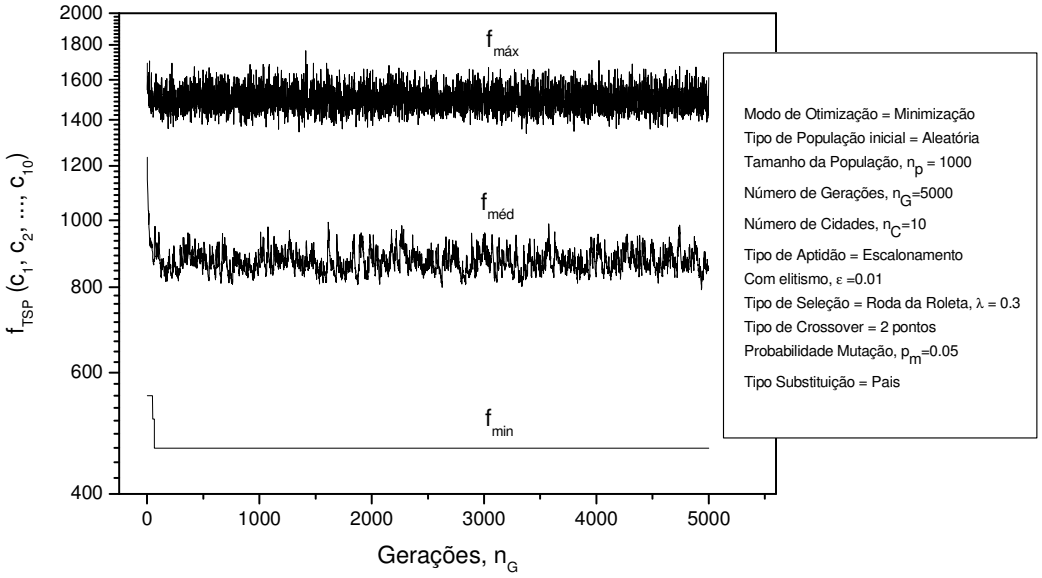


Figura 4.2 – Valores mínimos, médios e máximos da função objetivo para o TSP de 10 cidades.

Para mostrar a evolução do algoritmo genético na solução de TSPs apresenta-se os indivíduos mais aptos encontrados nas gerações (a) 10, (b) 20, (c) 50 e (d) 200 de uma dada rodada. Mesmo para a décima geração o indivíduo mais apto já representava um caminho relativamente otimizado. Para as gerações 20 e 50 os indivíduos mais aptos já estavam mais otimizados. Por fim, para a geração 200 ou mesmo para alguma geração anterior, o algoritmo genético já tinha encontrado o trajeto ótimo, Figura 4.3(d).

4.6 TSP para 20 cidades

De modo semelhante ao descrito para o TSP de 10 cidades, executou-se o algoritmo genético para o TSP de 20 cidades. Para este caso, de acordo com a Tabela 4.1, leva-se alguns dias para se resolver este TSP usando algoritmo combinatoriais.

Nota-se na Figura 4.4 que o algoritmo genético gastou aproximadamente duzentas gerações para que o indivíduo mais apto atingisse o valor mínimo da função objetivo e a solução se estabilizasse.

Na Figura 4.5 se vê que o algoritmo genético resolveu este TSP também de maneira exata, utilizando uma população com 1000 indivíduos, probabilidade de mutação de 10%, taxa de elitismo de 1%, taxa de *crossover* de 30%, *crossover* de 2 pontos, e substituição dos pais pelos filhos.

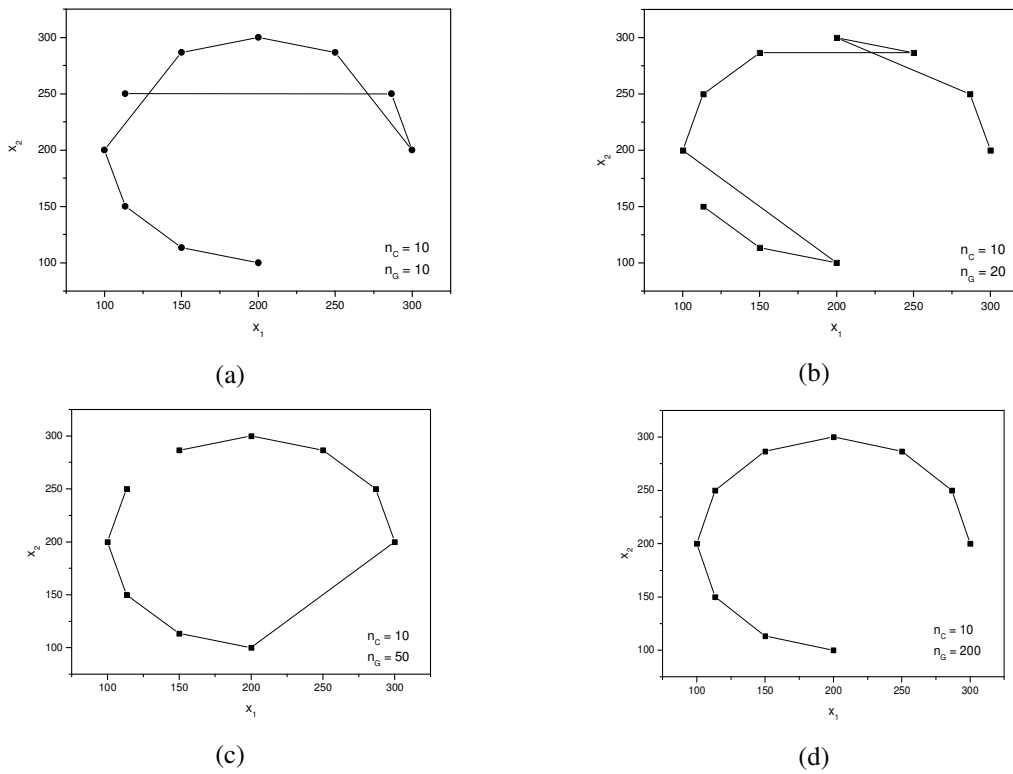


Figura 4.3 – Caminhos (indivíduos mais aptos) para diferentes gerações: 10, 20, 50 e 200

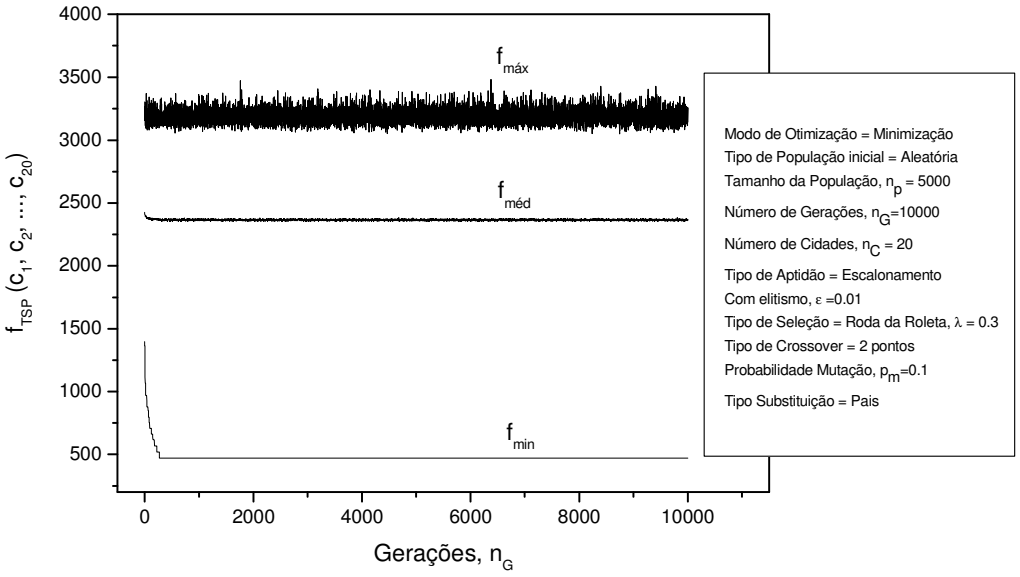


Figura 4.4 - Valores mínimos, médios e máximos da função objetivo para o TSP de 20 cidades.

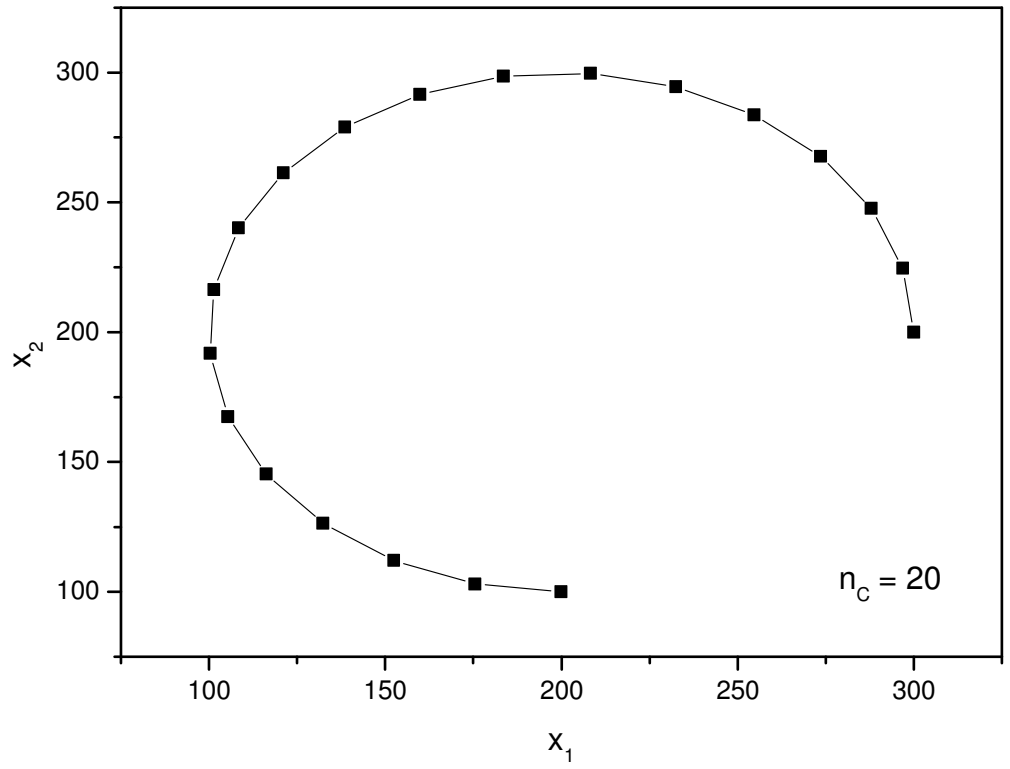


Figura 4.5 – Indivíduo mais apto para o TSP de 20 cidades

4.7 TSP para 30 cidades

No caso do TSP para 30 cidades o tempo de cálculo utilizando-se técnicas combinatoriais levaria milênios para se chegar à solução exata, Tabela 4.1.

Com a utilização do algoritmo genético pode-se obter soluções aproximadas ou mesmo a solução exata do TSP. Na Figura 4.6 são apresentados os valores mínimos, correspondentes ao indivíduo mais apto da população. O valor da função objetivo decresceu monotonicamente até por vota da geração 2000, depois estabilizou e não mais se alterou.

Na Figura 4.7 observa-se que a solução obtida pelo algoritmo genético não foi a exata, porém, foi bastante aproximada, tendo apenas um trecho entre duas cidades errado.

Provavelmente, com populações maiores, e também experimentando com os parâmetros: probabilidade de mutação, taxa de seleção, entre outros, seja possível resolver com valor exato este TSP, com um tempo de processamento factível, e a um custo baixo.

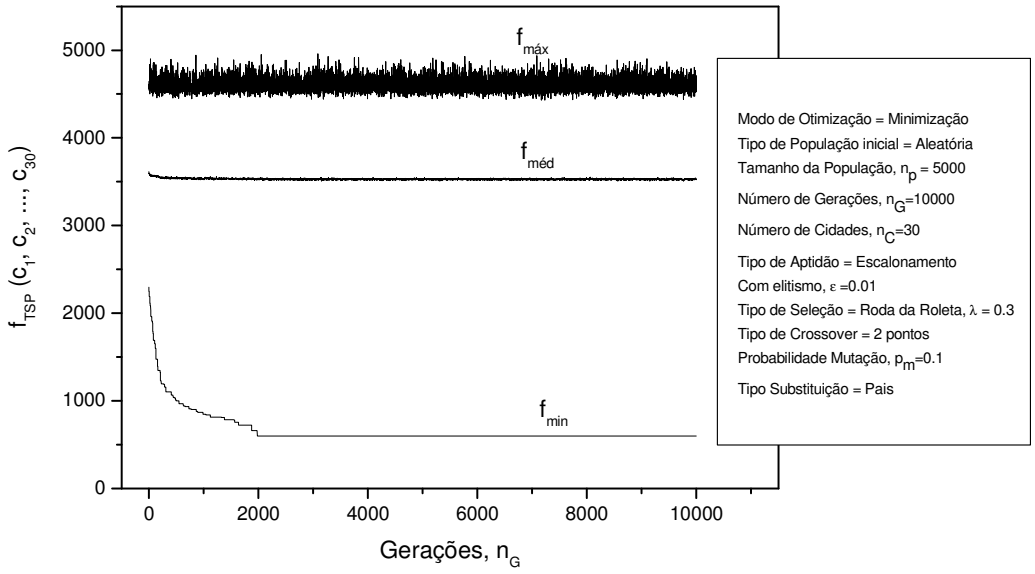


Figura 4.6 - Valores mínimos, médios e máximos da função objetivo para o TSP de 30 cidades.

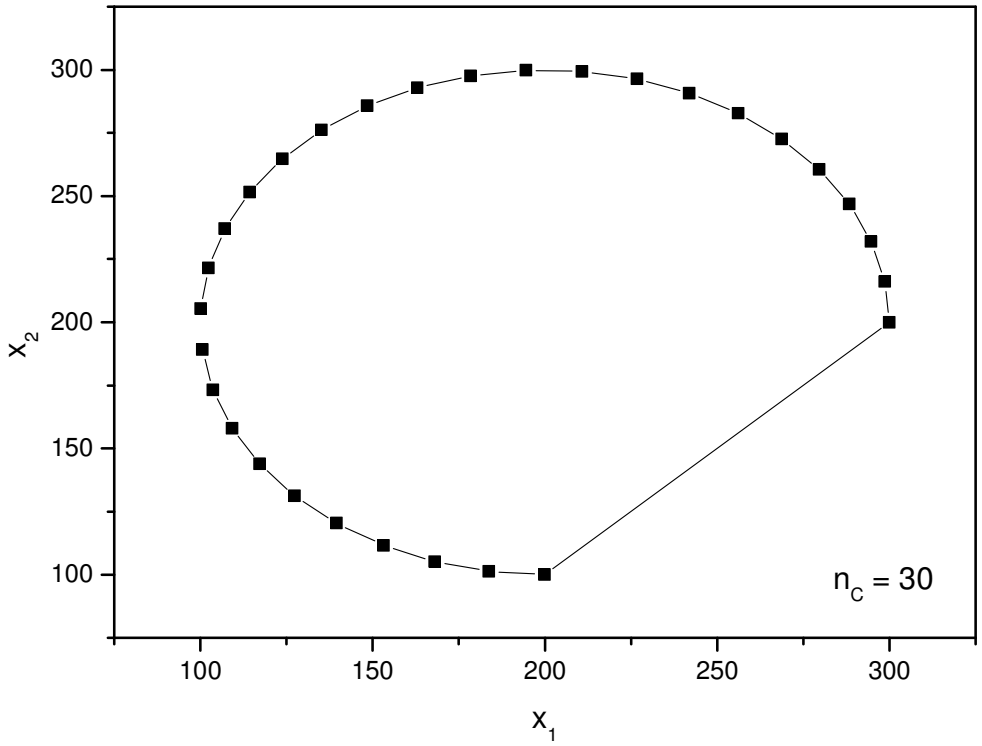


Figura 4.7 - Indivíduo mais apto para o TSP de 30 cidades

4.8 TSP para 40 cidades

Conforme explicado na dedução da Tabela 4.1 à medida que a quantidade de cidades aumenta, a dificuldade de solução do TSP aumenta fatorialmente. Se para 30 cidades o algoritmo genético não encontrou a solução exata, então será improvável que a encontre para o TSP de 40 cidades.

Na Figura 4.8 são apresentados os valores mínimos da função objetivo, correspondentes ao indivíduo mais apto. Do início da evolução da população o algoritmo genético reduziu o valor da função até estabilizar após aproximadamente duas mil gerações.

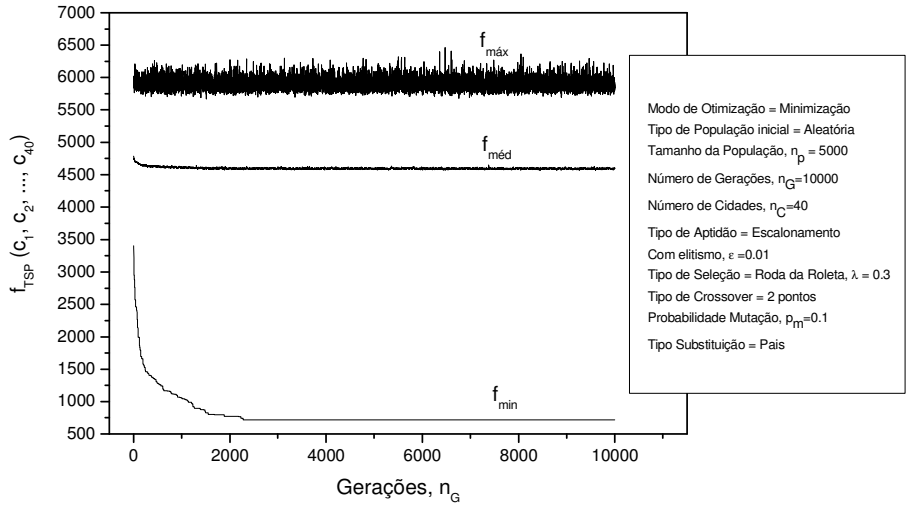


Figura 4.8 - Valores mínimos, médios e máximos da função objetivo para o TSP de 40 cidades.

A solução aproximada obtida pelo algoritmo genético, para o TSP de 40 cidades, pode ser vista na Figura 4.9. Embora não seja a solução exata, o caminho apresentado já está bastante otimizado e não apresenta grandes deslocamentos inúteis, como, por exemplo, atravessando pelo meio do círculo que as cidades formam.

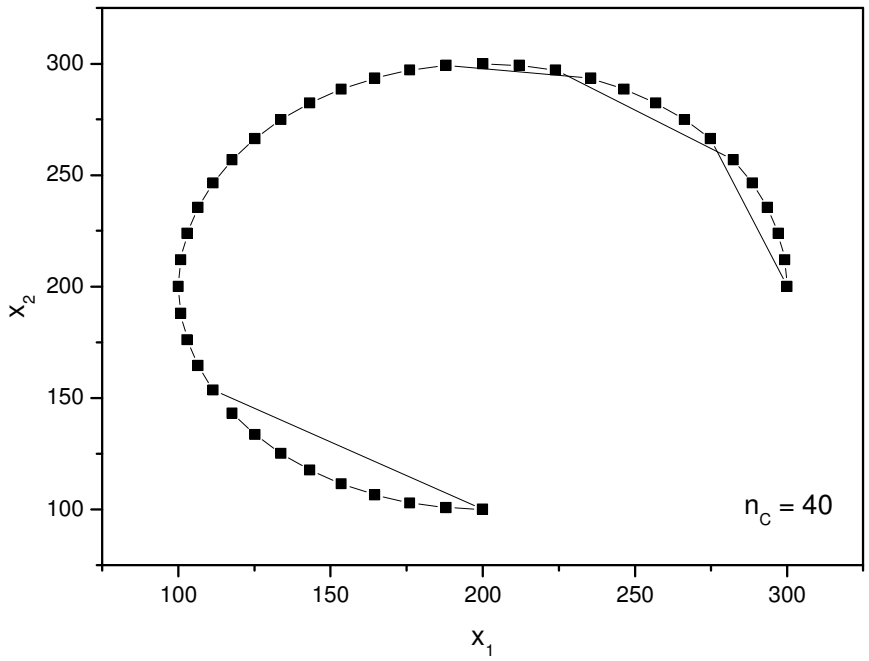


Figura 4.9 - Indivíduo mais apto para o TSP de 40 cidades

5. HILL-CLIMBING

Hill-climbing são algoritmos destinados a aumentar a convergência dos algoritmos genéticos utilizando informações adicionais sobre a função objeto. São estratégias "morro acima" para problemas de maximização ou "morro abaixo" para problemas de minimização. Conforme estabelecido nos capítulos 2 e 3, neste trabalho, todos os problemas de otimização são tratados como maximização, então, sempre estará se fazendo algoritmos do tipo "morro acima", conhecidos na literatura como *hill-climbing*.

Neste capítulo trata-se três tipos de algoritmos de *hill-climbing*, dois para problemas de otimização multidimensional e um para o TSP. Dos dois destinados a otimização multidimensional, um é de natureza genética e opera diretamente com o material genético dos indivíduos (alelos), e pode ser aplicado a todo tipo de função objetivo, o outro é baseado no gradiente da função e só se aplica a problemas cujas funções objetivo sejam de classe C^1. O *hill-climbing* associado ao TSP realiza operações diretamente nas seqüência de cidades, ou seja, o material genético de cada indivíduo.

Pode-se interpretar *hill-climbing* como sendo um operador de mutação ao "estilo Lamarqueano". Na mutação ao "estilo Darwiniano" os alelos são mutados diretamente por algum mecanismo físico, químico, biológico ou bioquímico, a mutação será passada adiante no material genético dos filhos, se o indivíduo gerar descendentes. A mutação também poderá se perder se o indivíduo for descartado pelo operador substituição. A mutação alterará o valor da avaliação do indivíduo e por conseqüência também alterará o valor de sua aptidão e portanto sua posição "social" na população. Em função de sua nova aptidão o indivíduo poderá ser com maior ou menor probabilidade selecionado para a reprodução (*crossover*). Em resumo a mutação ao "estilo Darwiniano" tem a sua efetivação através de mecanismo sutil que passa por aptidão, seleção para reprodução e por fim *crossover*. Já na mutação ao "estilo Larmarqueano" as variações no fenótipo de um indivíduo são imediatamente transformadas em mudanças no material genético, desde que produzam novos indivíduos mais aptos. Em realidade, em mutação "novos indivíduos" significam "o mesmo" indivíduo, porém com parte de seu material genético modificado.

Computacionalmente, *hill-climbing* funciona como se fosse um segundo operador de mutação. Basta adicionar a funcionalidade ao código já existente que não utilizasse o algoritmo.

5.1 Hill-climbing binário

O *hill-climbing* binário associado a problemas de otimização bidimensional consiste, computacionalmente, em fazer uma cópia do material genético de um determinado indivíduo e então promover mudanças no material genético do indivíduo-cópia. Após as alterações o indivíduo-cópia, agora já mutado, é avaliado e sua avaliação é comparada com a avaliação do indivíduo original. Se o indivíduo-cópia for mais apto do que o indivíduo original então o material genético do indivíduo original é substituído pelo material genético do indivíduo-cópia. Caso não seja, a operação termina sem alterar o material genético do indivíduo original. Este processo é realizado sobre todos os indivíduos da população, em todas as gerações.

A idéia contida no parágrafo anterior pode ser implementada em uma infinidade de maneiras. Descrever-se-á agora como foi implementada neste trabalho.

Seja um indivíduo original e um indivíduo cópia mostrados na Tabela 5.1.

Tabela 5.1 – Cromossomos de indivíduo original e de sua cópia, ambos com 16 alelos.

Indivíduo	Genótipos															
Original	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
Cópia	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**

Sorteia-se, aleatoriamente, um gene para se realizar a mudança no seu alelo. Considere, por exemplo, que o gene sorteado seja o oitavo, da esquerda para direita na Tabela 5.1. O alelo sorteado está sublinhado no genótipo do indivíduo cópia da Tabela 5.2. A primeira mudança consiste em alterar o estado do alelo. Se for **1** muda-se para **0**, e vice-versa. No caso o alelo sorteado é 1, então é alterado para 0, o material genético resultante pode ser visto no indivíduo "mudança 1" da Tabela 5.2. O indivíduo "mudança um" é avaliado, se ele for mais apto do que o indivíduo cópia então o processo continua, caso contrário, o processo é abortado. Em não sendo abortado, na seqüência se o primeiro alelo à esquerda do alelo referência (oitavo) for diferente deste então seu valor é mantido, caso contrário é mudado. No caso o alelo referência é **1** e o sétimo alelo de "mudança 1" é **0**, então o alelo fica inalterado, conforme mostra-se em "mudança 2". Prosseguindo o processo, como o sexto alelo de "mudança 2", **0**, é diferente do alelo referência **1**, seu valor fica inalterado, conforme pode ser visto no indivíduo "mudança 3" da Tabela 5.2. De maneira semelhante compara-se o quinto alelo, **1**, de "mudança 3" com o alelo referência, como são diferentes, altera-se o quinto alelo de "mudança 3" de **1** para **0**, conforme pode ser visto na Tabela 5.2. Como houve mudança no

material genético de “mudança 3” então este indivíduo é avaliado e comparado com o último indivíduo que teve mudança bem sucedida. Se “mudança 3” for o mais apto ele passa a ser a última mudança bem sucedida. E assim, repete-se o processo até cinco vezes, enquanto a mudança for neutra ou produzir indivíduos mais aptos, Tabela 5.2.

