Ana Isabel Rojão Lourenço Azevedo

INTRODUÇÃO AO ESTUDO DA PARALELIZAÇÃO DE ALGORITMOS DE PLANEAMENTO OPERACIONAL COM MÉTODOS GENÉTICOS

Tese de mestrado em Engenharia Electrotécnica e de Computadores na Área de Informática Industrial
apresentada à Faculdade de Engenharia da Universidade do Porto

Setembro, 1997

ERRATA

Pág. 1 - linha 13

Onde se lê resolver estes problemas, deve ler-se resolver problemas.

Pág. 2 - linha 26

Onde se lê Caixeiro Viajente, deve ler-se Caixeiro Viajante.

Pág. 58 - linha 1

Onde se lê Algoritmos Genéticos Canónicos, deve ler-se Algoritmo Genético Canónico.

Pág. 63 - figura 3.20

Onde se lê pte, deve ler-se pc.

Pág. 66 - figura 3.26

Onde se lê Óptimo (?), deve ler-se Óptimo.

Pág. 67 - tabela 3.2 - linha 4

Onde se lê 100, deve ler-se 1000.

Pág. 68 - tabela 3.3

Onde se lê Tamanho do Problema, deve ler-se Tamanho da População.

Resumo

Os Algoritmos Genéticos são técnicas de optimização, baseadas na selecção natural, que se têm vindo afirmar nos últimos anos. Existem já diversas aplicações a situações reais, com resultados bastante bons. Actualmente, com a popularização de pacotes de software para paralelização de aplicações, como o PVM, surge como natural a sua utilização na paralelização dos algoritmos genéticos. Existem várias modelos para a paralelização de algoritmos genéticos.

Neste trabalho apresenta-se um estudo do software PVM, utilizando-o, numa primeira fase, para a paralelização do algoritmo de ordenação MergeSort, o que nos serviu como um bom teste para ele. No que se seguiu, verificamos a existência de viabilidade na paralelização do algoritmo genético canónico, utilizando um sistema com quatro estações de trabalho (workstations). Para isso foi feito o estudo do Algoritmo Genético Canónico, não especializado, para a resolução de um problema do caixeiro viajante de 70 cidades, sendo este bastante complexo. Posteriormente paralelizamos o algoritmo, utilizando o modelo do algoritmo genético grosseiro. Realizamos testes alterando os valores dos vários parâmetros e verificamos a obtenção de melhores resultados com o aumento do número de ilhas e com migração. Verificamos ainda que as melhorias em relação ao algoritmo sequencial foram mais acentuadas com a utilização de populações pequenas - da ordem dos 12%; com a utilização de populações grandes as melhorias foram poucas - cerca de 2%. Quanto ao tempo de execução em paralelo, verificamos que o tempo necessário à obtenção de soluções análogas às do algoritmo sequencial, corresponde aproximadamente à divisão do tempo sequencial pelo número de processadores.

Verificamos que é útil a paralelização de algoritmos genéticos, no nosso sistema, utilizando o algoritmo genético grosseiro. As vantagens existem quer nos resultados obtidos quer no tempo necessário à obtenção de um determinado valor.

PALAVRAS-CHAVE: PVM
Algoritmos Genéticos
Algoritmos Genéticos Paralelos

Abstract

Genetic Algorithms are optimization techniques, based on natural selection, which have been growing in recent years. There are already several applications to real situations, with very good results. Nowadays, due to the spreading of software packages for parallel applications, such as PVM, their use in making genetic algorithms parallel becomes natural. There are several models for making genetic algorithms parallel.

In this work, we present a study of PVM software, using it, in a first stage, to make the MergeSort sort algorithm parallel, which was a good test. Afterwards, we checked the feasibility of making the canonical genetic algorithm parallel, by using a system of four workstations. For that purpose, we have studied the Canonical Genetic Algorithm, non-specialized, in solving a problem of the travelling salesman of 70 cities, which was extremely complex. After that, we have made the algorithm parallel, using the coarse grained genetic algorithm model. We have made some tests, changing the values of the several parameters, and we realized that we had better results with the increasing of the number of islands and with migration. We also verified that the improvements concerning the sequential algorithm were more striking when small populations (about 11%) were used; with large populations the improvements were few (about 2%). As to the time of execution in parallel, we realized that the necessary time to obtain solutions similar to those of the sequential algorithm is almost the same as the division of the sequential time by the number of processors.

We verified that making genetic algorithms parallel is useful in our system, using the coarse grained genetic algorithm. There are advantages both in the results obtained and in the time necessary to obtain a certain value.

KEY-WORDS: PVM
Genetic Algorithms
Parallel Genetic Algorithms

Résumé

Les Algorithmes Génetiques ce sont des techniques d'optimisation fondées à la sélection naturelle et que se sont affirmées depuis des années. Il y a déjá beaucoup d'applications rèélles avec des résultats très positifs. Actuellement, avec la généralisation des paquets de software pour la parallélisation d'applications, comme le P.V.M., son utilisation nous paraît naturelle dans la parallélisation des algorithmes génétiques.

Dans ce travail, je présente une étude du software P.V.M., l'utilisant, dans une première étape, dans la parallélisation de l'algorithme d'ordination «MergeSort», ce que nous a bien servi pour le tester. Ensuite, nous avons vérifié l'éxistence de la possibilité de la parallélisation de l'algorithme génétiques canonique, en utilisant une méthode avec quatre bureaux de travail (wokstations). Pour cela, nous avons fait l'étude de l'algorithme génétique canonique, non spécialisé, pour la résolution d'un problème très complexe de commis voyageur de 70 villes.

Plus tard, nous avons parallélisé l'algorithme génétique grossier.

Nous avons réalisé des tests changeant les valeurs de plusieurs paramètres et nous avons vérifié l'obtention de meilleurs résultats en augmentant le nombre d'îles et avec migration.

Nous avons encore vérifié que, par rapport à l'algorithme séquentiel, les améliorations se sont accentuées en utilisant des populations petites - vers 11%; en utilisant des grandes populations nous n'avons obtenu que 2% d'améliorations. En ce qui concerne le temps d'exécution en parallèle, nous avons vérifié que le temps nécessaire à l'obtention de solutions identiques à celles d'algorithme séquentiel, correspond à peu près à la division du temps séquentiel par le nombre de processeurs.

Nous avons constaté l'utilité de la parallélisation d'algorithmes génétiques dans notre système, utilisant l'algorithme génétique grossier. Les avantages existent soit dans les travaux obtenus, soit dans le temps nécessaire à l'obtention d'une certaine valeur.

MOT-CLEF: P.V.M.

Algorithmes Génétiques

Algorithmes Génétiques parallèles

Dedicado à minha família e
em memória da minha
madrinha e da minha avó.

Agradecimentos

Ào meu orientador, o Professor Doutor João Falcão e Cunha, pelo apoio incondicional que me prestou durante a realização deste trabalho;

Ao Professor Doutor Raúl Vidal pelas facilidades concedidas na utilização da biblioteca do INESC;

À Joana e à Cristina por todo o apoio que me deram e pela disponibilidade que sempre tiveram, mesmo quando o trabalho era muito;

A todos os que trabalham no CICA, pela ajuda que sempre me prestaram durante a realização deste trabalho;

À Alice, pela sua colaboração;

À Paula, pela revisão cuidada deste trabalho;

À minha família, por toda a disponibilidade que tiveram durante este período, pela paciência e por tudo o resto. Agradeço especialmente ao meu marido e aos meus filhos pelas horas que deveriam ter sido passadas com eles e não o puderam ser.

1. Introdução 1

1.1. Objectivos do nosso trabalho 2

1.2. Estrutura da tese 3

2. Paralelizar utilizando PVM 4

2.1. O PVM 5

2.1.1. Configurar o PVM 6

2.1.2. Algumas Considerações 7

2.1.2.1. Modelos de Programação 7

2.1.2.2. TID's 9

2.1.2.3. Trocando Mensagens 10

2.1.3. Como é constituído o PVM 11

2.1.3.1. O *daemon* 12

2.1.3.2. A consola 13

2.1.3.3. A biblioteca de funções 15

2.2. Uma Aplicacação PVM: paralelização do Algoritmo MergeSort 18

2.2.1. As limitações da paralelização utilizando PVM 21

2.3. Mais sobre PVM 24

2.3.1. Detecção de erros 24

2.3.2. Outros projectos baseados no PVM 25

2.3.3. E o futuro? 25

2.4 Outros Sistemas para Paralelização de Aplicações 26

2.4.1. O Linda 26

2.4.2. O Sistema P4 26

2.4.3. O Express 27
2.4.4. O MPI 27
2.5. Notas Finais 27

3. Algoritmos Genéticos 29
3.1. Os Algoritmos Genéticos: descrição e contextualização 31
3.1.1. Algoritmos Genéticos na optimização 33
3.1.2. Perspectiva Histórica 36
3.2. Como funciona um Algoritmo Genético 37
3.2.1. Avaliação/Adaptação 38
3.2.2. Operadores Genéticos 41
3.2.2.1. Selecção 41
3.2.2.2. Cruzamento 43
3.2.2.3. Mutação 44
3.2.2.4. A geração Seguinte 45
3.2.2.5. Fim da Evolução 45
3.2.3. Fundamentos Matemáticos dos Algoritmos Genéticos 46
3.2.3.1. Notações e considerações gerais 47
3.2.3.2. O Teorema dos Esquemas 49
3.2.3.3. Críticas ao Teorema das Esquemas 50
3.2.4. A Codificação 51
3.2.5. Parametrização de um Algoritmo Genético 51
3.3. Alguns Conceitos Avançados 55
3.3.1. Algoritmos Genéticos Híbridos 55
3.3.2. Programação Genética 56
3.3.3. Problemas baseados em Permutações 57
3.3.3.1. O Problema do Caixeiro Viajante 58
3.3.3.2. A Tabela de Inversões 60
3.3.3.3. Aplicação ao Problema do Caixeiro Viajante 62
3.4. Notas Finais 68

4. Algorimos Genéticos Paralelos 70

4.1. Abordagem tradicional à paralelização de Algoritmos Genéticos 71

4.2. Aproximação à paralelização de Algoritmos Genéticos por decomposição 74

4.2.1. Algoritmos Genéticos Grosseiro 74

4.2.2. Algoritmos Genéticos Finos 76

4.2.3. Algoritmos Genéticos Paralelos Híbridos 77

4.3. Aplicação ao Problema do Caixeiro Viajante 78

4.3.1. Valores obtidos pelo algoritmo.........................78

4.3.2. Tempos de execução/ Comunicação 85

4.4. Notas finais 88

5. Conclusões e Perspectivas futuras 90

Bibliografia 92

Anexo A Matrizes dos Problemas do Caixeiro Viajante utilizados 102

Anexo B Códigos da implementação do algoritmo Genético Grosseiro 105

Lista de figuras

Figura 2.1. Um sistema PVM 6
Figura 2.2. Modelo Mestre_Escravo 8
Figura 2.3. Modelo em Árvore 8
Figura 2.4. Modelo Híbrido 9
Figura 2.5. Componentes de um sistema PVM 12
Figura 2.6. Algoritmo MergeSort 19
Figura 2.7. Algoritmo MergeSort Paralelo 19
Figura 2.8. Modelo de implementação do algoritmo Merge-Paralelo 19

Figura 3.1. Fluxograma de um Algoritmo Genético canónico 31
Figura 3.2. Classes de Técnicas de Pesquisa 34
Figura 3.3. Gráfico da Função a Optimizar 37
Figura 3.4. Codificação contida num cromossoma 38
Figura 3.5. O aparecimento de uma nova geração 40
Figura 3.6. Distribuição dos pontos na primeira geração 40
Figura 3.7. Distribuição dos pontos na oitava geração 41
Figura 3.8. Fluxograma do algoritmo de selecção através da roleta viciada...... 42
Figura 3.9. Fluxograma do algoritmo de um torneio estocástico de ordem 2 42
Figura 3.10. Operador de cruzamento com um ponto 43
Figura 3.11. Operadores de Mutação 44
Figura 3.12. Evolução do Algoritmo Genético Tradicional 46
Figura 3.13. Visualização dos Esquemas como Hiperplanos no espaço tridimensional 48

Figura 3.14. Algoritmo Genético: variação de p_c;

p_m=0,01; p_{te}=0,8;pop=50 53

Figura 3.15. Algoritmo Genético: variação de p_m;

p_c=0,8; p_{te}=0,8;pop=50........... 53

Figura 3.16. Algoritmo Genético: variação de p_{te};

p_c=0,8; p_m=0,01;pop=50..........54

Figura 3.17. Algoritmo Genético: variação de *pop*;

p_c=0,8; p_m=0,01;p_{te}=0,8........... 54

Figura 3.18. Algoritmo para converter uma tabela de inversões na permutação correspondente..61

Figura 3.19. Variação do parâmetro p_c para o Problema do Caixeiro Viajante de tamanho 17; p_m=0,01; p_{te}=0,8;pop=500........................ 63

Figura 3.20. Variação do parâmetro p_c para o Problema do Caixeiro Viajante de tamanho 70; p_m=0,01; p_{te}=0,8;pop=4000........................ 63

Figura 3.21. Variação do parâmetro p_m para o Problema do Caixeiro Viajante de tamanho 17; p_c=0,8; p_{te}=0,8;pop=500 64

Figura 3.22. Variação do parâmetro p_m para o Problema do Caixeiro Viajante de tamanho 70; p_c=0,8; p_{te}=0,8;pop=4000.........................64

Figura 3.23 Variação do parâmetro p_{te} para o Problema do Caixeiro Viajante de tamanho 17; p_c=0,8; p_m=0,01;pop=500 65

Figura 3.24. Variação do parâmetro p_{te} para o Problema do Caixeiro Viajante de tamanho 70; p_c=0,8; p_m=0,01;pop=4000 65

Figura 3.25. Variação do parâmetro pop para o Problema do Caixeiro Viajante de tamanho 17; p_c=0,8; p_m=0,01;p_{te}=0,8.......................... 66

Figura 3.26. Variação do parâmetro pop para o Problema do Caixeiro Viajante de tamanho 70; p_c=0,8; p_m=0,01;p_{te}=0,8.......................... 66

Figura 4.1. Algoritmo Genético Paralelo................................. 71

Figura 4.2. Algoritmo Genético Assíncrono...............................72

Figura 4.3. Modelo de um Algoritmo Genético Paralelo..................... 73

Figura 4.4. Representação de um modelo de um Algoritmo Genético Grosseiro. 75

Figura 4.5. Modelo de um Algoritmo Genético Fino. 76

Figura 4.6. Comparação dos valores de 8 ilhas de tamanho 32000 cromossomas. 84

Figura 4.7. Comparação dos valores de 4 ilhas de tamanho 64000 cromossomas. 85

Lista de tabelas

Tabela 2.1. Rotinas PVM para Controlo de processos 15

Tabela 2.2. Rotinas PVM para Gestão de Informação 16

Tabela 2.3. Rotinas PVM para Configuração Dinâmica 16

Tabela 2.4. Rotinas PVM para Sinalização 16

Tabela 2.5. Rotinas PVM para Estabelecer e Mostrar Opções 16

Tabela 2.6. Rotinas PVM de Passagem de Mensagens 17

Tabela 2.7. Rotinas PVM para Grupos de processos 18

Tabela 2.8. Características do hardware utilizado 20

Tabela 2.9. Tempos de execução do Algoritmo MergeSort 21

Tabela 2.10. Tempos reais de execução do Algoritmo Merge-Paralelo 24

Tabela 3.1. Informação sobre o Problema do Caixeiro Viajante 59

Tabela 3.2. Tempos de execução do Algoritmo 67

Tabela 3.3. Qualidade das soluções obtidas 68

Tabela 4.1. Soluções obtidas pelo algoritmo; **TS**=2000; **IM**=100 79

Tabela 4.2. Soluções obtidas pelo algoritmo; **TS**=2000; **IM**=200 79

Tabela 4.3. Soluções obtidas pelo algoritmo; **TS**=2000; **IM**=500 79

Tabela 4.4. Soluções obtidas pelo algoritmo; **TS**=4000; **IM**=100 79

Tabela 4.5. Soluções obtidas pelo algoritmo; **TS**=4000; **IM**=200 80

Tabela 4.6. Soluções obtidas pelo algoritmo; **TS**=4000; **IM**=500 80

Tabela 4.7. Soluções obtidas pelo algoritmo; **TS**=8000; **IM**=100 80

Tabela 4.8. Soluções obtidas pelo algoritmo; **TS**=8000; **IM**=200 80

Tabela 4.9. Soluções obtidas pelo algoritmo; **TS**=8000; **IM**=500 81
Tabela 4.10. Soluções obtidas pelo algoritmo; **TS**=16000; **IM**=100 81
Tabela 4.11. Soluções obtidas pelo algoritmo; **TS**=16000; **IM**=200 81
Tabela 4.12. Soluções obtidas pelo algoritmo; **TS**=16000; **IM**=500 81
Tabela 4.13. Comparação das populações panmixias com ilhas do mesmo tamanho, isoladas 82
Tabela 4.14. Comparação das populações panmixias com ilhas do mesmo tamanho, com migrantes 81
Tabela 4.15. Tempos de comunicação do Algoritmo Genético Paralelo, calculados com o modelo teórico...........................86

1 Introdução

Charles Darwin e Alfred Russel Wallace[1], há pouco mais de um século, sublinharam o facto de a natureza ser prolífera e de nascerem muito mais animais e plantas do que aqueles que têm possibilidade de sobreviver. O próprio ambiente selecciona as variedades mais adaptadas à sobrevivência, o que resulta numa série de lentas transformações de uma forma de vida para outra, e, consequentemente, dá origem a novas espécies. As mutações, alterações súbitas na hereditariedade, transmitem-se à descendência sendo elas a matéria prima da própria evolução. Actualmente utilizam-se os Algoritmos Genéticos como métodos de pesquisa eficientes inspirados nos processos de selecção natural e em populações genéticas, tendo sido já aplicados com sucesso nas mais diversas área, desde as ciências da computação às ciências económicas.

Nos últimos anos, os algoritmos genéticos paralelos têm sido utilizados para resolver estes problemas difíceis. Problemas difíceis necessitam de populações grandes o que se traduz directamente em elevados custos computacionais. Os primeiros estudos na área, realizados por Grefenstette, visaram a redução do tempo de execução necessário para a obtenção de uma solução satisfatória, tendo sido alcançado esse objectivo através da sua implementação em diversas arquitecturas. Verificou-se mesmo, em alguns casos, a obtenção de melhores soluções em paralelo [Cantú-Paz95]. Actualmente, tem-se vindo a assistir a um crescimento na pesquisa relacionada com os algoritmos genéticos paralelos, tendo surgido várias variantes do modelo inicial, com bons resultados.

[1] *In* Enciclopédia Britânica (Tradução da autora).

Uma rede de computadores pode ser vista como uma máquina paralela potencial, que oferece um grande poder computacional. Este pode ser aproveitado, através da paralelização, de forma a que as suas capacidades estejam disponíveis para as várias aplicações, gerindo os recursos que num determinado momento se encontram disponíveis. Com os recursos disponíveis no CICA[2] e com o suporte do software Parallel Virtual Machine (PVM) [Geist94a] é possível a execução de aplicações paralelas. O PVM é um pacote (package) de software que tem como principal objectivo a utilização de uma rede heterogénea de computadores de forma a que, para o utilizador, tenha a aparência de uma só máquina. A grande popularidade do PVM deve-se a vários factores, nomeadamente à sua simplicidade, à existência de um modelo de computação natural e geral, à robustez da implementação e à facilidade na utilização.

1.1. Objectivos do nosso trabalho

No nosso trabalho pretendemos averiguar a existência, ou não, de vantagens na paralelização de algoritmos genéticos utilizando os recursos disponíveis no CICA. Vamos analisar vários modelos de paralelização, segundo critérios tais como a adaptação do modelo aos recursos disponíveis e o tempo de execução. Quanto ao modelo de paralelização, vamos averiguar se é mais vantajoso utilizar uma só população e actuar sobre ela em paralelo, ou ter várias sub-populações sobre as quais o algoritmo vai actuar de forma independente, trocando periodicamente informação. Pretendemos ainda determinar quais os melhores parâmetros para o algoritmo genético paralelo para o modelo escolhido, nomeadamente no que diz respeito ao tamanho das populações.

Para a realização dos testes utilizámos o Problema do Caixeiro Viajante (TSP=Traveling Salesman Problem). O Problema do Caixeiro Viajente [Lawler85] é um dos problemas clássicos de optimização combinatória. Um caixeiro viajante dedica-se a percorrer várias cidades comprando e vendendo produtos. O seu interesse é o de percorrer um certo número de cidades, de forma a utilizar o mais curto trajecto total

[2] Centro de Informática Doutor Correia de Araújo, Faculdade de Engenharia da Universidade do Porto

possível. De forma mais concreta, o problema consiste em encontrar um caminho no qual, partindo de uma cidade determinada, se percorram todas as cidades restantes, voltando à cidade de partida e minimizando a distância total percorrida. Considera-se o Problema do Caixeiro Viajante, como representativo dos problemas pertencentes ao domínio daqueles cuja resolução é de extrema complexidade [Lawler85], Cap. 3. Por este motivo, consideramo-lo como um bom desafio para os algoritmos genéticos.

1.2. Estrutura da Tese

Vamos iniciar o nosso trabalho fazendo uma breve descrição, no segundo capítulo, do software PVM. Aí será, apresentada a implementação do algoritmo de ordenação MergeSort, com o qual pretendemos, devido ao seu elevado número de tarefas, testar o limite dos recursos do sistema PVM. Será feita ainda uma breve referência a outros ambientes de paralelização, além do PVM.

No Capítulo três, faremos uma introdução aos algoritmos genéticos. Estudaremos primeiro o caso de uma função de optimização simples, à medida que for sendo desenvolvido o tema. Por fim, aplicaremos o algoritmo ao Problema do Caixeiro Viajante tomado nas suas várias dimensões.

No quarto capítulo, será abordado o tema dos algoritmos genéticos paralelos. Veremos os modelos de paralelização mais utilizados e, considerando as suas características, veremos qual poderá ser o mais adaptado ao nosso sistema. Serão realizados testes considerados pertinentes para que se possam atingir os objectivos propostos.

Por fim, no quinto capítulo, serão retiradas as conclusões finais do nosso trabalho e serão apontadas algumas directivas para futuros trabalhos.

2 Paralelizar utilizando o PVM

Nos últimos anos, tem-se verificado um interesse crescente na paralelização de aplicações[1] e subsequente desenvolvimento das mesmas. Este facto deve-se principalmente a dois factores:

- o aparecimento dos computadores massivamente paralelos (MPP's);
- o grande desenvolvimento e, digamos, vulgarização de sistemas distribuídos.

Os primeiros apresentam grande capacidade de cálculo, pelo que são muito utilizados em cálculos científicos de grande exigência (por exemplo, na previsão do tempo), cálculos esses que dificilmente poderiam ser executados num computador sequencial vulgar; são, no entanto, extremamente caros.

Os cientistas começaram a aperceber-se que os sistemas distribuídos (o segundo factor) poderiam ser aproveitados de forma a apresentarem uma capacidade idêntica aos computadores massivamente paralelos. Porém seria necessário que os dados ou as várias tarefas fossem paralelizadas, de forma a aproveitar da melhor forma os recursos disponíveis numa rede heterogénea de computadores. Isto tem particular interesse, pois os sistemas distribuídos apresentam várias vantagens, nomeadamente no que diz respeito, por exemplo, a custos, acessibilidade e à facilidade de desenvolvimento de ferramentas que permitam fazer a detecção de erros.

Surge assim a necessidade de uma aproximação sistemática, que permita implementar esta ideia de utilização de sistemas distribuídos para paralelizar aplicações. Geist et all ([Geist94a]) desenvolveram um pacote (package) de software que permite que os computadores de uma rede heterogénea tenham, para o utilizador, a

[1] Para aprofundar ideias relativas a paralelização, ver [Lester93] e [Harel92]

aparência de uma só máquina, trabalhando em paralelo (figura 2.1.) - o PVM (Parallel Virtual Machine), que vamos apresentar no decorrer deste capítulo da seguinte forma:.