Tabela 5.2 – Cromossomos do indivíduo cópia e de suas alterações à esquerda.

Indivíduo	Genótipos															
Cópia	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
Mudança 1	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
Mudança 2	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
Mudança 3	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
Mudança 4	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
Mudança 5	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**

Quando o processo de mudança produzir um indivíduo menos apto, então, pára-se o processo à esquerda do alelo de referência e passa-se a um processo semelhante, de refinamento local, procurando alterar os alelos à direita do alelo referência. O processo é semelhante, porém da esquerda para a direita. Também são realizadas até cinco vezes, enquanto as mudanças forem neutras ou forem melhorias na aptidão. Quando houver a produção do primeiro indivíduo menos apto do que o indivíduo produzido na última mudança bem sucedida o processo é terminado e o indivíduo original é substituído pelo indivíduo cópia que estará acumulando a última mudança bem sucedida. Caso não haja nenhum mudança bem sucedida o processo de *hill-climbing* terá falhado. O processo à direita do alelo de referência e seus resultados são mostrados na Tabela 5.3. Para iniciar o processo à direita considerou-se como exemplo que a última mudança bem sucedida à esquerda tenha gerado o indivíduo mais apto “mudança 5”.

Tabela 5.3 – Cromossomos do indivíduo cópia e de suas alterações á esquerda.

Indivíduo	Genótipos															
Cópia	**1**	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
Mudança 5	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
Mudança 6	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
Mudança 7	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**1**
Mudança 8	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**	**0**	**1**	**1**
Mudança 9	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**1**
Mudança 10	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**

O processo de mudança para a direita é continuado por até cinco vezes, desde que produza indivíduos neutros ou mais aptos do que o produzido na última mudança bem sucedida, caso contrário o processo é terminado. Se houve alguma mudança bem sucedida o indivíduo original é substituído pelo último mais apto encontrado. Se não se encontrou nenhum indivíduo mais apto então o indivíduo original fica inalterado. Note-se que ao final deste processo a avaliação do indivíduo já está realizada, restando apenas ao operador seleção realizar o cômputo das aptidões de todos os indivíduos, quando de sua operação.

Repete-se este processo para todos os indivíduos da população e sucessivamente para todas as gerações.

5.1.1 Hill-climbing binário na otimização multidimensional, para 10 variáveis

O algoritmo de *hill-climbing* binário descrito, sumariamente, na seção anterior foi utilizado em conjunto com os outros operadores já descritos nos capítulos anteriores e obteve-se os resultados apresentados nas Figuras 5.1 a 5.5 para minimização, para 10 variáveis, das funções $f_5(\mathbf{x})$ a $f_{10}(\mathbf{x})$, exceto para $f_8(\mathbf{x})$.

Comparando-se os resultados das Figuras 5.1 a 5.5 com os respectivos obtidos sem o uso de *hill-climbing*, apresentados nas Figuras 3.50, 3.52, 3.54, 3.58 e 3.60, nota-se que para as funções $f_5(\mathbf{x})$, $f_6(\mathbf{x})$ e $f_{10}(\mathbf{x})$ os resultados apresentados a seguir foram melhores do que aqueles apresentados no capítulo 3.

Já para as funções $f_7(\mathbf{x})$ e $f_9(\mathbf{x})$ os resultados foram semelhantes com e sem o uso de *hill-climbing*.

Nota-se na Figura 5.5 que para os valores mínimos da função objetivo, $f_{10}(\mathbf{x})$, da ordem de 10^{-14} a 10^{-20} a curva apresenta uns repiques que teoricamente não deviam acontecer, uma vez que se está utilizando elitismo sem janelamento. Pode ser que o algoritmo de *hill-climbing* esteja destruindo, sob certas condições muito específicas, o indivíduo mais apto da população. Pela faixa de valores em que ocorre poderia ser também algum problema associado com a representação numérica do valor computado.

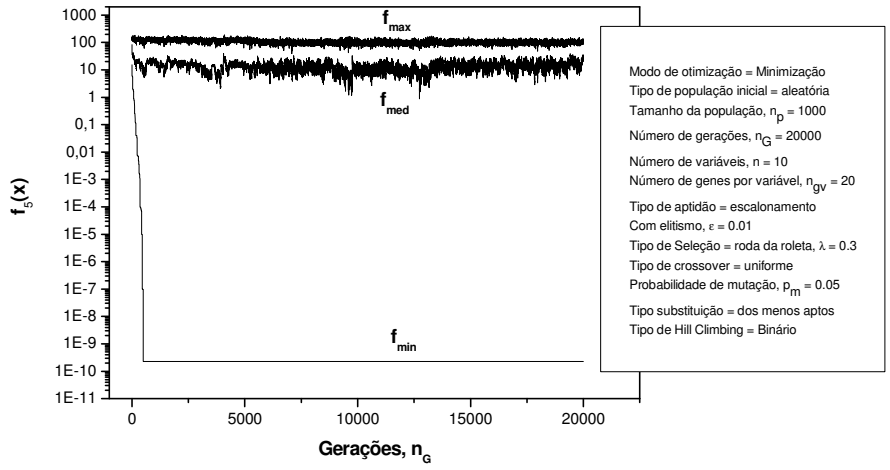


Figura 5.1 – Valores mínimos, médios e máximos da função objetivo $f_5(\mathbf{x})$, para 10 variáveis, com o uso de *hill-climbing* binário.

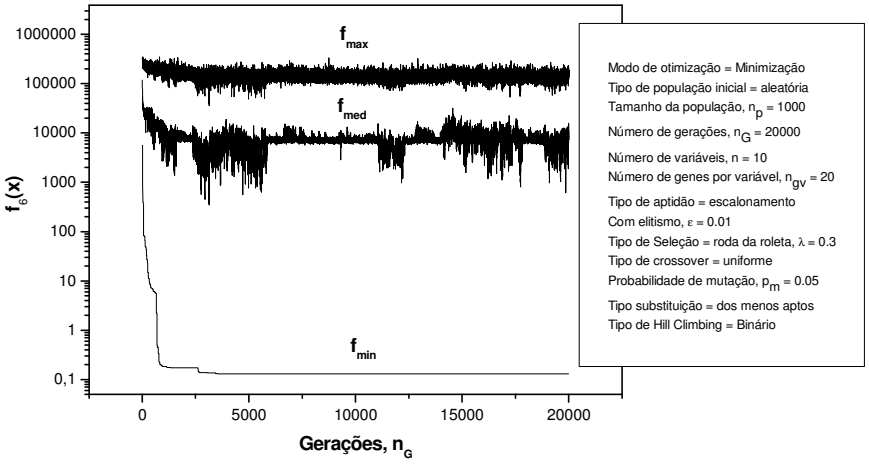


Figura 5.2 - Valores mínimos, médios e máximos da função objetivo $f_6(\mathbf{x})$, para 10 variáveis, com o uso de *hill-climbing* binário.

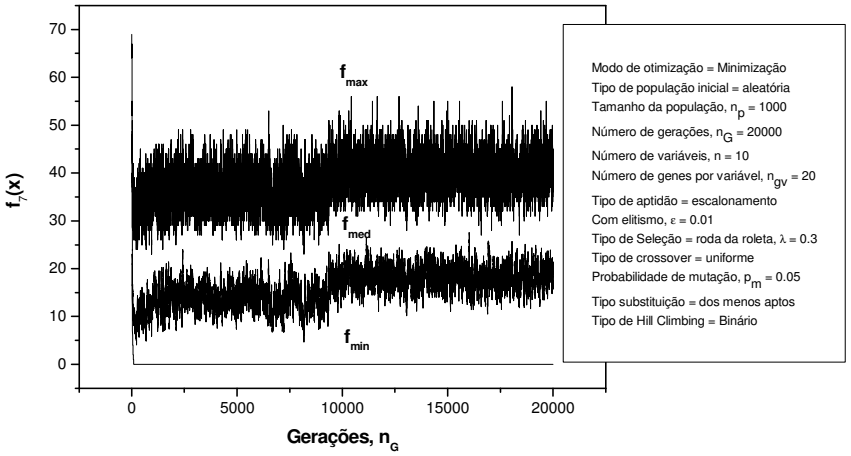


Figura 5.3 - Valores mínimos, médios e máximos da função objetivo $f_7(\mathbf{x})$, para 10 variáveis, com o uso de *hill-climbing* binário.

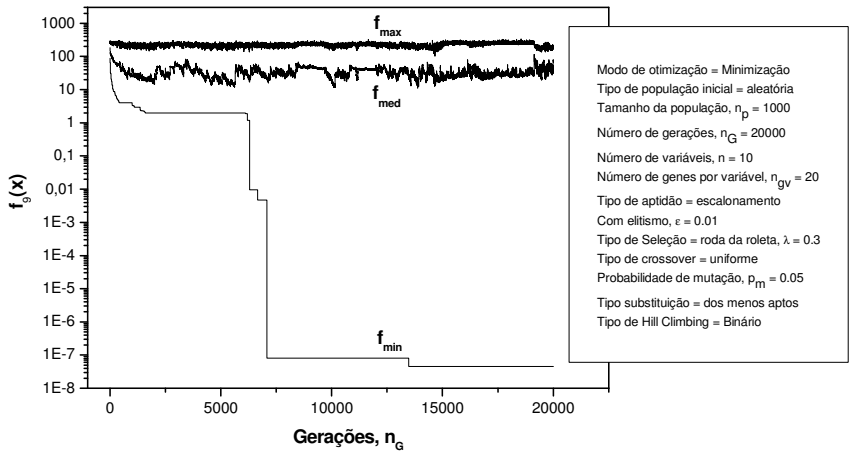


Figura 5.4 - Valores mínimos, médios e máximos da função objetivo $f_9(\mathbf{x})$, para 10 variáveis, com o uso de *hill-climbing* binário.

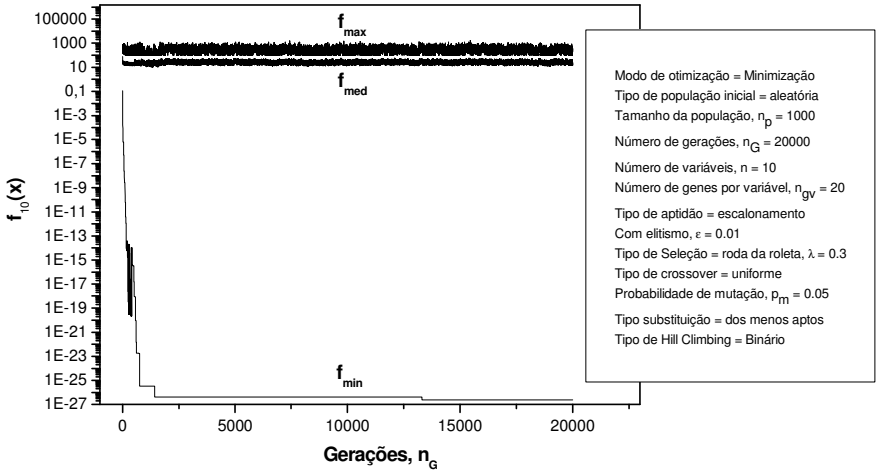


Figura 5.5 - Valores mínimos, médios e máximos da função objetivo $f_{10}(\mathbf{x})$, para 10 variáveis, com o uso de *hill-climbing* binário.

5.1.2 Hill-climbing binário na otimização multidimensional com e sem janelamento, para 100 variáveis

Conforme ficou estabelecido no capítulo anterior a solução de problemas de otimização multidimensional de dimensionalidade mais alta, por exemplo 100 ou 1000 variáveis, naturalmente são mais difíceis. Nesta subseção apresenta-se os resultados da minimização das funções $f_5(\mathbf{x})$ a $f_{10}(\mathbf{x})$, exceto $f_8(\mathbf{x})$, utilizando-se *hill-climbing* binário com e sem janelamento.

Para a função objetivo $f_5(\mathbf{x})$, Figura 5.6, nota-se que os resultados foram bons com *hill-climbing*, e muito bons com *hill-climbing* e janelamento. Reduziu-se o valor da função

objetivo até a ordem de 10^{-18}. Fato raro, pois, a representação dos números *floating-point* em geral suporta bem números pequenos não nulos até 10^{-15}. Também a distância do indivíduo mais apto até a solução exata da minimização da função objetivo, Figura 5.7, apresenta resultados expressivamente bons, tendo reduzido a distância a 10^{-9}. Nota-se que para o caso com *hill-climbing* e sem janelamento a solução foi evoluindo até a geração 1000, a partir da qual a resposta estabilizou-se e não mais reduziu nem o valor da função objetivo nem a distância do indivíduo mais apto até a solução. Isso, provavelmente, deve-se ao fato de que no modo sem janelamento o hiper-volume de busca fica constante, e no caso com janelamento o hiper-volume vai diminuindo sua dimensão. Foi deduzido no capítulo 2 que a acurácia de uma solução pode ser melhorada com a diminuição do domínio de busca, entre outras.

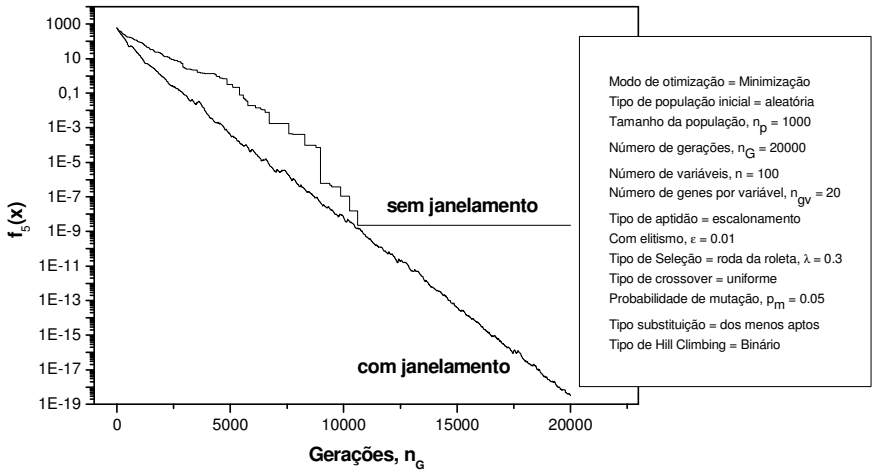


Figura 5.6 - Valores mínimos, médios e máximos da função objetivo $f_5(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* binário e janelamento.

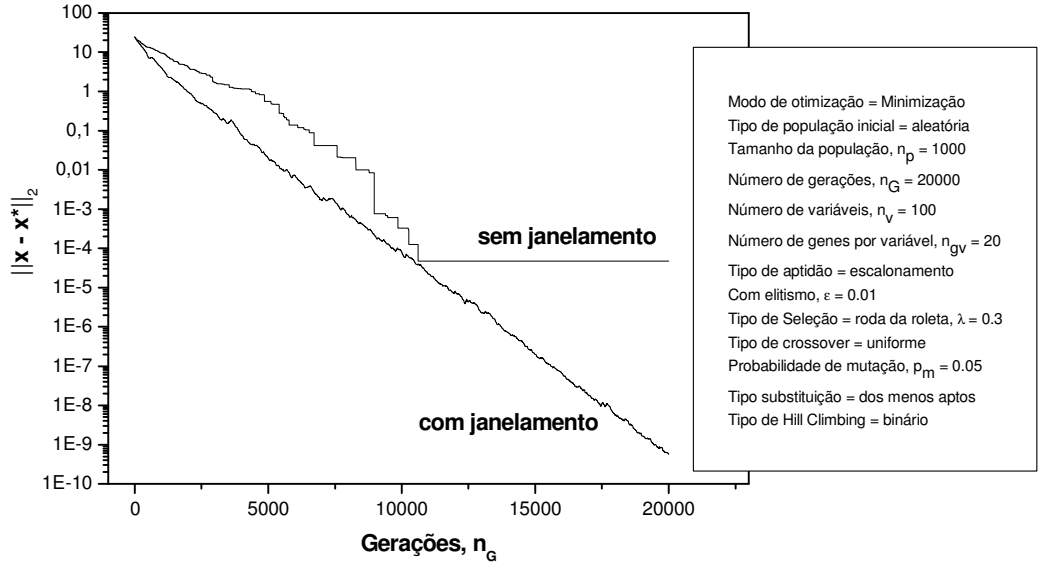


Figura 5.7 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_5(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* binário e janelamento.

Nos testes anteriores, a função $f_6(\mathbf{x})$ foi a que apresentou mais dificuldade para ser minimizada, quando usando *hill-climbing* e janelamento também esta função foi a que apresentou as maiores dificuldades para se reduzir o seu valor, bem como diminuir a distância entre o indivíduo mais apto e o ponto de mínimo exato. Note que os resultados apresentados nas Figuras 5.8 e 5.9 são relativamente semelhantes àqueles apresentados às Figuras 3.64 e 3.65, para 100 variáveis. Embora no caso com *hill-climbing* e janelamento percebe-se alguns aspectos que são melhores, por exemplo, na Figura 5.9 a menor distância atingida é da ordem de 5, enquanto na Figura 3.65 seja da ordem de 10.

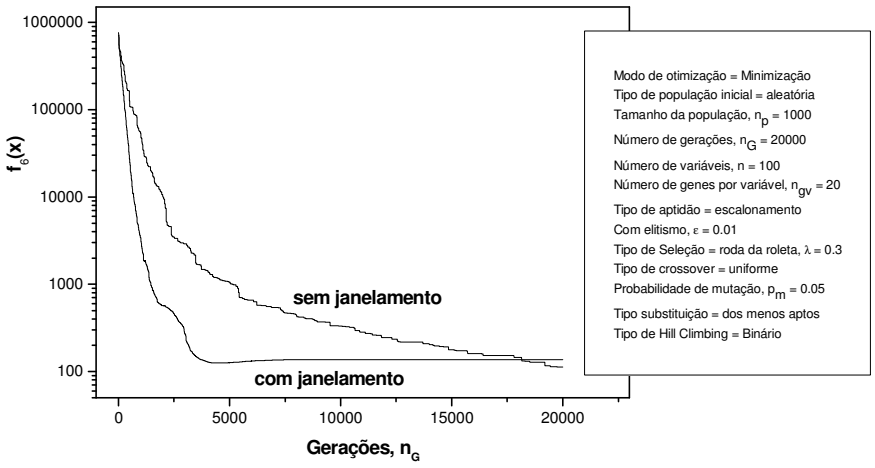


Figura 5.8 - Valores mínimos, médios e máximos da função objetivo $f_6(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* binário e janelamento.

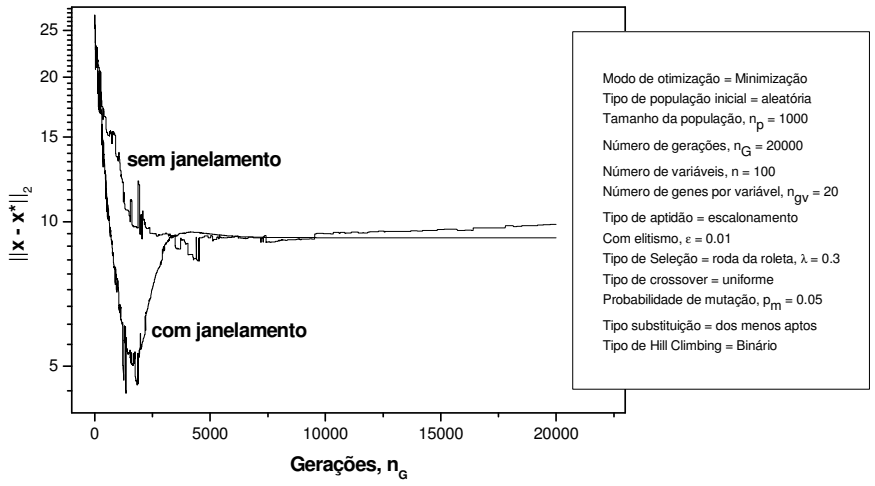


Figura 5.9 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_6(\mathbf{x})$, para 100 variáveis, com o uso de hill-climbing binário e janelamento.

De acordo com o estabelecido no capítulo 3 a função $f_7(\mathbf{x})$, provavelmente, é a de mais fácil minimização utilizando-se o algoritmo genético. Isso fica claro nos resultados obtidos com *hill-climbing* binário e sem janelamento. Os resultados para a função objetivo bem como para a distância até a solução exata tendem para zero, Figuras 5.10 e 5.11. No entanto, neste caso ocorreu um fato interessante, a solução com *hill-climbing* binário e janelamento produziu resultados piores do que os obtidos com *hill-climbing* binário e sem janelamento. Significa que o janelamento acabou deixando de fora do domínio de busca a região de mínimo da função. Esta região é vizinha ao canto (-5; -5; ...; -5), e o algoritmo de janelamento não estava adequadamente construído ou sintonizado para este caso.

Também os repiques nas curvas das Figuras 5.8 e 5.9 referentes à função objetivo $f_6(\mathbf{x})$, provavelmente, são devidos ao janelamento ter excluído o ponto do domínio de busca.

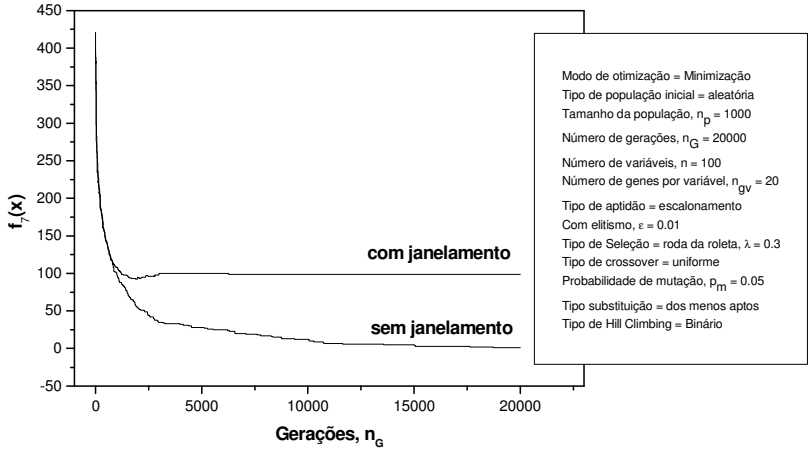


Figura 5.10 - Valores mínimos, médios e máximos da função objetivo $f_7(\mathbf{x})$, para 100 variáveis, com o uso de hill-climbing binário e janelamento.

Apesar de apresentar grande número de mínimos e máximos locais a função $f_9(\mathbf{x})$ juntamente à função $f_5(\mathbf{x})$ são as que apresentam resultados bastante acurados, Figura 5.12 e 5.13. O valor da função objetivo foi reduzido até a ordem de 10^{-13} e a distância até o mínimo exato foi reduzida até a ordem de 10^{-7}.

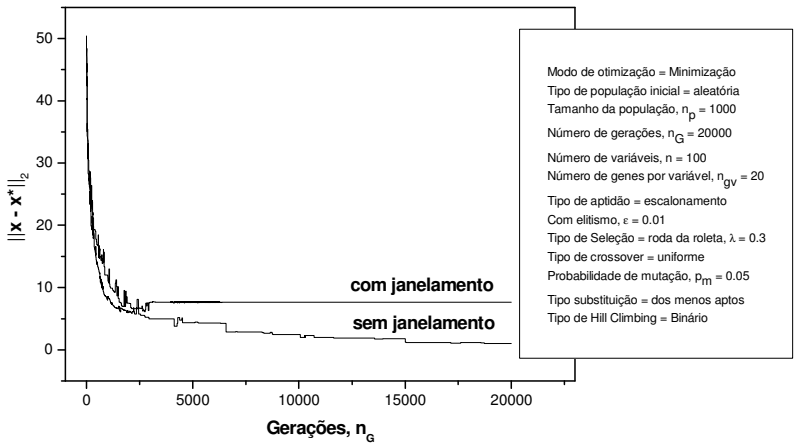


Figura 5.11 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_7(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* binário e janelamento.

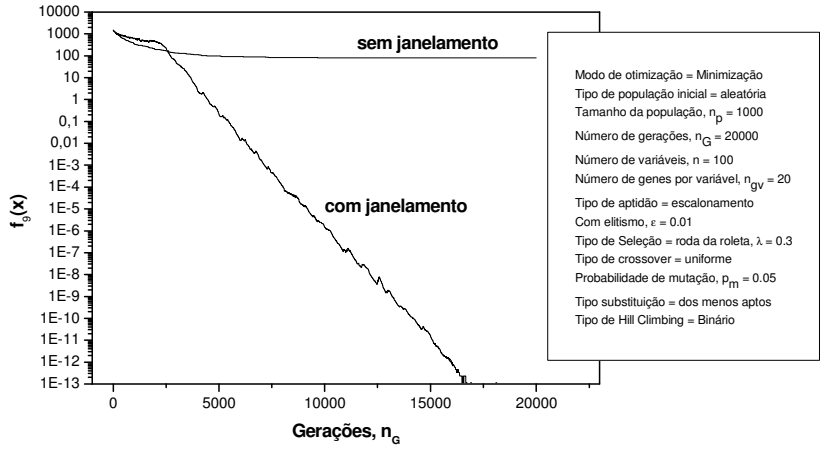


Figura 5.12 - Valores mínimos, médios e máximos da função objetivo $f_9(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* binário e janelamento.

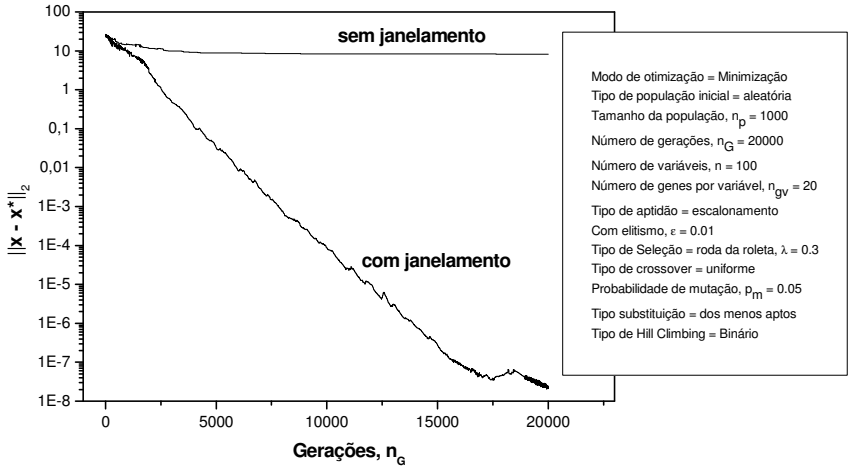


Figura 5.13 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_9(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* binário e janelamento.

O algoritmo de pênalti é ativado na minimização da função objetivo $f_{10}(\mathbf{x})$ porque ela não é definida em boa parte do espaço de busca. Também está ativo o algoritmo de *hill-climbing* binário, além de todas outras características do algoritmo genético. Nas Figuras 5.14 e 5.15 nota-se que quando também está atuando o mecanismo de janelamento, os resultados convergem para a diminuição dos valores da função objetivo e também da distância do indivíduo mais apto até a solução exata da otimização. No entanto existe alguma interação, mal controlada entre todos estes mecanismos que fazem com que o resultado obtido operando sem janelamento apresente acréscimos e decréscimos na função objetivo, fato que não deveria ocorrer uma vez que utiliza-se o elitismo, e o individuo mais apto não deveria ser perdido.

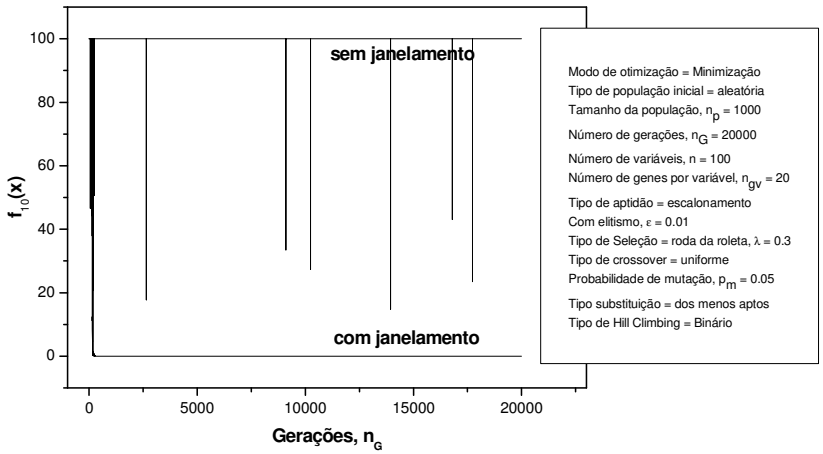


Figura 5.14 - Valores mínimos, médios e máximos da função objetivo $f_{10}(\mathbf{x})$, para 100 variáveis, com o uso de hill-climbing binário e janelamento.

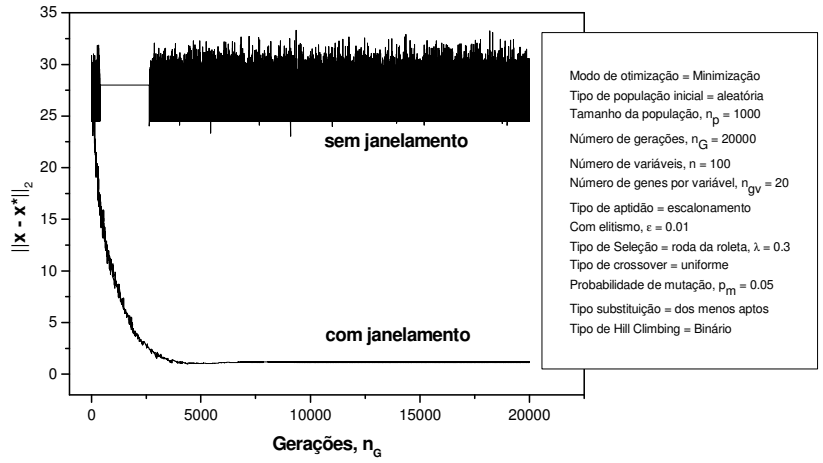


Figura 5.15 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_{10}(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* binário e janelamento.

5.2 *Hill-climbing* de gradiente

Funções objetivo de classe C^1 ou superior podem ser otimizadas utilizando-se métodos clássicos baseados no seu gradiente. Estes métodos, em geral, falham completamente quando aplicados na otimização de funções objetivo de classe C^0.

Nesta seção desenvolveu-se um algoritmo de *hill-climbing* inspirado no método da descida máxima, que é uma das variantes da grande família dos métodos do gradiente. Este *hill-climbing* de gradiente só pode ser aplicado à otimização de funções de classe C^1 ou superior, tais como as funções objetivo $f_5(\mathbf{x})$, $f_6(\mathbf{x})$ e $f_9(\mathbf{x})$.

O algoritmo do *hill-climbing* de gradiente é descrito, sucintamente, como segue. Para um indivíduo original $\mathbf{x}$ pode criar uma cópia modificada, $\tilde{\mathbf{x}}$, através da seguinte expressão

$$\tilde{\mathbf{x}} = \mathbf{x} + \alpha \mathbf{g}(\mathbf{x}), \quad \begin{cases} Minimização & \Rightarrow \quad \alpha = -1 \\ Maximização & \Rightarrow \quad \alpha = +1 \end{cases}, \tag{5.1}$$

na qual

$$\mathbf{g}(\mathbf{x}) \equiv \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \cdots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right]^T \tag{5.2}$$

Se $f(\tilde{\mathbf{x}}) > f(\mathbf{x})$ para maximização ou $f(\tilde{\mathbf{x}}) < f(\mathbf{x})$ para minimização, então $\mathbf{x}$ é substituído por $\tilde{\mathbf{x}}$, isto é $\mathbf{x} \leftarrow \tilde{\mathbf{x}}$ e $f(\mathbf{x}) \leftarrow f(\tilde{\mathbf{x}})$, e o processo continua por até cinco vezes. Caso a desigualdade não se verifique em algum passo o processo é descontinuado.

Após a realização desta primeira fase do processo, realiza-se então uma segunda fase de refinamento local, similar à primeira, seguindo o formalismo

$$\bar{\mathbf{x}} = \mathbf{x} + \beta \mathbf{g}(\mathbf{x}), \quad \begin{cases} Minimização & \Rightarrow \quad \beta = -1/10 \\ Maximização & \Rightarrow \quad \beta = +1/10 \end{cases}, \tag{5.3}$$

Se $f(\bar{\mathbf{x}}) > f(\mathbf{x})$ para maximização ou $f(\bar{\mathbf{x}}) < f(\mathbf{x})$ para minimização, então $\mathbf{x}$ é substituído por $\bar{\mathbf{x}}$, isto é $\mathbf{x} \leftarrow \bar{\mathbf{x}}$ e $f(\mathbf{x}) \leftarrow f(\bar{\mathbf{x}})$, e o processo continua por até cinco vezes. Caso a desigualdade não se verifique em algum passo o processo é descontinuado.

Este processo é realizado para todos os indivíduos da população, para todas as gerações.

5.2.1 Hill-climbing de gradiente em otimização multidimensional, para 10 variáveis

Novamente, a minimização da função objetivo $f_5(\mathbf{x})$, para 10 variáveis, utilizando o *hill-climbing* de gradiente apresenta resultados bem acurados, sendo que o valor da função convergiu para algo da ordem de 10^{-9}. Os resultados são muito semelhantes àqueles apresentados na Figura 5.1 obtidos usando-se *hill-climbing* binário.

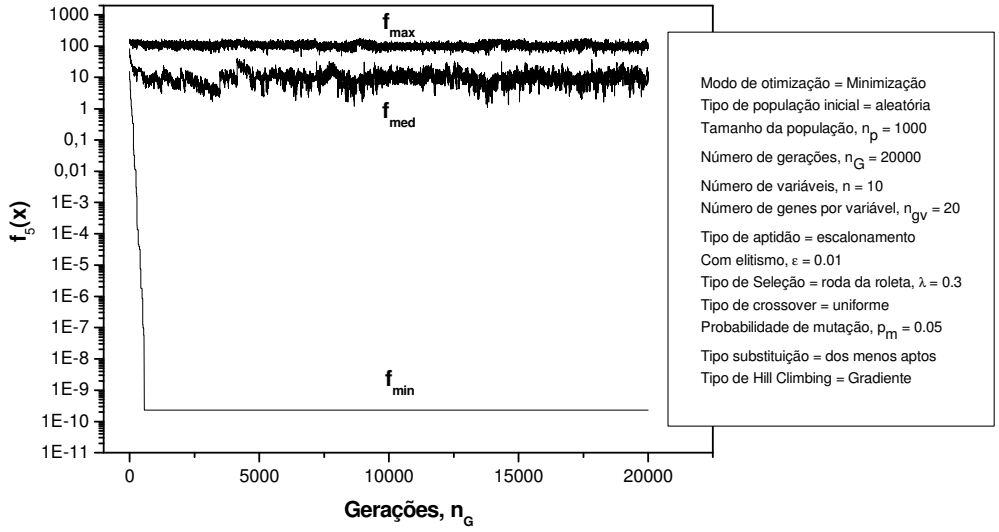


Figura 5.16 - Valores mínimos, médios e máximos da função objetivo $f_5(\mathbf{x})$, para 10 variáveis, com o uso de *hill-climbing* de gradiente.

Quando o *hill-climbing* de gradiente é aplicado para a minimização da função objeto $f_6(\mathbf{x})$ os valores correspondentes ao indivíduo mais apto convergiram para uma valor ligeiramente menor do que 0,1. A curva dos valores mínimos da função objetivo, Figura 5.17 é muita parecida com aquela apresentada na Figura 5.2 obtida usando o *hill-climbing* binário. Os melhores valores constantes na Figura 5.7 são ligeiramente maiores do 0,1, assim, os resultados produzidos utilizando-se *hill-climbing* de gradiente são ligeiramente melhores do que aqueles obtidos usando-se o *hill-climbing* binário.

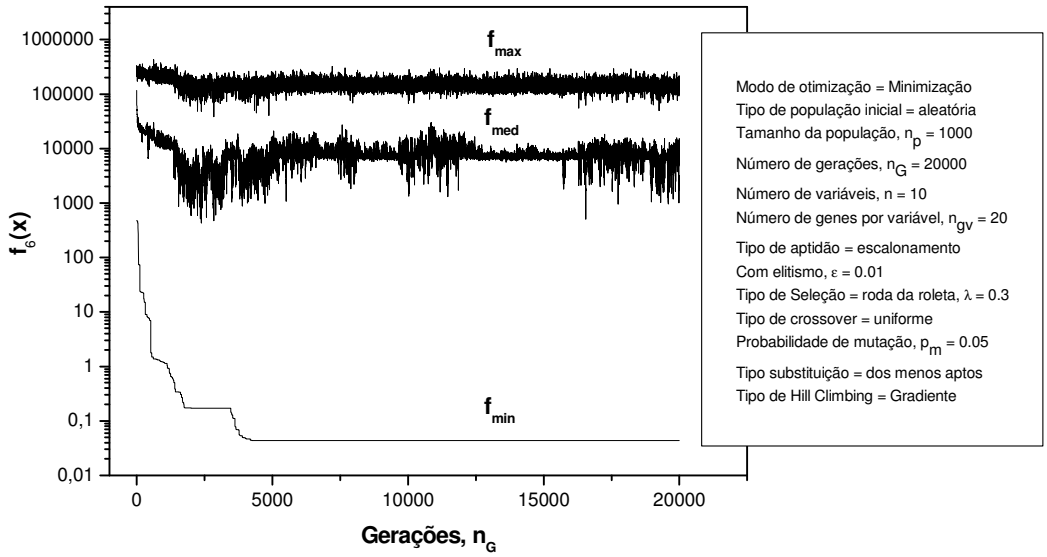


Figura 5.17 - Valores mínimos, médios e máximos da função objetivo $f_6(\mathbf{x})$, para 10 variáveis, com o uso de *hill-climbing* de gradiente.

Os resultados obtidos para a minimização da função objeto $f_9(\mathbf{x})$ para 10 variáveis são razoavelmente acurados com o valor mínimo da função convergindo para 10^{-7}. Os resultados, obtidos utilizando-se o *hill-climbing* de gradiente, são absolutamente semelhantes àqueles apresentados na Figura 5.3 e obtidos usando o *hill-climbing* binário.

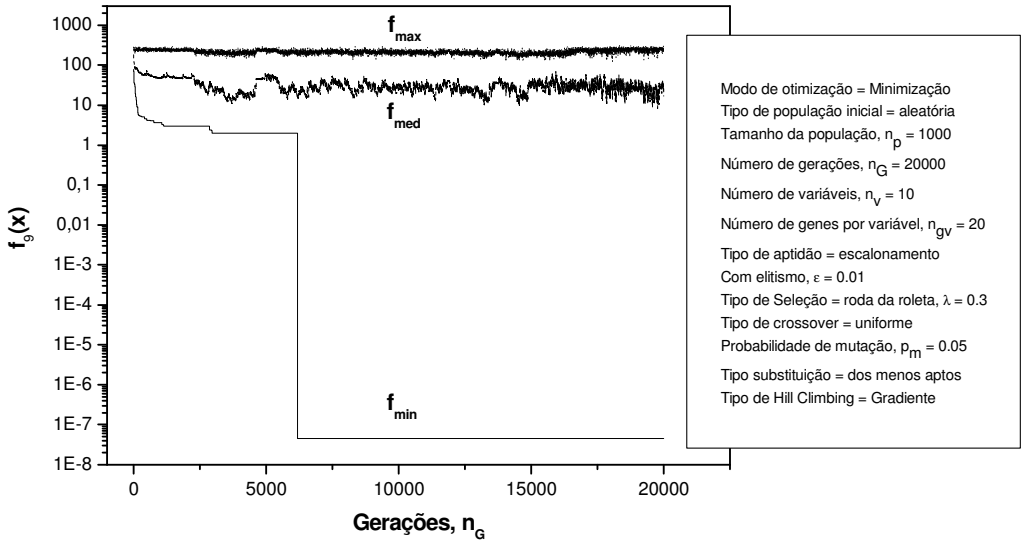


Figura 5.18 - Valores mínimos, médios e máximos da função objeto $f_9(\mathbf{x})$, para 10 variáveis, com o uso de *hill-climbing* de gradiente.

5.2.2 Hill-climbing de gradiente em otimização multidimensional, para 100 variáveis

Aplicou-se também o algoritmo genético com *hill-climbing* de gradiente com e sem janelamento para a minimização das funções objetivo de classe C^1, $f_5(\mathbf{x})$, $f_6(\mathbf{x})$ e $f_9(\mathbf{x})$, os resultados para as funções objetivo e também para as distâncias do indivíduo mais apto até as respectivas soluções exatas são apresentados nas Figuras 5.19 a 5.24. Em geral a combinação do *hill-climbing* de gradiente com janelamento produziu resultados melhores do que aqueles obtidos apenas com o *hill-climbing*. A eficiência na solução de um dado problema depende fortemente na natureza da função objetivo, neste trabalho observou-se que as soluções das minimizações das funções $f_5(\mathbf{x})$, $f_7(\mathbf{x})$ e $f_9(\mathbf{x})$ geraram resultados mais acurados e convergiram mais rapidamente do que nos casos das soluções da otimização de $f_6(\mathbf{x})$ e $f_8(\mathbf{x})$. Por exemplo, na Figura 5.22 tem-se os resultados para a distância do indivíduo mais apto até a solução exata da minimização da função $f_6(\mathbf{x})$. Os melhores resultados apontam

para uma distância mínima da ordem 10, enquanto que os valores correspondente à função $f_5(\mathbf{x})$, Figura 5.20 são da ordem de 10^{-5}.

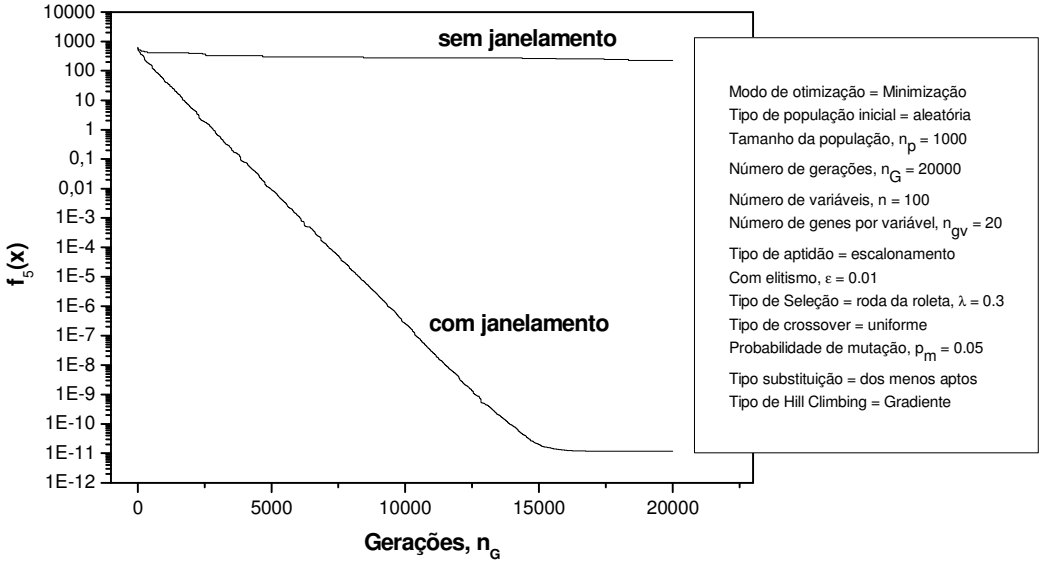


Figura 5.19 - Valores mínimos da função objetivo $f_5(\mathbf{x})$, para 100 variáveis, com o uso de hill-climbing de gradiente e janelamento.

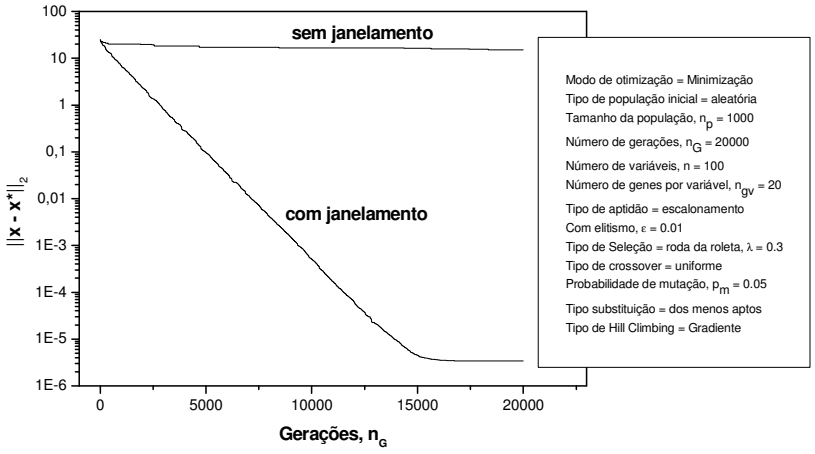


Figura 5.20 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_5(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* de gradiente e janelamento.

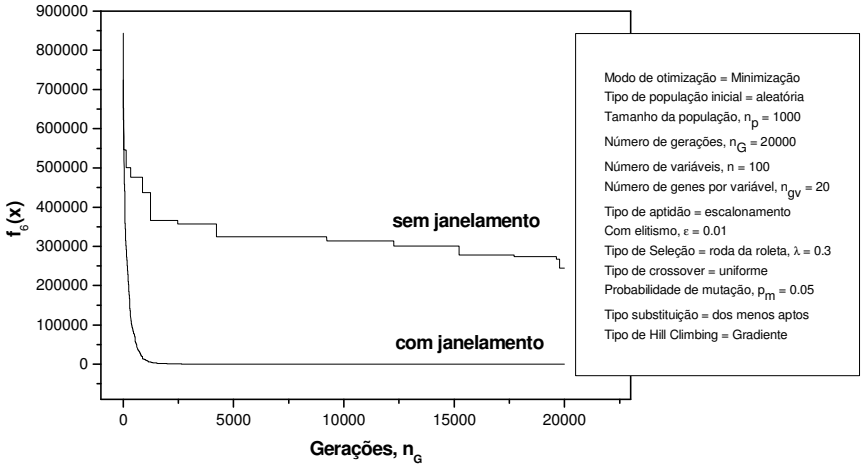


Figura 5.21 - Valores mínimos da função objetivo $f_6(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* de gradiente e janelamento.

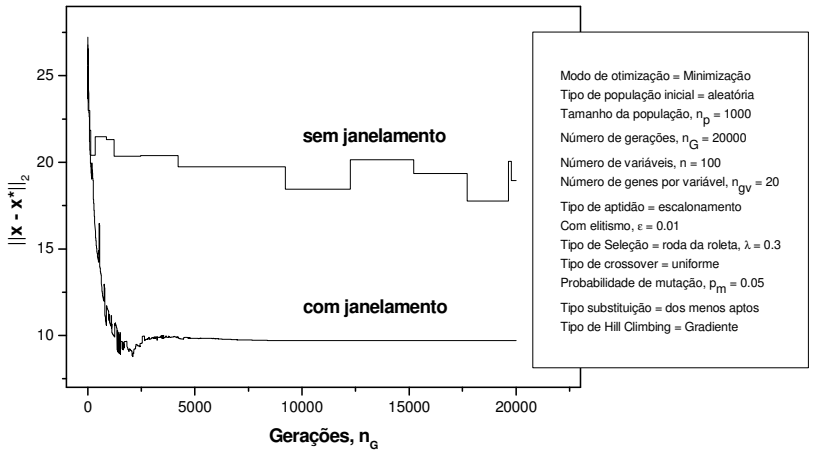


Figura 5.22 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_6(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* de gradiente e janelamento.

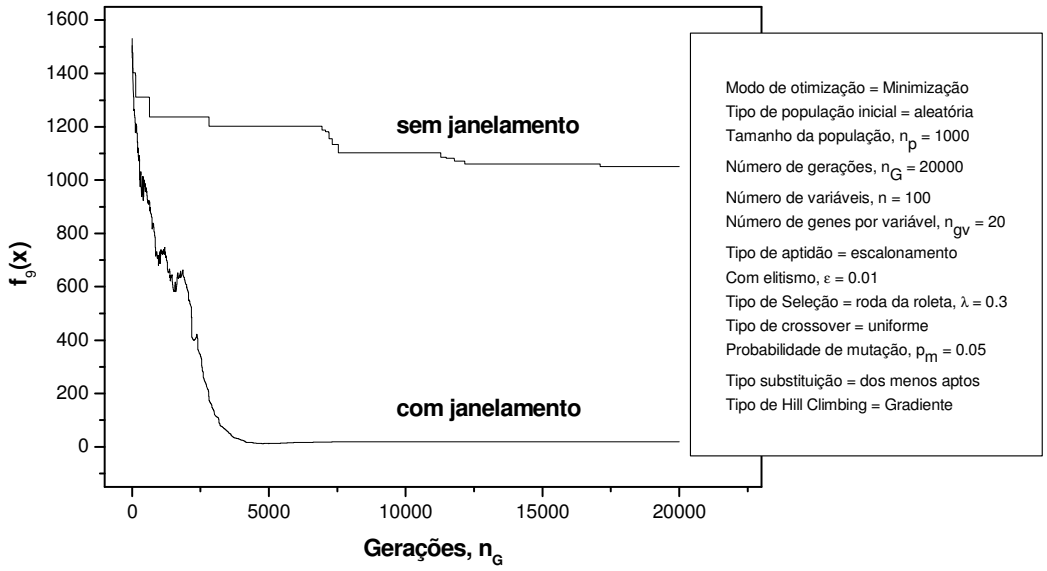


Figura 5.23 - Valores mínimos da função objetivo $f_9(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* de gradiente e janelamento.