No ponto 2.1. vamos fazer uma breve descrição do PVM. No ponto 2.2. é feita a paralelização do algoritmo de ordenação MergeSort. Esse algoritmo é implementado utilizando PVM. No ponto 2.3. veremos como pode ser feita a detecção de erros de uma aplicação PVM e quais as futuras linhas de orientação do sistema PVM. No ponto 2.4. faremos uma breve incursão através de outros sistemas para paralelização de aplicações.

2.1. O PVM

O projecto PVM começou em 1989 no Oak Ridge National Laboratory, surgindo o protótipo PVM 1.0, que nunca chegou a ser difundido. Em Março de 1991 surge, na universidade do Tenesse, a versão 2.0, que veio a sofrer vários melhoramentos (PVM 2.1-2.4). Foi feita uma restruturação total, surgindo a versão 3.0 em Fevereiro de 1993 tendo esta também já sofrido alguns melhoramentos. Neste trabalho vamos ter por base a versão 3.3 do PVM [Geist94a], que se encontra disponível no CICA.

O PVM é um pacote (package) de software, de distribuição livre, que se encontra grandemente divulgado, quer devido à facilidade da sua obtenção, quer à sua grande qualidade e facilidade de utilização.

Um dos grandes objectivos do PVM é a utilização de todas as potencialidades de uma rede heterogénea de computadores, de forma a parecer ao utilizador uma só máquina (figura 2.1.). A sua heterogeneidade está presente a três níveis:

- ao nível das aplicações, pois as tarefas podem ser colocadas no processador mais adequado;
- ao nível das máquinas, pois podem ser utilizados computadores com diferentes formatos de dados, diferentes arquitecturas e diferentes sistemas operativos;
- ao nível das redes, pois pode abarcar vários tipos de redes.

A última versão de PVM e documentação com ele relacionada está sempre disponível através do *netlib*, na Internet. Qualquer utilizador pode aceder aos ficheiros

de PVM utilizando o endereço http://www.netlib.org/pvm3/index.html. Para mais pormenores consultar [Geist94a] pp. 5-7, onde se encontra a forma de obter e instalar o PVM.

2.1.1. Configurar o PVM

Cada utilizador do sistema PVM tem acesso a uma máquina virtual, que é constituída por várias máquinas UNIX.

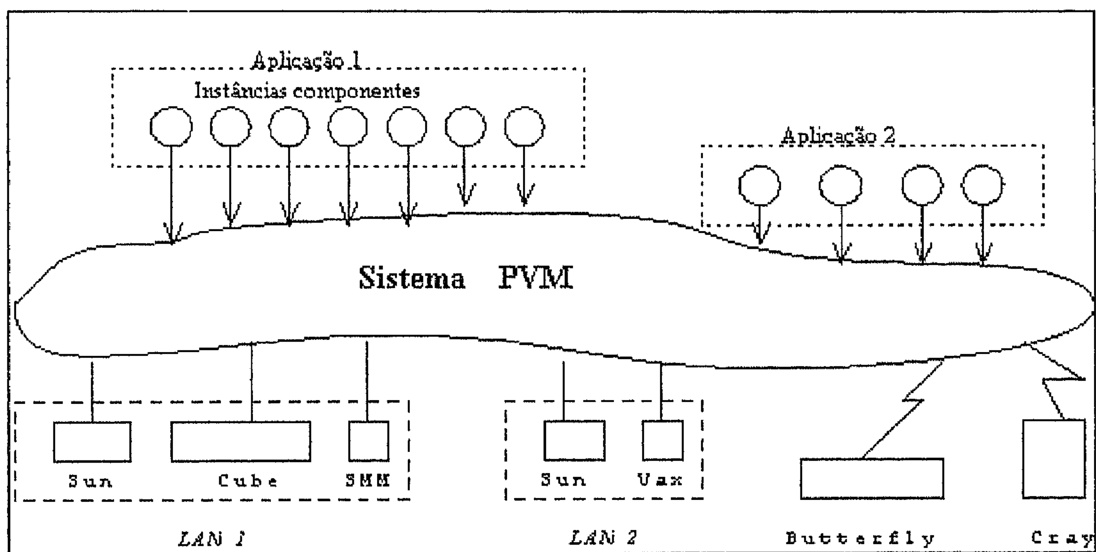


Figura 2.1. Um sistema PVM [2]

A cada uma das máquinas que o PVM vai utilizar chama-se máquina hospedeira (host). Apresenta-se a possibilidade de utilizar vários tipos de máquinas hospedeiras, com arquitecturas distintas. O tipo de arquitectura utilizado, assim como outros parâmetros, devem ser estabelecidos pelo utilizador no seu ambiente, depois de instalado o PVM.

Vejamos, de seguida, quais são os parâmetros a ter em consideração pelo utilizador:

- A variável PVM_ROOT, que indica ao sistema onde está o directório contendo os ficheiros do sistema PVM, já instalado, normalmente no

[2] Adaptado de [Geist91]

directório pvm3. Deverá acrescentar ao ficheiro ***.cshrc*** a linha ***setenv PVM_ROOT /usr/local/pvm3***

- A variável PVM_ARCH, que diz qual a arquitectura do hospedeiro e, por conseguinte, quais os executáveis que o sistema deve escolher do directório PVM_ROOT. Para isso deverá acrescentar ao ficheiro ***.cshrc*** a linha: ***source $PVM_ROOT/lib/cschr.stub.*** Desta forma, o sistema determina automaticamente a variável. Há, porém, vários tipos de arquitecturas suportadas pelo PVM3 que podem ser encontradas em [Geist94b] pp. 21.
- O ficheiro ***.rhosts*** deve ser criado contendo o nome dos outros hospedeiros que deseja utilizar, que será utilizado pelo UNIX.

```
tom.fe.up.pt> more .rhosts
crazy.fe.up.pt
jerry.fe.up.pt
riff.fe.up.pt
```

- Um ficheiro de configuração ***hostfile*** deverá, também, ser criado contendo todos os hospedeiros que vai utilizar[3], que será utilizado pelo PVM.

```
tom.fe.up.pt> more hostfile
tom.fe.up.pt
crazy.fe.up.pt
jerry.fe.up.pt
riff.fe.up.pt
```

- O directório ***pvm3/bin/ALPHA***, depois de criado deverá conter todos os executáveis, criados na altura da compilação.

2.1.2. Algumas considerações

2.1.2.1. Modelos de programação

O PVM baseia-se num dos paradigmas da programação paralela mais utilizados que é o da programação por passagem de mensagens. Este consiste, basicamente, na existência de vários processos trabalhando em simultâneo, cada um ocupando o seu espaço de memória próprio, e que comunicam entre si, trocando mensagens, sempre que necessário.

[3] Existem várias opções que pode utilizar ao criar a hostfile. Consultar [Geist94b] pp. 29-32

O PVM considera três modelos de programação [Geist94b]:

1. O modelo multidão (crowd) representado na figura 2.2. Este modelo consiste numa colecção de processos fortemente relacionados, que normalmente executam o mesmo código sobre dados diferentes, trocando-os periodicamente. Este modelo pode ser subdividido em:
   - mestre-escravo - existe um processo separado, o mestre, que controla a geração, iniciação, recolha e amostragem dos resultados. Os programas escravos executam os cálculos reais;
   - nó-para-nó (node-to-node) - são executadas múltiplas instâncias do mesmo programa, com um dos processos tomando as responsabilidades não computacionais, assim como as responsabilidades computacionais;

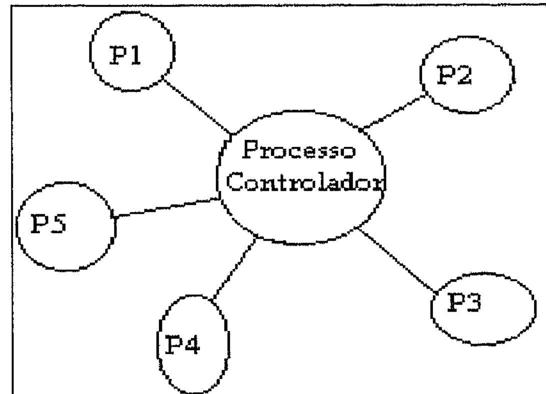


Figura 2.2. Modelo Multidão (Variante Mestre -Escravo)

2. O modelo em árvore, representado na figura 2.3. - neste modelo os processos são gerados dinamicamente, conforme os cálculos crescem, existindo entre eles uma relação pai-filho.

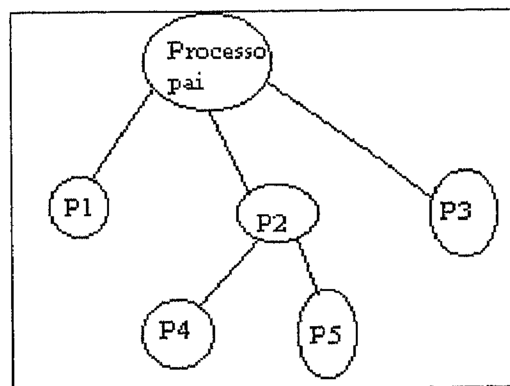


Figura 2.3. Modelo em Árvore

3. O modelo híbrido representado na figura 2.4. - este modelo é uma combinação dos dois modelos, encontrando-se um esquema possível representado na figura sendo neste caso os processos gerados de forma arbitrária.

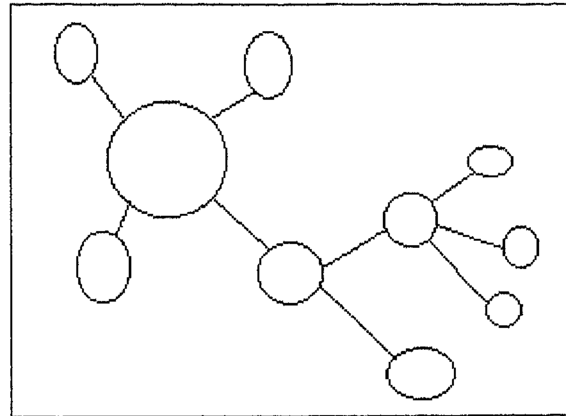


Figura 2.4. Modelo Híbrido

O primeiro modelo utiliza-se quando pretendemos executar a mesma tarefa ou várias tarefas, diversas vezes, sendo as várias tarefas independentes umas das outras. O tempo de execução é sempre determinado pela tarefa mais lenta. O segundo modelo é utilizado quando uma determinada tarefa (pai) depende, de alguma forma, de uma ou várias outras (filhos). Neste modelo o "pai" só pode terminar o seu trabalho, quando todos os "filhos" o tiverem feito, pelo que se podem verificar grandes períodos de espera. Assim sendo, este modelo só deverá ser utilizado quando uma tarefa (pai) dependa realmente do trabalho realizado por outra(s) (filho(s)). Caso contrário, deverá ser utilizado o primeiro modelo. Na maior parte das aplicações reais, existem os dois tipos de tarefas referidas, pelo que teremos presente o modelo híbrido.

2.1.2.2. TID's

O componente básico endereçável mais pequeno de uma aplicação PVM é uma tarefa. Cada tarefa de uma aplicação PVM é identificada pelo sistema, através da utilização de um identificador de tarefas: o TID (task identifier). Existe ainda a possibilidade de criar grupos de tarefas. Neste caso cada uma das tarefas do grupo tem ainda um identificador no grupo: GID (group identifier). O TID é sobretudo necessário para identificar tarefas aquando do envio e recepção de mensagens.

2.1.2.3. Trocando mensagens

O PVM consiste, basicamente, num conjunto de tarefas que comunicam entre si trocando mensagens. Aqui, tal como em todos os outros aspectos, segue-se a filosofia de manter a interface o mais simples possível com o utilizador, deixando para o sistema a árdua tarefa de gerir todos os recursos da forma mais eficiente. Assim, o PVM fornece ao utilizador as primitivas básicas para enviar e receber mensagens.

Enviar uma mensagem no sistema PVM é um processo de três passos. O primeiro passo consiste na criação de um depósito (buffer). De seguida, os dados a enviar são empacotados no depósito para finalmente, serem enviados para um processo ou para um grupo de processos. Receber uma mensagem é um processo de dois passos. O primeiro consiste na recepção da mensagem e o segundo no desempacotamento da mesma. Todos estes passos são executados através de várias rotinas PVM apropriadas, as quais serão referidas no ponto 2.1.3.3.

Centremo-nos, agora, no modelo de comunicação do PVM. Este modelo apresenta várias características:

- supõe que qualquer tarefa pode enviar uma mensagem para qualquer outra tarefa;
- supõe que não há limite para o tamanho da mensagem, partindo do princípio de que há sempre memória disponível;
- faz a atribuição de espaço para o depósito de mensagens de forma dinâmica, e assim, o tamanho das mensagens que podem chegar ao mesmo tempo a um hospedeiro é apenas limitado pelo espaço de memória disponível;
- fornece o envio assíncrono bloqueante das mensagens, a recepção assíncrona bloqueante e não-bloqueante das mesmas; nas operações de comunicação em modo assíncrono, não é esperado que o receptor envie a informação de que recebeu a mensagem; nas comunicações bloqueantes, apenas há retorno quando a rotina encarregada da comunicação termina por completo o seu trabalho; nas comunicações não-bloqueantes, há retorno imediatamente após a chamada da rotina;

- garante que a ordem das mensagens é preservada;
- permite a utilização de um tipo especial de mensagem, associada a uma etiqueta (tag), sendo que o PVM retorna a primeira mensagem com essa etiqueta.

É de referir ainda, a possibilidade da existência de comunicação colectiva, ou seja, envolvendo grupos de processos. Existem três tipos de operações colectivas ([MacDonald94]):

- barreira: esta operação sincroniza os processos, bloqueando o processo que a chama até que todos os processos do grupo tenham chamado a rotina *barreira* (barrier);
- difusão: nesta operação, um processo envia uma mensagem para um grupo de processos com uma única operação;
- aglutinação: este tipo de operações caracteriza-se por tomar itens de dados de vários processos, e os reduzir a um único item, o qual fica normalmente disponível a todos os participantes do grupo (por exemplo, tendo vários valores espalhados pelos processos, podemos seleccionar o maior deles através de uma única operação de aglutinação).

2.1.3. Como é constituído o PVM

Todas as aplicações PVM são escritas pelo utilizador como programas sequenciais vulgares, sendo a paralelização conseguida através da introdução de funções adequadas da biblioteca do PVM. O ambiente PVM permite a utilização das linguagens de programação FORTRAN, C e C++.

Uma aplicação PVM é constituída por várias tarefas, tendo cada uma delas a responsabilidade de realizar uma parte do trabalho total. Quando várias tarefas realizam o seu trabalho simultaneamente, dizemos que existe uma paralelização das mesmas. O paralelismo pode ser conseguido de duas formas [Geist94a]:

- paralelismo funcional: cada tarefa realiza um trabalho diferente. Por exemplo, uma responsabiliza-se pela entrada dos dados, outra pela realização dos cálculos e outra pela saída dos resultados;

- paralelismo dos dados: verifica-se, neste caso a existência de várias instâncias da mesma tarefa, mas cada uma delas realiza o seu trabalho, apenas sobre uma parte dos dados.

Ao nível da sua implementação, o PVM é constituído por duas partes (figura 2.5.):

- o "daemon";
- uma biblioteca de funções.

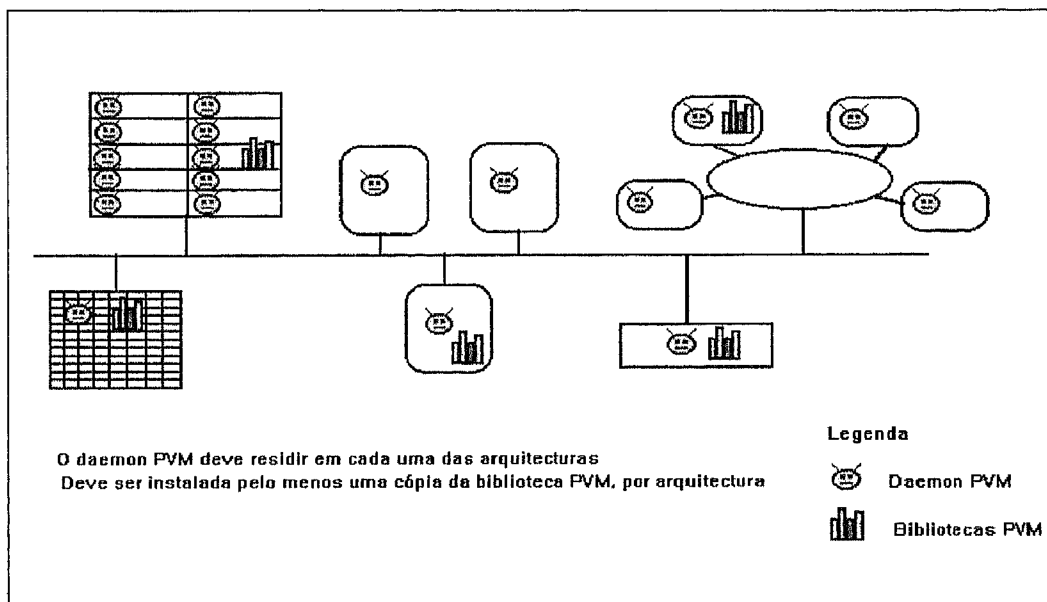


Figura 2.5. Componentes de um sistema PVM[4]

2.1.3.1. O *daemon*

Uma das formas de inicializar o sistema PVM é escrevendo *pvm hostfile* na prompt de um dos hospedeiros. Nesse momento, é criado um "*daemon*" *pvmd3* que vai fazer a gestão das tarefas e da comunicação entre elas. Mais especificamente, o daemon *pvmd3*:

- corre em cada um dos hospedeiros da máquina virtual;
- funciona como ponto de contacto entre os vários hospedeiros;
- faz a autenticação das várias tarefas;

[4] Adaptado da página web http://aixport.sut.ac.jp/advsys/mhppc/pvm/daemonlib.gif

- executa os processos;
- detecta as falhas;
- sendo sobretudo um encaminhador de mensagens, também envia e sincroniza as mesmas.

Os "*daemons's*" são propriedade de cada utilizador, não havendo qualquer interacção entre "*daemon's*" de utilizadores diferentes.

Existe um "*daemon*" (mestre) que vai inicializar todos os outros (escravos).

2.1.3.2. A consola

A inicialização do sistema PVM faz-se, no geral, utilizando o comando *pvm [-n<hostname>] [hostfile]*[5]. Nesse momento, o daemon *pvmd3* inicia a consola. Esta responde com a prompt:

```
tom.fe.up.pt> pvm hostfile
pvm>
```

A consola pvm tem várias funções, nomeadamente, configurar a máquina virtual, iniciar ou parar as várias tarefas e receber informações e mensagens de erro. Pode-se iniciar e parar a consola múltiplas vezes, em qualquer dos hospedeiros onde corre o PVM.

Existem vários comandos aceites pela consola, podendo estes ser introduzidos através do teclado (standard input). Podem-se encontrar detalhes sobre esses comandos em [Geist94a] pp.7-9. No que se segue, referiremos apenas alguns dos mais utilizados, ou que consideramos de algum interesse.

- conf : lista a configuração da máquina virtual

```
pvm> conf
4 hosts, 1 data format
             HOST       DTID     ARCH    SPEED
     tom.fe.up.pt      40000    ALPHA     1000
   crazy.fe.up.pt      80000    ALPHA     1000
   jerry.fe.up.pt      c0000    ALPHA     1000
    riff.fe.up.pt     100000    ALPHA     1000
```

[5] a opção -n é útil para especificar um nome alternativo para o pvmd mestre.

- delete: retira hospedeiros à máquina virtual

```
pvm> delete crazy.fe.up.pt
1 successful
                      HOST   STATUS
            crazy.fe.up.pt   deleted
```

- add : adiciona hospedeiros à máquina virtual

```
pvm> add crazy.fe.up.pt
1 successful
                      HOST     DTID
            crazy.fe.up.pt   180000
```

- ps -a : fornece uma listagem de todas as tarefas actuais da máquina virtual

```
pvm> ps -a
             HOST      TID   FLAG 0x COMMAND
jerry.fe.up.pt           0      8/a -
jerry.fe.up.pt       c000b    6/c,f ordenar_paralelo20
jerry.fe.up.pt       c000c    6/c,f ordenar_paralelo20
jerry.fe.up.pt       c000d    6/c,f ordenar_paralelo20
jerry.fe.up.pt       c000e    6/c,f ordenar_paralelo20
jerry.fe.up.pt       c000f    6/c,f ordenar_paralelo20
jerry.fe.up.pt       c0010    6/c,f ordenar_paralelo20
jerry.fe.up.pt       c0011    6/c,f ordenar_paralelo20
 riff.fe.up.pt           0      8/a -
 riff.fe.up.pt      10000b    6/c,f ordenar_paralelo20
 riff.fe.up.pt      10000c    6/c,f ordenar_paralelo20
 riff.fe.up.pt      10000d    6/c,f ordenar_paralelo20
 riff.fe.up.pt      10000e    6/c,f ordenar_paralelo20
 riff.fe.up.pt      10000f    6/c,f ordenar_paralelo20
crazy.fe.up.pt       8000b    6/c,f ordenar_paralelo20
crazy.fe.up.pt       8000c    6/c,f ordenar_paralelo20
crazy.fe.up.pt       8000d    6/c,f ordenar_paralelo20
crazy.fe.up.pt       8000f    6/c,f ordenar_paralelo20
crazy.fe.up.pt       80010 16/o,c,f ordenar_paralelo20
crazy.fe.up.pt       80011      2/f ordenar_paralelo20
  tom.fe.up.pt       4000d      4/c -
  tom.fe.up.pt       4000e    6/c,f ordenar_paralelo20
  tom.fe.up.pt       4000f    6/c,f ordenar_paralelo20
  tom.fe.up.pt       40010    6/c,f ordenar_paralelo20
  tom.fe.up.pt       40011 16/o,c,f ordenar_paralelo20
  tom.fe.up.pt       40012    6/c,f ordenar_paralelo20
```

- quit : sai da consola, deixando todos os *daemons* e todas as tarefas a correr. Se voltarmos a escrever pvm voltamos à consola

```
pvm> quit

pvmd still running.
tom.fe.up.pt> pvm
pvmd already running.
pvm>
```

- halt : "mata" todos os processos, incluindo a consola e fecha o PVM

```
pvm> halt
tom.fe.up.pt>
```

O PVM suporta a utilização de uma multiplicidade de consolas. É possível correr uma consola em qualquer hospedeiro de uma máquina virtual existente e também iniciar uma consola no meio de uma aplicação PVM para verificar os seus progressos.

2.1.3.3. A Biblioteca de Funções

A biblioteca do PVM está organizada pelas funções das rotinas que a constituem. Todas as rotinas estão escritas em C e estão contidas na biblioteca **libpvm3**. As aplicações em C++ podem utilizar as mesmas rotinas. As aplicações em Fortran podem chamar esta biblioteca através da interface, fornecida com a fonte de PVM, **libfpvm3**. As funções relativas a processos de grupo dinâmicos são construídas no topo do núcleo das rotinas PVM, estando contidas na biblioteca **libgpvm3**.

Vamos apenas descrever as rotinas da biblioteca **libpvm3** e **libgpvm**, pois será esse o âmbito do nosso trabalho. De seguida, referir-nos-emos às rotinas que consideramos mais importantes, agrupadas pelas suas funções e, suncitamente, tentaremos descrever o seu desempenho. Para mais informações deverá ser consultado [Geist94b], capítulo 5.