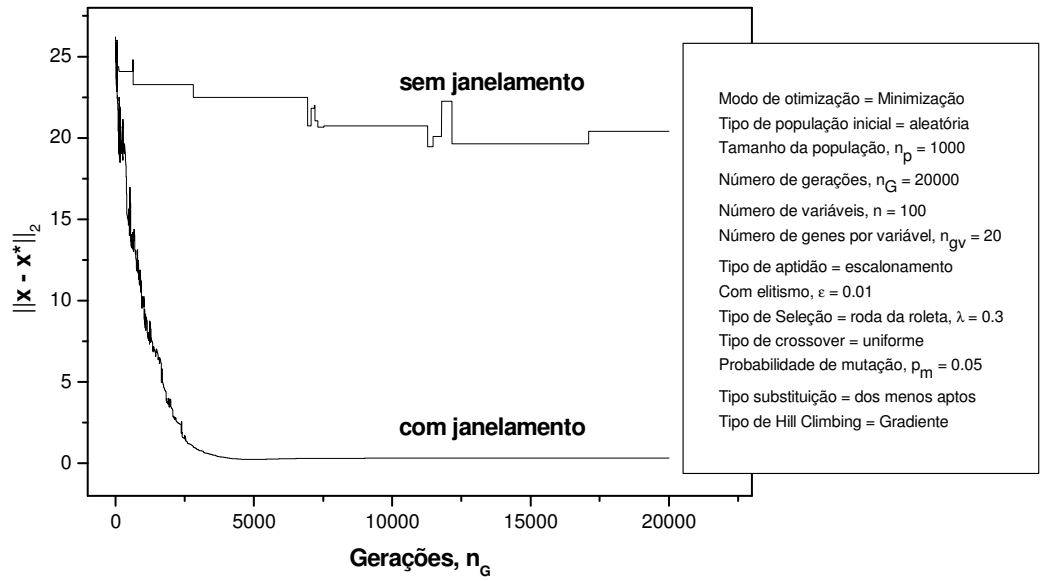


Figura 5.24 - Valores mínimos da distância do indivíduo mais apto até a solução da função $f_9(\mathbf{x})$, para 100 variáveis, com o uso de *hill-climbing* de gradiente e janelamento.

5.3 *Hill-climbing* combinatório

O *hill-climbing* combinatório utilizado neste trabalho sobre TSP consiste da simulação da troca de dois alelos subseqüentes de um dado indivíduo. Se o virtual novo indivíduo for mais apto do que o original a troca final é realizada, caso contrário a troca é descontinuada.

Repete-se este processo dos dois primeiros alelos até os dois últimos, em todas os indivíduos da população e para todas as gerações. Adicionalmente, ao final do cromossomo analisa-se uma troca extra que corresponde à troca do primeiro alelo pelo último. É como se estivesse efetuando as trocas em um cromossomo circular. Este processo de troca é sempre realizado da esquerda para a direita do cromossomo.

5.4 *Hill-climbing* combinatório de 30 cidades

No capítulo 4 o algoritmo genético para o TSP de 30 cidades não havia resolvido de forma exata a minimização da função custo. No entanto havia produzido resultados aproximados bastante adequados, em um tempo factível e a um custo computacional pequeno.

Aquele TSP de 30 cidades é então revisitado utilizando-se o *hill-climbing* combinatório. Desta vez o algoritmo genético conseguiu diminuir ainda mais os valores da função objeto, Figura 5.25, e o algoritmo genético que já apresentava uma solução com apenas um trecho errado, agora conseguiu obter a solução exata. O indivíduo mais apto que corresponde à seqüência de cidades que produz o menor valor para a função custo foi então obtido. A solução pode ser vista na Figura 5.26.

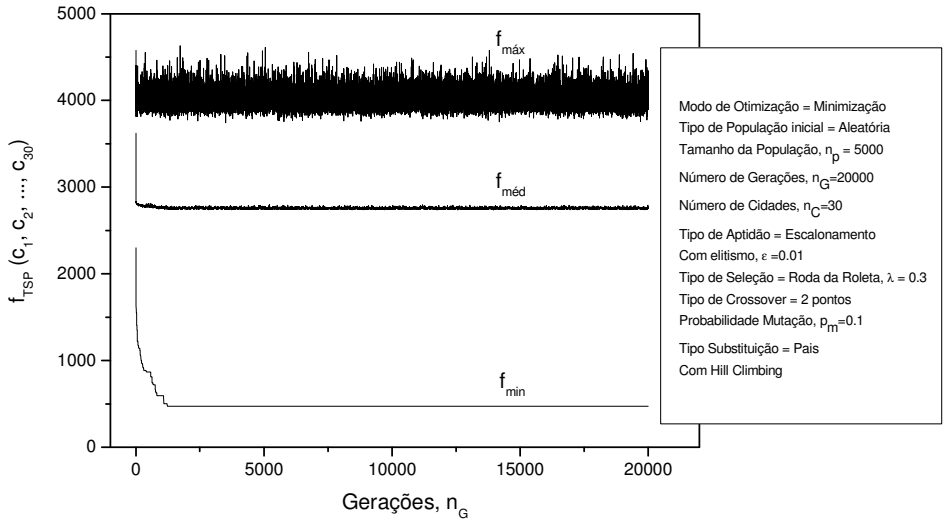


Figura 5.25 - Valores mínimos, médios e máximos da função objetivo para o TSP de 30 cidades com *hill-climbing* combinatório.

Aplicou-se também o *hill-climbing* combinatório na solução do TSP de 40 cidades. Neste caso não conseguiu chegar à solução exata, mas houve melhoria na solução quando

comparada com a apresentada no capítulo 4. Nota-se na Figura 5.27 que o algoritmo genético foi diminuindo o valor da função objetivo até atingir um patamar e não mais se alterou. Neste caso específico, Figura 5.28, uma dificuldade que surge é o fato de que todas as cidades, exceto duas, estão a uma distância constante da cidade mais próxima, gerando uma dificuldade de discernimento para o algoritmo genético. De qualquer forma existe apenas um erro na solução apresentada, falta pouco para a solução exata.

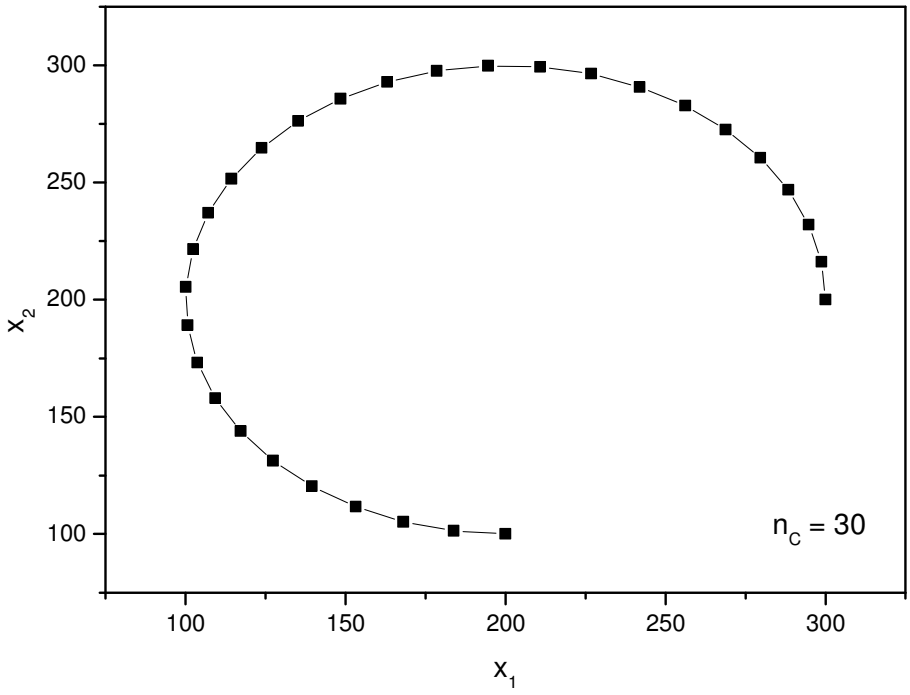


Figura 5.26 - Indivíduo mais apto para o TSP de 30 cidades com *hill-climbing* combinatório.

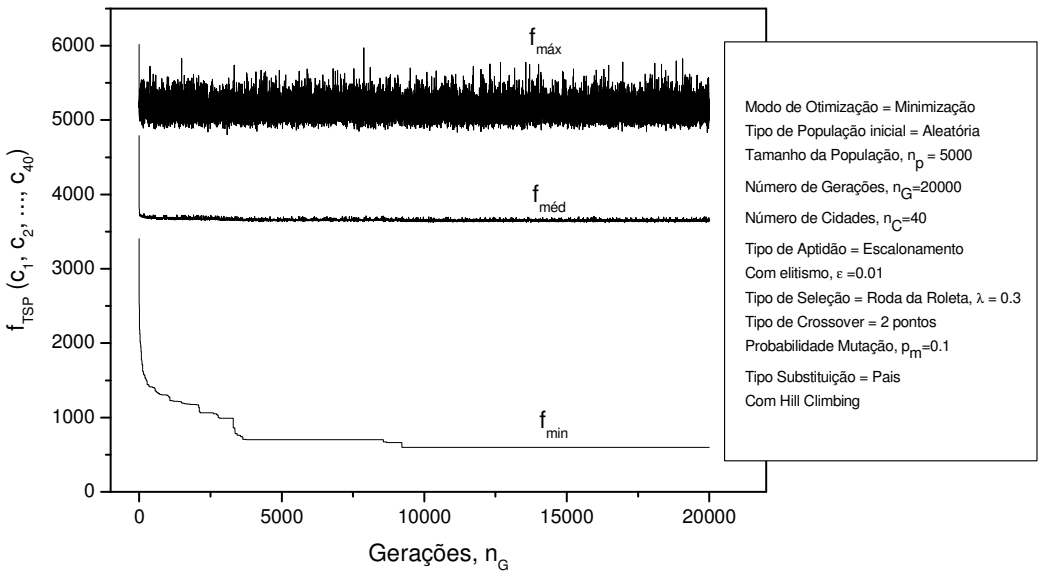


Figura 5.27 - Valores mínimos, médios e máximos da função objetivo para o TSP de 40 cidades com *hill-climbing* combinatório.

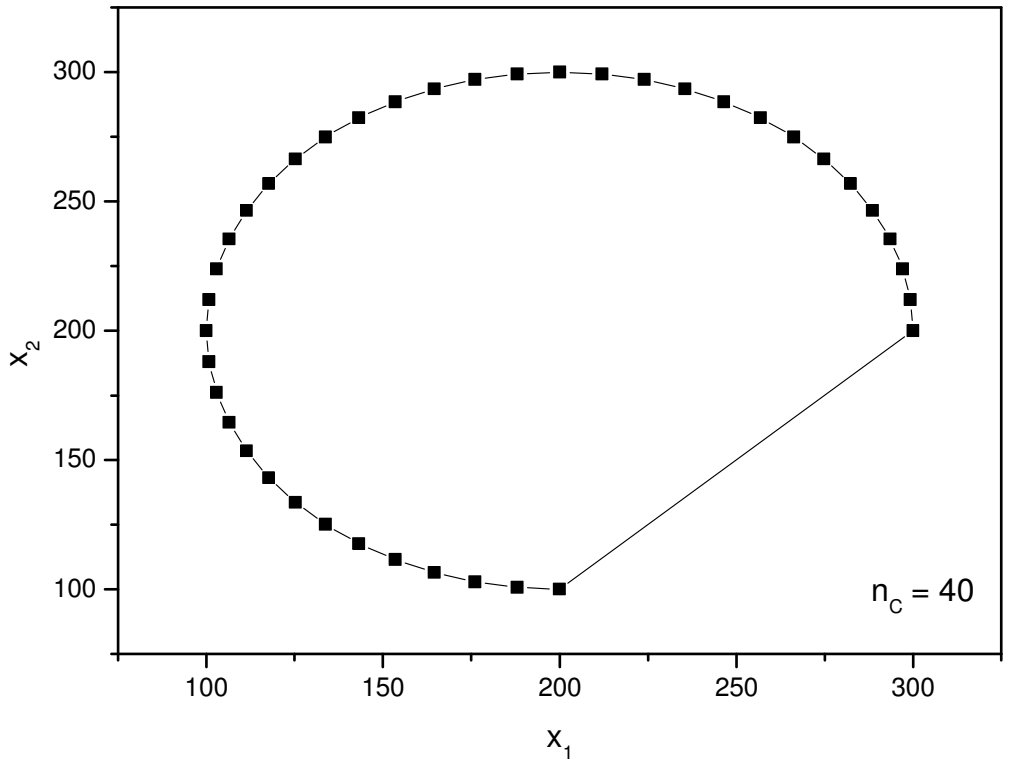


Figura 5.28 - Indivíduo mais apto para o TSP de 40 cidades com *hill-climbing* combinatório.

6. ALGORITMOS GENÉTICOS E PROCESSAMENTO PARALELO

6.1 – O Processo Reprodutivo

Holland (1962a) foi o primeiro a imaginar o processo de tornar paralelo um Algoritmo Genético, reconhecendo de forma indireta a eficiência do processamento paralelo para esse tipo de algoritmo. Ainda indo mais longe, discutiu sobre o mapeamento de planos reprodutivos para um tipo de computação chamada de circuito iterativo (Tomasz, 2006), que não será tratada nesse trabalho por não fazer parte do escopo do mesmo.

Existiram outros pesquisadores que logo no início da pesquisa com algoritmos genético atentaram para o fato de que o paralelismo combinaria de forma ideal com o que se espera de um algoritmo genético. Sivanandam e Deepae (2007) exibiram algumas possibilidades a respeito de algoritmos genéticos em processamento distribuído ou paralelo. Alguns protótipos de algoritmos genéticos foram nomeados por Haupt (2004), através de exames e implementações. São eles:

- Sincronizado *master-slave*
- Semi-sincronizado *master-slave*
- Distribuído, não sincronizado concorrente
- Rede (*Network*)

6.2 – Sincronizado master-slave

Nesse protótipo da Figura 6.1, chamado de *master-slave*, um processo principal denominado *master* é responsável pela coordenação e gerenciamento dos outros processos, chamados de escravos. O processo principal é quem controla a seleção. todo o processo e performance das operações genéticas que venham a ser realizadas. Os outros processos, os escravos, realizam a avaliação das funções em questão. A implementação desse tipo de paralelismo é relativamente simples, porém precisam ser levantados nesse momento os pontos que tornam esse processo problemático quando tratado em termos da execução do mesmo.

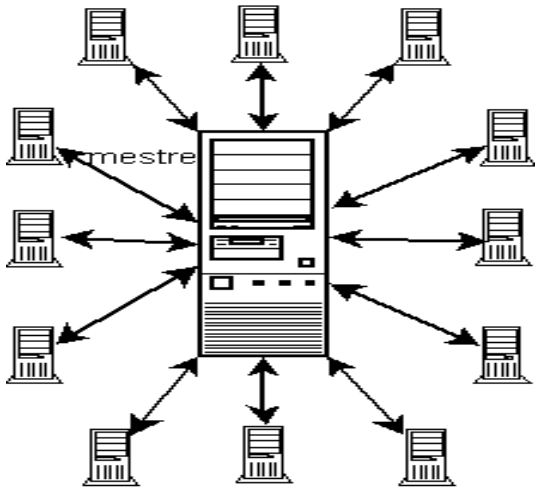


Figura 6.1 – Comunicação de nós clientes com o nó servidor (mestre).

Se existe muita variação na avaliação por parte dos nós escravos, a quantidade de tempo cresce exponencialmente, tornando o processo lento. Além disso, o algoritmo não pode ser avaliado como confiável, pois o processo principal, ou *master*, é vital para o funcionamento do algoritmo genético nesse tipo de processamento. Vindo o mesmo a falhar, todo o sistema irá, indiretamente, falhar.

6.3 – Semi-sincronizado master-slave

Como já foi mencionado no item anterior, o problema de tempo gasto entre a avaliação da função e o retorno do resultado é crucial e determinante para o fator tempo.
Nessa implementação da Figura 6.2, as operações sincronizadas são substituídas, em partes, pela seleção de alguns nós que complementam o trabalho, ou seja, a questão da dependência do sincronismo diminui de forma interessante.

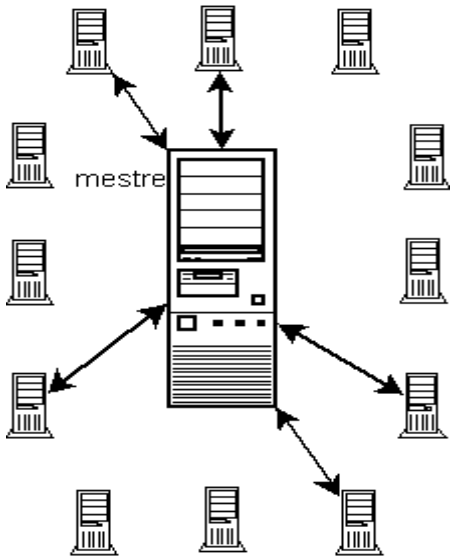


Figura 6.2 – Comunicação de alguns nós clientes com o nó servidor (mestre).

Mesmo com esse tratamento melhor das tarefas a serem realizadas pelos nós, ainda existe a necessidade da centralização dos processos por um um processo principal, ou *master*, tornado-se o ponto mais problemático dessa implementação.

6.4 – Distribuído, não sincronizado concorrente

Na implementação desse ítem, como pode ser visto na Figura 6.3, existem processos idênticos em cada um dos nós realizando as mesmas operações genéticas e avaliando funções, sempre executando de maneira individual, porém fazendo uso de uma mesma memória em comum que é compartilhada em favor de todos os nós do cluster responsáveis por tal processamento. O acesso a esse tipo de memória deve ser feito de forma a evitar acessos simultâneos a locais idênticos de memória, para que não ocorram problemas com o acesso.

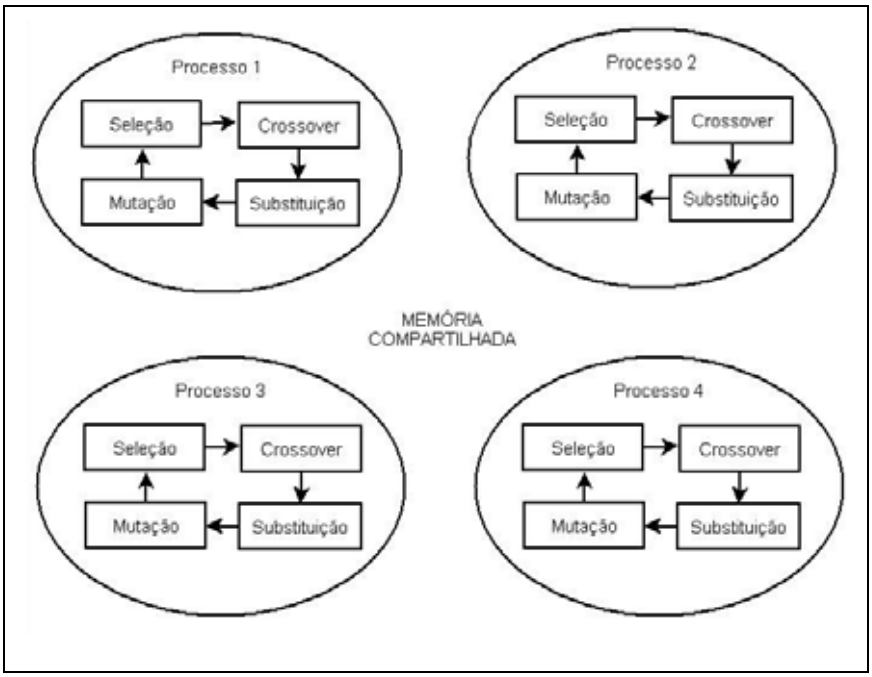


Figura 6.3 - Processos fazendo uso de memória compartilhada.

Esse procedimento é um pouco mais complexo de ser implementado do que os outros procedimentos cidados nos dois últimos itens, mas existe uma credibilidade maior no sistema como um todo, pois o processamento concorrente é independente de outros resultados, tendo como únicda dependência a memória compartilhada. Os nós geralmente estão realizando algum processamento útil, graças a independência entre eles.

6.5 – Rede (*Network*)

Em se tratando da implementação em Rede existem algoritmos genéticos independentes rodando em cada um dos nós da rede, tal qual o procedimento da Figura 6.4, as operações genéticas e avaliações de funções são independentes.

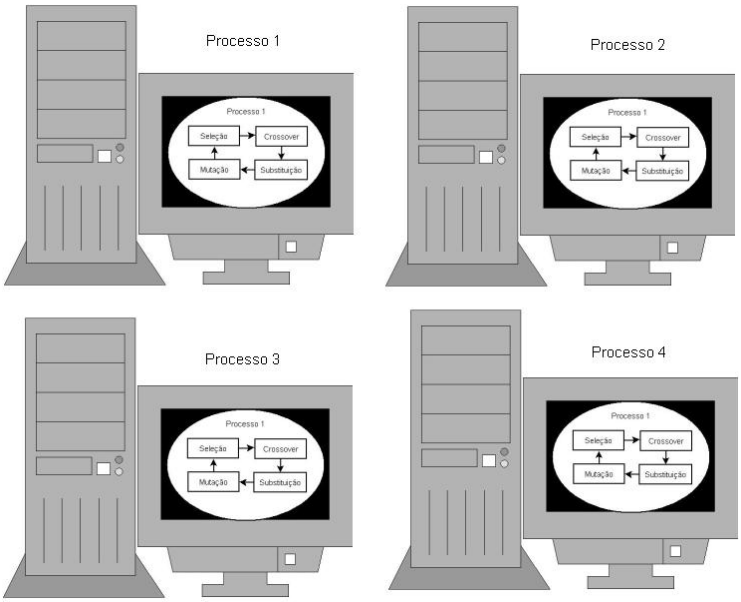


Figura 6.4 – Memórias e Algoritmos Genéticos independentes.

Os nós e seus respectivos processos trabalham localmente, porém a cada geração, os melhores indivíduos são enviados as outras sub-populações da estrutura do cluster ou algo que o valha.

Nesse tipo de implementação a taxa de transferência de dados deve ser alta, tendo em vista que constantemente bons resultados de nós precisam ser enviados a outros nós (Goldbeg, 1989).

6.6 – Paradigma do Arquipélago com Proximidade Geográfica

O paradigma do arquipélago utilizando servidor consiste em uma implementação do protótipo de Rede. A estrutura física e lógica representa um arquipélago. Cada ilha tem sua própria população e seus próprios indivíduos.

A medida que a população de uma ilha se desenvolve, ela envia seu melhor indivíduo para a ilha geograficamente mais próxima e esta ilha realiza procedimento semelhante com a ilha que lhe enviou o individuo. Dessa maneira, ambos passam a ter os melhores indivíduos de cada uma das ilhas, como pode ser visto na Figura 6.5.

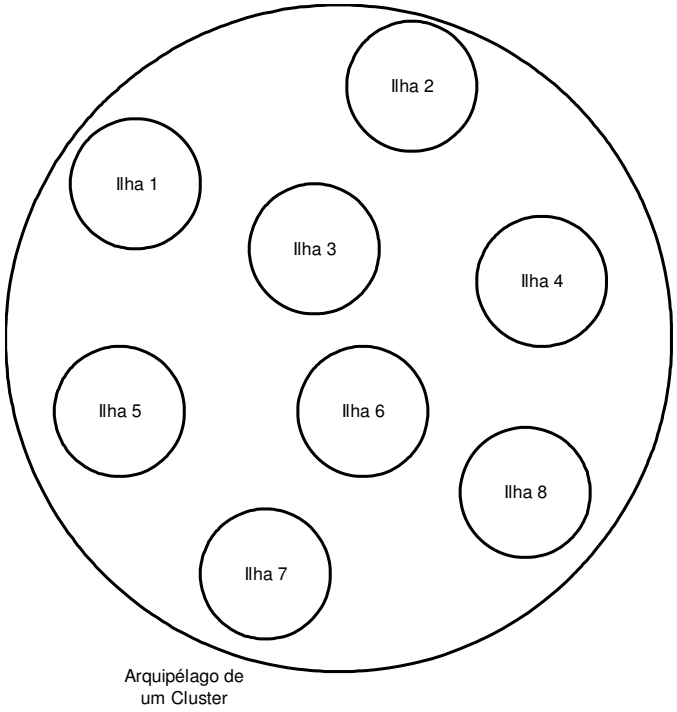


Figura 6.5 – Figura representando o Paradigma do Arquipélago de um Cluster.

Ao final de todas as gerações, os melhores indivíduos do arquipélago são exibidos e o melhor deles é selecionado, ou ainda, após um atingir um dado resultado esperado o algoritmo finaliza.

6.7 – Paradigma do Arquipélago utilizando *Token Ring*

Baseado no conceito descrito anteriormente, existe uma implementação que muda a maneira como os nós se comunicam entre si, baseado em um conceito de uma topologia de rede chamada Anel que utiliza o protocolo de rede *token ring*.