Controlo de Processos

NOME	DESEMPENHO
pvm_mytid	retorna o *tid* deste processo
pvm_exit	diz ao *pvmd* local que este processo vai deixar o PVM
pvm_spawn	gera novas tarefas
pvm_kill	termina um processo PVM

Tabela 2.1. Rotinas PVM para controlo de processos

Informação

NOME	DESEMPENHO
pvm_parent	dá-nos o *tid* do processo que gerou esta tarefa
pvm_tidtohost	retorna o *tid* do hospedeiro no qual a tarefa especificada corre
pvm_config	retorna informação sobre o estado actual da máquina virtual
pvm_tasks	retorna informação sobre as tarefas a correr na máquina virtual

Tabela 2.2. Rotinas PVM para gestão de informação

Configuração Dinâmica

NOME	DESEMPENHO
pvm_addhosts	adiciona um ou mais hospedeiros à máquina virtual
pvm_delhosts	apaga um ou mais hospedeiros da máquina virtual

Tabela 2.3. Rotinas PVM de configuração dinâmica

Sinalização

NOME	DESEMPENHO
pvm_sendsig	envia um sinal a outro processo
pvm_notify	solicita notificações de eventos PVM tais como falhas de hospedeiros

Tabela 2.4. Rotinas PVM para sinalização

Opções

NOME	DESEMPENHO
pvm_setopt	estabelece várias opções da *libpvm*
pvm_getopt	mostra várias opções da *libpvm*

Tabela 2.5. Rotinas PVM para estabelecer e mostrar opções

Passagem de Mensagens

	NOME	DESEMPENHO
Manipulação de depósitos	**pvm_initsend**	limpa o depósito de envio e especifica o tipo de codificação
	pvm_mkbuf	cria um novo depósito de mensagens
	pvm_freebuf	disponibiliza um depósito de mensagens
	pvm_getsbuf	retorna o identificador do depósito de envio activo
	pvm_getrbuf	retorna o identificador de depósito de recepção activo
	pvm_setsbuf	muda o depósito de recepção activo e guarda o anterior
	pvm_setrbuf	muda o depósito de envio activo
Empacotamento e desempacotamento de dados	**pvm_pk***	carrega o depósito de mensagens activo, com *arrays* do tipo de dados descrito
	pvm_upk*	descarrega o depósito de mensagens activo, de *arrays* do tipo de dados descrito
Enviar e Receber Dados	**pvm_send**	envia os dados do depósito de mensagens activo
	pvm_mcast	distribui os dados do depósito de mensagens activo para um conjunto de tarefas
	pvm_psend	empacota e envia os dados numa só chamada
	pvm_recv	recebe uma mensagem
	pvm_nrecv	recepção não bloqueante
	pvm_probe	verifica se uma mensagem chegou
	pvm_trecv	recepção com tempo limitado
	pvm_precv	recebe uma mensagem directamente do depósito

Tabela 2.6. Rotinas PVM de passagem de mensagens

Grupos de Processos

NOME	DESEMPENHO
pvm_joingroup	inicia o processo que a chama num dado grupo
pvm_lvgroup	retira o processo que a chama dum dado grupo
pvm_gettid	retorna o *tid* do processo identificado pelo nome de um grupo e por uma instância
pvm_getinst	retorna o número de uma instância num grupo de processos PVM
pvm_gsize	retorna o número de processos actualmente num dado grupo
pvm_barrier	bloqueia o processo que a chama até que todos os processos num grupo a tenham chamado
pvm_bcast	difunde os dados do depósito de mensagens activo
pvm_reduce	executa uma operação de redução em todos os membros de um grupo especificado

Tabela 2.7. Rotinas PVM para grupos de processos

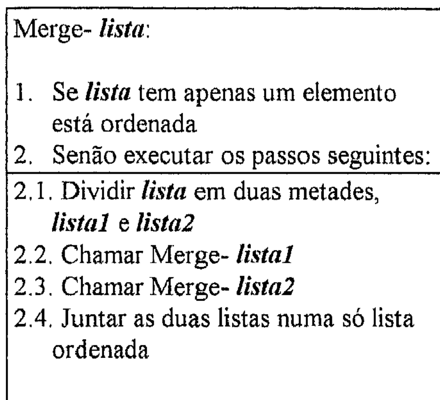
2.2. Uma aplicação PVM: Paralelização do algoritmo MergeSort

Uma aplicação PVM é escrita como um programa sequencial vulgar, por exemplo em linguagem C, estando contidas nesse programa as chamadas às rotinas da biblioteca PVM. Existem nomes pré-definidos para estas rotinas. Pode-se encontrar o nome das rotinas mais utilizadas, assim como uma pequena frase que as descreve, nas tabelas 2.1. a 2.7. As rotinas estão incluídas no ficheiro ***pvm3/include/pvm3.h***, pelo que todas as aplicações devem incluir a linha ***#include <pvm3.h>***. O utilizador para compilar qualquer aplicação escrita em linguagem C deverá escrever, na shell do Unix, depois de inicializar o PVM, o seguinte:

cc -o nome-a-dar -I$PVM_ROOT/include nome-da-aplicação -L$PVM_ROOT/lib/ALPHA -lpvm3

para cada uma das aplicações. De seguida deverá copiar os executáveis para o directório *$PVM_ROOT/bin/ALPHA*, pois será nesse directório que o pvmd os irá procurar. Será aconselhável utilizar uma makefile.

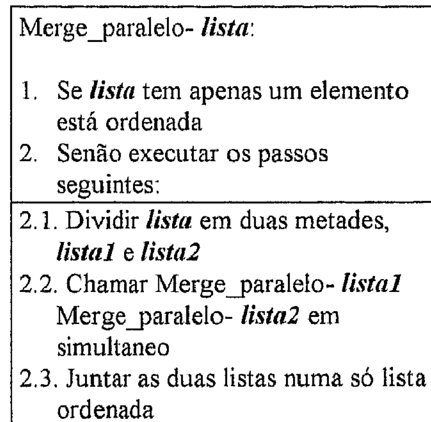
Para nossa primeira aplicação PVM, escolhemos implementar o algoritmo MergeSort (figura 2.6.), em paralelo [Darel92] - figura 2.7. A escolha prende-se com o facto de, neste algoritmo, serem originadas muitas rotinas, dependentes umas das outras. Pensamos que constituí um bom teste para determinar as limitações do PVM.


Merge- ***lista***:

1. Se ***lista*** tem apenas um elemento está ordenada
2. Senão executar os passos seguintes:

2.1. Dividir ***lista*** em duas metades, ***lista1*** e ***lista2***
2.2. Chamar Merge- ***lista1***
2.3. Chamar Merge- ***lista2***
2.4. Juntar as duas listas numa só lista ordenada

Figura 2.6. Algoritmo MergeSort


Merge_paralelo- ***lista***:

1. Se ***lista*** tem apenas um elemento está ordenada
2. Senão executar os passos seguintes:

2.1. Dividir ***lista*** em duas metades, ***lista1*** e ***lista2***
2.2. Chamar Merge_paralelo- ***lista1*** Merge_paralelo- ***lista2*** em simultaneo
2.3. Juntar as duas listas numa só lista ordenada

Figura 2.7. Algoritmo MergeSort Paralelo

Vamos agora proceder à paralelização do algoritmo de ordenação MergeSort, cujos passos estão representados na figura 2.6. O algoritmo paralelo está descrito na figura 2.7. O modelo de programação utilizado para a implementação do algoritmo paralelo é o modelo híbrido. Inicialmente temos um mestre, que inicia o vector a ordenar e o envia para uma rotina de ordenação, que por seu lado vai criar um modelo em árvore, recebendo no final o vector já ordenado, segundo o esquema da figura 2.8.

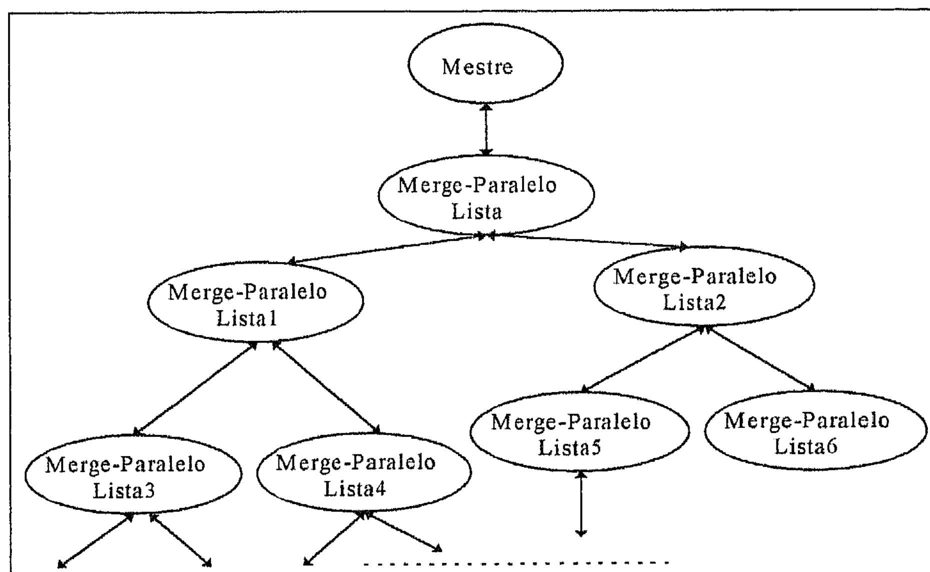


Figura 2.8. Modelo de implementação do algoritmo Merge-Paralelo

Implementámos os algoritmos das figuras 2.7. utilizando o PVM sobre quatro dos nós do *cluster UNIX* do CICA. A saber: tom.fe.up.pt; crazy.fe.up.pt; jerry.fe.up.pt e riff.fe.up.pt, todos eles baseados no processador Alpha de 64 bits, e interligados em rede FDDI a um GIGASwitch [Peres95]. As características do seu hardware estão especificadas na tabela 2.8.

Nome / Modelo Hardware	tom.fe.up.pt crazy.fe.up.pt DEC3000/500	jerry.fe.up.pt AS1000 4/233	riff.fe.up.pt AS1000 4/266
Velocidade de Relógio (ns)	6,7	4,3	3,8
Performance pico (Mflops)	150	233	266
Ram (Mbytes)/ Processador	128	128	128

Tabela 2.8. Características do Hardware utilizado

Quanto ao software, o sistema operativo que se encontra disponível nas quatro máquinas é o Digital UNIX nas versões 3.2G, 3.2A e 4.0A, para o tom.fe.up.pt crazy.fe.up.pt; o jerry.fe.up.pt e o riff.fe.up.pt, respectivamente. Trata-se de um sistema multiutilizador e multitarefa. O número de utilizadores varia significativamente ao longo de um dia, influenciando o tempo necessário à execução de um determinado processo; de forma mais especifica, este tempo depende do número de processos simultâneos em execução numa mesma máquina. O nó tom.fe.up.pt é, normalmente, o mais sobrecarregado. Assim sendo, apesar de os nós tom.fe.up.pt e crazy.fe.up.pt terem as mesmas características de hardware, o primeiro apresenta tempos de execução superiores.

Executámos a nossa aplicação nas quatro máquinas, utilizando vectores de números inteiros com tamanhos diferentes, várias vezes e em horas diferentes do dia, que correspondem a cargas de utilização diferentes. Medimos os tempos de execução em cada um dos casos, encontrando-se as médias desses tempos representadas na tabela 2.9.

Tamanho dos Vectores	Média dos tempos no tom.fe.up.pt	Média dos tempos no crazy.fe.up.pt	Média dos tempos no jerry.fe.up.pt	Média dos tempos no riff.fe.up.pt
20	$2{,}40 \times 10^{-3}$ s	$2{,}19 \times 10^{-3}$ s	$1{,}14 \times 10^{-3}$ s	$1{,}01 \times 10^{-3}$ s
40	$2{,}99 \times 10^{-3}$ s	$2{,}65 \times 10^{-3}$ s	$2{,}22 \times 10^{-3}$ s	$1{,}39 \times 10^{-3}$ s
100	$3{,}76 \times 10^{-3}$ s	$3{,}71 \times 10^{-3}$ s	$3{,}51 \times 10^{-3}$ s	$2{,}21 \times 10^{-3}$ s

Tabela 2.9. Tempos de execução do Algoritmo MergeSort

Consultando a tabela 2.9., podemos verificar que se confirma o facto esperado de o nó tom.fe.up.pt ser aquele que apresenta tempos de execução mais elevados; as diferenças relativamente ao nó crazy.fe.up.pt não são significativas. Quanto aos nós jerry.fe.up.pt e riff.fe.up.pt, apresentam melhores tempos de execução, particularmente o riff.fe.up.pt, que apresenta tempos de execução de cerca de metade daqueles que são apresentados pelo tom.fe.up.pt e pelo crazy.fe.up.pt.

2.2.1. As limitações da paralelização utilizando PVM

Normalmente a paralelização de um algoritmo faz equacionar uma série de situações habituais, porém no caso particular da paralelização utilizando PVM existem mais duas situações a questionar. Uma delas refere-se ao número de tarefas simultâneas que cada processador vai suportar e a outra diz respeito ao tempo necessário para a comunicação das tarefas necessárias à implementação do algoritmo. No PVM, o limite daquele número de tarefas depende não só do número de processos disponibilizados no sistema operativo para o utilizador mas também do número de "*file descriptors*" disponíveis para o *pvmd*, sendo que o PVM fica "engarrafado" quando muitas tarefas tentam comunicar entre si. No entanto, este número não é fundamental, dado que não faz sentido ter um grande número de tarefas simultâneas no mesmo processador. No nosso sistema, o PVM poderá suportar, no máximo, 30 tarefas simultâneas, dependendo das comunicações existentes entre elas.

O número de tarefas envolvidas, aquando da ordenação pelo algoritmo MergeSortParalelo é elevado. Sendo ***n*** o tamanho do vector e uma potência de dois, vem que o número de tarefas, $\boldsymbol{T}$, é dado pela fórmula:

$$T(n)=\sum_{i=0}^{\log_2 n} 2^i \qquad \text{[Fórmula 2.1]}$$

Por exemplo, se n=8, vem $T(8)=\sum_{i=0}^{3} 2^i = 1+2+4+8=15$; caso ***n*** não seja uma potência de dois, e sendo ***k*** a potência de dois imediatamente a seguir a ***n***, o número de tarefas é dado por:

$$T(n)=\sum_{i=0}^{\log_2 k} 2^i - 2(k-n) \qquad \text{[Fórmula 2.2]}$$

Por exemplo, se n=7, vem $T(7)=\sum_{i=0}^{3} 2^i - 2(8-7)=15-2=13$.

Facilmente se demonstra, por indução matemática, que $T(n)=2n-1$. Como dispomos de quatro processadores, isso implica que a partir de um certo tamanho do vector, o número de tarefas necessárias à implementação do algoritmo se torna incomportável para o PVM. Por exemplo, para n=20, são necessárias $T(20)=2\times 20-1=39$ tarefas, correspondendo a dez tarefas por processador; para n=40 são necessárias $T(40)=2\times 40-1=79$ tarefas, o que corresponde, aproximadamente, a vinte tarefas por processador; para n=100 são necessárias $T(100)=2\times 100-1=199$ o que corresponde a cerca de 50 tarefas por procesador, já por si só incomportável.

Com o objectivo de estimar o tempo de execução em paralelo, necessitamos de conhecer os tempos de comunicação entre os vários nós utilizados. Para isso baseamo-nos em [Guedes95]. Neste trabalho são determinados modelos teóricos dos tempos de comunicação entre os vários nós existentes no CICA, à altura da sua realização. O tempo de execução de uma aplicação em paralelo depende fortemente do desempenho da pior máquina. Este facto acontece especialmente quando existe sincronização entre as várias tarefas, pois as rotinas que são executadas nas máquinas mais rápidas terão de esperar pelas mais lentas. No entanto, mesmo que não exista sincronização, para que uma aplicação termine, terão de terminar todas as rotinas que a constituem e deste

modo sendo, o seu tempo de execução depende das rotinas mais lentas. No nosso caso, como temos várias instâncias das mesmas rotinas, a mais lenta é a que corre na pior máquina. Assim sendo, vamos basear os nossos cálculos nos resultados da pior máquina, ou seja, do modelo DEC3000/500.

Segundo [Guedes95] pp. 51-55, o tempo de comunicação entre dois processadores P_1 e P_2 para o envio de uma mensagem de ***m*** *bytes* é dada pela fórmula $T(m) = T_0 + m\beta$, sendo T_0 o tempo, em segundos, requerido para iniciar o envio da mensagem e β o tempo, em segundos, de envio de um *byte* pela rede. Para o caso particular do modelo DEC3000/500, obteve-se a fórmula:

$$T_c(m) = 0{,}727 \times 10^{-3} + 0{,}60 \times 10^{-6} m \quad \text{[Fórmula 2.3]}$$

Vamos agora estimar, utilizando esta fórmula, o tempo necessário para a comunicação na ordenação de um vector de tamanho ***n***. No caso de ***n*** ser uma potência de dois obtemos $T_c(n) = 2 \times \sum_{i=1}^{\log_2 n} 2^i \cdot T_c\left(\frac{2n}{2^i}\right)$. Por exemplo, para n=8 vem $T_c(8) = 2 \times \left(2 \times T_c(8) + 4 \times T_c(4) + 8 \times T_c(2)\right) = 2{,}03 \times 10^{-2}$. Realizando os cálculos para n=20 e n= 40, obtemos $T_c(20) \approx 4{,}09 \times 10^{-2}$ e $T_c(40) \approx 0{,}19$. Não foram realizados cálculos para n=100 pois, como já verificámos, o número de tarefas necessárias não é suportado pelo nosso sistema. Antes de passar ao teste da aplicação, referimos a previsão de tempos reais de execução muito superiores aos tempos de comunicação calculados, apesar de cada uma das tarefas ser de rápida execução. Este aspecto deve-se ao facto de que o Merge-Paralelo exige grandes tempos de espera, pois em cada nível da árvore de execução o "pai" tem de esperar que todos os "filhos" e "netos" executem o seu trabalho computacional e também devido ao facto de o PVM começar a ficar "engarrafado", dado o elevado número de tarefas por processador.

Os testes realizaram-se utilizando os recursos já referidos, várias vezes, em horas diferentes do dia, correspondendo a cargas de utilização diferentes. Os resultados obtidos estão registados na tabela 2.10.

Tamanho do vector	Média dos tempos reais de execução (s)
20	1,91
40	4,14
100	Engarrafou

Tabela 2.10. Tempos reais de execução do Merge-Paralelo

Podemos verificar que os tempos reais são muito superiores aos tempos calculados, confirmando-se as previsões já efectuadas. Também se confirma que o PVM não consegue comportar o número de tarefas geradas ao ordenar um vector de tamanho 100. Assim sendo, concluímos que o algoritmo MergeSort não é um algoritmo que deva ser paralelizado nas condições presentes.

2.3. Mais sobre PVM

2.3.1. Detecção de erros

O PVM fornece um sistema simples para detecção de erros. Ao chamar a rotina **pvm_spawn**, fazêmo-lo colocando o seu parâmetro *flag* = *PvmTaskDebug*. Sendo assim, a tarefa é iniciada através de um roteiro de detecção de erros [Geist94b], Cap.9.

Uma forma popular de fazer a detecção de erros em programas paralelos é o rastreio (tracing). O PVM 3.3 suporta directamente o rastreio das suas aplicações. Para isso, é necessária a existência de ambientes completos de programação que incluam, pelo menos, um detector de erros, um utilitário para rastreio e um analisador de qualidade. Existem vários ambientes disponíveis, sendo os mais conhecidos o XPVM, o XAB, o Hence e o ParaGraph ([Chergui95]). Têm sido feitos esforços no sentido da criação deste tipo de ambientes, não havendo, no entanto, uniformização nem integração de uns com os outros.

2.3.2. Outros projectos baseados no PVM

Não queremos deixar de referir a existência de vários projectos baseados no PVM, desenvolvidos por vários investigadores em centros espalhados por todo o mundo. Incorrendo no risco de sermos um pouco nacionalistas, salientamos o projecto WPVM (Windows Parallel Virtual Machine) desenvolvido no Departamento de Engenharia Informática da Universidade de Coimbra. O sistema WPVM é uma implementação do PVM para o Windows da Microsoft. Para obtenção de mais informações consultar [Alves95] e [Alves96].

2.3.3. E o futuro ?

Existem algumas limitações do sistema PVM que se têm vindo a tentar superar, por exemplo, no que diz respeito à chamada de rotinas que contenham passagem de mensagens, à comunicação entre tarefas PVM e na detecção de erros nas aplicações PVM. Assim sendo, a versão 3.4 do PVM poderá conter [Geist95]:

- Introdução de um contexto de comunicação que permitirá a chamada, de forma segura, a rotinas existente em bibliotecas que contenham passagem de mensagens.
- Alteração da forma de codificação dos dados no envio de mensagens, o que levará a uma maior eficiência. No que diz respeito à portabilidade e ao número de rotinas necessárias à implementação do novo esquema, subsistem ainda alguns problemas a superar.
- Melhoramentos no rastreio, fornecendo maior flexibilidade e maior eficiência ao acesso da informação durante a execução do programa.
- Criação de um serviço de nomes para as tarefas, levando a uma maior segurança na comunicação entre elas.
- Possibilidade de grupos estáticos, o que, em alguns casos, levará à obtenção de melhores rendimentos.

- Possibilidade de correr aplicações MPI (ver secção 2.4.) a partir da consola PVM, aproveitando as vantagens dos dois sistemas.
- Introdução de níveis de segurança na transmissão de mensagens.

2.4. Outros Sistemas para Paralelização de Aplicações

Têm vindo a ser desenvolvidos vários projectos de software baseados no mesmo princípio que o PVM: utilizar sistemas distribuídos para paralelizar aplicações. Os mais divulgados são o Linda ([Carriero90]), o P4 ([Butler92]), o Express ([Flower91]) e o MPI ([Preston96]). O sistema Linda baseia-se num modelo de memória distribuída virtual e os outros três num modelo de passagem de mensagens.

2.4.1. O sistema Linda

O Linda centra-se num modelo de memória distribuída, que consiste num espaço de tuplos (tuple space), sendo estes a sua unidade básica, surgindo uma memória distribuída virtual. Os tuplos não são acedidos através de endereços, mas por um nome lógico que é qualquer selecção dos seus valores. Há três tipos de operações possíveis para acesso à memória: ler, adicionar e retirar. Isto permite o acesso simultâneo de vários processos à memória. Os processos não comunicam directamente: um processo com dados, para comunicar, escreve-os na memória; um processo que necessite de dados vai buscá-los à memória. Se a este núcleo de operações, no espaço de tuplos, adicionarmos uma linguagem, obtemos um dialecto para programação em paralelo (por exemplo, C-Linda).

2.4.2. O sistema P4

O P4 foi desenvolvido no Argonne National Laboratory, para programar uma variedade de máquinas em paralelo. É constituído por uma biblioteca de macros e subrotinas. Permite um modelo de memória partilhada baseado em monitores e um modelo de memória distribuída baseado na passagem de mensagens, permite ainda uma

combinação dos dois modelos. Pretende-se que este sistema seja portável, simples de instalar e usar e que seja eficiente. Permite a utilização das linguagens C e FORTRAN.

2.4.3. O sistema Express

O Express é um software comercial da Parasoft Corporation. A sua filosofia é a seguinte: inicia-se com uma versão sequencial de uma aplicação que, seguindo o desenvolvimento recomendado para o seu ciclo de vida, culmina numa versão paralela que aponta para aquilo que os autores chamam o "óptimo". Para ajudar o programador através dos vários estados do desenvolvimento de uma aplicação paralela, o Express fornece várias ferramentas.

2.4.4. O MPI

O MPI (Message Passing Interface) é uma especificação padrão que surge como uma grande uniformização das implementações de passagem de mensagem existentes (PVM, P4, Express, ...). As especificações foram completadas em Abril de 1994. Pretende-se que seja não uma biblioteca escrita numa determinada linguagem, mas sim uma camada de software, situada sobre o hardware, que faça a interface de comunicações. Prevê-se para breve a sua implementação.

2.5. Notas Finais

Relativamente a todos os sistemas enunciados, a grande popularidade do PVM deve-se a vários factores, nomeadamente a sua simplicidade, existência de um modelo de computação natural e geral, robustez da implementação, facilidade na utilização e alto grau de portabilidade. Devem, no entanto, ser tomados alguns cuidados na sua utilização na implementação de algoritmos paralelos, que nem sempre são os mais adequados ao sistema. Estes cuidados relacionam-se principalmente com o número de tarefas por processador disponível e com a necessidade de minimizar a comunicação entre os vários processadores.

Pelo que nos foi dado observar as duas grandes bibliotecas de passagem de mensagens sobre as quais se situam os grandes esforços são o PVM e o MPI, tendendo as outras ao desaparecimento. Pensa-se que nenhuma delas sobrevirá por si só, pois encaminham-se para uma uniformização/aglutinação.

Prevêem-se evoluções importantes nomeadamente a utilização de "processos ligeiros" (threads) [Tanebaum90] e a introdução de metodologias de orientação ao objecto.

3 Algoritmos Genéticos

O DNA constitui o suporte universal de toda a informação biológica que define as características de cada organismo. Este princípio aplica-se a todas as células vivas, desde as que constituem os seres mais simples aos dotados de uma maior complexidade genética. O DNA é um polímero, em que as unidades básicas que o constituem são nucleótidos. Cada nucleótido é constituído por uma base azotada (Adenina(A), Guanina(G), Citosina(C) e Timina(T)), uma desoxirribose e um ácido fosfórico. Embora exista um pequeno número de tipos diferentes de nucleótidos (apenas quatro), é possível uma grande diversidade de moléculas de DNA, pois cada nucleótido pode estar presente um grande número de vezes ou sujeitar-se a diferentes ordenações. Esta combinação permite que cada indivíduo tenha o seu próprio DNA.

Os genes são segmentos de moléculas de DNA que contêm determinada informação. Cada gene pode conter milhares de pares de bases. O número de bases, a sua natureza e a ordem pela qual estão organizadas diferem de gene para gene. O conjunto de genes que constitui a informação genética de um organismo tem o nome de Genoma ou Cromossoma. Cada organismo vivo, desde a bactéria até ao homem, tem o seu genoma específico.

A replicação de DNA (através da reprodução) assegura a conservação e transmissão do património genético próprio de cada espécie. Os seres vivos reproduzem-se de duas formas distintas: normalmente os seres mais simples, como as bactérias, conseguem cópias exactas de si próprios (reprodução assexuada); os seres mais complexos reproduzem-se através da combinação das características de mais de um indivíduo (reprodução sexuada).

Salvo raras excepções, verifica-se uma linguagem (código de 4 bases) comum a quase todas as células. Esta linguagem que a célula utiliza na transferência de informação genética, designa-se código genético. A universalidade do código genético é uma forte indicação de que toda a vida na terra evoluiu a partir de uma única fonte. Todos os habitantes da terra são parentes próximos. Todos têm uma química orgânica e uma herança evolutiva comuns. À grande descoberta da selecção natural como mecanismo da evolução estão associados os nomes de Charles Darwin e Alfred Russel Wallace. Há pouco mais de um século, eles sublinharam o facto de a natureza ser prolífera e de nascerem muito mais animais e plantas do que aqueles que têm possibilidade de sobreviver, logo, o ambiente selecciona as variedades mais adaptadas à sobrevivência. O ambiente selecciona, num dado momento, as poucas de entre elas que apresentam as maiores probabilidades de sobrevivência, o que resulta numa série de lentas transformações de uma forma de vida para outra, a origem de novas espécies. As mutações, alterações súbitas na hereditariedade, transmitem-se à descendência. São eles a matéria prima da evolução. Os segredos da evolução são a morte e o tempo: a morte de um sem fim de espécies mal adaptadas ao ambiente e o tempo necessário à lenta acumulação de padrões de mutação favoráveis[1].

Inspirado nestas ideias, Jonh Holland publicou em 1975, o seu livro "Adaptation in Natural and Artificial Systems", lançando assim as bases da teoria sobre algoritmos genéticos. Outros pioneiros como Goldberg, De Jong, Grefenstette, Davis e outros, alimentaram o crescimento dos algoritmos genéticos.

É neste linha que neste capítulo vamos explicar o funcionamento dos algoritmos genéticos. No ponto 3.1. será feita uma breve abordagem ao tema. No ponto 3.2. será descrito o funcionamento de um algoritmo genético canónico. Em 3.3. serão abordados alguns conceitos avançados, tais como algoritmos genéticos híbridos e programação genética e em 3.3.3. faremos uma abordagem a problemas baseados em permutações, com aplicação ao problema do Caixeiro Viajante.

[1] in Enciclopédia Britânica

3.1. Os algoritmos genéticos: descrição e contextualização

As aplicações que utilizam algoritmos genéticos estendem-se já a uma grande variedade de áreas, desde as ciências da computação, às ciências económicas e passando mesmo pela música. O interesse manifestado pela comunidade científica, relativamente à utilização de algoritmos genéticos, tem vindo a crescer e consequentemente, tem vindo a acontecer uma grande expansão nesta área.

Os algoritmos genéticos são, no essencial, uma versão em programa de computador dos processos evolutivos que acontecem na natureza. Acontece, no entanto, que o processo pode ser fortemente acelerado pelo computador, através da manipulação dos princípios evolucionários: os membros da população crescem, combinam-se e morrem em poucos microssegundos; este facto origina que, em pouco tempo, existam grandes mudanças. A figura 3.1. apresenta o funcionamento de um algoritmo genético tradicional.

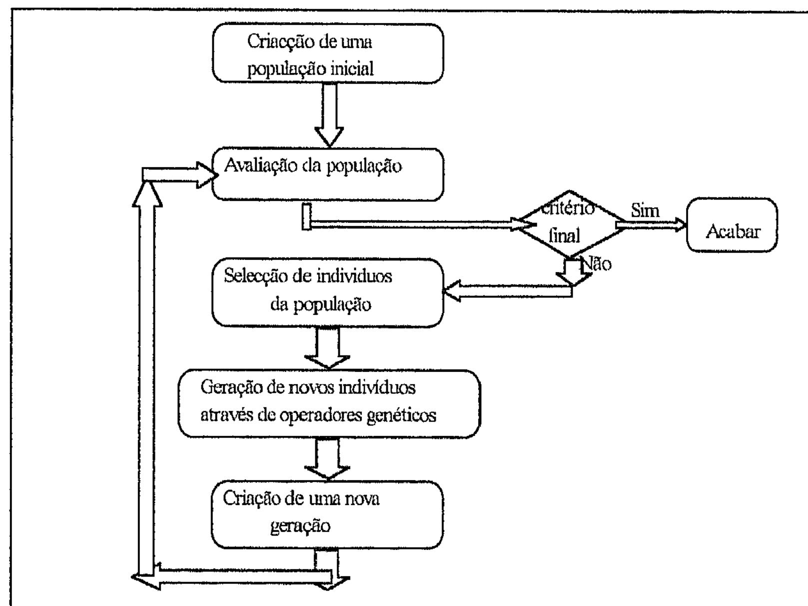


Figura 3.1. Fluxograma de um algoritmo genético tradicional

Um algoritmo genético necessita, antes de mais nada, de um conjunto de indivíduos que formam uma população, sendo cada um dos indivíduos um membro dessa população [Whitley93]. Cada membro da população é uma tentativa de resolução do problema em estudo e está associado a um resultado (função de saída do algoritmo),

que descreve o grau de adaptação do indivíduo. Ao conjunto desses resultados chama-se função de adaptação (fitness). As entradas para a função de adaptação são, habitualmente, designadas por genes ou cromossomas. Cada gene contém informação sobre uma dada característica do problema a resolver e cada indivíduo da população contém os genes necessários à caracterização da solução do problema; portanto, cada membro da população contém o mesmo número de genes.

O trabalho dos algoritmos genéticos inicia-se com tentativas de solução relativamente pobres, isto é, os membros da população têm uma adaptação fraca. São permitidos três processos básicos: **união** (mate), **mutação** e **sobrevivência** dos mais adaptados. O primeiro processo, união, envolve a troca de informação entre membros da população, atribuindo-se-lhe o nome de cruzamento (crossover). Quando há união entre membros da população, estes trocam entre si informações, o que resulta numa mistura de informações dos valores dos genes, criando uma descendência nova e diversa.

Um membro da população pode também submeter-se aos processos de mutação. Na mutação, os valores individuais dos genes dos membros da população podem ser alterados. A mutação é importante porque são introduzidos novos genes, aumentando a diversidade.

Uma vez feitos os cruzamentos e as mutações, os novos membros da população substituirão os velhos se forem melhores. Desta forma, é melhorada a adaptação dos membros da população em cada novo ciclo (ou geração). Estamos perante o terceiro processo básico: sobrevivência dos mais adaptados.

O processo descrito gera, normalmente, soluções cada vez melhores para o problema, num tempo surpreendentemente curto [Davis91]! Existem, no entanto, alguns problemas considerados difíceis para os algoritmos genéticos [Grefenstetteb].

3.1.1. Algoritmos genéticos na optimização

Os algoritmos genéticos impõem-se cada vez mais como uma técnica de optimização, apresentando algumas diferenças essenciais relativamente às técnicas de optimização tradicionais [Goldberg89]. Essas diferenças são as seguintes:

- utilização de uma codificação (binária ou não) dos valores dos parâmetros, sendo a codificação irrelevante para o algoritmo;
- utilização de uma pesquisa a partir de vários pontos e não a partir de um só ponto, diminuindo a probabilidade de convergência para óptimos locais;
- utilização da função de adaptação (fitness) como único recurso, em vez de derivadas ou outras informações parciais nem sempre possíveis de obter, permitindo assim a abordagem de uma gama mais diversificada de problemas;
- transações baseadas em regras probabilísticas, conseguidas através dos operadores genéticos de selecção, cruzamento e mutação, diminuindo a probabilidade de convergência para óptimos locais.

Tendo por base estes quatro pontos, os algoritmos genéticos podem ser considerados como métodos fracos (weak) e robustos: fracos, pois exigem poucas restrições ao domínio do problema que pretendem resolver, e robustos, pois podem ser utilizados em problemas variados com bons resultados. Os algoritmos genéticos não pretendem atingir o óptimo para um determinado tipo de problemas, pretendem sim obter bons resultados para uma vasta gama de problemas, em tempo considerado razoável para esse problema.

Os algoritmos genéticos estão englobados nas técnicas de pesquisa. Estas são divididas em três grandes classes [Goldberg89]:

- baseadas no cálculo,
- enumerativas,
- guiadas aleatoriamente,

segundo o diagrama em árvore da figura 3.2.

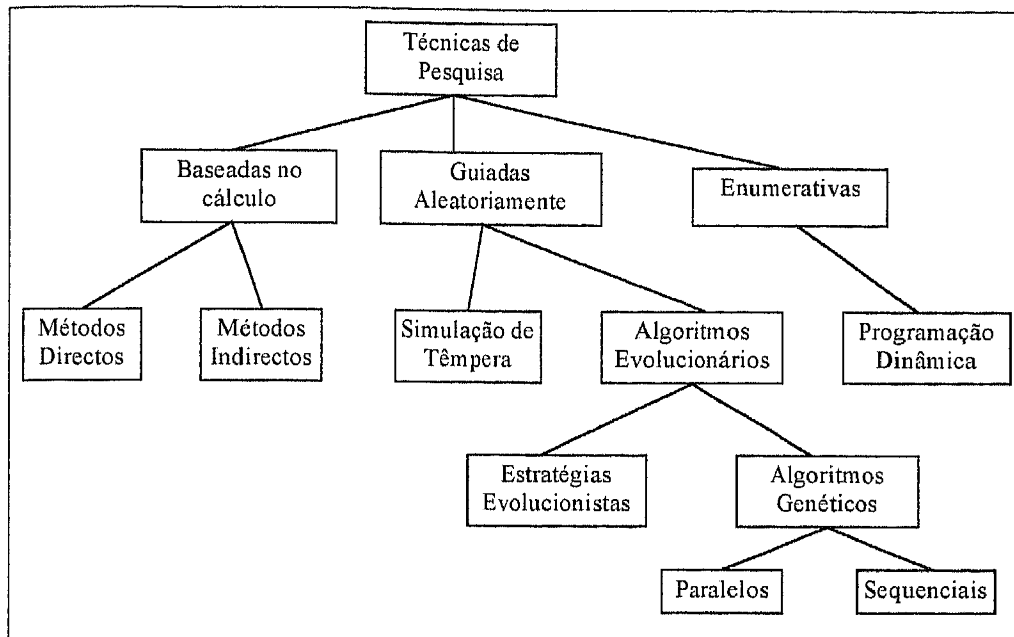


Figura 3.2. Classes de Técnicas de Pesquisa

As técnicas baseadas no cálculo exigem que sejam satisfeitas, um conjunto de condições necessárias/suficientes. Estas técnicas subdividem-se em métodos directos e indirectos. Os métodos indirectos procuram óptimos locais, habitualmente através da resolução de um conjunto de equações não lineares, resultantes do facto de igualar o gradiente da função objectivo a zero. A procura das soluções possíveis (picos da função) iniciam-se pela restrição da pesquisa a pontos com declive nulo em todas as direcções. Os métodos directos tais como os de Newton e Fibonacci, procuram extremos "saltando" pelo espaço de soluções e acedendo ao gradiente da cada novo ponto, que vai orientar a direcção da pesquisa. É esta a noção de "subir a colina" (hill-climbing), através do qual se encontra o melhor ponto local subindo na direcção do maior gradiente possível. No entanto, estes aplicam-se apenas a um conjunto restrito de problemas bem comportados.

As técnicas enumerativas procuram todos os pontos relacionados com o espaço de soluções de uma função objectivo (finito ou discreto), um ponto de cada vez. São muito simples de implementar, mas podem requerer um trabalho computacional significativo. O espaço de soluções de muitas aplicações é demasiado grande para uma

pesquisa, utilizando estas técnicas. A programação dinâmica é um bom exemplo de uma técnica enumerativa.

As técnicas de pesquisa guiadas aleatoriamente são baseadas nas técnicas enumerativas, mas utilizam informação adicional para guiar a pesquisa. O seu alcance é bastante generalizado, sendo capaz de resolver problemas bastante complexos. As duas maiores subclasses são os Algoritmos de Simulação de Têmpera[2] (Simulated Anneling) e os Algoritmos Evolutivos (Evolutionary Algorithms), embora ambos sejam processos evolutivos. Os primeiros utilizam um processo de evolução termodinâmica para procurar estados de energia mínimos, enquanto que os algoritmos evolutivos utilizam princípios baseados na selecção natural. Esta última forma de pesquisa desenvolve-se com gerações do princípio ao fim, melhorando as capacidades das soluções potenciais através de operações inspiradas biologicamente. Estas técnicas subdividem-se ainda em Estratégias Evolutivas (Evolution Strategies) e Algoritmos Genéticos. As primeiras foram propostas no início da década de 70 por Rechenberg e Schwefel e apresentam a capacidade de adaptar o processo de evolução artificial aos requisitos do ambiente. Isto significa que as estratégias evolutivas são capazes de adaptar os seus parâmetros estratégicos às características topológicas locais da função objectivo, o que representa uma diferença significativa para os algoritmos genéticos tradicionais. Têm sido propostas muitas alterações aos algoritmos genéticos canónicos, inicialmente introduzidos por Holland. Existe, no entanto, uma capacidade importante que distingue todos os algoritmos genéticos: trata-se da sua técnica de lidar com populações. Nos algoritmos genéticos tradicionais adoptou-se uma política na qual toda a população é substituída em cada geração. Inversamente as políticas "*steady-state*", usadas por muitos algoritmos genéticos, empregam uma substituição selectiva da população, sendo possível manter indivíduos na população durante várias gerações, enquanto eles forem melhores que o resto da população.

[2] Têmpera do aço: tratamento térmico que se aplica ao aço visando obter uma dureza muito alta.

3.1.2. Perspectiva histórica

Pensamos que neste ponto é importante deixar algumas notas sobre a evolução histórica dos algoritmos genéticos, não pretendendo, no entanto, ser exaustivos, uma vez que está fora do âmbito deste trabalho. Informação mais detalhada sobre este assunto pode ser encontrada em [Goldberg89], cap.4.

Já antes da utilização dos algoritmos genéticos alguns biólogos como Barricelli (1975;1962) e Fraser (1960;1962) entre outros, utilizaram os computadores digitais para a simulação de ambientes genéticos.

Foi no entanto Holland, em 1962, a reconhecer as grandes potencialidades da aplicação de operadores de base genética a sistemas artificiais de adaptação. Bagley, em 1967, utilizou pela primeira vez o termo "Algoritmo Genético". O seu trabalho é extremamente importante, dado que contém já a utilização dos três operadores selecção, cruzamento e mutação[3] - reconhecendo, além disso, a necessidade de utilizar operadores genéticos mais poderosos .

Em 1971, Hollstien, na sua dissertação, faz pela primeira vez, a aplicação de algoritmos genéticos a um problema de optimização matemática pura. Por este época, Holland surge com o importante Teorema dos Esquemas[4], identificando pela primeira vez a importância fundamental da recombinação estruturada para alcançar o paralelismo implícito dos algoritmos genéticos e solidificando a sua base teórica.

O ano de 1975 é bastante profícuo para os algoritmos genéticos, tendo sido publicadas duas obras muito importantes: Adaptation in Natural and Artificial Systems, por Holland e An Analisys of the behavior of a class of Genetic Adaptative Systems, por De Jong. A obra de De Jong é fulcral, dado que ele faz a combinação da Teoria dos Esquemas de Holland com a sua experimentação computacional cuidadosa, em aplicações relativas à optimização de funções, solidificando assim toda a teoria dos algoritmos genéticos.

Várias pessoas têm vindo a seguir estes estudos, sugerindo vários melhoramentos aos algoritmos básicos e outros trabalhos específicos foram

[3] Em análise em 3.2.2.
[4] Ver secção 3.2.3.

desenvolvidos com algum sucesso. É, no entanto, uma área jovem com um vasto campo teórico ainda por explorar e da qual se esperam muitos desenvolvimentos.

3.2. Como funciona um algoritmo genético

Vamos, a partir de agora, descrever o funcionamento de um algoritmo genético, servindo-nos de um exemplo simples - a função real de duas variáveis reais[5]:

$$f(x,y) = \sin(0{,}8x+6)\cdot\cos y(2\cos(0{,}2x+16))$$
$$x \in [0;6{,}4]$$
$$y \in [0;6{,}4], \text{ com a precisão de } 0{,}1$$

O gráfico desta função está representado na figura 3.3[6]. Verificamos que a função apresenta, neste intervalo, um máximo, pelo que vamos maximizá-la. Por observação do gráfico, verificamos que a função apresenta declives acentuados nos extremos, pelo que os métodos tradicionais poderão ser facilmente enganados[7]. Apresentamo-la aqui para servir de exemplo para o funcionamento de um algoritmo genético que, como veremos, descobrirá o óptimo com facilidade.

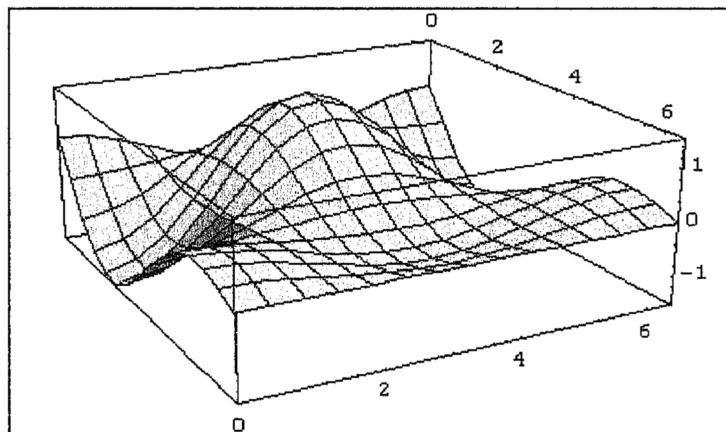


Figura 3.3. Gráfico da função a optimizar

Há apenas duas componentes principais da maioria dos algoritmos genéticos que dependem do problema: a sua codificação e a função de avaliação. Podemos ver um

[5] Função criada pela autora da tese, com o auxilio do software Mathematica, Versão 2.2.3., sobre Windows95.
[6] O gráfico foi obtido com o software referido na nota 5.
[7] Verificámos este facto. Utilizaámos o software *Mathematica* para determinar o máximo e ele não conseguiu, tendo havido necessidade de diminuir a amplitude do intervalo inicial. Utilizámos ainda o software Scientifc WorkPlace, que determinou vários pontos "máximos", incluindo o verdadeiro máximo; este é de 1,49 , com a precisão de duas casas decimais.

algoritmo genético como uma caixa preta com uma série de mostradores de controlo, que representam os vários parâmetros (codificados), e com uma única saída, um valor que indica quão bem uma combinação particular de valores dos parâmetros resolvem o problema de optimização. Vamos tentar descobrir como funciona a "caixa preta" e como é feita a ligação com a sua entrada (a codificação dos vários parâmetros) e a sua saída (a avaliação).

O primeiro passo na implementação de algoritmo genético é a geração de uma população inicial. No algoritmo genético canónico, cada membro da população vai ser uma cadeia de tamanho L, que corresponde à codificação do problema. Cada cadeia pode ser designada como genótipo (Holland,1975) ou alternativamente como cromossoma (Schaffer, 1987). Na maioria dos casos a população inicial é gerada de uma forma aleatória. Um cromossoma, num algoritmo genético canónico, é constituído por uma cadeia de *bits* de informação contendo um dos valores binários 0 ou 1. É possível utilizar outros tipos de codificação, conforme veremos mais adiante, mas no que se segue, utilizaremos a codificação tradicional.

No nosso exemplo, o tamanho do cromossoma terá de ser capaz de representar $\frac{6{,}4-0}{0{,}1}=64$ possíveis valores para cada valor de x e de y. Portanto, o tamanho do cromossoma vai ser de $2\times\log_2 64 = 2\times 6 = 12$, conforme representado na figura 3.4. . Após a criação da população inicial cada cromossoma é avaliado, sendo-lhe atribuído um valor de adaptação.

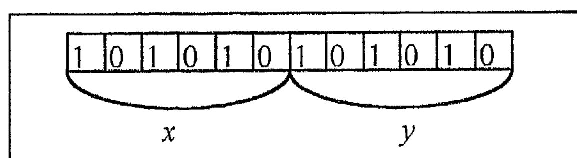


Figura 3.4. Codificação contida num cromossoma

3.2.1. Avaliação / Adaptação

As noções de avaliação (evaluation) e adaptação (fitness) são por vezes usadas alternadamente. Temos, no entanto, de distinguir entre função de avaliação e função de

adaptação [Whitley93]. A função de avaliação corresponde à função objectivo que se pretende maximizar ou minimizar, com respeito a um conjunto particular de parâmetros. No nosso exemplo, a avaliação inicia-se pela descodificação dos cromossomas:

$$x = \frac{\sum_{i=0}^{5} c_i \cdot 2^i}{10}; y = \frac{\sum_{i=6}^{11} c_i \cdot 2^{i-10}}{10}; \text{ sendo } c_i \text{ valor do } bit \text{ i do cromossoma c}$$

De seguida substituímos os valores de x e de y assim obtidos na expressão que define a função. Obtemos, então, o valor de avaliação desse cromossoma.

A função de adaptação transforma esta medida de rendimento numa atribuição de oportunidades reprodutivas. A avaliação de um cromossoma é indiferente da avaliação de qualquer outro cromossoma. No entanto, a adaptação desse cromossoma é sempre definida em relação a outros membros da população corrente. Nos algoritmos genéticos canónicos a adaptação é definida por :

$\frac{f_i}{\bar{f}}$ onde f_i é a avaliação associada ao cromossoma i

e $\bar{f}$ é a média das avaliações de todos os cromossomas da população

A verdade é que existem inúmeras formas de definir a função de adaptação, mas geralmente, esta é definida por:

$$u(x) = g(f(x)),$$

sendo u a função de adaptação

f a função de avaliação

e g transforma cada valor da função objectivo num número não negativo

Estes métodos apresentam o inconveniente de que ao fim de algumas gerações, os valores dos vários cromossomas são muito semelhantes, levando a uma convergência prematura do algoritmo. Existem métodos que pretendem obviar a este problema, como por exemplo, o escalonamento (scaling) e a normalização linear[8]. No método do

[8] Para mais informações, consultar [Davis91] e [Goldberg89]

escalonamento é encontrada a avaliação mínima da função e a adaptação de cada cromossoma é igual à quantidade pela qual a sua avaliação excede o mínimo. No método da normalização linear, os cromossomas são ordenados por ordem crescente da avaliação. A adaptação começa com um valor constante e decresce linearmente.