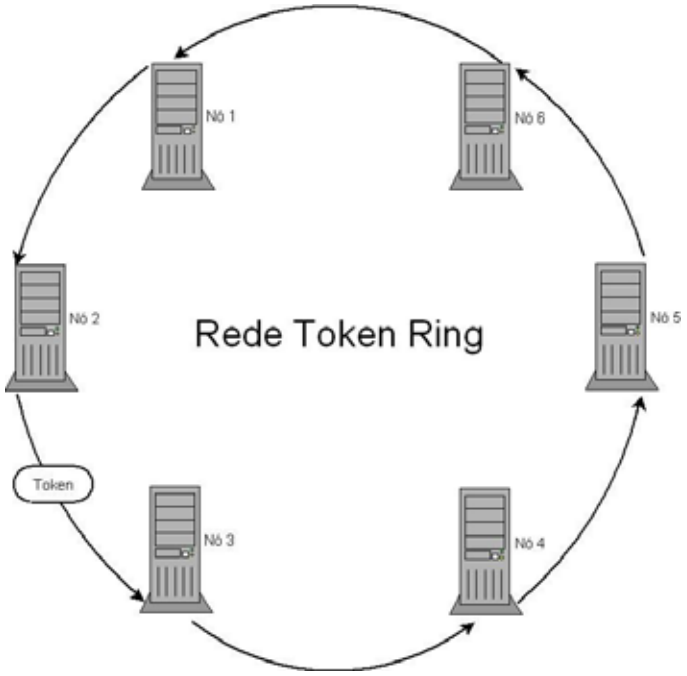


Figura 6.6 – Exemplo de uma rede Token Ring

O paradigma do arquipélago com *token ring* utiliza o conceito modificado de *token* (testemunho) que circula numa topologia de rede do tipo anel, em que cada estação deve aguardar a recepção para transmitir. Quem detêm o *token* é quem transmite em um dado tempo, como pode ser visto na Figura 6.6.

A modificação ocorre no sentido de que a implementação paralela baseada nesse conceito, dispensa a confirmação de recebimento para que a próxima estação ou nó passe a transmitir, dessa forma, a comunicação ocorre de forma não bloqueante, ou seja, um nó desempenha seu trabalho independente de aguardar pelo *token* ou algo que o valha.

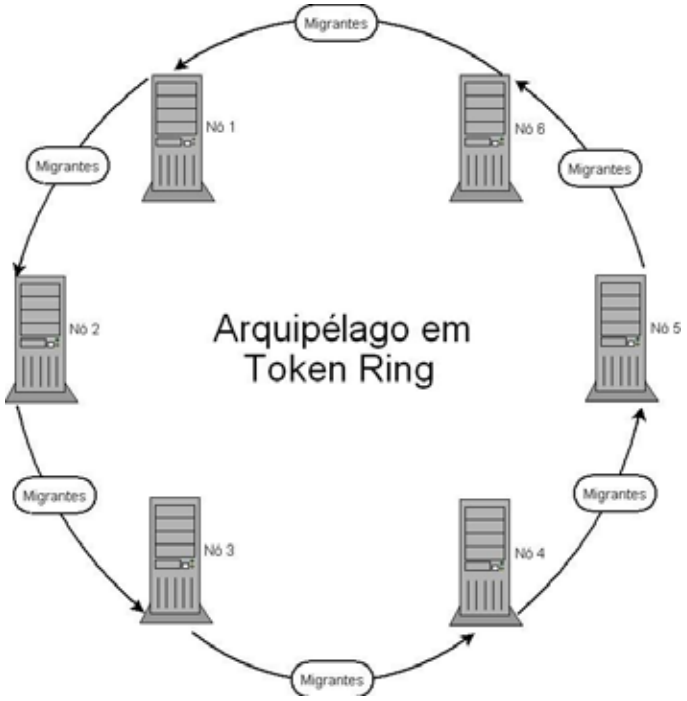


Figura 6.7 – Exemplo do Paradigma do Arquipélago em uma rede Token Ring

Note-se que os indivíduos que migram, referenciados como Migrantes na Figura 6.7, não dependem de nenhum *token* para dar andamento às suas respectivas migrações, tendo em vista que isso impediria um funcionamento não sincronizado e essencial para a execução do algoritmo genético no modo aqui empregado.

Se a comunicação ocorresse de forma bloqueante utilizando um token, por exemplo, os nós estariam impedidos de buscarem a melhor solução independentemente da chegada de indivíduos de outras ilhas, o que acarretaria um bloqueio dos outros nós, saindo, em partes, do conceito de Rede explicado no item 6.5 desse capítulo.

6.8 – O Rank como identificador da Ilha

Cada nó possui uma identificação para o programa e para a biblioteca de programação paralela, nesse caso a MPI. Essa identificação recebe o nome de *rank* do processo, ou simplesmente *rank*.

Para entender o processo é necessário que se pense de forma paralela, ou seja, escreva-se os códigos como se o mesmo fosse rodar em máquinas distintas, o que geralmente ocorre.

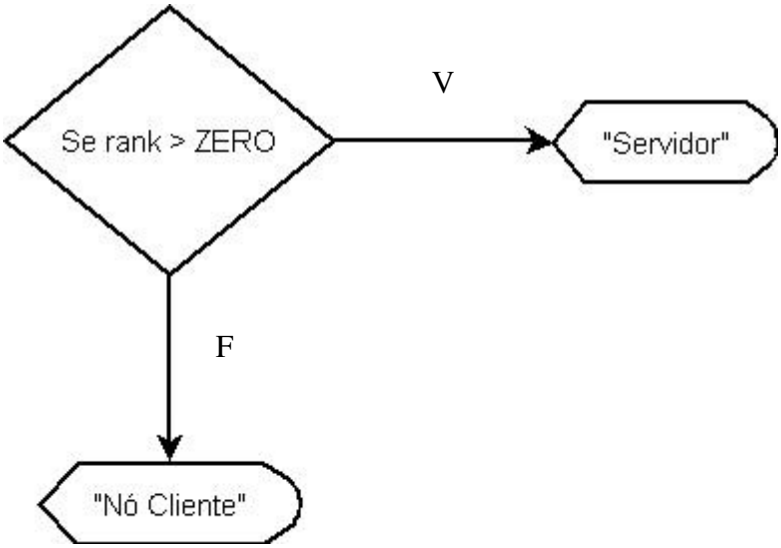


Figura 6.8 – Estrutura de decisão em um ambiente paralelo

Por exemplo, observando o fluxograma da Figura 6.8, nota-se que uma estrutura de decisão muda o fluxo do programa baseado no *rank* do processo.

Percebe-se claramente que o fluxo irá para lugares distintos caso a variável de controle seja o servidor do cluster ou simplesmente mais um nó. Todo o código deve ser escrito de maneira a perceber claramente que "o mesmo" está sendo executado em outras máquinas simultaneamente.

6.9 – A clonagem de indivíduos para migração

Quando ocorre o processo migratório, esperado no paradigma da ilha baseado em *token ring*, supõem-se que os melhores indivíduos de cada nó, ou ilha, são enviados ao nó numericamente mais próximo.

Porém, pode-se ter a falsa impressão de que esses indivíduos estão sendo perdidos a cada migração por geração. Na verdade, ocorre que uma cópia, ou clone, desses melhores indivíduos é feita, e então enviada para o nó em questão.

Dessa forma evita-se que as boas soluções do problema sejam perdidas ao longo dos processos migratórios que por ventura venham a ocorrer na execução do algoritmo genético.

6.10 – Implementação Paralela

Baseado no protótipo de Rede e no algoritmo genético local descrito anteriormente foi criado um código no qual o processamento ocorre de forma distribuída.

Os dados principais são informados na máquina servidora, que por sua vez distribui os dados para todos os nós da rede (Talbi et al., 1995). Esses dados são: o tamanho da população, a quantidade de gerações, o tamanho do cromossomo, a taxa de *crossover,* a taxa de mutação, a existência ou não de elitismo, a taxa de elitismo, o tipo de seleção, o tipo de substituição e o tipo de aptidão.

Observe-se o seguinte fragmento de código em que ocorre o envio dos melhores indivíduos:

```
for (k=ZERO; k<qtmigra; k++)
{
  if (myid == (numprocs-UM))
    destino = UM;
  else
    destino = myid + UM;
  for (i=ZERO; i<tam_cromo; i++)
    ind[i] = p1[selmigra[k]].cromoss[i];
  MPI_Send(ind,tam_cromo,MPI_CHAR,destino,tagenv,MPI_COMM_WORLD);
  tagenv++;
}
```

Figura 6.9 - Código fonte da migração de indivíduos

Em primeiro lugar pode ser observado que de acordo com o valor da variável *myid*, se o seu valor for igual ao correspondente ao último nó do *cluster*, então o envio ocorre para o primeiro nó, fechando assim o anel proposto pelo conceito de *token ring*.

Caso não seja ainda o último nó do *cluster*, então o envio ocorre para o próximo nó, numericamente mais próximo.

O comando utilizado para o envio é o *MPI_Send*, que corresponde a um comando não bloqueante, quer dizer, a execução do código continua independentemente do destino ter ou não recebido aquele “pacote”.

Na seqüência pode ser observado o fragmento de código correspondente ao recebimento de indivíduos migrantes de outros nós do cluster.

```
tagrec = ((myid - UM) * qtmigra)+UM;
for (k=ZERO; k<qtmigra; k++)
{
  MPI_Irecv(ind2,tam_cromo,MPI_CHAR,MPI_ANY_SOURCE,MPI_ANY_TAG,
      MPI_COMM_WORLD,&request);
  MPI_Test(&request,flag,MPI_STATUS_IGNORE);
  for (i=ZERO; i<tam_cromo; i++)
    pbuff[qtsel+k].cromoss[i] = ind2[i];
  tagrec++;
}
```

Figura 6.10 - Código fonte da recepção de indivíduos

Nesse código utilizou-se um tipo de recebimento de mensagem que é chamado de não bloqueante, o *MPI_Irecv*. Quando utilizado, tal qual o *MPI_Send*, permite que o código na seqüência seja executado, e não fique bloqueado aguardando por algo que pode demorar ou simplesmente não ocorrer, que seria o caso do uso de *MP_Recv*.

Existe, porém, a necessidade do uso de *MPI_Test*, comando que fornece uma finalização para o *MPI_Irecv*, mesmo que a finalização seja um dado que não chegou. É indispensável o seu uso, ou o uso de um comando equivalente, ou seja, que finalize o processo de *MPI_Irecv*.

Da maneira como está estruturado o código, pode ser que em um dado momento algum indivíduos migrantes se percam em uma migração, e não cheguem aos seus destinos, porém em uma próxima execução desse trecho, ou seja, em uma próxima migração outros indivíduos chegarão.

Deve-se ficar atento ao fato de que os indivíduos migrantes, são clones dos melhores de cada nó, ou seja, os nós que enviaram seus melhores indivíduos, não os perderam de sua população.

6.11 – Testes

Vários testes foram feitos para verificar a capacidade de processamento do algoritmo genético paralelo, fazendo uso de suas respectivas características, mudando as variáveis tamanho da população, quantidade de gerações e tamanho do cromossomo. O arquipélago

dos testes foi composto por 5 ilhas, sendo uma delas para simplesmente exibir os melhores resultados.

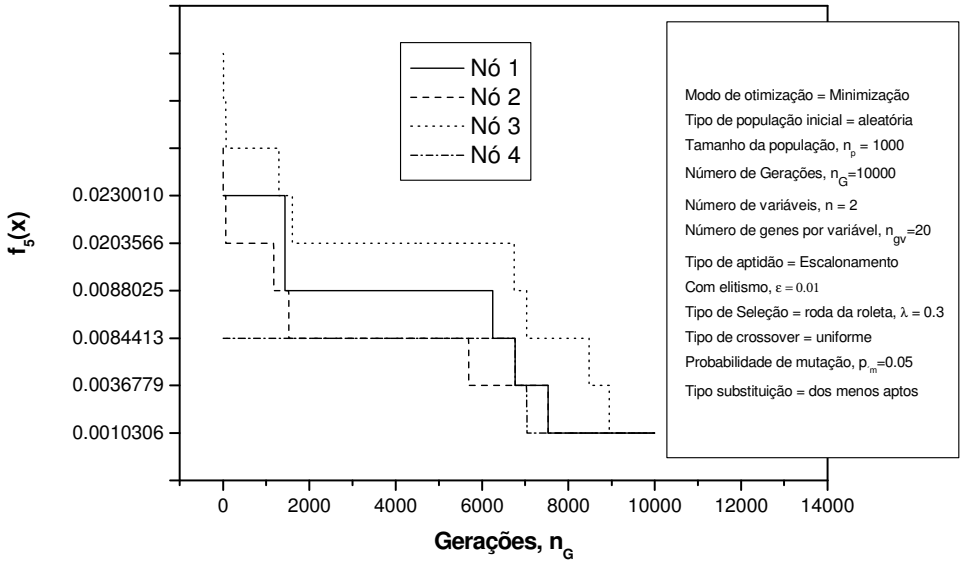


Figura 6.11 – Melhores indivíduos por nó - Função $f_5(\boldsymbol{x})$ com 2 variáveis.

A Figura 6.11 mostra a evolução do indivíduo mais apto da população de cada ilha ou nó do cluster. Com duas variáveis, foi executada a função $f_5(\boldsymbol{x})$ com os parâmetros padrões adotados no capítulo 3.

Devido a alguma deficiência na implementação computacional do algoritmo em paralelo a taxa de convergência piorou quando comparada com aquela apresentada no processamento seriado, embora o processo funcionasse. A melhoria nesta taxa de convergência deficiente não foi possível ao tempo da escrita desse documento.

Além disso, quando um indivíduo migrante chega até o nó de destino, caso o mesmo fosse do tipo elitista, essa propriedade não seria mantida a principio, ou seja, seu cromossomo pode ser modificado ao longo dos processos genéticos, e ocasionalmente ele pode ser avaliado como um dos melhores desse novo nó em que ele passa a habitar e então ser marcado novamente como elitista, caso esse parâmetro esteja ativado no algoritmo.

Na próxima figura, o número de variáveis foi elevado para 10 e o problema foi resolvido novamente.

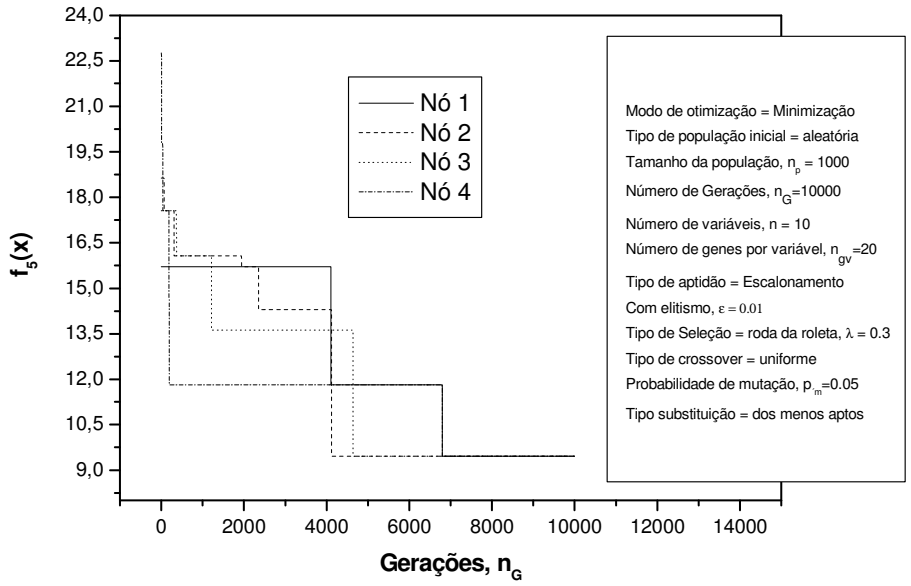


Figura 6.12 – Melhores indivíduos por nó - Função $f_5(\boldsymbol{x})$ com 10 variáveis.

Na Figura 6.12 pode-se observar que o mesmo problema com 10 variáveis, dificulta a solução, porém deve ser observado que o nós enviam seus melhores indivíduos e quando a geração 7000 é alcançada, os valores obtidos são praticamente iguais em cada nó, em conseqüência do processo migratório.

Observando a Figura 6.13 tem-se um efeito claro obtido pelo paralelismo implementado para a solução de um problema através de um algoritmo genético. A cada geração, os indivíduos migrantes vão chegando aos nós destinos e vão fazendo com que cada nó evolua individualmente com seu processamento, porém fazendo uso de bons indivíduos migrantes que chegam da ilha ou nó mais próximo numericamente.

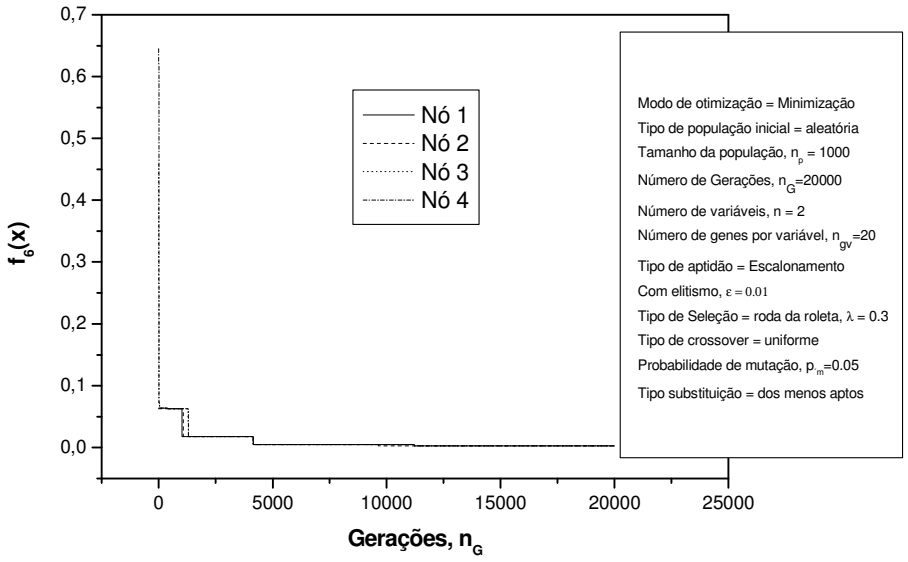


Figura 6.13 – Melhores indivíduos por nó - Função $f_6(\boldsymbol{x})$ com 2 variáveis

O mesmo problema da figura anterior à Figura 6.14 agora é abordado com 10 variáveis. O resultado final obtido está longe de solução esperada, como pode ser visto no capítulo 3, porém tem-se o mesmo efeito exibido na Figura 6.11, ou seja, os nós progressivamente em sua evolução enviam seus melhores indivíduos, fazendo com que cada nó caminhe de maneira similar à sua.

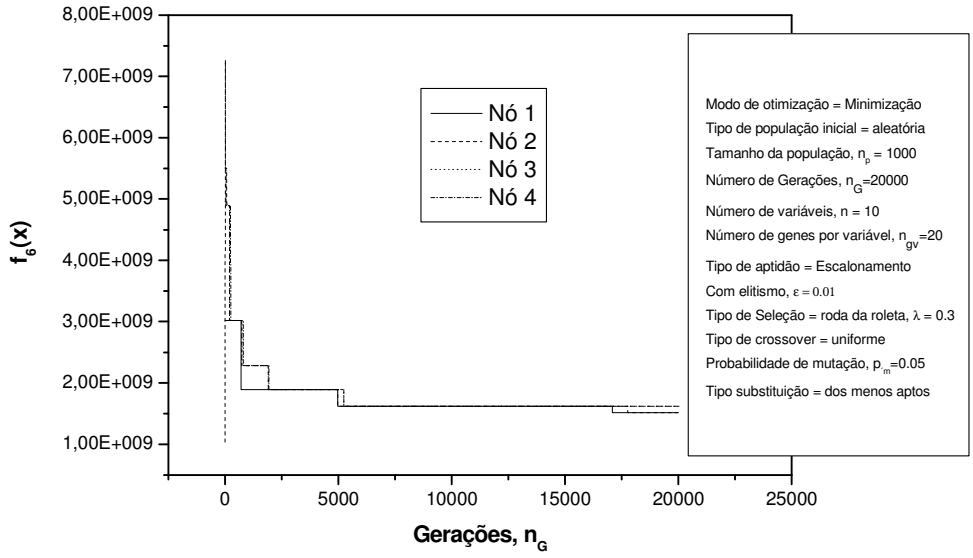


Figura 6.14 – Melhores indivíduos por nó - Função $f_6(\boldsymbol{x})$ com 10 variáveis.

Nesse exemplo da Figura 6.15 pode ser visto que ocorre a super convergência do algoritmo genético paralelo em questão, ou seja, o resultado esperando é obtido rapidamente e no caso do paralelismo, esse resultado é enviado de uma dada maneira a todos os outros nós, fazendo com que todos os outros alcancem a super convergência.

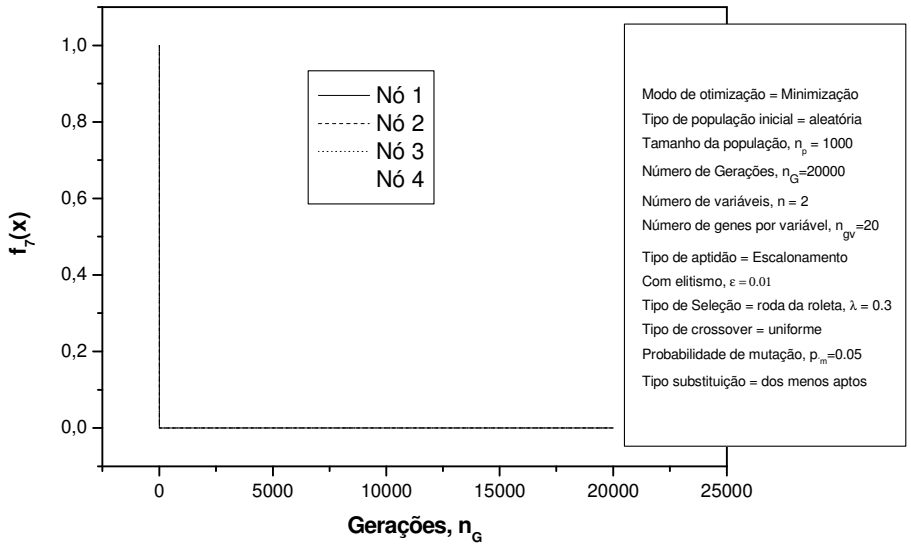


Figura 6.15 - Melhores indivíduos por nó - Função $f_7(\boldsymbol{x})$ com 2 variáveis.

Tal qual o ocorrido em outros exemplos desse mesmo capítulo, a Figura 6.16 exibe com clareza a migração dos bons indivíduos de cada um dos nós para os outros nós do cluster, mesmo não obtendo o resultado esperado, todos os nós caminharam de uma maneira muito similar em seu processo evolutivo.

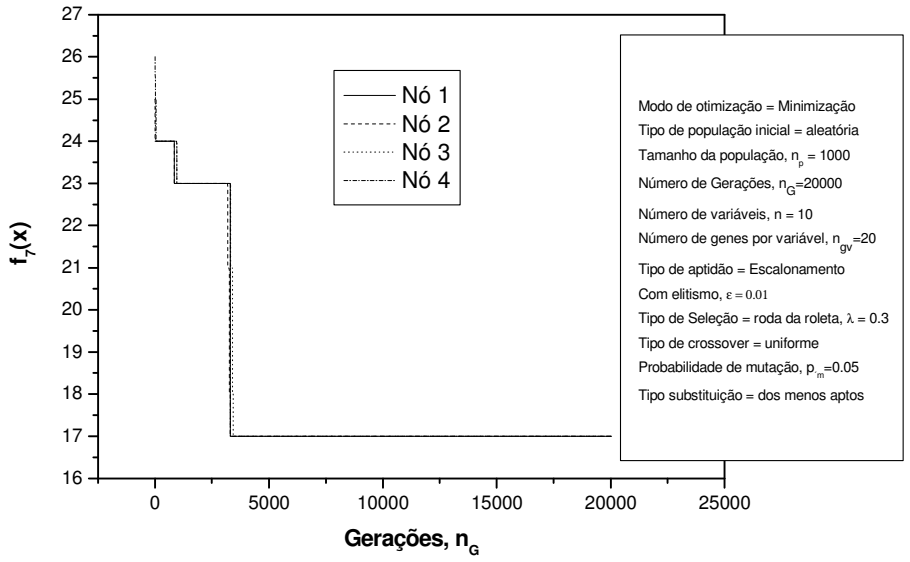


Figura 6.16 - Melhores indivíduos por nó - Função $f_7(\boldsymbol{x})$ com 10 variáveis

Na função $f_9(\boldsymbol{x})$ representada na Figura 6.17 o resultado obtido foi bem próximo ao esperado e nota-se claramente que os nós andaram de maneira relativamente uniforme na busca pelo melhor resultado.

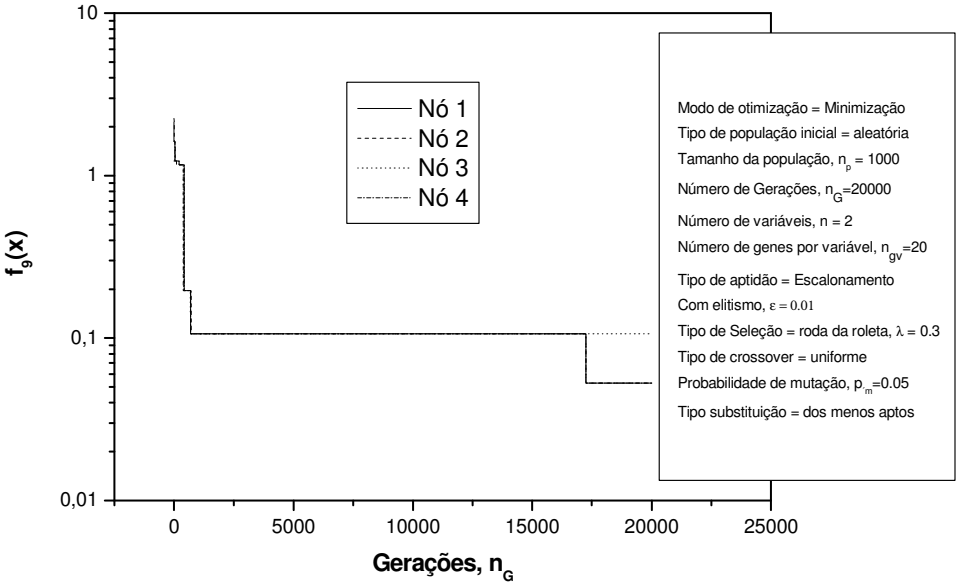


Figura 6.17 – Melhores indivíduos por nó - Função $f_9(\boldsymbol{x})$ com 2 variáveis.