Depois de calculada a função de adaptação, passamos ao processo de selecção na população corrente criando uma população intermédia; através de cruzamentos e mutações aplicados a esta população intermédia obtemos a população seguinte, conforme está esquematizado na figura 3.5.

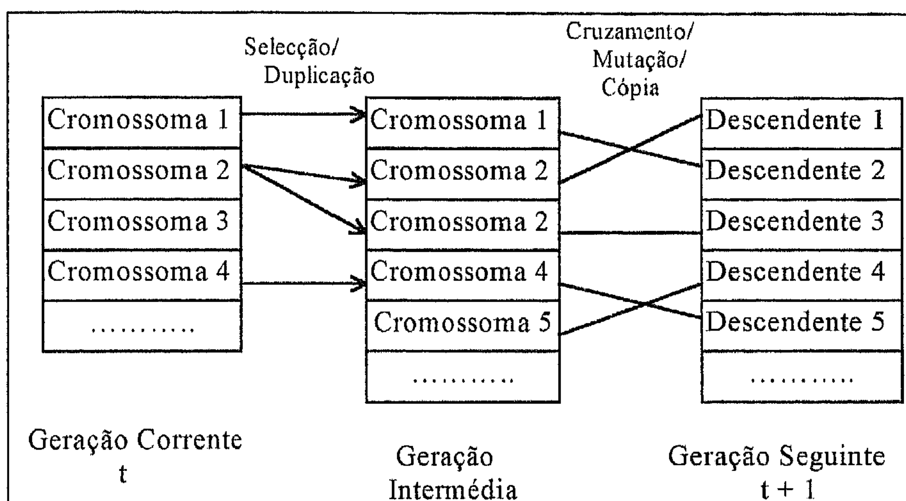


Figura 3.5. O aparecimento de uma nova geração

O processo de passar da população corrente para a população seguinte constitui uma geração na execução do algoritmo genético.

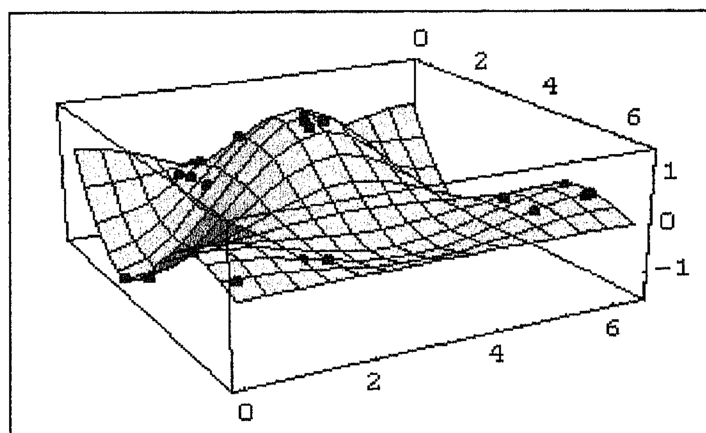


Figura 3.6. Distribuição dos pontos na primeira geração

No decorrer das gerações as soluções vão-se aproximando do máximo da função, conforme se pode visualizar através da comparação das figura 3.6. e 3.7.

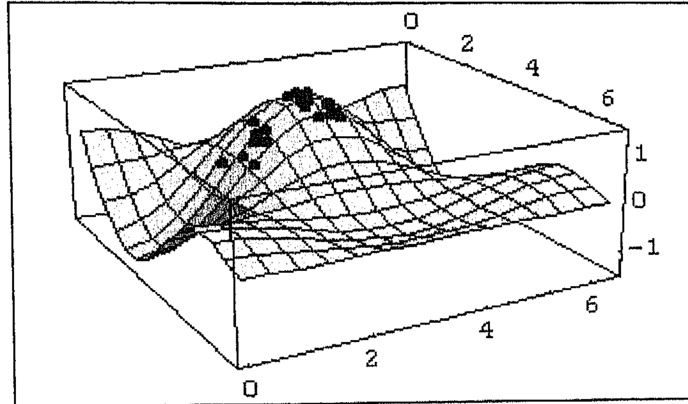


Figura 3.7. Distribuição dos pontos na oitava geração

3.2.2. Operadores genéticos

Vamos neste ponto explicar o funcionamento dos operadores genéticos de selecção, cruzamento e mutação. Utilizaremos apenas os operadores definidos para o algoritmo genético canónico. Nalguns casos é útil utilizar outros operadores, existentes também na selecção natural, tais como recessão e domínio, partilha e inversão [Goldberg89]. No que se segue vamos cingir-nos aos operadores genéticos canónicos.

3.2.2.1. Selecção

Com o processo de selecção pretendem-se atribuir mais hipóteses reprodutivas aos membros da população mais adaptados, existindo formas diversas para a sua implementação, sendo todas elas utilizadoras da função de adaptação. As duas formas mais utilizadas num processo de selecção são os da "roleta viciada" (roulette whell selection) e do "torneio estocástico" (stochastic tournament) [Goldberg89].

No processo da "roleta viciada", associa-se a cada cromossoma da população uma ranhura duma roleta, sendo o tamanho da ranhura proporcional ao valor da função de adaptação do cromossoma. Simulando o funcionamento de uma roleta,

seleccionamos os indivíduos de que necessitamos para os restantes operadores genéticos. Um fluxograma de um algoritmo para a selecção de um indivíduo através do processo da roleta viciada está representado na figura 3.8.

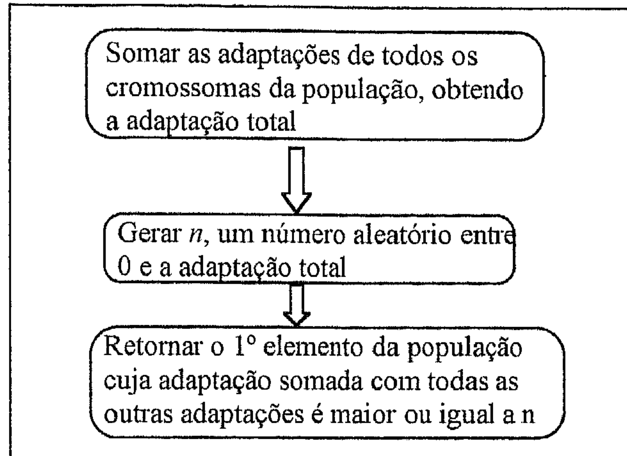


Figura 3.8. Fluxograma do algoritmo de selecção através da roleta viciada

No processo do "torneio estocástico" as probabilidades de selecção são calculadas como habitualmente, de forma proporcional à função de avaliação. Após o cálculo das probabilidades é seleccionado de forma aleatória um número de indivíduos igual à ordem do torneio, seleccionando o melhor deles, com uma determinada probabilidade (p_{te}). Num torneio de ordem dois são utilizados valores típicos entre 0,7 e 0,9. Apresenta-se o fluxograma de um algoritmo na figura 3.9.

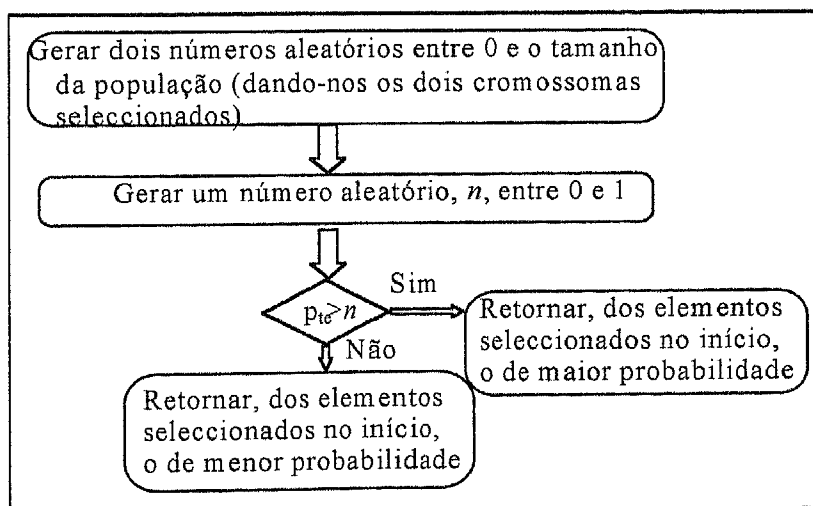


Figura 3.9. Fluxograma de um Algoritmo de um torneio estocástico de ordem 2

Ambos os métodos apresentam algumas desvantagens. No método da "roleta viciada", à medida que as gerações vão evoluindo, vão sendo eliminados os cromossomas com menor função de avaliação, os seus valores de adaptação aproximam-se, tornando-se a roleta viciada num método de selecção puramente aleatório, implicando a possível convergência prematura do algoritmo. Existem métodos, já referidos, para calcular a função de adaptação, de modo a evitar esta situação. No torneio estocástico existe maior facilidade de propagação de más soluções, embora este efeito tenda a diluir-se à medida que o número de gerações aumenta. Além disso, visto que necessita apenas de informação local, é fácil de implementar em paralelo. Uma vez que o âmbito desta tese é o da paralelização, escolhemos o torneio estocástico de ordem 2 como processo de selecção.

Após o processo de selecção ter levado a cabo a construção da população intermédia, esta pode ser sujeita aos operadores de cruzamento e mutação.

3.2.2.2. Cruzamento

O operador de cruzamento é efectuado a partir de dois cromossomas escolhidos aleatoriamente na população intermédia, a que chamamos "pais", dando lugar a dois novos cromossomas a que chamamos "descendentes". O operador de cruzamento mais simples inicia-se pela escolha aleatória de uma posição de cruzamento, k, efectuando-se de seguida a recombinação dos dois cromossomas, conforme apresentado na figura 3.10.

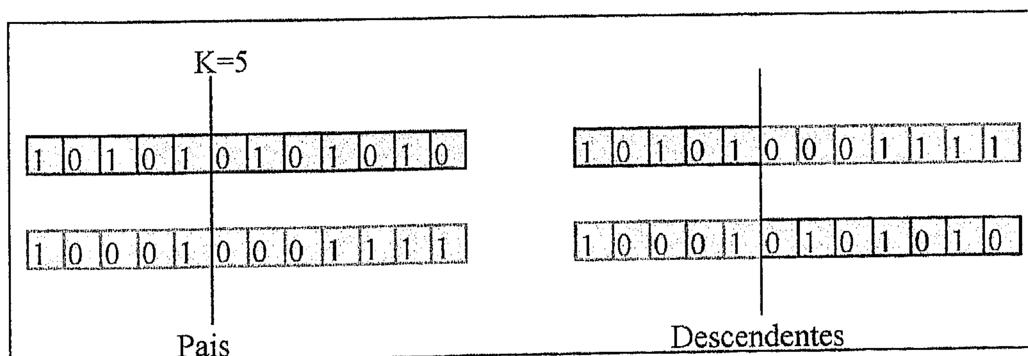


Figura 3.10. Operador de cruzamento, com um ponto

Existem operadores de cruzamento nos quais se define o número de pontos de cruzamento. De Jong, no seu importante trabalho de experimentação, concluiu que o desempenho dos algoritmos diminui à medida que aumentam o número de pontos de cruzamento.

Nem todos os cromossomas são sujeitos a este operador sendo, nesse caso, copiados integralmente para a geração seguinte. Ao operador de cruzamento está associada uma determinada probabilidade, p_c, que toma tipicamente valores próximos de 0,8. Será este o valor que consideraremos neste exemplo.

3.2.2.3. Mutação

O operador de mutação consiste simplesmente na troca de um dos *bits* escolhidos aleatoriamente num cromossoma, com uma probabilidade muito baixa, p_m. Tipicamente p_m toma valores próximos a 0,01. Na figura 3.11. encontra-se o esquema de um operador de mutação.

Apesar de apresentar baixa probabilidade, o operador de mutação é muito importante, na medida em que introduz material genético novo que, ou se perdeu pela utilização de critérios de selecção demasiado rigorosos, ou nunca fez parte do espaço de possíveis soluções.

Nalguns casos, o operador de mutação consiste na troca de dois *bits* entre si. Na prática, este operador consiste em aplicar o primeiro duas vezes, pelo que a probabilidade de mutação deve ser aproximadamente 0,005.

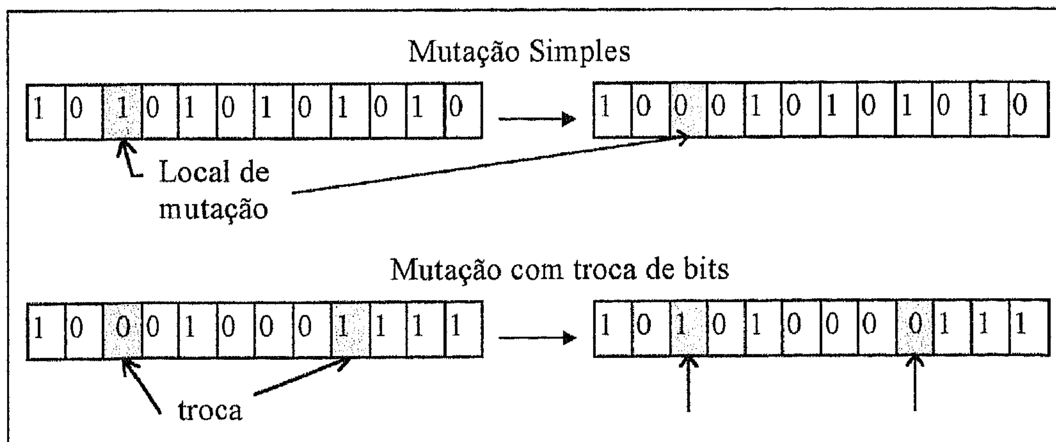


Figura 3.11. Operadores de Mutação

Após a completa aplicação destes três operadores (selecção, cruzamento e mutação) temos uma nova população que pode ser avaliada. Este processo dá lugar a uma nova geração do algoritmo genético.

3.2.2.4. A Geração Seguinte

A geração seguinte vai apresentar soluções que se aproximaram do óptimo, relativamente à geração anterior. Nos gráficos apresentados nas figuras 3.6. e 3.7. podemos verificar como os valores da função dos vários cromossomas se vão aproximando do máximo.

Nos processos descritos até este momento, podemos substituir todos os cromossomas presentes numa determinada geração, para obtenção da geração seguinte, mas pode acontecer que neste processo se percam as melhores soluções do problema! Existem algumas estratégias que pretendem evitar que isto aconteça [Davis91]. Uma dessas estratégias chama-se elitismo e consiste em copiar para a geração seguinte o melhor cromossoma da geração corrente. Vamos utilizá-la na implementação do nosso exemplo.

3.2.2.5. Fim da Evolução

O resultado final, que se pretende seja o "*cromossoma perfeito*", depende dos vários parâmetros. Mas como sabemos quando parar? Existindo vários critérios de paragem salientamos os mais usados, podendo sempre acontecer a combinação dos vários critérios:

- estabelecer um número determinado de gerações;
- estabelecer um tempo limite;
- estabelecer um limite de qualidade;
- convergência dos valores da população, utilizando por exemplo o desvio padrão.

No nosso exemplo, utilizamos o primeiro critério, parando ao fim de 30 gerações. A evolução do algoritmo é apresentada no gráfico da figura 3.12. Podemos

verificar que o algoritmo evoluiu rapidamente para o óptimo, tendo este sido atingido por volta da geração 11. Verificamos, ainda que a partir de determinada geração (cerca da geração 22) todos os cromossomas coincidem com o óptimo. Concluímos que se trata de um comportamento muito bom. Na secção 3.2.5. vamos ver qual a variação do comportamento do algoritmo quando sujeito a variações nos seus diversos parâmetros e verificaremos que continua com um comportamento muito bom.

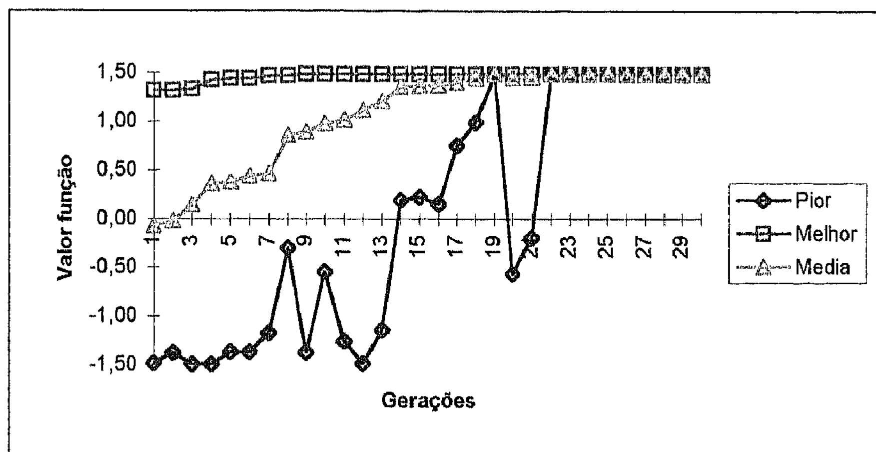


Figura 3.12. Evolução do algoritmo genético tradicional

3.2.3. Fundamentos Matemáticos dos Algoritmos Genéticos

A questão que a maioria das pessoas que se inicia no estudo dos algoritmos Genéticos se coloca nesta altura é "Porque é que um processo como este faz algo de útil? Porque devo acreditar que isto vai resultar numa forma de optimização?" Holland, em 1975, propôs uma teoria para explicar como é que um algoritmo genético pode resultar numa pesquisa complexa e robusta, no espaço de soluções. O modelo teórico é importante pois permite uma maior compreensão do funcionamento dos algoritmos genéticos, o que pode ajudar a melhorá-los.

Um algoritmo genético procura semelhanças entre cromossomas de uma população e relações entre estas semelhanças e uma adaptação elevada. Surge assim a noção de esquema (Schema). Um esquema é um padrão de semelhanças que descreve um conjunto de cadeias com similaridades em determinadas posições.

3.2.3.1. Notações e considerações gerais

Dado que uma grande parte dos estudos teóricos efectuados sobre algoritmos genéticos se baseia na codificação binária, vamos considerar, sem perda de generalidade, que as nossas cadeias de informação são constituídas por elementos do alfabeto $V = \{0,1\}$. O número de cadeias de tamanho L que se podem construir utilizando este alfabeto é 2^L. [Whitley93]

Para ajudar a definir o conceito de esquema (também designado por padrão ou hiperplano) acrescentamos ao alfabeto V o símbolo "*", um metacaracter que pode tomar qualquer valor do alfabeto V. Podemos agora criar cadeias no alfabeto ternário $V' = \{0,1,*\}$ designadas por esquemas (schemata). Por exemplo, $\langle *1* \rangle$ e $\langle *10 \rangle$ são esquemas no alfabeto V′. Um cromossoma específico pode ser interpretado como uma representação de um grande número de esquemas, ou seja, o cromossoma $\langle 110 \rangle$ é uma representação dos dois esquemas referidos (entre outros). Assim, $\langle *10 \rangle = \{\langle 110 \rangle, \langle 010 \rangle\}$ e $\langle *1* \rangle = \{\langle 010 \rangle, \langle 011 \rangle, \langle 110 \rangle, \langle 111 \rangle\}$.

Verificamos que os esquemas não são todos iguais, sendo uns mais específicos do que outros. Por exemplo, considerando cadeias de tamanho 5, $\langle 1*01* \rangle$ é mais "definitivo" do que $\langle 1**** \rangle$. Por outro lado, alguns esquemas ocupam uma percentagem maior do comprimento total da cadeia. Por exemplo, $\langle 1***0 \rangle$ é mais "comprido" do que $\langle 1*0** \rangle$. Surge assim a necessidade de introduzir duas propriedades dos esquemas: a ordem e o comprimento.

A ordem de um esquema, *H*, que representaremos por *O(H)*, é o número de posições fixas presentes. Assim, $O(\langle 1*0** \rangle) = 2$ e $O(\langle 1***0 \rangle) = 2$. O comprimento de um esquema, que representaremos por $\delta(H)$ é a distância entre a primeira e a última posições fixas da cadeia. Vem então que, $\delta(\langle 1***0 \rangle) = 5 - 1 = 4$ e $\delta(\langle 1*0** \rangle) = 3 - 1 = 2$.

Podemos obter uma visualização geométrica destes esquemas (de tamanho 3), como hiperplanos num espaço tridimensional [Goldberg89]. Na figura 3.13. temos a representação do espaço de pesquisa no espaço tridimensional. Os pontos no espaço são cadeias tridimensionais ou esquemas de ordem três, as linhas são esquemas de ordem dois, os planos são esquemas de ordem um e a totalidade do espaço é o esquema de ordem zero, $\langle *** \rangle$.

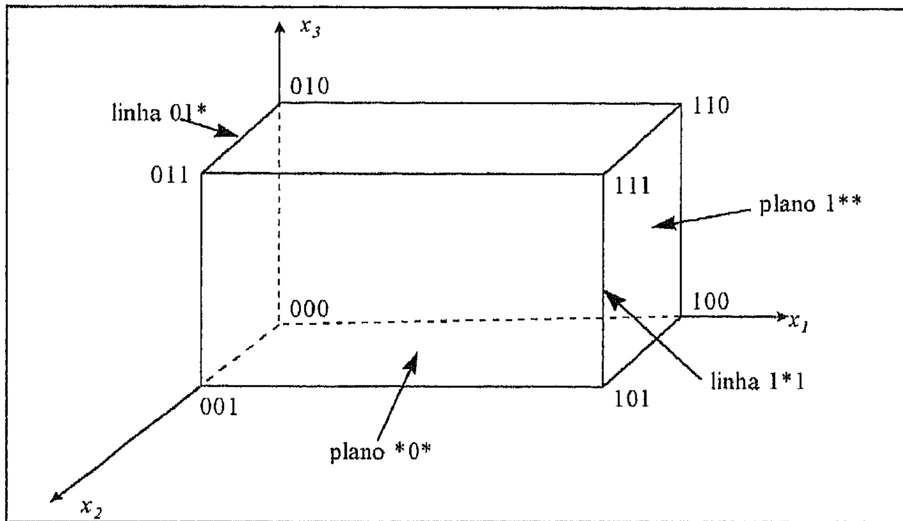


Figura 3.13. Visualização dos esquemas como hiperplanos no espaço tridimensional

Este resultado generaliza-se a espaços de dimensão *n*, sendo *n* o tamanho das cadeias (cromossomas). Podemos assim pensar num algoritmo genético como abrindo caminho através de diferentes hiperplanos, tentando melhorar o seu rendimento.

Quantos esquemas existem? No caso considerado, em que temos um alfabeto de cardinalidade[9] $k=2$ e cadeias de comprimento $L=3$, vem que o número de esquemas é $(2+1)^3 = 3^3$. No caso geral, temos que o número de cadeias que podemos construir, com um alfabeto de cardinalidade k, é k^L e o número de esquemas é $(k+1)^L$. Parece-nos porém, que as coisas se complicam, ao invés de se tornarem mais simples. Acontece que os esquemas e as suas propriedades são ferramentas notacionais importantes, visto permitirem considerar informação extra na análise do algoritmo genético, pois além da informação contida nas cadeias de informação, consideram as relações entre elas. Esta análise, que Holland explorou com o teorema dos esquemas

[9] Cardinalidade de um alfabeto é o número de símbolos que o constituem.

(que apresentaremos na secção seguinte), passa por verificar o efeito dos vários operadores genéticos sobre os esquemas.

3.2.3.2. O Teorema dos Esquemas

Baseando-se nos processos de selecção e nos operadores de cruzamento e mutação, Holland em 1975, estabeleceu o Teorema Fundamental dos Esquemas. Este estabelece o efeito, sobre o valor esperado para o número de esquemas, dos vários operadores genéticos em conjunto.