Nesse processamento houve um sincronismo indireto muito grande no envio e recebimento dos melhores resultados de cada nó, a não ser no final em que os resultados variaram no nó número 2.

Mais uma vez, como pode ser visto na Figura 6.18, os melhores valores foram passados de nó para nó, porém não foi obtido o resultado ótimo esperado, tendo em vista que a complexidade do programa cresce na medida em que o número de variáveis também aumenta.

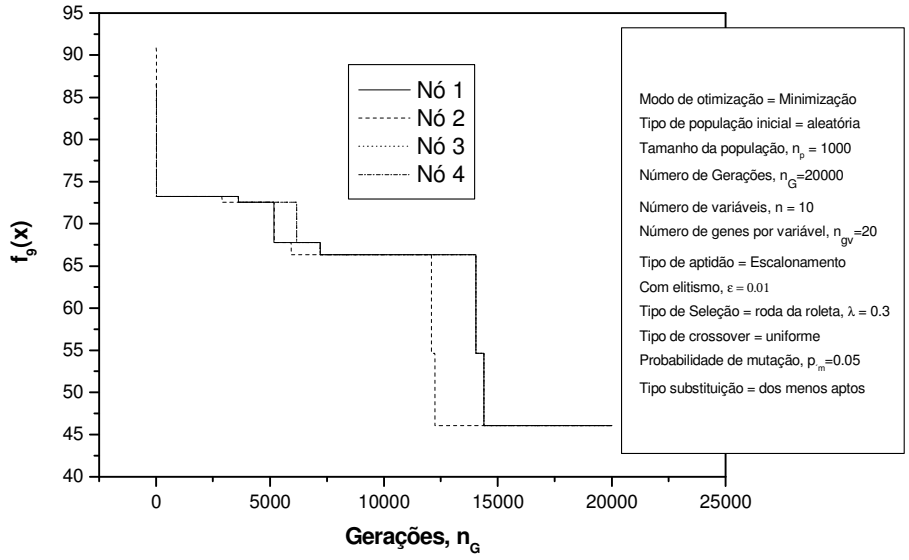


Figura 6.18 – Melhores indivíduos por nó - Função $f_9(\boldsymbol{x})$ com 10 variáveis.

6.12 – Problema do Caixeiro Viajante em Paralelo

Como visto no capítulo 4, o problema do caixeiro viajante consiste em buscar o melhor caminho a ser percorrido, ou o caminho ótimo possível levando em consideração outras variáveis, como tempo de processamento, a tolerância ao erro e a quantidade de cidades.

Nos próximos itens serão tratados os fatores que envolvem o problema do caixeiro viajante no aspecto paralelo ou distribuído, e quais as medidas que devem ser tomadas para a execução do mesmo.

6.13 – Busca pelo melhor caminho

A Figura 6.19 ilustra simbolicamente a situação da busca pelo melhor caminho. As cidades de número 1 e 2 estão mais próximas, e o caminho a ser percorrido deveria passar primeiramente entre as duas e em um próximo passo para a cidade número 3.

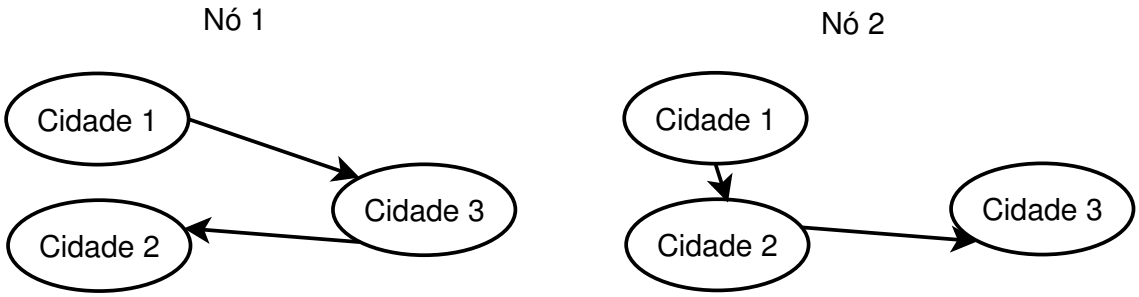


Figura 6.19 - Nós buscando a solução do problema

Porém o nó 1 não consegue resolver o problema e faz com que ocorra uma viagem até a cidade número 3 e depois uma viagem de volta até a cidade de número 2, gerando dois erros no caminho final a ser percorrido.

O caminho deveria ser cidade 1, cidade 2 e cidade 3, ou ainda, cidade 2, cidade 1 e cidade 3, como pode ser visto no nó 2.

6.13 – Enviando o melhor caminho para o caixeiro vizinho

Tendo em vista que se caminhassem de maneira paralela, porém sem comunicação entre si, os algoritmos genéticos para solução do problema do caixeiro viajante não teriam um ótimo funcionamento, ou ainda não teriam o melhor funcionamento possível.

Deve existir um processo migratório em que a melhor solução é enviada para o nó numericamente mais próximo, como pode ser visto na Figura 6.20.

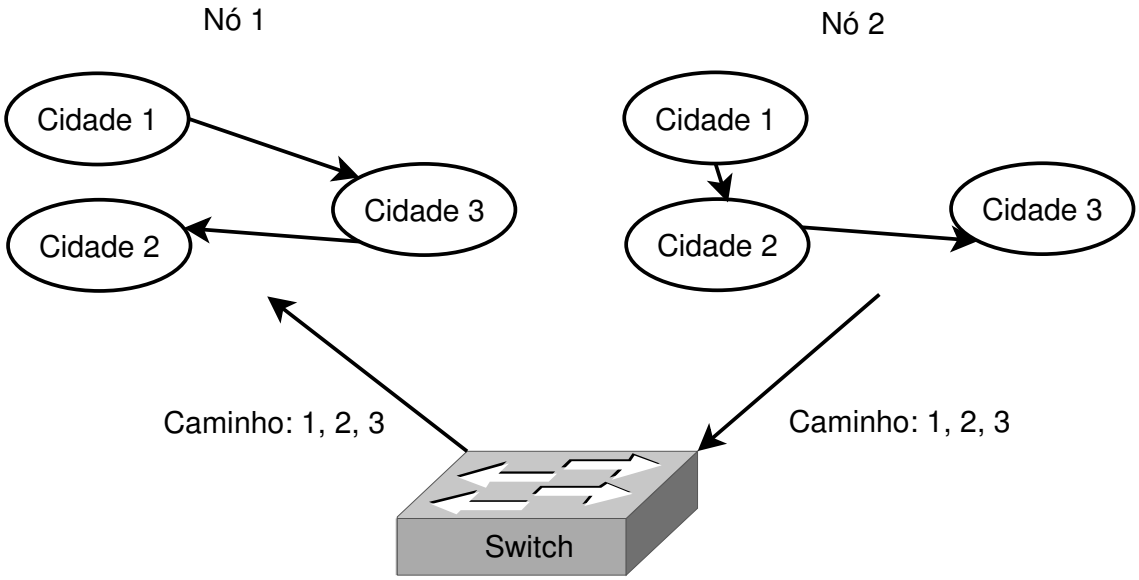


Figura 6.20 – Envio do melhor indivíduo para o nó mais próximo

Note que em uma etapa de migração, esse resultado deve ser enviado para o nó numericamente mais próximo no processo de *token ring*, já comentado nos primeiros itens desse capítulo.

A Figura 6.20 também ilustra o uso de um *switch* como concentrador de dados para o envio de um dos melhores caminhos a serem percorridos.

6.14 – Testes do TSP

Tal qual o algoritmo genético multidimensional paralelo testado anteriormente, o algoritmo desenvolvido para a solução do problema do caixeiro viajante também foi modificado para atuar em um ambiente de programação distribuído.

A Figura 6.21 mostra parte dos resultados para o problema do caixeiro viajante com dez cidades, na qual pode-se observar que cada nó enviou, em um tempo suficientemente sincronizado com seu nó mais próximo, os melhores indivíduos da população.

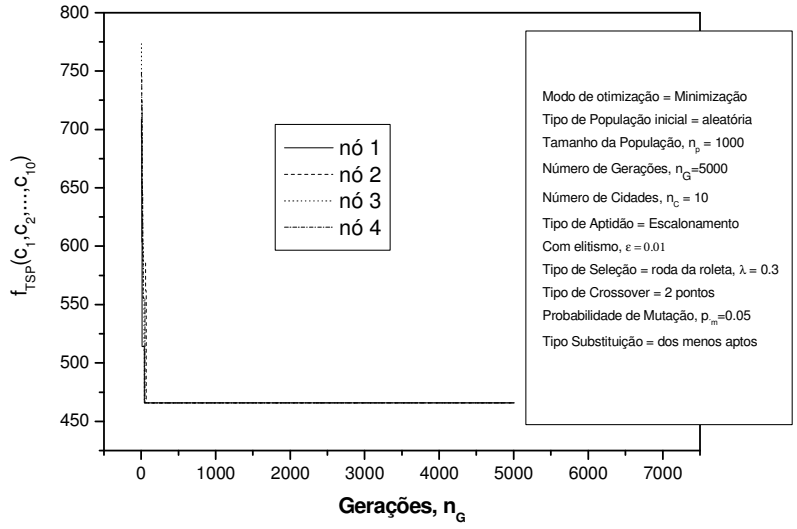


Figura 6.21 – Melhores indivíduos por nó – TSP de 10 cidades.

Na Figura 6.22 pode-se observar que devido a heterogeneidade computacional dos nós do cluster em questão, adicionando-se também o fato de que o enviou do melhor indivíduo não é um processo bloqueante sincronizado, os nós não obtiveram todos o mesmo resultado.

Via de regra, isso quer dizer que quando envia-se um dado não bloqueante em um instante, nem sempre esse dado chegará ao destino instantes depois, pois não existe garantia de que o processo está exatamente no ponto de recepção no nó destino e o dado tende a se perder.

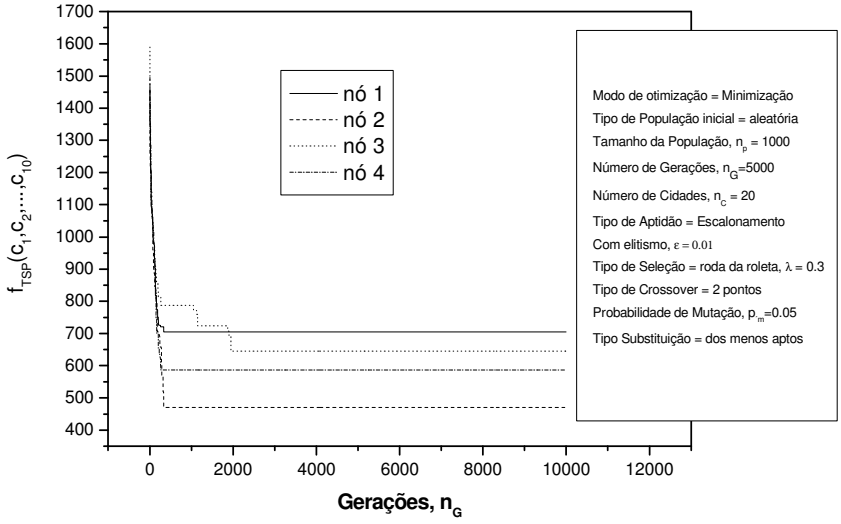


Figura 6.22 – Melhores indivíduos por nó – TSP de 20 cidades.

O próximo resultado é a solução do problema do TSP para 30 cidades, como pode ser visto na Figura 6.23. O resultado obtido não foi tão distante da solução desejada porém, mais uma vez, a informação ótima obtida por um dos nós, não conseguiu chegar aos outros, devido aos aspectos já citados nos comentários da figura anterior.

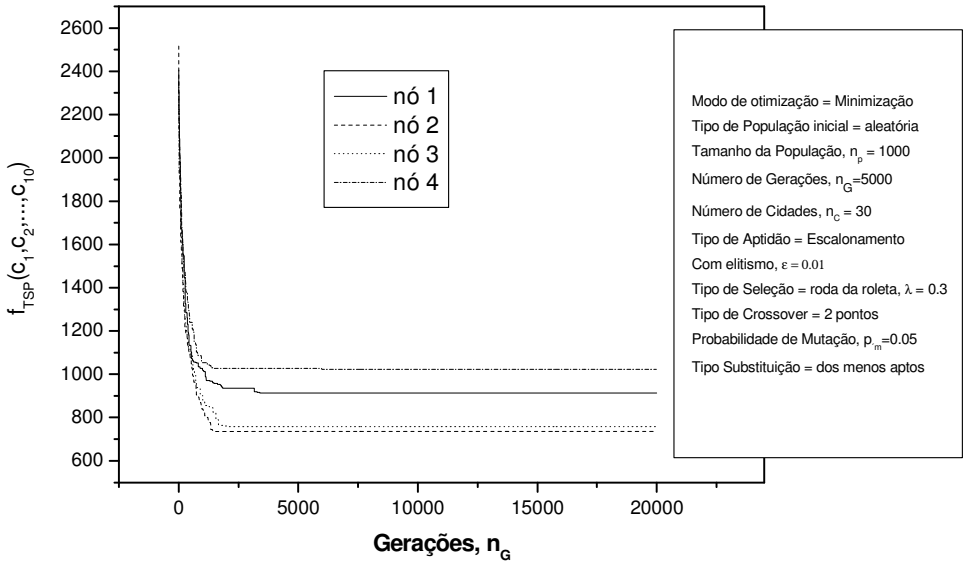


Figura 6.23 – Melhores indivíduos por nó – TSP de 30 cidades

A Figura 6.24 ilustra o melhor indivíduo obtido para o TSP de 30 cidades em paralelo, que obteve resultado idêntico a sua execução de maneira serial. Deve-se relembrar nesse ponto que, se a comunicação nesse processo fosse do tipo bloqueante, não existiria garantia com relação ao tempo de espera, por outro lado teria-se a certeza de que o dado sempre chegaria ao seu destino.

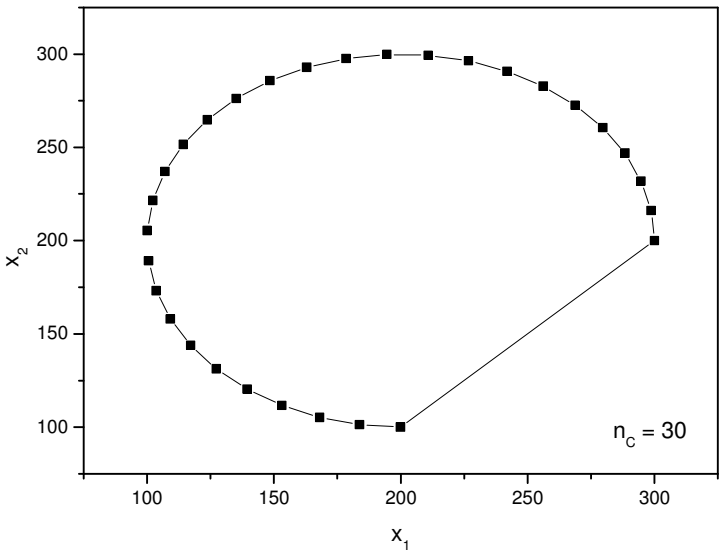


Figura 6.24 - Indivíduo mais apto para o TSP de 30 cidades em paralelo.

O último teste realizado no paradigma paralelo foi o problema do TSP de 40 cidades, ilustrado na Figura 6.25, que em métodos determinísticos tem um custo em termos de tempo muito maior do que a sua codificação em um algoritmo genético.

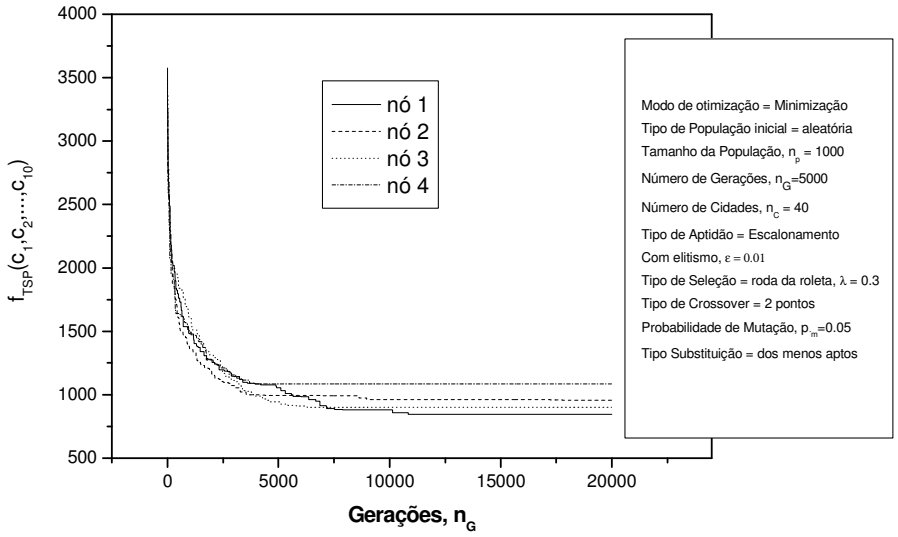


Figura 6.25 – Melhores indivíduos por nó – TSP de 40 cidades.

Os resultados obtidos foram muito bons, como pode ser visto na Figura 6.26, e o tempo de processamento também não foi alto se comparado a outros métodos. Deve-se

também levar em consideração que nesse tipo de codificação, se um nó terminou seu processamento antes dos outros nós, mesmo não obtendo o melhor resultado, seus dados podem ser utilizados antes do processamento paralelo terminar como um todo.

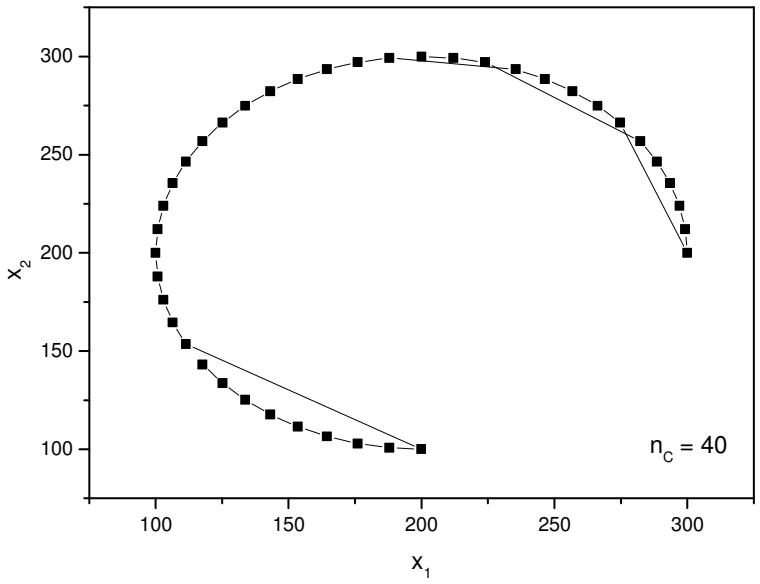


Figura 6.26 - Indivíduo mais apto para o TSP de 40 cidades em paralelo.

A informação do último parágrafo é interessante se for levado em consideração o tempo de processamento diferente para cada nó em um ambiente de programação paralelo não homogêneo, que é o caso deste trabalho.

CONCLUSÃO

Esse trabalho teve como foco a avaliação dos operadores de um algoritmo genético e alguns aspectos do comportamento paralelo no ambiente de computação distribuída de um algoritmo genético.

Pode-se notar e comprovar a robustez do algoritmo genético, tendo em vista que ele foi utilizado ao longo do trabalho para resolver problemas de funções de dimensões diferenciadas, além de ser empregado para funções com comportamento explicitamente distinto.

Por exemplo, a função $f_3(x_1, x_2)$, também chamada de função degrau, tem um comportamento diferente da função f_1, que é quadrática e simétrica. A primeira não admite nenhuma derivada contínua, portanto não pode ser resolvida por nenhum método baseado em gradiente, somente métodos de busca direta, que é o caso do algoritmo genético. Já a segunda é infinitamente derivável e de fácil solução. Fica claro e comprovado no capítulo 3 que ambas funções podem ser resolvidas pelo algoritmo genético, comprovando sua robustez.

Pode-se também observar em vários momentos o comportamento da função $f_9(\mathbf{x})$, conhecida na literatura como **Função de Rastrigin,** para a qual se conseguiu obter bons resultados utilizando o algoritmo genético mesmo quando executada com 10 variáveis, levando em consideração sua característica de possuis vários pontos de mínimos e máximos locais, sendo uma função de difícil solução.

No capítulo 4 o algoritmo genético foi adaptado para a realidade de um problema do caixeiro viajante, também chamado de TSP (*Traveler Salesman Problem*), conhecido por sua alta aplicabilidade e dificuldade em obter uma solução eficiente.

Em teorias de análise combinatória um problema de 20 cidades gastaria aproximadamente 28,2 dias, porém o algoritmo genético implementado nesse trabalho mostrou sua eficiência e conseguiu resolver o problema em um tempo muito inferior a este.

Para problemas com um maior número de cidades, o algoritmo genético não chegou na melhor solução, como por exemplo com 40 cidades, porém obteve um resultado muito próximo ao desejado, mais uma vez em um curto espaço de tempo se comparado com outros métodos.

No capítulo 5, foram implementados métodos que utilizam estratégias de “morro acima”, que se mostraram muito eficientes em sendo utilizadas juntamente com algoritmos genéticos. Conseguiu-se resolver problemas multidimensionais para 100 variáveis, algo complexo, mas que se tornou possível devido a estratégia de janelamento combinada com o *hill-climbing*.

No sexto e último capítulo do presente trabalho, o algoritmo genético foi codificado de maneira distribuída ou paralela, e o processamento dos problemas citados anteriormente nessa conclusão foi realizado por recursos compartilhados de mais de um computador. Os resultados obtidos nesse processo não foram significativamente melhores para justificar o uso de algoritmos genéticos em paralelo, levando em consideração a estrutura heterogênea do ambiente paralelo utilizado ao longo dos testes desse trabalho. Aparentemente, houve algum erro de implementação computacional não resolvido em tempo. Adicionalmente, os nós do cluster utilizados eram muito heterogêneos dificultando seu uso.

REFERÊNCIAS

ALVES, R.A., **Algoritmos Genéticos**. São Paulo: Edições Inteligentes, 2005.

BAGLEY, J.D., **The behavior of adaptive systems which employ genetic and correlation algorithms**. Doctoral Dissertation - University of Michigan, Michigan, 1967.

BARICELLI, N. A., **Symbiogenetic evolution processes realized by artificial methods**. New York: Methods 9, 1957.

BARICELLI, N. A., **Numerical testing of evolution theories**. New York: ACTA Biotheoretica, 1962

CISCO, **CCNA.** Disponível em: <http://www.cisco.com>. Acesso em: 15 dez. 2004.

DARWIN, C., **A Origem das Espécies** (Versão traduzida do original de 1859). São Paulo: Martin Claret, 2006.

DE JONG, K., **An analysis of the behaviour of a class of genetic adaptive systems**. PhD thesis – University of Michigan, Michigan, 1975.

FRASER, A. S., Simulation of genetic systems by automatic digital computers. 5-linkage, dominance and epistasis. In O. Kempthorne (Ed.), **Biometrical genetics** (pp. 70-83). New York: Macmillan. 1960.

FRASER, A. S., Simulation of genetic systems. New York: **J. Theoret. Biol.**. 1962.

GOLDBERG, D. E., **Genetic algorithms in search, optimization, and machine learning**. Washington: Addison-Wesley, 1989.

GOLDBERG, D. E., SASTRY, K., **Genetic Algorithms: The Design of Inovation.** Washington: Springer, 2007.

GRIFFIHS, A.J.F; MILLER, J.H.; SUZUKI, D.T.; LEWONTIN, R.C.; GELBART, W.M., **Introdução a genética.** Sétima edição. Rio de Janeiro: Guanabara Koogan, 2000.

HAUPT, R.L., , **Pratical Genetic Algorithms**. New York: Wiley-Interscience, 2004.

HOLLAND, J. H., Concerning efficient adaptive systems. In M. C. Yovits, G. T. Jacobi, & G. D. Goldstein (Eds.), **Self-organizing systems** (pp. 215-230). Washington: Spartan Books. 1962a.

HOLLAND, J. H., Information processing in adaptive systems. Information Processing in the Nervous System, **Proceeding of the International Union of Physiological Sciences**, New York, v. 3, p. 330-339. 1962b.