Este teorema assume como pressupostos a existência de uma codificação binária, a existência de um processo de selecção proporcional e operadores de cruzamento e mutação clássicos [Goldberg89].

Teorema de Holland:

O número esperado de cópias, *m*, do esquema *H*, na geração *t+1*, é limitado pela expressão:

$$m(H,t+1) \geq m(H,t)\frac{f(H)}{\bar{f}}\left(1 - p_c\frac{\delta(H)}{L-1} - p_m O(H)\right)$$

sendo

$m(H,t)$ o número de exemplares do esquema H no instante t

$f(H) = \frac{\sum_{c_i \in H} f(c_i)}{m(H,t)}$ a adaptação média do esquema H; c_i é uma cadeia, representando o esquema H

$\bar{f} = \frac{\sum_{j=1}^{n} f_j}{n}$ a adaptação média da população $A(t)$ de tamanho n

p_c a probabilidade de cruzamento

p_m a probabilidade de mutação

$\delta(H)$ o comprimento do esquema H

$O(H)$ a ordem do esquema H

Prova-se, através deste teorema [Goldberg89], que esquemas curtos, de pequena ordem e com adaptação acima da média, têm uma maior probabilidade de sobrevivência, o que leva a que o seu número aumente exponencialmente de geração em geração, difundindo-se rapidamente pela população. Os esquemas com estas características chamam-se "blocos com significado" (building blocks). Prova-se ainda, que o número de esquemas processados de forma útil em cada geração é da ordem de n^3, enquanto que o algoritmo genético processa apenas n em cada geração. Dado que esta vantagem no processamento é tão importante (e aparentemente só presente nos algoritmos genéticos), atribuiu-se-lhe o nome de paralelismo implícito.

3.2.3.3. Críticas ao Teorema dos Esquemas

Apesar dos algoritmos genéticos serem usados com sucesso numa grande variedade de problemas o Teorema dos Esquemas não é, em si mesmo, uma condição necessária ou suficiente de convergência para um problema arbitrário. Foram-lhe feitas algumas críticas, nomeadamente por [Peek93]:

- a falta de exactidão da desigualdade é tal que, tentando utilizar o teorema para prever a representação de um esquema particular ao longo de várias gerações, o resultado dessa previsão seria em muitos casos inútil ou ilusório;
- o teorema não está adaptado ao comportamento dinâmico do algoritmo genético;
- há dificuldades de adaptação do teorema às modificações ao algoritmo genético canónico.

Em consequência destas críticas surgiram novos trabalhos, pretendendo criar uma teoria mais robusta e completa. Enquanto que alguns trabalhos visam adaptar o Teorema dos Esquemas a novos operadores ou melhorar as aproximações que ele fornece [Peek93], outros tentam integrar outros métodos, tais como os algoritmos de simulação de têmpera [Mahfouhd93a]. Estamos perante uma área muito vasta e da qual se espera ainda uma grande evolução. O teorema dos esquemas teve, apesar de tudo, o

grande mérito de criar uma base teórica para os algoritmos genéticos, base esta que é fundamental para o seu desenvolvimento.

3.2.4. A codificação

Num certo sentido, a codificação de um problema, para uma pesquisa com um algoritmo genético não é problemática, estando amplamente limitada pela imaginação do programador. No entanto, como os algoritmos genéticos são robustos, as decisões de codificação não são demasiado importantes.

Existem alguns aspectos úteis na escolha da codificação para um algoritmo genético[Goldberg89]:

- deve ser seleccionado um alfabeto de tamanho mínimo que permita uma expressão natural para o problema; prova-se que num alfabeto de menor cardinalidade, a quantidade de informação disponível para ser processada pelo algoritmo genético é mais elevada ;
- a codificação deve ser fechada relativamente aos operadores de cruzamento e mutação, isto é, os "descendentes" por eles gerados não devem ser "impossíveis"; caso não seja possível evitar esta situação, os "descendentes" devem ser fortemente penalizados pela função de avaliação;
- as semelhanças entre conceitos idênticos devem ser preservadas ; por exemplo, utilizando código Gray e não codificação binária pura.

Seguindo estas orientações é fácil , normalmente, encontrar uma boa codificação de um qualquer problema através de uma cadeia de bits (cromossomas).

3.2.5. Parametrização de um algoritmo genético

A parametrização de um algoritmo genético consiste na definição de alguns parâmetros do algoritmo que habitualmente são os seguintes:

- Tamanho da população - *pop*

  este depende da complexidade do problema - populações de pequena dimensão convergem rapidamente mas a informação genética pode ser

insuficiente para gerar soluções satisfatórias; por seu lado populações de dimensão elevada, apesar da diversidade de informação que contêm podem não convergir em tempo útil. 100 é um valor típico para n, embora em problemas complexos possamos considerar populações de 1000 ou mais indivíduos.

- Número de gerações - G

  está relacionado com o grau de convergência do algoritmo e, por sua vez, com a complexidade do problema e com o tamanho da população; até certo ponto a evolução é muito rápida, tendendo depois para a estabilização.

- Probabilidade de cruzamento - p_c

  como já vimos, a probabilidade de cruzamento deve ser elevada; para aumentar a recombinação de soluções, um valor típico é 0,8; não deve ser demasiado elevada pois aumenta a probabilidade de destruição de bons esquemas já construídos.

- Probabilidade de mutação - p_m

  como já vimos, a probabilidade de mutação deve ser mantida em valores baixos, pois o seu aumento tende a transformar o algoritmo genético num algoritmo de pesquisa aleatória. Um valor típico é 0,01.

Todos estes parâmetros estão dependentes dos operadores utilizados e do ponto de execução do algoritmo. Em [Davis91] propõe-se que o próprio algoritmo genético faça a auto-adaptação dos valores dos parâmetros.

Apresentam-se, de seguida, os resultados da alteração dos três parâmetros, *pop*, p_c e p_m, no exemplo introduzido na página 37. Verificamos que, de uma forma geral, o algoritmo converge rapidamente para o valor máximo da função e, a partir de determinada geração, todos os cromossomas são esse valor. Assim, utilizamos como medida de comparação a média de todos os valores em cada geração.

Quanto ao parâmetro p_c verificamos, por observação do gráfico da figura 3.14. que o algoritmo apresenta um bom comportamento global nos quatro casos considerados, acontecendo o melhor resultado para o valor 0,9 , confirmando-se a

necessidade de valores elevados para o parâmetro. É de salientar o bom comportamento do algoritmo para uma probabilidade de cruzamento de 0,6.

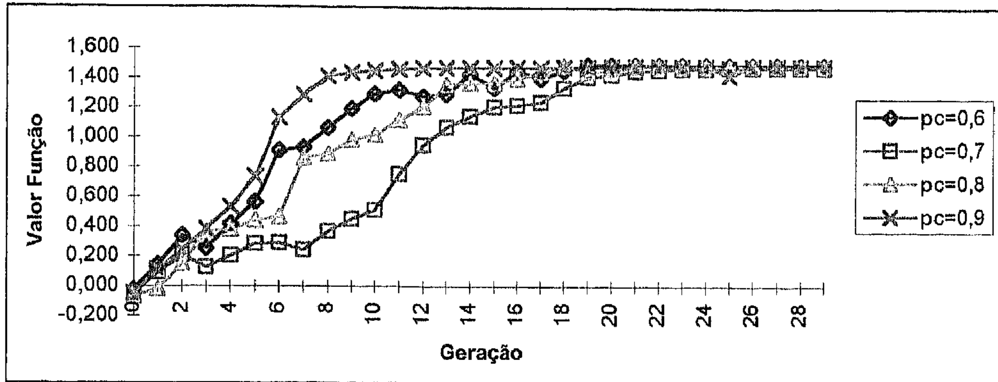


Figura 3.14. Algoritmo Genético: variação de p_c; p_m=0,01; p_{te}=0,8; pop=50

Quanto à variação da probabilidade de mutação, cujos efeitos estão registados no gráfico da figura 3.15., verificamos que não há praticamente diferença na evolução do algoritmo, quando a probabilidade de mutação toma os valores 0,005 e 0,01. O melhor comportamento acontece para uma probabilidade de mutação de 0,04.

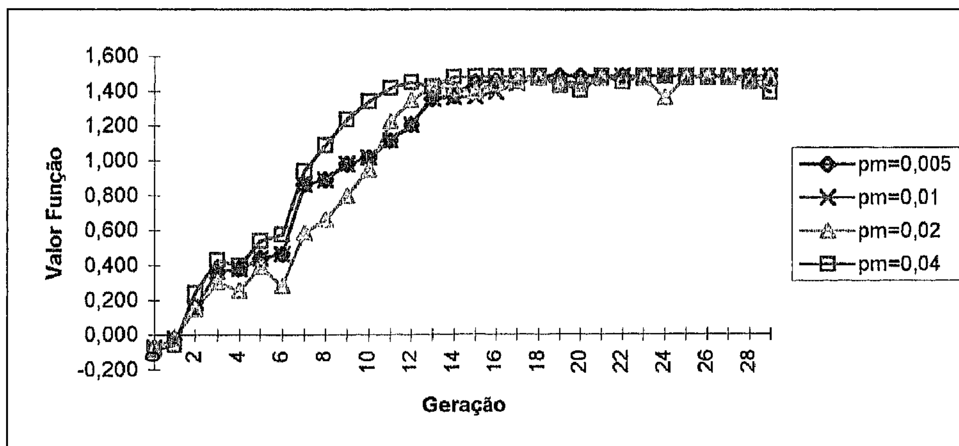


Figura 3.15. Algoritmo Genético: variação de p_m; p_c=0,8; p_{te}=0,8; pop=50

Quando a probabilidade de seleccionar o melhor no torneio estocástico é 0,9, verificamos uma melhoria significativa do comportamento do algoritmo, pelo que concluímos da vantagem de um torneio estocástico de ordem elevada (figura 3.16).

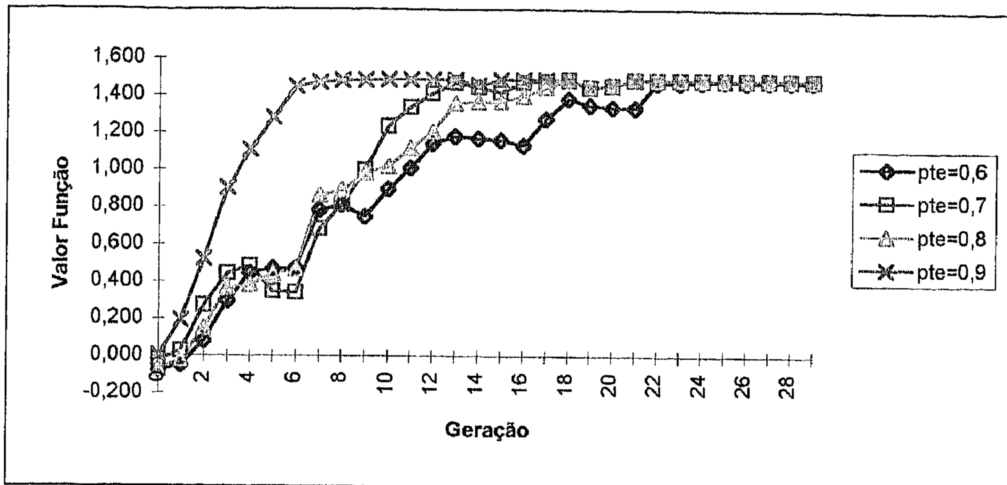


Figura 3.16. Algoritmo Genético: variação de p_{te}; p_c=0,8; p_m=0,01; pop=50

Por fim, testamos a evolução do algoritmo alterando o tamanho da população. Os resultados estão registados no gráfico da figura 3.17. Verifica-se que o algoritmo converge para um ponto diferente do óptimo, quando o tamanho da população é pequeno, devido ao facto de esta não conter informação suficiente para a evolução do algoritmo na forma desejada. No caso de uma população grande, verifica-se que o algoritmo converge rapidamente para o óptimo. Assim sendo, verificamos que o tamanho da população é crucial para o bom desempenho de um algoritmo genético.

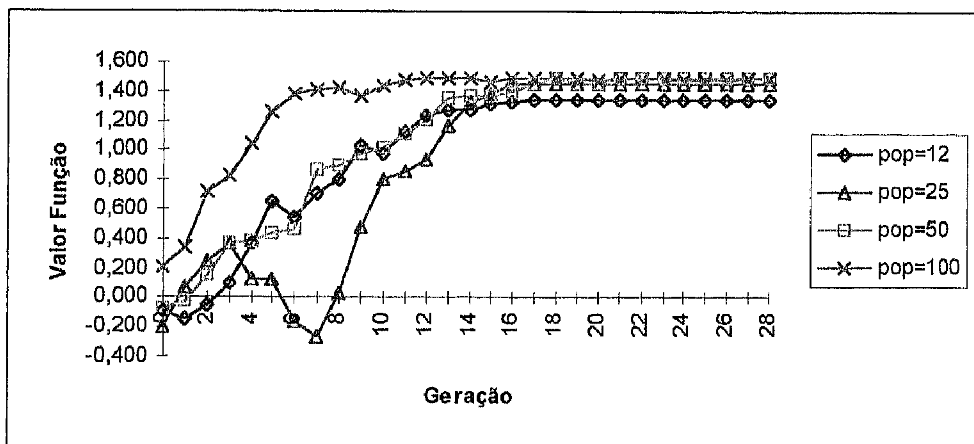


Figura 3.17. Algoritmo Genético: variação de pop; p_c=0,8; p_m=0,01; p_{te} =0,8

Confirma-se a necessidade de probabilidades de cruzamento elevadas, de baixas probabilidades de mutação e de que a probabilidade de escolher o melhor seja elevada.

Além disso, verifica-se uma forte influência do tamanho da população no comportamento do algoritmo.

3.3. Alguns conceitos avançados

Têm sido desenvolvidos alguns trabalhos que visam melhorar o desempenho dos algoritmos genéticos em grupos específicos de problemas. Vamos debruçar-nos sumariamente sobre alguns desses trabalhos. Em 3.3.1. é feita uma abordagem aos algoritmos genéticos híbridos. Na secção 3.3.2. é introduzida a programação genética, criada por Koza. Para finalizar, na secção 3.3.3. vamos falar de um tipo especial de problemas baseados em permutações.

3.3.1. Algoritmos Genéticos Híbridos

Como já vimos atrás, um algoritmo genético não é, necessariamente, o melhor método para resolver certos tipos de problemas. A sua capacidade de, independentemente do problema, pesquisar através do espaço de soluções, procurando a melhor solução (apesar da vantagem da robustez do método) apresenta a desvantagem de não utilizar de forma útil toda a informação sobre o problema a resolver. Surgem assim os algoritmos genéticos híbridos. A filosofia subjacente a este tipo de algoritmos é a de aproveitar as boas características de algum bom método já conhecido para o problema em causa e também as boas características dos algoritmos genéticos. Combinando os dois métodos da melhor forma, obtemos melhores soluções do que aqueles que obteríamos se os utilizássemos separadamente.

A forma mais utilizada consiste na combinação do algoritmo genético com um método de "subir a colina". Dado que o sucesso deste tipo de métodos depende do ponto escolhido inicialmente, podemos utilizar em primeiro lugar o algoritmo genético para descobrir uma boa solução inicial e depois "subir a colina".

Os algoritmos genéticos híbridos foram já utilizados com sucesso em alguns casos práticos [Davis91]; a teoria sobre o assunto ainda está pouco explorada existindo, no entanto, alguns trabalhos na área, nomeadamente [Lobo96].

3.3.2. Programação Genética

Koza tem sido o percursor de um novo paradigma a que chamou Programação Genética [Koza92], baseando-se na ideia de que um programa pode evoluir através de uma evolução genética, adaptando-se da melhor forma ao seu ambiente. Muitos problemas da Inteligência Artificial, da regressão simbólica, da programação automática, entre outros, podem ser vistos como programas que requerem a descoberta de outros programas de computador que produzam saídas desejáveis como resposta a entradas particulares. Koza reformulou este tipo de problemas, como sendo uma pesquisa (no espaço de todos os programas possíveis), daqueles que melhor se adaptam aos resultados pretendidos. A linguagem de programação utilizada, para os programas gerados pela Programação Genética é o LISP, devido à sua fácil utilização para o problema em causa. Segundo Koza o paradigma da Programação Genética gera programas de computador, para resolver problemas, através da execução dos três passos seguintes:

1. Gerar uma população inicial de composições aleatórias de funções e terminais do problema (programa de computador);
2. Executar iterações, até atingir determinado critério, dos seguintes sub-passos:
    a) Executar cada programa da população e atribuir-lhe um valor de adaptação, de acordo com o modo de resolver o problema;
    b) Criar uma nova população de programas de computador, aplicando as duas operações primárias [em baixo (i e ii)]. As operações são aplicadas ao(s) programa(s) na população, escolhidos com uma probabilidade baseada na adaptação:
        i) Copiar programas existentes para a nova população;
        ii) Criar novos programas de computador através da recombinação genética de partes do programa, escolhidos aleatoriamente;
3. O melhor programa que até ao momento tenha aparecido em qualquer geração é designado como sendo o resultado da Programação Genética. Este pode ser a solução do problema.

Este paradigma tem sido aplicado com sucesso a grande variedade de problemas em vastas áreas da programação [Koza92].

3.3.3. Problemas baseados em permutações

Existem certos tipos de problemas como por exemplo, o problema do Caixeiro Viajante [Lawler85] ou o de Sequencialização de Tarefas [Anderson94], dependentes da ordem, sendo a sua representação baseada em permutações. Este tipo de problemas cria algumas dificuldades ao algoritmo genético canónico. Tomemos o exemplo do problema do caixeiro viajante, já sumariamente descrito no capítulo 1. Representando cada cidade por um número, uma permutação das cidades indica um percurso do Problema do Caixeiro Viajante.

Sendo 1, 2, 3, 4, 5, 6, 7, 8, 9 nove cidades a percorrer, podemos fazê-lo das seguintes formas (entre outras):

9 8 5 3 1 2 4 6 7 e 1 3 5 7 9 2 4 6 8

Considerando estas permutações como cromossomas do algoritmo genético e efectuando um simples cruzamento, de um ponto com k=4, obtemos os descendentes seguintes:

9 8 5 3 9 2 4 6 8 e 1 3 5 7 1 2 4 6 7

Por simples observação dos descendentes verificamos que estes não poderão constituir soluções admissíveis para o problema, pois, por exemplo, segundo o primeiro descendente visitamos duas vezes a cidade 9. Vários autores encontraram diversos operadores de cruzamento e mutação especiais, para obviar a esta situação. Destes destacamos os seguintes: o "cruzamento de correspondências parciais" (partially matched crossover), o "cruzamento baseado na ordem" (order crossover), o "cruzamento cíclico" (cycle crossover) e operadores de inversão (inversion). [Goldberg89] [Davis91]

Visto que nós pretendemos estudar o comportamento de um algoritmo genético quando ele é paralelizado, pensamos ser de grande utilidade a utilização do algoritmo genético canónico e, sendo assim, vamos concentrar os nossos esforços numa representação (codificação) para as permutações, a tabela de inversões [knuth73], que

permite a utilização do Algoritmos Genéticos Canónicos. Vamos, em primeiro lugar, fazer uma abordagem ao Problema do Caixeiro Viajante em 3.3.3.1. e, de seguida, em 3.3.3.2., falaremos sobre a tabela de inversões. Por fim, em 3.3.3.3., descreveremos a nossa aplicação ao Problema do Caixeiro Viajante.

3.3.3.1. O problema do Caixeiro Viajante

O problema do Caixeiro Viajante é um dos problemas de optimização combinatória mais conhecidos. Pertence à classe dos problemas que são fáceis de colocar e difíceis de resolver. Trata-se de minimizar os custos dos percursos que consistem em percorrer um conjunto de cidades, voltando à cidade inicial e passando apenas uma vez por cada cidade, o que corresponde a minimizar os ciclos hamiltonianos num grafo conexo [Ross92].

Este problema, que à partida parece muito fácil, é difícil de resolver, pertencendo ao conjunto dos problemas que são NP-Completos[10] (não são resolúveis em tempo polinomial). Este tipo de problemas são um desafio ideal para os algoritmos genéticos, dado que não se conhece nenhum algoritmo capaz de gerar (no caso geral) a solução óptima sem recorrer à enumeração sistemática das soluções. Para um problema de tamanho ***n***, a quantidade das soluções possíveis é ***n!***. Isto implica que, a partir de uma certa dimensão do problema, o espaço de pesquisa é de tal ordem, que se torna incomportável a determinação da solução óptima. Supondo que dispomos de um computador capaz de processar um milhão de soluções por segundo, obtemos a tabela 3.1[11]. Por observação desta tabela, podemos verificar que a partir de certo número de cidades é incomportável determinar o óptimo através da enumeração de todos os casos possíveis.

[10] Ver [Harel92]
[11] Adaptado da página web http://gaia.uc3m.es/~jmaw/p_lisp1.html

Número de Cidades	Número de soluções	Tempo	Unidade
4	**3**	**0,000003**	**Segundos**
5	**12**	**0,000012**	**Segundos**
6	**60**	**0,000060**	**Segundos**
7	**360**	**0,000360**	**Segundos**
8	**2520**	**0,002520**	**Segundos**
9	**20160**	**0,020160**	**Segundos**
10	**181440**	**0,181440**	**Segundos**
11	**1814400**	**1,814400**	**Segundos**
12	**19958400**	**19,9584**	**Segundos**
13	**239500800**	**3,99168**	**Minutos**
14	**3113510400**	**51,89184**	**Minutos**
15	**$4{,}3589 \times 10^{10}$**	**12,108096**	**Horas**
16	**$6{,}5384 \times 10^{11}$**	**7,56756**	**Dias**
17	**$1{,}0461 \times 10^{13}$**	**121,08096**	**Dias**
18	**$1{,}7784 \times 10^{14}$**	**5,635527**	**Anos**
19	**$3{,}2012 \times 10^{15}$**	**101,43949**	**Anos**
20	**$6{,}0823 \times 10^{16}$**	**19,273503**	**Séculos**
21	**$1{,}2165 \times 10^{18}$**	**385,47006**	**Séculos**
...	...	...	...

Tabela 3.1. Informação sobre o Problema do Caixeiro Viajante

Apesar desta situação, existem casos especiais do Problema do Caixeiro Viajante, para os quais se conhecem métodos que permitem obter uma solução de forma fácil e eficiente. Estes casos especiais acontecem quando a matriz (a_{ij}), contendo os custos de deslocamento da cidade i para a cidade j, apresenta características especiais, por exemplo, é triangular superior. Estes casos especiais podem ainda acontecer quando o Problema do Caixeiro Viajante é resolvido numa rede com características especiais [Lawler85], Cap. 4. O maior Problema do Caixeiro Viajante para o qual se conhece uma solução tem 3038 cidades. A solução foi obtida utilizando técnicas de programação inteira e foi necessário um ano e meio de tempo de computação[12].

Uma outra abordagem, consiste na determinação de boas aproximações num tempo razoável. São conhecidas várias heurísticas eficientes, que permitem atingir este

[12] De CRPC Newsletter; página web http://www.crpc.rice.edu/CRPC/newsletter/Jan93/news.tsp.html

objectivo [Lawler85] Caps.5,6 e 7. Os algoritmos genéticos encaixam-se neste tipo de abordagem, permitindo, em tempo razoável, a determinação de boas aproximações e, por vezes, a determinação do óptimo.

3.3.3.2. A tabela de inversões

Seja $a_1 a_2 \cdots a_n$ uma permutação do conjunto $\{1,2,\ldots,n\}$. O par (a_i,a_j) é uma inversão da permutação se $i<j$ e $a_i > a_j$. Por exemplo, a permutação 1 3 5 7 9 2 4 6 8 tem dez inversões: *(3,2)*; *(5,2)*; *(5,4)*; *(7,2)*; *(7,4)*; *(7,6)*; *(9,2)*; *(9,4)*; *(9,6)*; *(9,8)*. Cada inversão é um par de elementos que está "fora de ordem". A cada permutação, $a_1 a_2 \cdots a_n$, podemos associar uma tabela de inversões, $b_1 b_2 \cdots b_n$, sendo que cada b_j é o número de inversões cujo segundo componente é j, ou seja, o número de elementos à esquerda de j que são maiores que j. Por exemplo, a permutação 1 3 5 7 9 2 4 6 8 tem como tabela de inversões 0 4 0 3 0 2 0 1 0.

A partir desta definição podemos concluir que uma tabela de inversões que represente uma permutação terá de satisfazer as condições:

$$\begin{array}{c} 0 \le b_n \le n-1 \\ 0 \le b_2 \le n-2 \\ \cdots \\ 0 \le b_{n-1} \le 1 \\ b_n = 0 \end{array} \qquad \text{[Condição 3.1.]}$$

Um resultado importante é que uma tabela de inversões define de forma única uma permutação. Assim, a partir de uma tabela de inversões podemos obter de forma única a permutação que lhe corresponde determinando sucessivamente as posições dos elementos *n*, *n-1*, ... *1*, conforme o algoritmo representado na figura 3.18.

Por exemplo, considerando a tabela de inversões

3 2 2 1 4 3 2 1 0

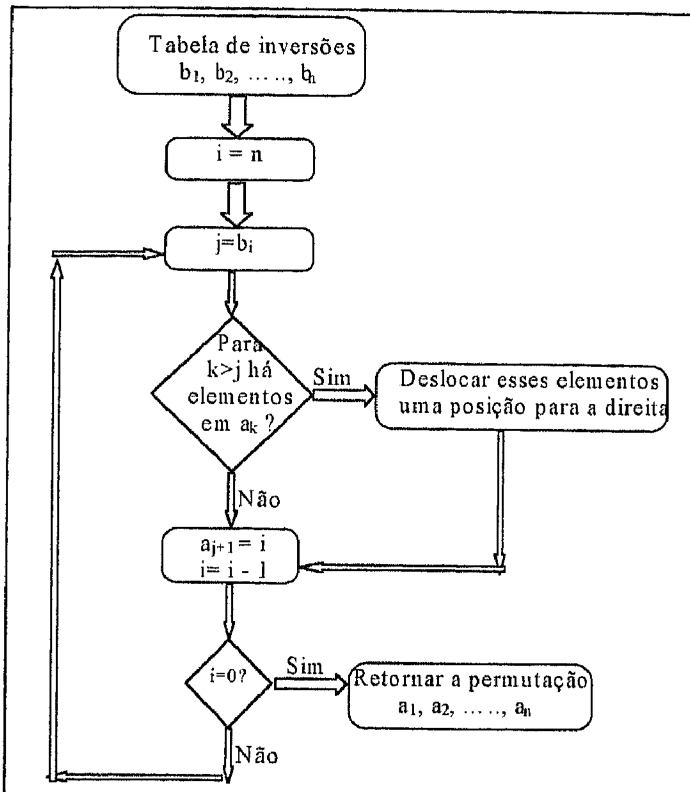


Figura 3.18. Algoritmo para converter uma tabela de inversões na permutação correspondente

vem:

Passo 1	9	
Passo 2	9 8	(porque b_8=1)
Passo 3	9 8 7	(porque b_7=2)
Passo 4	9 8 7 6	(porque b_6=3)
Passo 5	9 8 7 6 5	(porque b_5=4)
Passo 6	9 4 8 7 6 5	(porque b_4=1)
Passo 7	9 4 3 8 7 6 5	(porque b_3=2)
Passo 8	9 4 2 3 8 7 6 5	(porque b_2=2)
Passo 9	9 4 2 1 3 8 7 6 5	(porque b_1=3)

Obtemos, desta forma, a permutação correspondente à tabela de inversões dada. A aplicação que relaciona permutações com tabela de inversões é bijectiva pelo que podemos utilizar indiferentemente uma das duas representações, dado que elas são equivalentes. Podemos, então, utilizar esta nova representação nos algoritmos genéticos.

Vejamos quais são os efeitos dos operadores genéticos de cruzamento e mutação sobre a tabela de inversões. Para isso relembremos que as únicas restrições, na tabela de inversões, estão expressas através de [Condição 3.1.]. Quanto ao operador de cruzamento, não surge qualquer problema pois, como se verifica facilmente, as restrições mantêm-se para os dois descendentes. Quanto ao operador de mutação, ao seleccionar um bit para mutação, *k*, ele não pode ser substituído por um número maior do que *n-k*, pois caso contrário não seriam satisfeitas as condições de [Condição 3.1.]. Assim sendo, podemos utilizar os operadores do algoritmo genético, fazendo apenas uma pequena alteração no operador de mutação: o novo *bit* a ocupar a posição de mutação deve ser menor ou igual a n-k. Este facto poderá elevar a probabilidade de mutação.

3.3.3.3. Aplicação ao problema do Caixeiro Viajante

Implementamos de seguida o algoritmo genético canónico para resolução de dois problema do Caixeiro Viajante, retirados da biblioteca TSPLIB[13]. Estes são assimétricos, isto é, a matriz que contém os custos de deslocamento de uma cidade i para uma cidade j não é simétrica, ou seja, $a_{ij} \neq a_{ji}$. Seleccionamos um problema com 17 (br17) cidades e um outro com 70 cidades (ft70), sendo 39 e 38673 as melhores soluções conhecidas, respectivamente [Fischetti96]. Os testes foram realizados utilizando os recursos do CICA, já descritos em 2.2. Utilizamos como representação para os cromossomas a tabela de inversões. Todo o processo é como descrito no algoritmo genético canónico, representado na figura 2.1, utilizando como codificação para os cromossomas a tabela de inversões, sendo que a função de avaliação faz uma descodificação prévia da tabela de inversões para depois calcular os custos associados e introduziu-se a pequena alteração no operador de mutação já referida em 3.3.3.1.

Com o objectivo de estimar a melhor parametrização para o algoritmo, utilizamos vários valores de cada parâmetro, mantendo os outros constantes. Os resultados estão representados nos gráficos das figuras 3.19 a 3.26.

[13] Gerhard Reinhelt; página web http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html

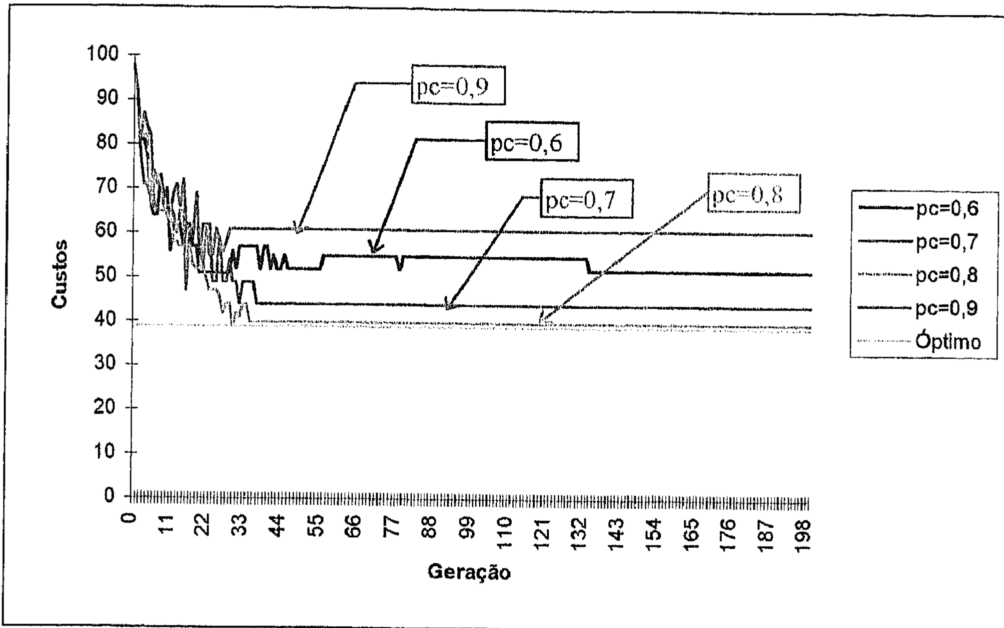


Figura 3.19. Variação do parâmetro p_c para o problema do Caixeiro Viajante de tamanho 17: p_m=0,01;p_{te}=0,8;pop=500

Observando o gráfico da figura 3.19, podemos concluir que quer a probabilidade de cruzamento aumente quer diminua, o comportamento do algoritmo piora. Este facto é mais acentuado no caso da diminuição do parâmetro. Observando a figura 3.20.,

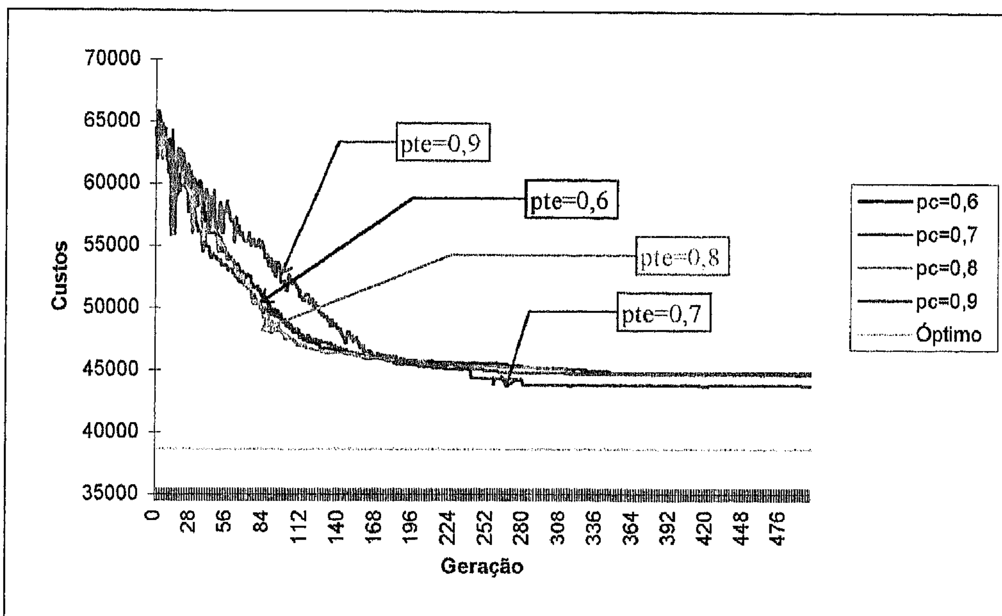


Figura 3.20. Variação do parâmetro p_c para o problema do Caixeiro Viajante de tamanho 70: p_m=0,01;p_{te}=0,8;pop=4000

confirma-se este facto, embora as diferenças não sejam tão acentuadas. Assim,

podemos concluir que um valor de 0,7 ou 0,8 para a probabilidade de cruzamento será a melhor opção.

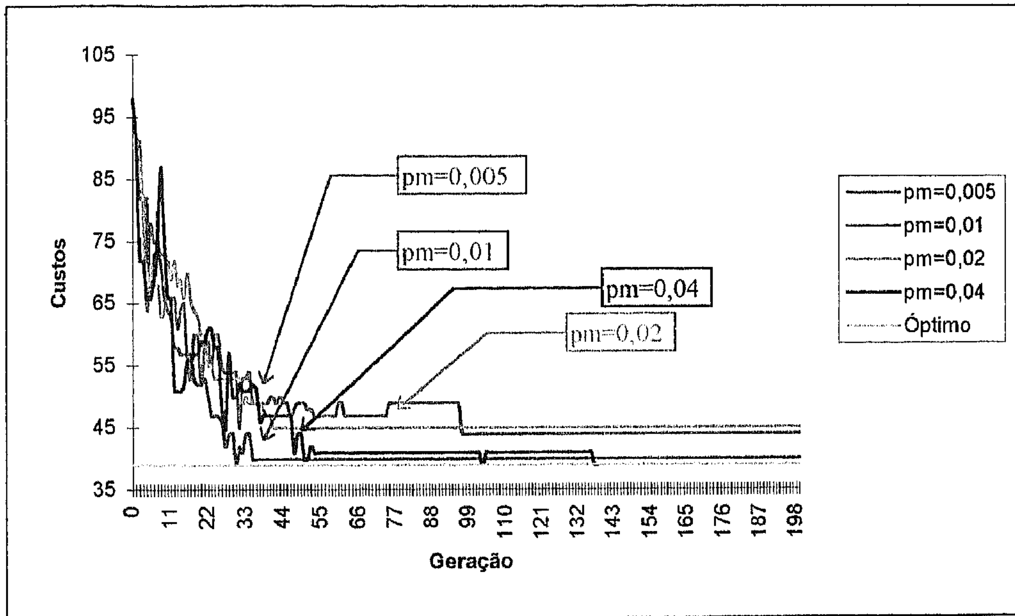


Figura 3.21. Variação do parâmetro p_m para o problema do Caixeiro Viajante de tamanho 17: p_c=0,8;p_{te}=0,8;pop=500

Quanto ao valor da probabilidade de mutação, observando as figuras 3.21 e 3.22, verificamos que no caso da diminuição do parâmetro o comportamento do algoritmo

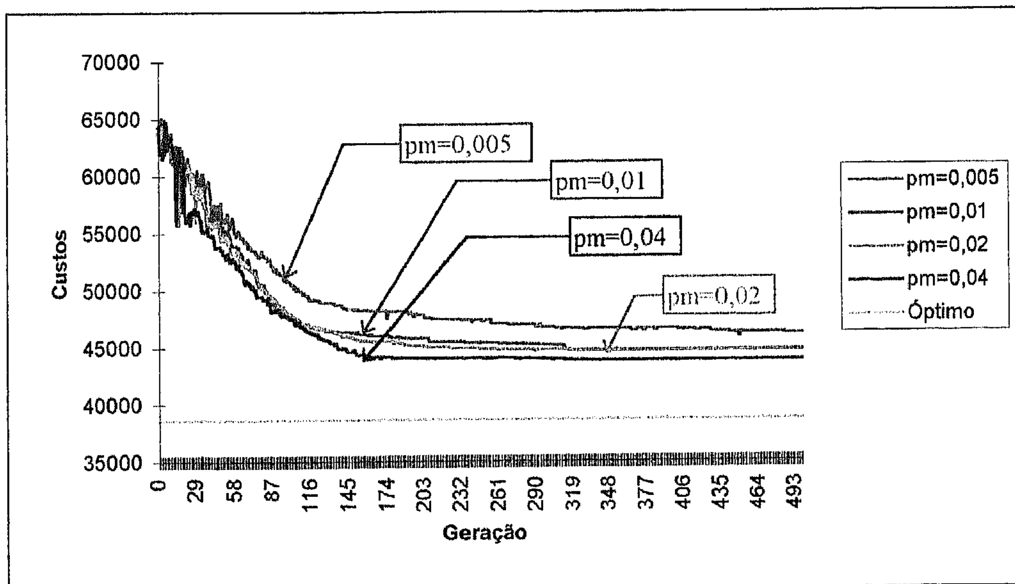


Figura 3.22. Variação do parâmetro p_m para o problema do Caixeiro Viajante de tamanho 70: p_c=0,8;p_{te}=0,8;pop=4000

piora. No caso do aumento do valor do parâmetro o comportamento do algoritmo melhora, levando a que, no caso de tamanho 17 o algoritmo atinja o óptimo bastante

cedo. Assim, pensamos que um bom valor para este parâmetro será um valor de 0,04. Testamos ainda o comportamento do algoritmo para um valor de 0,05 mas ele piorou significativamente.

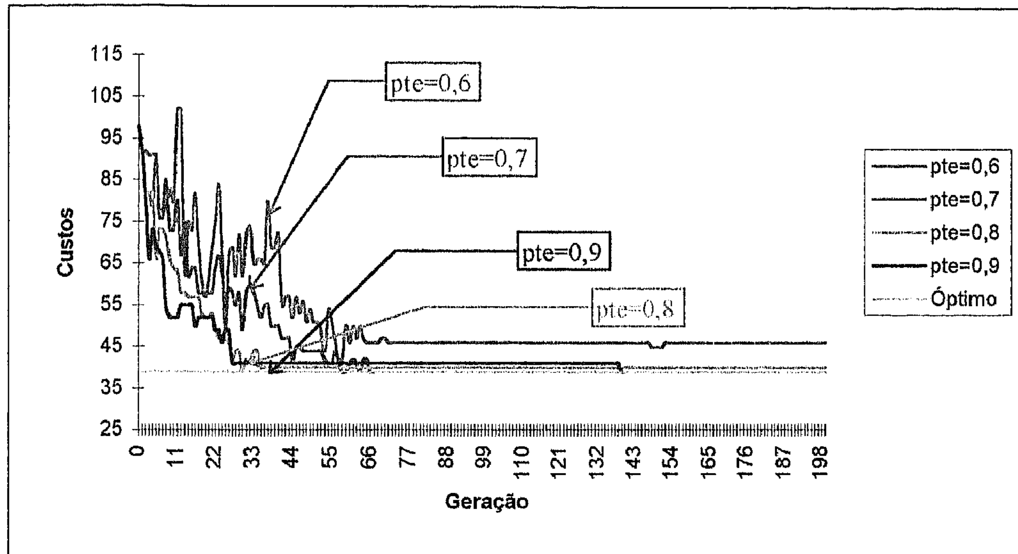


Figura 3.23. Variação do parâmetro p_{te} para o problema do Caixeiro Viajante de tamanho 17: p_c=0,8;p_m=0,01;pop=500

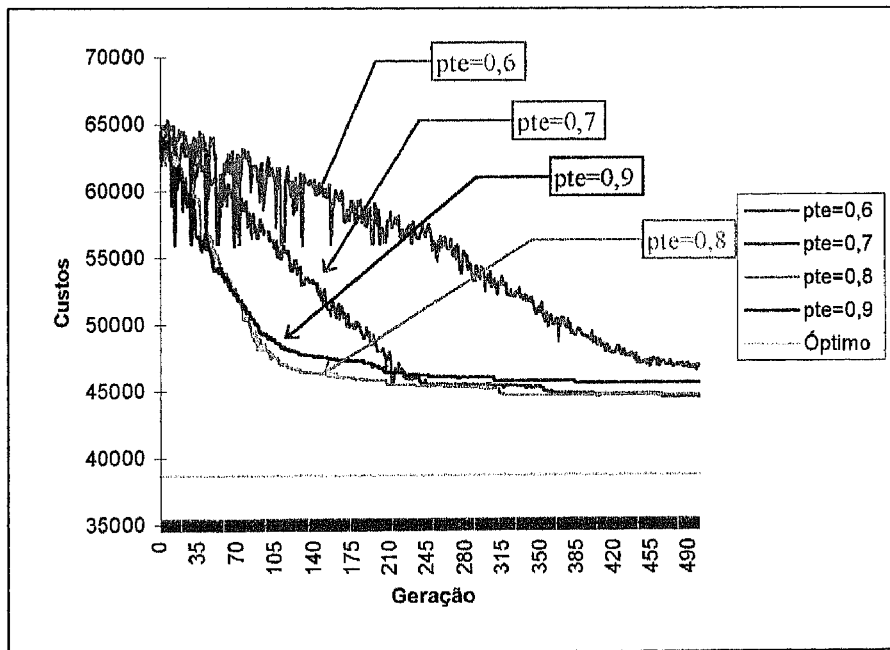


Figura 3.24. Variação do parâmetro p_{te} para o problema do Caixeiro Viajante de tamanho 70: p_c=0,8;p_m=0,01;pop=4000

O comportamento do algoritmo é fortemente influenciado pela probabilidade de, no torneio estocástico, seleccionar o melhor indivíduo (figuras 3.23 e 3.24). No caso em que a probabilidade de seleccionar o melhor é baixa, verificamos que o algoritmo apresenta um comportamento muito fraco, quedando-se muito afastado dos valores

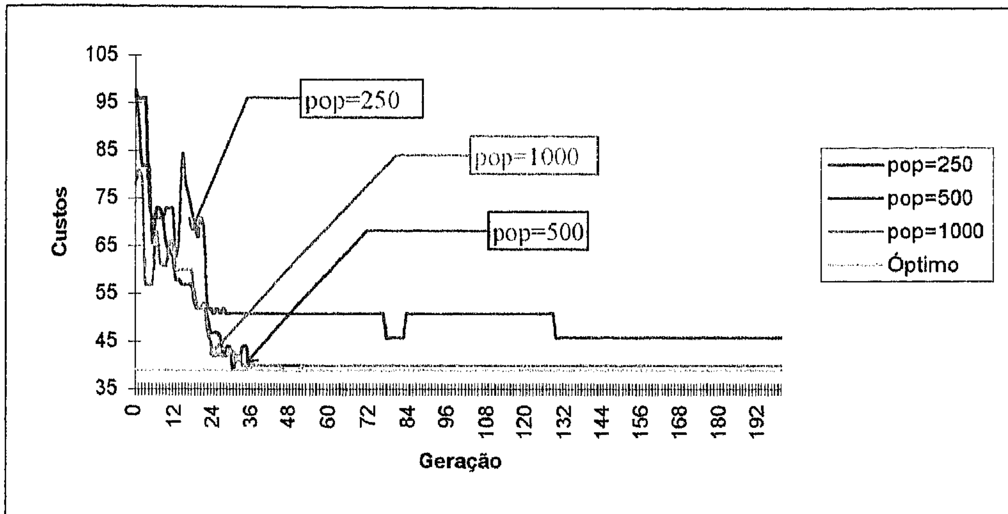


Figura 3.25. Variação do parâmetro *pop* para o problema do Caixeiro Viajante de tamanho 17: p_c=0,8;p_m=0,01;p_{te}=0,8

óptimos. No caso de valores elevados para o parâmetro, verifica-se uma melhoria acentuada. Contudo pensamos que uma probabilidade demasiado alta (0,9) se torna demasiado selectiva, dificultando a aproximação do óptimo, como podemos observar na figura 3.24. Este aspecto poderá ser superado aumentando o valor da probabilidade de mutação.

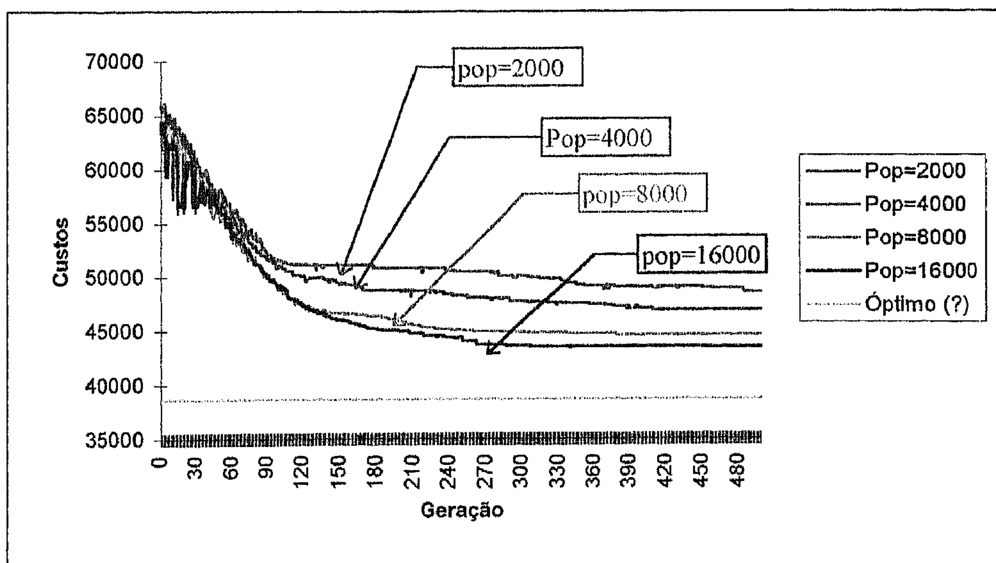


Figura 3.26. Variação do parâmetro *pop* para o problema do Caixeiro Viajante de tamanho 70: p_c=0,8;p_m=0,01;p_{te}=0,8

Verificamos, observando os gráficos das figuras 3.25 e 3.26, que o tamanho da população é de primordial importância para o comportamento do algoritmo. Um aumento no tamanho da população provoca o aproximar do valor óptimo, provocando, no entanto, um aumento significativo nos tempos de execução. Além disso, dado que o hardware tem limites não podemos aumentar indefinidamente o tamanho da população.