HOLLAND, J. H., Outline for a logical theory of adaptive systems. **Journal of the Association for Computing Machinery**, New York, v.3, p. 297-314. 1962c.

HOLLAND, J. H., **Adaptation in Natural and Artificial Systems**. Washington: MIT Press, 1975.

LANGDON, W.B., POLI, R., **Foundations of Genetic Programming**. Washington: Springer, 2002.

LINDEN, R., **Algoritmos Genéticos – Uma importante Ferramenta da Inteligência Computacional.** São Paulo: Brasport, 2006.

LYRA, J., **Beowulf e o Saci**. Disponível em: <http://www.latt.if.usp.br/pmc/Howto>. Acesso em: 10 jan. 2005.

MARTIN, F. G., Cockerham, C. C., High speed selection studies. In O. Kempthorne, ed., **Biometrical Genetics**. London: Pergamon. 1960.

MENDEL, G., **Versuche über Pflanzenhybriden**. Abhandlungen: Brüm, 1865.

MENDEL, G., **Versuche über Pflanzen-Hybriden: Verhandlungen des nature forschenden Vereines**. Abhandlungen: Brüm, 1866.

MITCHELL, M., **Introduction to genetic algorithms.** Cambridge: The MIT Press, 1999.

PITANGA, M., **Computação em Cluster: O Estado da Arte da Computação.** Rio de Janeiro: Brasport, 2003.

PITANGA, M., **Construindo Supercomputadores com Linux.** Segunda edição. Rio de Janeiro: Brasport, 2004.

ROSENBROCK, H. H., An automatic method for finding the greatest or least value of a function. **Computer Journal**, v. 3, 175-184, 1960.

SANTANA, H., **Estratégias de variação automática da pressão seletiva em uma classe de algoritmos evolutivos**. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Presbiteriana Mackenzie, São Paulo, 2004.

SIVANANDAM, S.N., DEEPA, S.N., **Introduction to Genetic Algorithms**. New York: Springer, 2007.

STERLING, T. L.; SALMON, J.; BECKER, D. J.; SAVARESE, D. F., **How to Build a Beowulf.** Cambridge: MIT Press, 1999.

TALBI, E.; BESSIERE, P.; AHUACTZIN, J., **Parallel Cooperating Genetic Algorithms: An application to robot motion planning**. In: CHAMBERS, L. (Org.). Pratical Handbook of Genetic Algorithms - vol II. New York: CRC Press, 1995.

TANENBAUM, A. S.; WOODHULL, Albert S. Woodhull, **Sistemas Operacionais: Projeto e Implementação.** Segunda edição. Porto Alegre: Bookman, 2000.

TOMASZ, D. G., **Genetic Algorithms Reference**. Washington: Tomasz Gwiazda, 2006.

ANEXO A

A. Técnicas de Programação Paralela e Redes

Nesse anexo serão abordados os aspectos e funcionalidades de uma estrutura de rede, seus padrões e modelos, bem como os elementos que compõem um cluster de computadores.

As definições de camadas são de essencial importância para conhecer bem o fluxo das informações, como elas trafegam em uma futura comunicação entre os micros ou nós, bem como onde ocorrerá a troca de informações em uma estrutura de processamento distribuído.

A.1 – Clusters Beowulf

O modelo de arquitetura baseado em *clusters* é representado principalmente pelos *clusters* Beowulf. Como definição, pode-se dizer que Beowulf é uma arquitetura de multicomputadores utilizados para computação paralela, que consiste em um conjunto de máquinas formado por um nó servidor e nós clientes conectados via rede (*Ethernet* ou outra topologia qualquer), rodando um sistema operacional que implemente estruturas de Máquinas Virtuais Paralelas (PVM - *Parallel Virtual Machines*) e/ou Passagem de Mensagens (MPI – Message Passing Interface.

O servidor tem a função de controlar todo o *cluster*, distribuir os arquivos e as tarefas para os nós clientes e servir de *gateway* para conexão externa. Existe a possibilidade de haver mais de um servidor em um *cluster* Beowulf, podendo ser dedicado à operações específicas como monitoração ou *gateway*, ou para se prestar somente como console.

Cada nó de um *cluster* Beowulf deve ser o mais simples possível. Recomenda-se utilizar máquinas com o menor número de acessórios possível, sem interfaces de vídeo ou áudio, unidades de disco removível, portas paralelas e outros. Todos os nós clientes são configurados a partir do servidor e só realizam o que lhes é determinado por ele, sendo totalmente dependentes, portanto simulando o comportamento de uma máquina única (Lyra, 2005).

A.2 – Histórico

O nome Beowulf foi tomado emprestado de um dos mais antigos poemas épicos da língua inglesa, que conta a estória de um herói de grande força e valentia em sua saga para derrotar o monstro de Grendel.

O primeiro *cluster* Beowulf foi desenvolvido por Sterling, Salmon, Becker e Savarese (1999), pesquisadores do Centro de Excelência em Dados Espaciais e Informações Científicas (CESDIS) da Agência Espacial Americana (NASA) no Centro Espacial Goddard em Greenbelt, Maryland, no ano de 1994. Foi implementado para utilização no projeto de Ciências Espaciais e Terrestres (ESS) em conjunto com o grupo de Computação e Comunicação de Alta Performance (HPCC) do qual os pesquisadores faziam parte.

Este *cluster* era formado por 16 processadores Intel™ 486DX4 100MHz conectados via rede *Ethernet* de 10Mbps. Na época, devido ao baixo *throghput* da rede e ao alto custo de um *switch*, foi necessário desenvolver um *driver* específico para as placas *dual ethernet* com a finalidade de distribuir o tráfego das mensagens na rede (criando a chamada *channel bonded Ethernet*, onde cada nó poderia acessar diretamente todos os demais). O sistema operacional utilizado foi o Linux, que já implementava as estruturas necessárias para passagem de dados (MPI) e para gerenciamento de máquinas virtuais paralelas (PVM). Os componentes de *hardware* eram comuns - cada nó do *cluster* era formado por um processador 486DX4 100MHz, placa mãe SiS471 com *cache* de 256KB, memória RAM de 16MB/60ns, disco rígido de 540MB IDE, placa de rede dual 10Mbps, de fácil aquisição e custo acessível. Estas facilidades contribuiram para o enorme sucesso e a popularização deste tipo de arquitetura que, em pouco tempo, já era implementada em diversas universidades e instituições de pesquisa. Em 1996, na feira Supercomputing '96, foi mostrado que um *cluster* de custo total inferior a $ 50,000.00 alcançaria velocidades superiores a 1GigaFLOP/s (1 bilhão de operações de ponto flutuante por segundo) (Lyra, 2005).

A.3 – Particularidades

Um *cluster* Beowulf não é um pacote de *software*, nem uma nova topologia de rede, nem um *hardware* especial e nem um sistema operacional com *kernel* diferenciado. É uma maneira de interligar estações que rodam Linux (ou algum outro sistema operacional) de forma a simular o comportamento e a performance de um supercomputador paralelo. Embora existam muitos pacotes de *software*, versões de *kernel*, bibliotecas PVM e MPI diversas e aplicativos de configuração que fazem a arquitetura de *cluster* mais rápida ou fácil de configurar, é possível construir um Beowulf com qualquer uma das distribuições atuais padrão do Linux e estações de trabalho comuns. Basicamente, uma configuração de duas ou mais máquinas rodando Linux em rede, compartilhando pelo menos uma estrutura de diretórios via NFS, uma relação de *trusting* que permita executar *shell* remoto (rsh) e bibliotecas de PVM ou MPI, já se constitui em um *cluster* Beowulf.

Na taxonomia tradicional de computação paralela, um *cluster* Beowulf pode ser situado como o meio-caminho entre as máquinas de processamento paralelo massivo (MPPs - *Massive Parallel Processors*), como o Cray ou os Hipercubos, e uma rede de estações (NOW - *network of workstations*), com características e benefícios de ambas. As MPPs são máquinas tipicamente maiores, mais caras e mais rápidas em sua conexão interna que os *clusters*. A programação é complexa pois é preciso preocupar-se com detalhes como localidade dos processos, balanceamento de carga, granularidade e *overhead* na comunicação para atingir a

performance ótima, dentre outras. Para a programação em uma rede de estações são necessários algoritmos bastante tolerantes quanto à problemas de distribuição da carga e latência provocada pela forma de comunicação. Para um *cluster* Beowulf, qualquer programa - tanto para MPPs como para NOWs - que não necessite de uma granularidade muito fina de processamento pode ser portado sem grandes dificuldades.

Um *cluster* Beowulf se diferencia de uma rede de estações (NOW) por algumas poucas, porém importantes, características. Em primeiro lugar, os nós de um *cluster* são dedicados exclusivamente ao processamento interno do *cluster*. Isso facilita no balanceamento de carga, já que a performance de cada nó não varia por força de fatores externos, uma vez que estão isolados do mundo exterior, rodando somente as aplicações determinadas pelo servidor. Em segundo lugar, nas redes de estações, além de existirem outras informações e processos que não pertencem ao domínio do programa que está sendo executado em paralelo e que podem diminuir a eficiência, também podem gerar um problema de segurança. Em um *cluster* todos os processos que rodam em qualquer nó são controlados através de um número de identificação gerado e controlado pelo servidor, não havendo portanto chance de execução de processos de atividade desconhecida. Finalmente, em uma rede de estações é preocupação do sistema operacional tornar a máquina mais amigável ao usuário, ocasionando desperdício de recursos que, no caso do *cluster* serão aproveitados para o processamento, já que os nós clientes não têm interação direta com o usuário (Pitanga, 2003).

A.4 – Estrutura de Redes

As redes locais (LANs) consistem em computadores, placas de rede, meios de rede, dispositivos de controle de tráfego de rede e dispositivos periféricos. As LANs permitem que as empresas que usam a tecnologia da computação compartilhem, de modo eficaz, itens como arquivos e impressoras e usem meios de comunicação como correio eletrônico. Elas reúnem: dados, comunicações, computação e servidores de arquivos.

As LANs são projetadas para executar as seguintes ações:

- Operar dentro de uma área geográfica limitada
- Permitir que usuários acessem meios de grande largura de banda
- Fornecer conectividade ininterrupta aos serviços locais
- Conectar dispositivos fisicamente adjacentes

Embora existam outros modelos, a maior parte dos atuais fabricantes de rede relaciona seus produtos ao modelo de referência OSI, que é fundamental para comunicações em rede, especialmente quando deseja instruir os usuários no uso dos produtos. O modelo OSI é considerado a melhor ferramenta disponível para ensinar às pessoas como os dados são enviados e recebidos através de uma rede.

O modelo de referência OSI permite que se visualize as funções de rede que acontecem em cada camada. Sobretudo, o modelo de referência OSI é uma estrutura que pode ser usada para entender como as informações trafegam através de uma rede. Além disso, pode-se usar o modelo de referência OSI para visualizar como as informações, ou pacotes de dados, trafegam desde os programas aplicativos (por exemplo, planilhas, documentos e outros), através de um meio de rede (como cabos e outros), até outros programas aplicativos localizados em um outro computador de uma rede, mesmo se o remetente e o destinatário tiverem tipos diferentes de meios de rede (CCNA, 2004).

A.5 – Modelo OSI

No modelo de referência OSI, existem sete camadas numeradas e cada uma ilustra uma função particular da rede. Essa separação das funções da rede é chamad*a* divisão em camadas. Dividir a rede nessas sete camadas oferece as seguintes vantagens:

- Decompõe as comunicações de rede em partes menores e mais simples.
- Padroniza os componentes de rede, permitindo o desenvolvimento e o suporte por parte de vários fabricantes.
- Possibilita a comunicação entre tipos diferentes de *hardware* e de *software* de rede.
- Decompõe as comunicações de rede em partes menores, facilitando sua aprendizagem e compreensão.

A tarefa de transferir informações entre computadores é dividida em sete problemas menores e mais gerenciáveis no modelo de referência OSI. Cada um desses problemas menores é representado por sua própria camada no modelo. As sete camadas do modelo de referência OSI são:

Camada 7: A camada de aplicação
Camada 6: A camada de apresentação
Camada 5: A camada de sessão
Camada 4: A camada de transporte
Camada 3: A camada de rede
Camada 2: A camada de enlace
Camada 1: A camada de física

Cada camada OSI individual tem um conjunto de funções que devem ser executadas para que os pacotes de dados trafeguem de uma origem a um destino em uma rede. A seguir, está uma breve descrição de cada camada no modelo de referência OSI, segundo o Cisco Systems (2004).

Camada 7: A camada de Aplicação

A camada de aplicação é a camada OSI mais próxima do usuário; fornece serviços de rede aos aplicativos do usuário. Ela se diferencia das outras por não fornecer serviços a nenhuma outra camada OSI, mas apenas a aplicativos fora do modelo OSI. Os programas de planilhas, os programas de processamento de texto e os programas de terminal bancário são exemplos desses processos de aplicativos. A camada de aplicação estabelece a disponibilidade dos parceiros de comunicação pretendidos, sincroniza e estabelece o acordo sobre os procedimentos para a recuperação de erros e o controle da integridade dos dados. Em poucas palavras é definida com o exemplo de navegadores para internet.

Camada 6: A camada de Apresentação

A camada de apresentação assegura que a informação emitida pela camada de aplicação de um sistema seja legível para a camada de aplicação de outro sistema. Se necessário, a camada de apresentação faz a conversão de vários formatos de dados usando um formato comum. Pode-se pensar na camada 6 com o mínimo de palavras, como um formato de dados comum.

Camada 5: A camada de Sessão

A camada de sessão, como está implícito no nome, estabelece, gerencia e termina sessões entre dois *hosts* que se comunicam. A camada de sessão fornece seus serviços à camada de apresentação. Ela também sincroniza o diálogo entre as camadas de apresentação dos dois *hosts* e gerencia a troca de dados entre eles. Além da regulamentação básica das sessões, a camada de sessão oferece recursos para a transferência eficiente de dados, classe de serviço e relatórios de exceção de problemas da camada de sessão, da camada de apresentação e da camada de aplicação. Para definir em poucas palavras a camada 5, é caracterizada em diálogos e conversações.

Camada 4: A camada de Transporte

A camada de transporte segmenta os dados do sistema *host* que está enviando e monta os dados novamente em uma seqüência de dados no sistema *host* que está recebendo. O limite entre a camada de transporte e a camada de sessão pode ser comparado ao limite entre os protocolos de aplicativos e os protocolos de fluxo de dados. Enquanto as camadas de aplicação, de apresentação e de sessão estão relacionadas a problemas de aplicativos, as quatro camadas inferiores estão relacionadas a problemas de transporte de dados.

A camada de transporte tenta fornecer um serviço de transporte de dados que isola as camadas superiores de detalhes de implementação de transporte. Especificamente, algumas questões, tais como realizar transporte confiável entre dois *hosts*, dizem respeito à camada de transporte. Fornecendo serviços de comunicação, a camada de transporte estabelece, mantém e termina corretamente circuitos virtuais. Fornecendo serviço confiável, são usados o controle do fluxo de informações e a detecção e recuperação de erros de transporte. Para definir em poucas palavras a camada 4, seria resumida em qualidade de serviços e confiabilidade.

Camada 3: A camada de Rede

A camada de rede é uma camada complexa que fornece conectividade e seleção de caminhos entre dois sistemas *hosts* que podem estar localizados em redes geograficamente separadas. Para definir em poucas palavras a camada 3, é o processo de seleção de caminhos, roteamento e endereçamento.

Camada 2: A camada de Enlace de Dados

A camada de enlace fornece trânsito confiável de dados por meio de um *link* físico. Fazendo isso, a camada de enlace trata do endereçamento físico (em oposição ao endereçamento lógico), da topologia de rede, do acesso à rede, da notificação de erro, da entrega ordenada de quadros e do controle de fluxo. Para definir em poucas palavras a camada 2, entenda-se por quadros e controle de acesso ao meio.

Camada 1: A camada Física

A camada física define as especificações elétricas, mecânicas, funcionais e de procedimentos para ativar, manter e desativar o *link* físico entre sistemas finais. Características como níveis de voltagem, temporização de alterações de voltagem, taxas de dados físicos, distâncias máximas de transmissão, conectores físicos e outros atributos similares são definidas pelas especificações da camada física. Para definir em poucas palavras a camada 1 como sinais e meios.

A.6 - Modelo TCP/IP

O Departamento de Defesa dos Estados Unidos (*DoD*) desenvolveu o modelo de referência TCP/IP porque queria uma rede que pudesse sobreviver a qualquer condição, mesmo a uma guerra nuclear. Para ilustrar melhor, imagine um mundo em guerra, entrecruzado por diferentes tipos de conexões: cabos, microondas, fibras óticas e *links* de satélites. Imagine, então, que existe a necessidade que informações/dados (na forma de pacotes) trafeguem, independentemente da condição de qualquer nó ou rede particular na *internetwork* (que, nesse caso, pode ter sido parcialmente destruída pela guerra). O Departamento de Defesa dos Estados Unidos queria que seus pacotes chegassem, todas as vezes, em qualquer condição, de um ponto a qualquer outro. Foi esse complexo problema de projeto que levou à criação do modelo TCP/IP e que se tornou, desde então, o padrão no qual a internet se desenvolveu.

Quando ler sobre as camadas do modelo TCP/IP, tenha em mente o objetivo inicial da Internet; isso vai ajudar no entendimento da razão de certas coisas serem como são. O modelo TCP/IP tem quatro camadas: a camada de aplicação, a camada de transporte, a camada de internet e a camada de acesso à rede. É importante notar que algumas das camadas do modelo TCP/IP têm o mesmo nome das camadas no modelo OSI. Porém não se pode confundir as camadas dos dois modelos, porque a camada de aplicação tem funções diferentes em cada um deles.

Camada de Aplicação

Os projetistas do TCP/IP decidiram que os protocolos de mais alto nível deviam incluir os detalhes da camada de apresentação e de sessão. Eles simplesmente criaram uma camada de aplicação que trata de protocolos de alto nível, questões de representação, codificação e controle de diálogo. O TCP/IP combina todas as questões relacionadas a aplicações em uma camada e garante que esses dados estejam empacotados corretamente para a próxima camada.

Camada de Transporte

A camada de transporte lida com questões de qualidade de serviços, de confiabilidade, controle de fluxo e de correção de erros. Um de seus protocolos, o *Transmission Control Protocol* (TCP), fornece formas excelentes e flexíveis de se desenvolver comunicações de rede confiáveis com baixa taxa de erros e bom fluxo. O TCP é um protocolo orientado para conexões. Ele mantém um diálogo entre a origem e o destino enquanto empacota informações da camada de aplicação em unidades chamadas segmentos. Orientado para conexões não significa que exista um circuito entre os computadores que se comunicam (o que poderia ser comutação de circuitos). Significa que segmentos da camada 4 do modelo OSI trafegam entre dois *hosts* para confirmar que a conexão existe logicamente durante um certo período. Isso é conhecido como comutação de pacotes.

Camada de Internet

A finalidade da camada de internet é enviar pacotes da origem de qualquer rede na *internetwork* e fazê-los chegar ao destino, independentemente do caminho e das redes que tomem para chegar lá. O protocolo específico que governa essa camada é chamado *Internet Protocol* (IP). A determinação do melhor caminho e a comutação de pacotes acontecem nessa camada. Pense nisso em termos do sistema postal. Quando alguém envia uma carta, não se sabe como ela vai chegar ao seu destino (existem várias rotas possíveis), mas, o que realmente importa, é que a carta chegue.

Camada de Acesso à Rede

O significado do nome dessa camada é muito amplo e um pouco confuso. É também chamada de camada host-rede. É a camada que se relaciona a tudo aquilo que um pacote IP necessita para realmente estabelecer um *link* físico e depois estabelecer outro *link* físico. Isso inclui detalhes de tecnologia de LAN e WAN e todos os detalhes nas camadas física e de enlace do OSI.

A.7 - Sistemas Operacionais

Segundo Tanenbaum e Woodhull (2000), sem *software*, um computador não teria utilidade, finalidade alguma. Com *software*, um computador pode armazenar, processar e recuperar informações, exibir documentos de multimídia, pesquisar na internet e envolver-se em muitas outras importantes atividades que justificam seu valor. O *software* de computador pode ser dividido, a grosso modo, em duas espécies: programas de sistema, que gerenciam a operação do computador em si, e programas aplicativos, que executam o trabalho que o usuário realmente deseja. O programa de sistema mais fundamental é o sistema operacional, que controla todos os recursos do computador e fornece a base sobre a qual os programas aplicativos podem ser escritos.

Um moderno sistema de computadores consiste em um ou mais processadores, alguma memória principal (também conhecida como RAM – Random *Access Memory*; Memória de Acesso Aleatório), discos, impressoras, interfaces de rede e outros dispositivos de entrada/saída. Em suma, um sistema complexo. Escrever os programas que controlam todos esses componentes e usá-los corretamente é um trabalho extremamente difícil. Se cada programador tivesse de se preocupar com o modo como as unidades de disco funcionam e com todas as dúzias de coisas que poderiam dar errado ao ler um bloco de disco, seria provável que muitos programas sequer pudessem ser escritos.

Há muitos anos tornou-se bastante evidente a necessidade de encontrar uma maneira de isolar os programadores da complexidade do *hardware*. A maneira com que isso se desenvolveu gradualmente foi colocar uma camada de *software* por cima do *hardware* básico para gerenciar todas as partes do sistema e oferecer ao usuário uma interface ou máquina virtual que é mais fácil de entender e de programar. Essa camada de *software* é o sistema operacional (Tanenbaum e Woodull, 2000).

A.7.1 Sistema Operacional como Máquina Virtual ou Máquina Estendida

O sistema operacional gerencia tudo em seu computador, cada *bit* que entra, cada *bit* que sai nos dispositivos de Entrada e Saída (E/S ou I/O). Ele também é responsável por cada tarefa que o usuário solicita para ser executada, qual virá primeiro, qual a importância dela, além do que ele administra todo o sistema de arquivos contidos no disco rígido, sem o qual seria impossível a criação de qualquer arquivo em seu computador.

Resumindo, o sistema operacional (S.O.) é um *software* cuja função é apresentar ao usuário o equivalente de uma máquina estendida ou máquina virtual que é mais fácil de programar que o *hardware* subjacente.

A arquitetura (conjunto de instruções, organização da memória, E/S e estrutura de barramento) da maior parte dos computadores no nível da linguagem de máquina é primitiva e desajeitada para programar, especialmente para entrada/saída.

Sem entrar em detalhes reais, deve estar claro que o programador médio provavelmente não quer ficar muito intimamente envolvido com a programação de disquetes, discos rígidos, que são igualmente complexos, apesar de bem diferentes. Em vez disso, o que o programador quer é uma abstração de ordem superior mais simples de lidar. No caso dos disco, uma abstração típica seria que o disco contém uma coleção de nomes de arquivos. Cada arquivo pode ser aberto para leitura ou gravação, então lido ou gravado e finalmente fechado. Detalhes como se a gravação deve ou não usar modulação de freqüência modificada e qual é o estado atual do motor não devem aparecer na abstração apresentada ao usuário.

O programa que realiza o mascaramento do *hardware* para o programador, apresentando uma simples visão de nomes de arquivos que podem ser lidos e gravados é, naturalmente, o sistema operacional. Assim como o sistema operacional mascara o *hardware* do programador e apresenta uma interface orientada para arquivo mais simples, ele também oculta muitas coisas complexas relacionadas com interrupções, temporizadores, gerenciamento de memória e outros recursos de baixo nível. Em cada caso, a abstração oferecida pelo sistema operacional é mais simples e mais fácil de utilizar do que o *hardware* subjacente (Tanenbaum e Woodull, 2000).

A.7.2 – Linux

O Linux surgiu de forma muito interessante. Tudo começou em 1991, quando um programador finlandês de 21 anos, Linus Benedict Torvalds, enviou a seguinte mensagem para uma lista de discussão na Internet: "Olá para todos que estão usando Minix. Estou fazendo um sistema operacional free (como passatempo) para 386, 486, AT e clones". Minix era um limitado sistema operacional baseado em Unix que rodava em microcomputadores como o AT. Linus pretendia desenvolver uma versão melhorada do Minix e mal sabia que seu suposto "passatempo" acabaria num sistema engenhoso. Muitos acadêmicos conceituados ficaram interessados na idéia do Linux e, a partir daí, programadores das mais variadas partes do mundo passaram a trabalhar em prol desse projeto. Cada melhoria desenvolvida por um programador era distribuída pela internet e integrada ao núcleo do Linux. No decorrer dos anos, este trabalho árduo e voluntário de centenas de programadores e colaboradores tornou-se num sistema operacional bem amadurecido.

O Linux tem crescido nos últimos anos e segundo o CEO (*Chief Executive Officer*) da Conectiva, empresa especializada em Linux, o número de cópias do sistema distribuídas por *downloads*, CDs ou revistas multiplicou-se oito vezes de 1998 para 1999 no Brasil. Este número cresceu de 45.000 para 400.000.

Uma característica muito atraente do sistema é o fato de o mesmo não ser um produto comercial, porém algumas empresas fornecem uma versão comercial específica. Se você quiser usar o Linux em casa ou no trabalho, não precisará pagar por ele desde que não esteja usando uma distribuição comercial e tenha conseguido sua cópia na Internet. Linus registrou seu produto sob os termos de licença da GNU, que podem ser resumidos assim: "O Linux pode ser usado, alterado e replicado quantas vezes necessário".

O Linux tem sido sucesso absoluto entre universitários e provedores de internet. Milhares de provedores ao redor do mundo estão abandonando seus Windows NT e migrando

para o Linux, que oferece muitas vantagens para administradores dos mesmos provedores quando o assunto é velocidade e segurança (Lyra, 2005).

A.8 - MPI – Interface de Passagem de Mensagens

De acordo com Pitanga (2004), o MPI é uma biblioteca com funções para troca de mensagens, responsável pela comunicação e sincronização de processos em um *cluster* paralelo. Dessa forma, os processos de um programa paralelo podem ser escritos em uma linguagem de programação seqüencial, tal como C ou Fortran. O principal objetivo do MPI é disponibilizar uma interface que seja largamente utilizada no desenvolvimento de programas que utilizem troca de mensagens. Além de garantir a portabilidade dos programas paralelos, essa interface deve ser implementada eficientemente nos diversos tipos de máquinas paralelas existentes (reais ou *clusters* de *workstations*).

O objetivo principal é que a MPI se torne o padrão de interface mais utilizado, tomando o espaço de bibliotecas com interfaces específicas, como PVM. O MPI define um conjunto de rotinas para facilitar a comunicação (troca de dados e sincronização) entre processos em memória, ela é portável para qualquer arquitetura, tem aproximadamente 129 funções para programação e ferramentas para análise de performance. A biblioteca MPI possui rotinas para programas em linguagem C / C++, Fortran 77 / 90. Os programas são compilados e ligados à biblioteca MPI. Inicialmente, na maioria das implementações, um conjunto fixo de processos é criado. Porém, esses processos podem executar diferentes programas. Por isso, o padrão MPI é às vezes referenciado como MPMD (Multiple Program Multiple Data).

Muitas implementações de bibliotecas do padrão MPI têm sido desenvolvidas, sendo algumas proprietárias e outras de código livre, mas todas seguindo o mesmo padrão de funcionamento e uso. O esforço de padronização MPI envolve cerca de 60 pessoas de 40 organizações, principalmente dos Estados Unidos e da Europa. A maioria dos vendedores de computadores paralelos e concorrentes estão envolvidos com o padrão MPI, por meio de pesquisas em Universidades e laboratórios governamentais e industriais. O processo de padronização começou com um seminário no Centro de Pesquisas em Computação Paralela, realizado em 29 e 30 de Abril de 1992 em Williamsburg, Virginia. Nesse seminário, as características essenciais para padronização de uma interface de passagem de mensagem foram discutidas, sendo estabelecido também um grupo de trabalho para continuar o processo de padronização. Em novembro de 1992 foi formalizado o processo de padronização, adotando-se o procedimento e a organização do HPF (*High Performance Fortran Forum*). O projeto do MPI padrão foi apresentado na conferência Supercomputing 93, realizada em novembro de 1993, do qual se originou a versão oficial do MPI (5 de maio de 1994).

Em 1994, no final do encontro do MPI-1, foi estabelecido que se deveria esperar mais experiências práticas com o MPI. A sessão do Forum-MPIF do Supercomputing 94 possibilitou a criação do MPI-2, que teve início em abril de 1995. No Supercomputing 96 foi apresentada a versão preliminar do MPI-2. Em abril de 1997 o documento MPI-2 foi votado e aceito de forma unânime. Com isso os fabricantes de computadores podem, agora, se concentrar na implementação eficiente da biblioteca, e não perder tempo especificando um padrão.

As funcionalidades que o MPI designa são baseadas em práticas comuns de paralelismo e outras bibliotecas de passagem de mensagem similares, como Express, NX/2, Vertex, Parmacs e P4. A filosofia geral do padrão MPI é implementar e testar características

novas que venham a surgir no mundo da computação paralela. Assim que uma determinada gama de características novas tenham sido testadas e aprovadas, o grupo MPI reúne-se novamente para considerar sua inclusão na especificação do padrão. Muitas das características do MPI têm sido investigadas e usadas por grupos de pesquisas por muitos anos, mas não em ambientes de produção ou comerciais. Sendo assim, existe um grande lapso de tempo entre novidades do padrão e sua implementação por vendedores e colaboradores. Entretanto, a incorporação desta características dentro do padrão MPI é justificada pelas expressivas inovações que elas trazem.

O MPI implementa excelente mecanismo de portabilidade e independência de plataforma computacional. Por exemplo, um código MPI que foi escrito para uma arquitetura IBM RS-6000 utilizando o sistema operacional AIX pode ser portado para uma arquitetura SPARC com o S.O. Solaris ou PC com Linux com quase nenhuma modificação no código fonte da aplicação.

O MPI-1.2 é uma extensão para o padrão MPI-1 de 1992. Há uma terceira especificação chamada MPI-2, que adiciona várias extensões à versão 1.2, sendo uma das mais importantes do controle de processos dinâmicos. Devido à dificuldade de se encontrar uma implementação MPI-2 que seja distribuída livremente, este trabalho restringe-se ao padrão MPI-1.2.

A execução de uma aplicação baseada nesta API (Interface de Programação de Aplicações) inicia um procedimento de disparar por meio de um processo "pai" seus processos "filhos" para serem executados remotamente nos computadores escravos do *cluster*.

A MPI fornece uma relação que permite a comunicação entre os processos de um programa paralelo com um outro processo, porém não especifica como os processos são criados, nem como ele estabelece comunicação. Além disso, uma aplicação MPI é estática, isto é, nenhum processo pode ser adicionado ou suprimido de uma aplicação depois de começada. As razões para adicionar um processo ao MPI são técnicas e práticas. As classes importantes de mensagens que passam pelas aplicações requerem um controle de processo. Esses incluem tarefas, aplicações de série com módulos paralelos e problemas que requerem uma avaliação em tempo de execução do número e do tipo de processos que devem ser iniciados.

Os elementos mais importantes em implementações paralelas são a comunicação de dados entre processos paralelos e o balanceamento de carga. Os processos podem usar mecanismos de comunicação ponto a ponto, com operações para enviar mensagens de um determinado processo a outro. Um grupo de processos pode chamar operações coletivas de comunicação para executar operações globais. O MPI é capaz de suportar comunicação assíncrona e programação modular, através de mecanismos de comunicadores que permitem ao usuário MPI definir módulos que englobem estruturas de comunicação interna.

Cada processo executa e comunica-se com outras instâncias do programa, possibilitando a execução no mesmo processador ou diferentes processadores. Essas instâncias podem se dar por meio de comunicação ponto a ponto ou coletivas. A melhor performance acontece quando esses processos são distribuídos entre diversos processadores. A comunicação básica consiste em enviar e receber dados de um processador para outro. Essa comunicação se dá por meio de uma estrutura de rede, na qual os processos estão em sistema de memória distribuída.

O pacote de dados enviados pelo MPI requer vários pedaços de informações: o processo transmissor, o processo receptor, o endereço inicial de memória onde os itens de dados deverão ser mandados, a mensagem de identificação e o grupo de processos que podem

receber a mensagem, ou seja, tudo isto ficará sob responsabilidade do programador em identificar o paralelismo ou implementar um algoritmo utilizando construções com o MPI.

Algumas implementações do MPI são:

- MPI-F: IBM Research
- MPICH: ANL/MSU – Argonne National Laboratory e Mississipi State University
- UNIFY: Mississipi State University
- CHIMP: Edinburgh Parallel Computing Center
- LAM: Ohio Supercomputer Center
- MPL: IBM Research

A.8.1 – Conceitos Importantes Sobre MPI

Segundo Marcos Pitanga (Pitanga, 2004), deve-se observar os seguintes conceitos para poder entender melhor a organização do MPI:

***Rank*:** Todo processo possui um único número de identificação, recebido pelo sistema quando no ato de sua inicialização. Esse número corresponde a um valor contínuo que varia de 0 até n-1 processos.

***Group*:** Grupo é um conjunto ordenado de n processos. Qualquer grupo é associado a um comunicador (*communicator*), e inicialmente, são membros de um grupo já preestabelecido.

***Communicator*:** Um comunicador é um conjunto de processos que consegue enxergar um ao outro. No MPI, as mensagens só podem ser transmitidas dentro de um comunicador. O MPI usa essa combinação de grupo e contexto para garantir uma comunicação confiável e segura, a fim de evitar problemas relacionados ao envio de mensagens entre os processos. Cada processo dentro de um comunicador é identificado por um número, chamado *rank*, que varia de 0 até n -1. Cada mensagem transmitida deve indicar ao comunicador para qual *rank* a mensagem será enviada. Um programa pode pertencer a mais de um comunicador simultaneamente e, nesse caso, ter mais de um *rank*. O remetente deve indicar a qual processo do comunicador a mensagem se destina. Ele faz isso definindo o *rank* do processo destinatário. 99% das rotinas do MPI exigem que seja especificado um comunicador como argumento, nesse caso o MPI_COMM_WORLD.

***Application Buffer*:** É um ambiente de memória, como uma variável, que armazena uma mensagem que o processo necessita enviar ou receber.

***System Buffer*:** É um endereço de memória reservado pelo sistema para armazenamento das mensagens. Dependendo do tipo de operação ("*send/receive*"), a mensagem no buffer de aplicação pode precisar ser copiado para o buffer do sistema. Neste caso está evidenciada uma comunicação assíncrona.

***Blocking Communication*:** uma rotina de comunicação é dita bloqueante quando a finalização de uma chamada precisa de determinados eventos. Por exemplo, um dado precisa ser enviado com sucesso, ou ter sido armazenado no *buffer* do sistema, informando que o endereço do buffer da aplicação pode ser utilizado novamente. Em uma rotina de recepção, a

mensagem deve ser armazenada no "system buffer", indicando que pode ser utilizado. A comunicação bloqueante é considerada síncrona.

***Non-Blocking Communication*:** uma rotina de comunicação é dita não bloqueante se a chamada retorna sem esperar qualquer evento que indique o fim ou o sucesso da rotina. Isso caracteriza uma comunicação assíncrona. Não se espera pela cópia de mensagens do *aplication buffer* para o *system buffer*, ou a indicação do recebimento de uma mensagem. É de inteira responsabilidade do programador a certeza de que o *aplication buffer* esteja disponível para ser reutilizado. Este tipo de comunicação é utilizada para melhorar a relação entre computação e comunicação para efeitos de ganho de performance.

***Synchronous Send*:** Bloqueia até que ocorra um *receive* correspondente no processo de destino. Pode ser *blocking* ou *non-blocking*. Operação útil quando a aplicação paralela necessita de garantias que o processo destino recebeu a mensagem. Dependendo da situação, o envio síncrono exige um longo tempo de espera em uma transmissão, o que pode acarretar problemas em *cluster*s, pois a rede é o gargalo do sistema.

***Buffered Send*:** MPI permite que se usem buffers para armazenar temporariamente as mensagens até que elas possam ser enviadas, liberando a aplicação. O programador cria um *buffer*, via função *malloc*, para o dado antes dele ser enviado. Necessidade de se garantir um espaço disponível para um "buffer", na incerteza do espaço do *System Buffer*. É avisado ao MPI o endereço do buffer que deve ser utilizado para envio "bufferizado".

***Standard Send*:** Operação básica de envio de mensagens usada para transmitir dados de um processo para outro. Pode ser "blocking" ou "non-blocking". O envio é completado assim que a mensagem for enviada, o que não implica no fato da mesma ter sido recebida.

***Ready Send*:** Se o destinatário tiver feito um receive, então a mensagem vai direto para ele. Caso contrário, o resultado não está definido e a mensagem é então destruída, o processo remetente não fica sabendo do resultado do envio.

Os comandos usados em programas que utilizam a biblioteca MPI podem ser divididos em dois grupos. O primeiro grupo possui os comandos relacionados à inicialização, identificação e finalização dos processos. O segundo grupo possui os comandos de comunicação.

A primeira regra em programas paralelos que usam MPI é que processos criados pelo comando *mpirun* representam instâncias diferentes de um mesmo programa. Apesar de usarem o mesmo código, são processos diferentes. Como os processos vão se comunicar, torna-se necessário algum esquema de endereçamento. Este endereço único de um processo em MPI é obtido pelo comando MPI_Comm_rank. Esta identificação será necessária quando os processos precisarem trocar mensagens. Outra informação importante é o número total de processos que foram criados. O comando MPI_Comm_size retorna este valor.

A.8.2 - Comandos de Comunicação Ponto a Ponto

Estes comandos são usados quando dois processos precisam se comunicar diretamente. Para cada comando de envio deve haver o respectivo comando de recepção, para que nenhuma mensagem seja perdida. Os dois comando principais são MPI_Send (envio de mensagens) e MPI_Recv (recepção de mensagens) (Pitanga, 2003).

Envio de Mensagens

Existem diversos comandos para envio de mensagens ponto a ponto. O mais utilizado é MPI_Send. Este comando é dito não bloqueante, o que significa que após ser colocado na fila de envio o programa continua sua execução. Este comando possui a seguinte estrutura:

int MPI_Send (void* *buffer*, int *contador*, MPI_Datatype *tipo*, int *destino*, int *tag*, MPI_Comm *comm*);

O campo *buffer* indica o local onde estão armazenados os dados a serem enviados, definido como uma estrura de ponteiros, ou seja, um vetor. O campo *contador* indica a quantidade de dados a serem enviados. O campo tipo identifica o tipo de dado que está sendo enviado. Alguns dos principais tipos em MPI e seus equivalentes em C podem ser vistos na Tabela 1.1. O campo destino indica o processo que receberá está mensagem. O campo *tag* representa um rótulo que identifica o tipo de mensagem que está sendo enviado. O campo *comm* identifica em que grupos estão os processos envolvidos nessa mensagem.

No exemplo a seguir, estão sendo enviados para o processo 0 (zero), 5 (cinco) variáveis inteiras, armazenadas no vetor local vet. O rótulo da mensagem foi enviado como 0 e foi usado o comunicador geral MPI_COMM_WORLD, onde são criados os processos.

MPI_Send (vet, 5, MPI_INT, 0, 0, MPI_COMM_WORLD);

Tabela A.1 - Tipos básicos em MPI e seus equivalentes em C.

Tipos em MPI	**Tipos em C**
MPI_CHAR	Char
MPI_INT	Int
MPI_SHORT	Short int
MPI_UNSIGNED_SHORT	Short int (sem sinal)
MPI_LONG	Long
MPI_UNSIGNED_LONG	Long int (sem sinal)
MPI_FLOAT	Float
MPI_DOUBLE	Double
MPI_LONG_DOUBLE	Long double

Recepção de Mensagens

De forma idêntica ao envio, existem diversos comandos para recepção de mensagens ponto a ponto. O comando mais utilizado é MPI_Recv. Este comando é dito bloqueante, que significa que o processo só continua sua execução após a mensagem ser lida. É comum o uso

em conjunto de envio não bloqueante com recepção bloqueante. O comando MPI_Recv possui a seguinte estrutura:

int MPI_Recv (void *_buffer_, int _contador_, MPI_Datatype _tipo_, int _origem_, int _tag_, MPI_Comm _comm_, MPI_Status *_status_);

O campo *buffer* indica o local onde serão armazenados os dados recebidos. O campo contador indica quantidade máxima de dados a serem recebidos. O campo tipo identifica o tipo de dado que está sendo recebido. O campo origem indica o processo de quem se espera a mensagem. O campo *tag* representa o rótulo que se espera receber. O campo *comm* identifica em que grupos de processos estão os processos envolvidos nessa comunicação. O campo status é objeto criado com o comando MPI_Status, e pode ser usado quando a quantidade de dados recebida ou quando a origem ou *tag* de uma mensagem possam variar.

No exemplo a seguir, espera-se receber do processo 0 (zero), 5 (cinco) variáveis inteiras, que serão armazenadas no vetor local vet. O rótulo da mensagem foi enviado como 0 e foi usado o comunicador geral MPI_COMM_WORLD.

```
MPI_Recv (vet, 5, MPI_INT, 0, 0, MPI_COMM_WORLD, &status);
```

Comandos de comunicação em grupo

Estes comandos são usados quando um processo necessita enviar a mesma mensagem para vários processos (1 -> N) ou quando vários processos devem enviar mensagens para um único processo (N ->1). O conceito de envio e recepção depende então do referencial, pois todos os processos estarão participando da comunicação, ou enviando ou recebendo mensagens. Os dois comandos principais são MPI_Bcast e MPI_Reduce.

MPI_Bcast

Existem diversos comandos que podem ser utilizados quando um processo deve enviar a mesma mensagem para vários processos. O comando mais utilizado é MPI_Bcast. Este comando possui a seguinte estrutura:

MPI_Bcast (void* *buffer*, int *contador*, MPI_Datatype *tipo*, int *root*, MPI_Comm *comm*);

O campo principal neste comando é *root*, pois indica qual processo estará enviando a mensagem. O campo *buffer* indica para o processo *root* o local onde estão os dados a serem enviados e para os outros processos onde eles devem armazenar os dados recebidos. O campo contador indica quantidade de dados enviados/recebidos. O campo tipo identifica o tipo de dado que está sendo enviado/recebido. O campo *comm* identifica em que grupos de processos estão os processos envolvidos nessa comunicação.

No comando a seguir, o processo 0 (zero) está enviando 50 (cinquenta) variáveis inteiras armazenadas no vetor local vet. Os outros processos receberão estes dados e armazenarão em seu vetor local vet. Foi usado o comunicador geral MPI_COMM_WORLD.

MPI_Bcast (vet, 50, MPI_INT, 0, MPI_COMM_WORLD);

MPI_Reduce

Existem diversos comando que podem ser utilizados quando um processo deve receber mensagens de vários processos. Um dos comandos mais utilizados é MPI_Reduce. Este comando possui a seguinte estrutura:

MPI_Reduce (void *_buffer_envio_, void *_buffer_recepcao_, int _contador_, MPI_Datatype _tipo_, MPI_Op _operacao_, int _root_, MPI_Comm _comm_)

No comando a seguir, o processo 0 realizará a soma de todos os valores enviados pelos demais processos. O processo 0 armazena esta soma no campo nocorrencias. O campo nvezes é usado pelos processos para armazenar o valor a ser enviado. Foi usado o comunicador geral MPI_COMM_WORLD.

MPI_Reduce (&nvezes, &nocorrencias, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

O campo principal neste comando também é _root_, pois indica qual processo receberá as mensagens. Outro campo importante é operacao, pois indica para o processo *root* o que fazer com os dados recebidos dos demais processos. Algumas das principais operações podem ser vistas na tabela 1.2. O campo buffer_envio é o local onde estão armazenados os dados a serem enviados para o processo *root*. O campo buffer_recepcao é usado pelo processo *root* para armazenar o resultado da operação indicada no campo operação. Os campos contador e *comm* possuem o mesmo significado do comando MPI_Bcast.

Tabela A.2 - Operações básicas utilizadas no comando MPI_Reduce.

Operação MPI_Reduce	**Descrição**
MPI_MAX	Seleciona o maior elemento dentre todos os recebidos
MPI_MIN	Seleciona o menor elemento dentre todos os recebidos
MPI_SUM	Calcula a soma de todos os valores recebidos
MPI_PROD	Calcula o produto de todos os valores recebidos

A.9 - Cálculo Computacional Paralelo

O uso de um sistema de processamento paralelo como o MPI requer um esforço de adaptação dos pesquisadores em relação aos hábitos tradicionais de programação. A arquitetura de um sistema deste tipo está baseado em novos paradigmas de programação, que são significativamente diferentes do paradigma tradicional usado em máquinas com um único processador, ou mesmo máquinas multiprocessadas. De fato, pode-se mesmo dizer que toda a estratégia algorítmica tradicional e mesmo a forma de se abordar alguns dos problemas precisam ser revistas.

Naturalmente, a forma mais simples de se começar a utilizar um sistema paralelo é usá-lo como um sistema computacional múltiplo, segundo a estratégia de *throughtput computing*. Isto tanto pode atender à demanda de um grupo de usuários que use individualmente e de forma independente nós do sistema, quanto um único usuário que utilize a máquina da mesma forma, devido às características do seu problema. Por exemplo, ele pode necessitar rodar um mesmo programa para um grande número de coleções diferentes de certos parâmetros, pois o que importa é que as necessidades computacionais sejam satisfeitas da melhor forma possível. De fato, se o seu problema pode ser resolvido dessa forma, não tenha dúvidas de fazê-lo, pois trata-se de um das formas mais eficientes de utilizar este tipo de recurso. Não complique o seu problema desnecessariamente somente porquê está usando uma máquina que permite processamento paralelo.

Este tipo de situação é caracterizado pelo fato de se realizar processamento simultâneo em conjuntos análogos mas diferentes de dados, sem que haja, em nenhum momento, necessidade de trocar quaisquer tipos de informações entre as diversas instâncias do processamento. Tudo que precisa ser feito é coletar os diversos resultados no final do processo. Pode-se dizer que não há necessidade de sincronizar os dados das diversas instâncias de processamento. Muitos programas são naturalmente deste tipo, ou podem ser abordados "algoritimicamente" de tal forma que possam ser tratados dessa maneira. Programas que fazem tarefas repetitivas de forma massiva, tais como procuras sobre ou análise de grandes quantidades de dados, sempre podem ser tratados dessa forma. Um exemplo é a renderização de imagem por meio de *ray tracing*. Existe um programa livre para este tipo de atividade, o Povray, que já vem com suporte para MPI incluído, o qual pode ser ativado por meio de uma simples opção do programa.

Outro exemplo simples de problema que pode ser tratado dessa forma, desde que use a abordagem apropriada é a maior parte dos problemas de simulação estocástica por meio do uso do método de Monte Carlo, em especial nos casos da simulação de distribuições de equilíbrio termodinâmico e do cálculo de integrais de dimensão alta. Em todos esses casos existe algum tipo de processamento estocástico que produz amostragens de algum observável. O processo é estatístico e quanto maior for o número de amostragens independentes obtidas, com mais precisão pode-se medir os observáveis. Para que as amostragens possam todas contribuir independentemente para o resultado final, o que pretende-se é que sejam estatisticamente independentes umas das outras, ou seja, a última coisa que deseja-se aqui é que existam dependências entre os diversos processos e trocas de informação entre eles. Uma situação ideal, portanto.

Entretanto, mesmo para problemas deste tipo pode ser vantajoso ir um pouco adiante e aprender a usar as bibliotecas de passagem de mensagem MPI e PVM. Por exemplo, se você estiver usando uma máquina de 100 nós, a simples tarefa de entrar em cada um deles para submeter um *job*, ou mesmo o uso de um *script* para fazer isto de forma mais automática, será

uma fatia muito grande do seu trabalho de simples execução do programa. Adicionando a isto a necessidade de se verificar que nós estão ou não disponíveis, ou com problemas, bem como a necessidade de se monitorar o transcorrer do processamento, existe uma grande quantidade de trabalho repetitivo. Através do uso das bibliotecas MPI ou PVM, pode-se escrever um programa *master* que não faz outra coisa além de realizar todo este trabalho de uma forma bem automática e muito confiável.

A escrita de um programa de controle (*master-program*), que não faz cálculos mas que controla e coordena um grande número de processos simultâneos que os realizam, é apenas uma de muitas estratégias possíveis de programação paralela. Ela também pode ser usada em casos em que é necessário fazer de tempos em tempos uma sincronização de dados entre todos os processos, com o uso de uma estratégia do tipo *scatter-gather*, na qual o programa *master* espalha tarefas de processamento pelos nós, recebe de volta resultados parciais, sincroniza os dados, volta a espalhar tarefas de processamento e assim por diante, ciclicamente, até que o problema tenha sido resolvido. Um exemplo de situação na qual se pode usar este tipo de estratégia é a resolução de problemas de eletrostática (condições de contorno para a equação de Laplace) pelo método de relaxação.

Basta observar que o critério crucial para que um esquema de programação funcione bem numa máquina com o MPI é a que diz respeito à questão da granularidade da programação. Uma máquina com o MPI é caracterizada por ter nós de processamento muito fortes e uma rede com capacidade de comunicação relativamente limitada. Outros esquemas, enfatizando enormes quantidades de nós de processamento fracos, bem como redes especiais extremamente rápidas, já foram tentados extensivamente no passado, levando à falência da maior parte das empresas que tentaram produzir e comercializar esses sistemas. Para que tenhamos sistemas paralelos grandes e acessíveis, é essencial o uso de componentes que sejam produzidos em massa para outros fins, dos quais decorrem as características arquiteturais básicas do MPI.

Para que um sistema paralelo desse tipo funcione com grande eficiência é importante que o "grão" de processamento que é atribuído a cada nó seja relativamente grande, podendo ocupar o nó por algum tempo antes que seja necessário passar mensagens e trocar dados com outros nós ou com o programa *master*. Se isto não for o caso, os nós estarão parados à espera de uma mensagem na maior parte do tempo e o desempenho do sistema cairá drasticamente. Assim, a filosofia de programação apropriada para o uso do MPI é re-examinar o seu problema à procura da forma de quebrá-lo em grãos grandes, que possam rodar bem em um nó forte. Em seguida, é necessário pensar nos métodos de passagem de mensagens e sincronização de dados que tenham a melhor chance de não deixar os nós esperando pela chegada da informação necessária para a continuidade do processamento. Em problemas mais difíceis de se paralelizar, fazer isso pode significar uma mudança completa dos algoritmos utilizados (Sterling et al., 1999).