Assim, pensando no factor tempo, medimos os tempos de execução em várias situações, com vista a abranger vários períodos de utilização dos recursos do CICA. O número de gerações é o mesmo que o utilizado até agora, sendo de 200 gerações no caso do problema de tamanho 17 e de 500 gerações no caso do problema de tamanho 70. As médias dos tempos obtidos estão registadas na tabela 3.2. Verificamos que quando a população aumenta para o dobro do tamanho, o tempo de execução do algoritmo também aumenta aproximadamente para o dobro.

Número cidades	Tamanho População	Média dos Tempos
17	250	7,1 ''
17	500	16,2''
17	100	34,4''
70	2000	7,8'
70	4000	21,9'
70	8000	35,5'
70	16000	52,8'

Tabela 3.2. Tempos reais de execução do algoritmo

É de salientar que, no caso do problema de tamanho 17, o óptimo foi atingido em cerca de 16 segundos o que consideramos muito bom. No caso do problema de tamanho 70, testamos o algoritmo (no limite dos recursos de memória do sistema) para uma população de 64000 cromossomas, com os valores de 0,8 para a probabilidade de cruzamento, 0,04 para a probabilidade de mutação e 0,8 para a probabilidade de no torneio estocástico ser seleccionado o melhor, pois conforme análise já realizada serão estes os melhores parâmetros para o bom desempenho do algoritmo. O número de gerações foi de 500. Obtivemos o valor de 42300 num tempo de 3h42 nó jerry.fe.up.pt. Verificámos que o algoritmo atingiu este valor, na geração 350 não se alterando mais, o que indica a convergência do algoritmo para esse valor.

Na tabela 3.3., apresentamos a qualidade das soluções obtidas pelo algoritmo para populações de tamanhos 2000, 4000, 8000, 16000 e 64000 cromossomas, indicando a percentagem pela qual a solução excede o óptimo em cada um dos casos.

Tamanho do problema	% pela qual a solução excede o óptimo
2000	25,6 %
4000	21,4 %
8000	15,35 %
16000	12,49 %
64000	9,38%

Tabela 3.3. Qualidade das Soluções obtidas

Podemos verificar que a qualidade das soluções aumenta com o tamanho da população. Os resultados são razoáveis atendendo à dificuldade associada ao Problema em causa e ao facto de utilizarmos apenas o Algoritmo Genético Canónico.

3.4. Notas Finais

Ao longo de todo este capítulo pudemos verificar que os Algoritmos Genéticos fornecem uma boa técnica de optimização, pois podem ser aplicados a uma grande variedade de problemas (desde aqueles que são mais fáceis de optimizar, e para os quais se conhecem já outras boas técnicas, até aqueles mais difíceis de optimizar como é o caso do problema do Caixeiro Viajante). Em ambos os casos, o Algoritmo Genético, por nós implementado, apresenta um bom comportamento.

Podemos assim concluir que os algoritmos genéticos são robustos e apresentam uma técnica de optimização aliciante, dada a sua facilidade de implementação e os bons resultados obtidos. Trata-se, no entanto, de uma área recente, pelo que se esperam grandes evoluções quer na sua base teórica quer na sua parte prática.

Ajustámos os parâmetros do algoritmo ao caso particular da nossa aplicação. Tal como na função real de duas variáveis reais da página 37, verifica-se, no caso do Problema do Caixeiro Viajante, que o melhor valor quer para a probabilidade de cruzamento, quer para a probabilidade de no torneio estocástico ser seleccionado o melhor, é de 0,8. Quanto à probabilidade de mutação, confirma-se a necessidade de

valores baixos, de 0,01 na função da página 37 e 0,04 no Problema do Caixeiro Viajante. Neste caso, o valor é mais elevado, como prevemos em 3.3.3.2.

Verificámos um aumento significativo quer da memória quer do tempo necessários para execução do algoritmo com o aumento do tamanho do problema. Uma forma de minorar estes problemas será a paralelização do algoritmo, que será estudada no capítulo seguinte. Com a paralelização pretendemos melhorar as soluções obtidas, aproximando-nos do óptimo. Pretendemos também que o tempo necessário à obtenção de uma determinada solução, pelo algoritmo genético canónico seja dividido pelo número de processadores, no algoritmo paralelo.

4 Algoritmos Genéticos Paralelos

Os Algoritmos Genéticos são, como já vimos, técnicas de pesquisa estocástica e de optimização, inspiradas pela evolução natural e pelas populações genéticas. Têm vindo a demonstrar a sua eficácia e robustez na pesquisa em espaços grandes e variados, num vasto campo de aplicações. Durante os últimos anos a área dos Algoritmos Genéticos em geral e o campo dos Algoritmos Genéticos Paralelos em particular, tem vindo a amadurecer até um ponto no qual é fiável a sua aplicação a problemas complexos do mundo real [Cantú-Paz95a].

A eficácia dos Algoritmos Genéticos está limitada pela sua capacidade de manter o equilíbrio entre a necessidade de uma amostragem diversificada de pontos e o desejo de atingir rapidamente um conjunto de potenciais soluções. Devido a exigências crescentes, tais como, espaços de pesquisa com funções de avaliação dispendiosas e populações com tamanhos grandes, existe uma necessidade crescente de implementações que permitam experiências rápidas e flexíveis. A maioria dos Algoritmos Genéticos, trabalha com uma grande população panmixia (panmictic population). Uma população panmixia, é uma população na qual qualquer indivíduo pode ser combinado com qualquer outro, com uma probabilidade que depende apenas da sua adaptação. O oposto é uma população dividida em subgrupos (subpopulações), evoluindo de forma paralela, onde os indivíduos apenas se podem combinar com os da sua subpopulação. Uma população panmixia apresenta o problema de a selecção se basear apenas na distribuição da adaptação ao longo de toda a população, levando a problemas tais como a convergência prematura. Pensamos que a paralelização é a via natural a explorar, sendo que representa melhor o que se passa na natureza, onde existem vários subgrupos populacionais evoluindo em paralelo [Levine94].

É importante distinguir entre duas aproximações aos Algoritmos Genéticos Paralelos [Cantú-Paz95a]: a aproximação tradicional, que utiliza o Algoritmo Genético Paralelo como um meio de implementar um Algoritmo Genético, e a aproximação por decomposição que utiliza o Algoritmo Genético Paralelo como um modelo particular de um Algoritmo Genético. No primeiro caso é paralelizado o ciclo que cria uma nova geração (secção 4.1.) enquanto que no segundo caso a população inteira aparece de uma forma distribuída (secção 4.2).

Vamos, no que se segue, descrever cada uma destas abordagens. Vamos ainda descrever a implementação por nós realizada, utilizando o equipamento do CICA. Em 4.1. vamos debruçar-nos sobre a abordagem tradicional à paralelização de Algoritmos Genéticos. Em 4.2. vamos descrever os vários modelos da paralelização de Algoritmos Genéticos por decomposição. No ponto 4.3. vamos descrever a implementação realizada, apresentando algumas conclusões em 4.4.

4.1. Abordagem tradicional à paralelização de Algoritmos Genéticos

Nesta abordagem à paralelização de Algoritmos Genéticos, o modelo sequencial é implementado num computador paralelo. Uma forma simples de o fazer é paralelizar o ciclo que cria uma nova geração, como se indica na figura 4.1., pois a maioria dos passos desse ciclo (avaliação, cruzamento, mutação e selecção) podem ser executados em paralelo [Whitley93].

1. Criação de uma população inicial.
2. Avaliação, em paralelo, dos elementos da população.
3. Efectuar, em paralelo, $\frac{pop}{2}$ vezes: (*pop* é o tamanho da população)
    - 3.1. Seleccionar dois indivíduos;
    - 3.2. Gerar dois novos indivíduos, através de operadores genéticos;
    - 3.3. Avaliar os dois novos indivíduos.
4. Substituir a população anterior pela nova população
5. Voltar a **3.**, até atingir o critério de paragem[1].

Figura 4.1. Algoritmo Genético Paralelo

[1] Ver secção 3.2.2.5.

Uma variante deste algoritmo é o chamado Algoritmo Genético Assíncrono [Zeigler91], que podemos ver na figura 4.2. Podemos verificar que os indivíduos são substituídos pelos seus descendentes logo após a avaliação. Este novo algoritmo tem a vantagem da interacção rápida dos indivíduos com os outros existentes na população, enquanto que no modelo anterior isso poderá acontecer só após algumas gerações podendo retardar o aparecimento de boas soluções.

1. Criação de uma população inicial.
2. Avaliação, em paralelo, dos elementos da população.
3. Efectuar, em paralelo, $\frac{pop}{2}$ vezes: (*pop* é o tamanho da população)
    3.1. Seleccionar dois indivíduos;
    3.2. Gerar dois novos indivíduos, através de operadores genéticos;
    3.3. Avalir os dois novos indivíduos.
    3.4. Seleccionar dois indivíduos para serem substituídos pelos novos;
4. Voltar a **3.**, até atingir o critério de paragem.

Figura 4.2. Algoritmo Genético Assíncrono

Para a implementação destes algoritmos, um processador mestre faz a supervisão da população total e faz a selecção, enquanto que os processos escravos recebem os indivíduos e combinam-nos criando "descendentes", que são depois avaliados, como podemos observar na figura 4.3.

Ao utilizar um computador de memória distribuída, os custos de comunicação associados à distribuição das estruturas de dados pelos vários processadores, à sincronização e à recolha dos resultados cresce bastante com o tamanho das populações. Este facto pode minimizar as melhorias no rendimento devido à utilização de vários processadores, a não ser que a função de avaliação consuma muito tempo de execução. São algoritmos que se adaptam a sistemas com muitos processadores. O número ideal seria de $\frac{pop}{2}$, o que corresponderia a um processador por cada escravo.

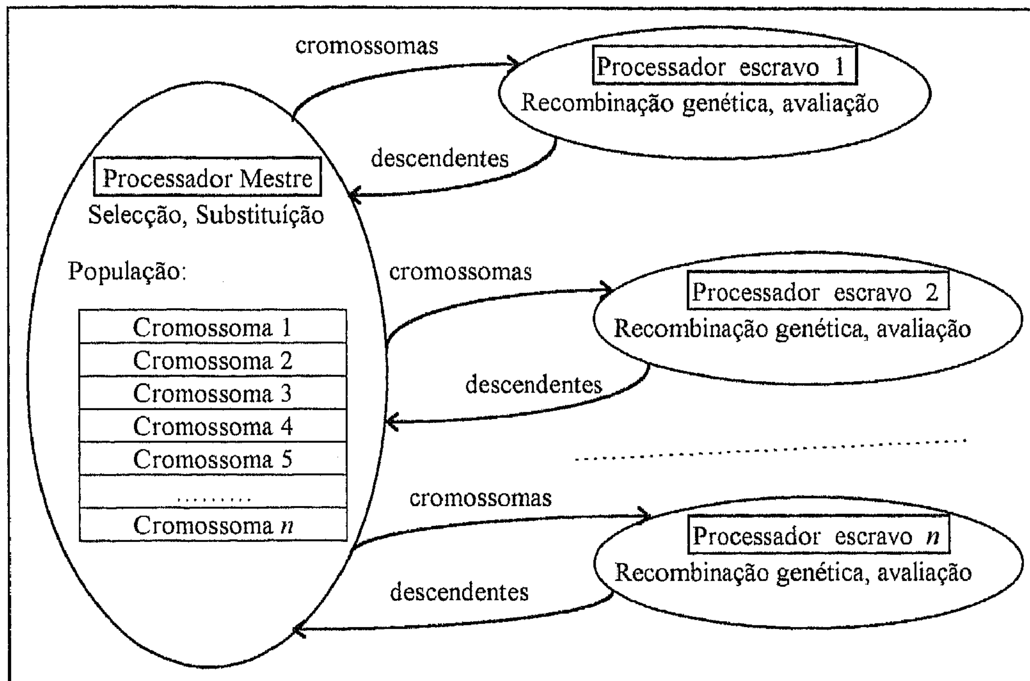


Figura 4.3. Modelo de um Algoritmo Genético Paralelo

Como exemplo de uma aplicação deste tipo de algoritmos referimos o trabalho de Abramson e Perkins [Cantú-Paz95a]. Eles utilizaram um Encore Multimax com 16 processadores, um Fusitsu AP1000 com 128 processadores e uma rede de estações de trabalho DEC, para implementar este algoritmos, aplicado ao cálculo de horários de comboio eficientes. As versões de memória partilhada (Encore) e de memória distribuída (Fujitsu), comportaram-se muito bem até aos 16 processadores. A versão Fujitsu AP1000 continuou a mostrar aumento de eficiência até aos 128 processadores, ficando, no entanto, muito abaixo do ideal com este número de processadores. As melhorias de rendimento com as estações de trabalho DEC, utilizando PVM, foram muito poucas, devido aos enormes custos associados com os protocolos de comunicação utilizados (*ethernet* e UNIX TCP/IP).

Como dispomos apenas de 4 processadores este modelo não será o mais adequado, pelo que não será o utilizado.

4.2. Aproximação à paralelização de Algoritmos Genéticos por decomposição

Neste tipo de aproximação a população inteira aparece de uma forma distribuída. Podem existir várias sub-populações, independentes ou interagindo umas com as outras ou pode existir uma só população na qual cada elemento interage apenas com um conjunto limitado de vizinhos. Geralmente, cada processador envia a outro as suas melhores soluções. Estas comunicações acontecem de acordo com a estrutura espacial da população. Estes Algoritmos podem ser classificados em Algoritmos Genéticos Grosseiros (coarse-grained), Algoritmos Genéticos Finos (fine-grained) e Algoritmos Genéticos Paralelos Híbridos [Cantú-Paz95a].

4.2.1. Algoritmos Genéticos Grosseiros

Num Algoritmo Genético Grosseiro, a população é dividida em várias subpopulações ou burgos (demes), cada uma das quais correndo o mesmo Algoritmo Genético independentemente e em paralelo - figura 4.4. Os processadores trocam entre si elementos das subpopulações - migrantes. Existem parâmetros importantes que devem ser estudados:

- o tamanho das subpopulações - **n**;
- os processadores que vão trocar dados;
- a frequência da troca de indivíduos - intervalo ou frequência de migrações (IM);
- a quantidade de indivíduos que os processadores trocam entre si - razão da migração (RM);
- a estratégia a utilizar na selecção dos indivíduos a migrar.

As populações isoladas ajudam a manter a diversidade genética, pois os indivíduos de cada subpopulação estão relativamente isolados dos indivíduos das outras subpopulações, explorando assim partes diferentes do espaço de entrada. Uma característica importante desta classe de algoritmos é a utilização de grandes burgos em número relativamente pequeno, e a introdução do operador genético de migração.

Em algumas implementações os indivíduos migrantes apenas se podem mover para populações vizinhas - modelo das alpondras (stepping stone). Noutras implementações, os indivíduos migrantes podem-se mover para qualquer população - modelo das ilhas. Estes modelos de paralelização de algoritmos genéticos, também conhecidos por algoritmos genéticos distribuídos, são muito populares, adaptando-se bem a sistemas com poucos processadores [Cantú-Paz95a].

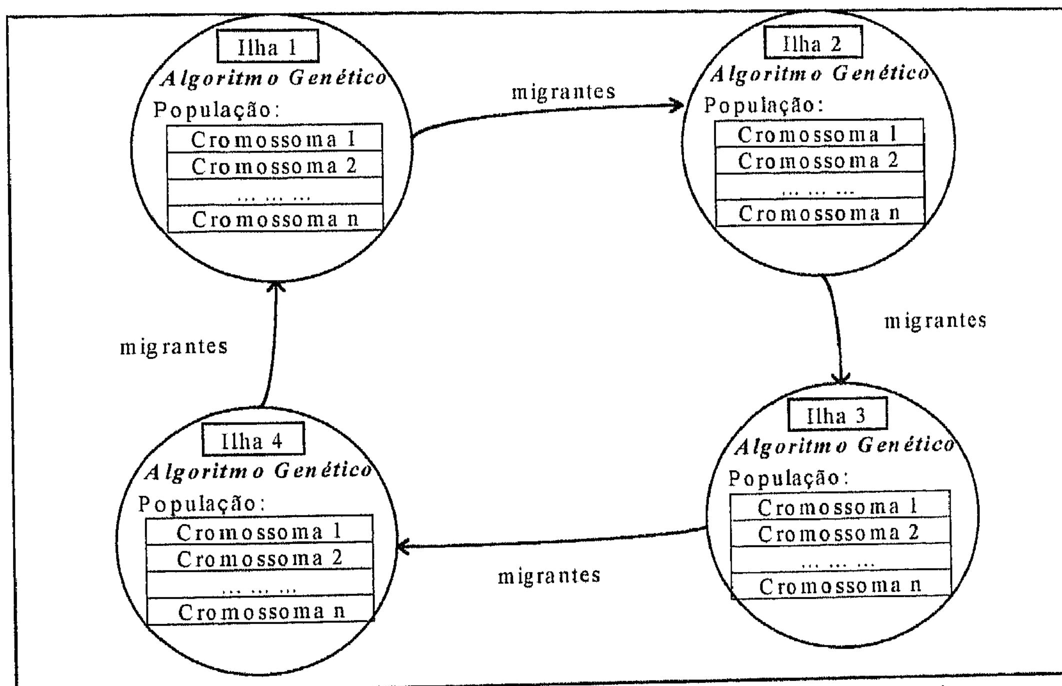


Figura 4.4. Representação de um modelo de um Algoritmo Genético Grosseiro

Um dos trabalhos pioneiros neste campo é a dissertação de Grosso (1985). Ele realizou simulações com indivíduos diplóides e a população foi dividida usando uma razão de migração fixa, que foi alterada em diferentes experiências. Grosso descobriu que as melhorias eram mais acentuadas nos pequenos burgos do que numa só população panmixia grande. No entanto, quando os burgos estavam isolados durante o processo de adaptação, este rápido crescimento parava em níveis mais baixos do que com grandes populações. Com valores intermédios nas taxas de migração, as subpopulações mostravam um comportamento similar às populações panmixias, enquanto que com valores mais baixos, apesar de terem a oportunidade de explorar regiões diferentes do espaço de pesquisa, os emigrantes não produziam efeitos significativos no burgo que os

recebia, nomeadamnete se esses valores fossem demasiado baixos. Estes resultados apontam para o facto da existência de uma razão de migração crítica, abaixo da qual o desempenho do algoritmos é obstruído pelo isolamento dos burgos e acima do qual as subpopulações se comportam como panmixias.

Vários trabalhos têm vindo a ser desenvolvidos na área, mas não têm sido conclusivos, principalmente no que diz respeito aos parâmetros relativos às migrações entre ilhas.

Devido às suas características é este o modelo que iremos utilizar.

4.2.2. Algoritmos Genéticos Finos

Num algoritmo genético paralelo fino a população é dividida em muitos burgos pequenos. A selecção, o cruzamento e a substituição apenas podem ser realizadas numa vizinhança limitada - figura 4.5. Como as vizinhanças se sobrepõem, os indivíduos

C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	...
...	...	...	...	...	...	...	...	...
...	...	...				...	...	...
...	...	...	..	.	..	...	...	...
...	...	...	C_{i-1}	C_i	C_{i+1}	...	...	...
...	...	...	..	.	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...	...
...	...	...	...	...	...	C_{n-2}	C_{n-1}	C_n

Figura 4.5. Modelo de um Algoritmo Genético Fino

propagam-se através de toda a população. O ideal será a atribuição de um indivíduo a um processador, o que apela à utilização de supercomputadores [Cantú-Paz95a].

Existem algumas escolhas importantes a fazer neste caso, nomeadamente:

- tamanho do burgo/vizinhança;
- topologia de conexão dos processadores;
- esquema da substituição dos indivíduos.

Apresentamos, como exemplo, a implementação de Baluja [Baluja92], utilizando o MasPar Mp-1 com 4096 processadores. Ele comparou topologias com diâmetros diferente, sendo duas delas variações de uma topologia linear e a outra uma

rede bidimensional. A rede, que tinha o menor diâmetro, deu os melhores resultados em quase todos os problemas testados.

Em alguns trabalhos têm sido feitas tentativas de comparação dos Algoritmos Genéticos Grosseiros e dos Algoritmos Genéticos Finos, tendo-se obtido resultados diferentes - enquanto que alguns trabalhos apontam para melhores resultados para os Algoritmos Genéticos Grosseiros, outros apresentam resultados contrários. Pensamos que estas comparações não poderão ser generalizadas, na medida em que as implementações dependem muito do sistema em causa.

Devido à necessidade de um grande número de processadores este modelo não se adapta ao sistema que iremos utilizar.

4.2.3. Algoritmos Genéticos Paralelos Híbridos

Existem alguns trabalhos que tentam combinar os vários modelos. A hibridação pode ser feita de várias formas. Por exemplo, podemos obter uma estrutura de ilhas sendo cada uma delas um Algoritmos Genético Fino. Gruau [Gruau94] utilizou um modelo deste tipo, num novo desenho de uma rede neuronal, com bons resultados. Um outro tipo de aplicação foi implementada por Neto [Neto95] que implementou um modelo de ilhas, sendo que cada uma delas correu um Algoritmo Genético Assíncrono. Com os testes realizados, o mesmo autor verificou que o tempo de processamento necessário à obtenção de boas soluções foi dividido pelo menos por um factor igual ao número de processadores do sistema. Quanto à qualidade das soluções finais, elas aproximaram-se muito da qualidade das soluções obtidas com o Algoritmo Genético Tradicional, nas mesmas condições de processamento.

Não vamos utilizar este modelo.

4.3. Aplicação ao problema do Caixeiro Viajante

Vamos passar a descrever a implementação por nós realizada, utilizando os recursos computacionais do CICA, já descritos no capítulo 2. Procedemos à implementação de um Algoritmo Genético Paralelo para a resolução do Problema do Caixeiro Viajante com 70 cidades, já referido no capítulo 3.

A primeira decisão a tomar diz respeito ao tipo de algoritmo a utilizar. Devido às características do sistema, optámos pelo modelo de ilhas. Este modelo adapta-se a sistemas com poucos processadores, como aquele que utilizámos. O modelo de comunicação adoptado é o de um anel unidirecional, representado na figura 4.4. O número de ilhas típico (**NI**) toma os valores 4, 8, 16, ...

Quanto ao número de migrantes (**NM**), considerámos primeiro o caso de ilhas isoladas, ou seja, o número de migrantes é zero e de seguida testámos o algoritmo para um migrante. Considerámos depois 10, 20 migrantes. Os indivíduos migrantes foram seleccionados aleatoriamente.

Outro parâmetro estudado foi o intervalo entre migrações (**IM**). Corremos a implementação do algoritmo durante 1000 gerações (**NG**) efectuando as migrações após 100 gerações, após 200 gerações e após 500 gerações. A sincronização entre as várias ilhas é feita aquando das migrações.

Variamos ainda o tamanho total de cada subpopulação (**TS**) iniciando com 2000 e depois com 4000, 8000, 16000 ... cromossomas.

A análise dos resultados obtidos foi feita atendendo a dois aspectos, sendo o primeiro deles as soluções obtidas pelo algoritmo e o segundo o tempo de execução.

4.3.1. Valores obtidos pelo algoritmo

Os valores obtidos a partir da execução do algoritmo, para a resolução do problema da caixeiro viajante com 70 cidades, já referido, estão apresentados nas tabelas 4.1. a 4.12. O valor apresentado em cada entrada das tabelas é o melhor de todas as ilhas.

Nº de ilhas / Nº migrantes	4	8	16
0	45430	45733	45714
1	45254	**43387**	44914
10	45590	44868	44682
20	45955	46299	45245

Tabela 4.1. Soluções obtidas pelo algoritmo; **TS**=2000; **IM**=100

Nº de ilhas / Nº migrantes	4	8	16
1	46123	46539	45925
10	45701	46420	**44702**
20	48154	46698	45686

Tabela 4.2. Soluções obtidas pelo algoritmo; **TS**=2000; **IM**=200

Nº de ilhas / Nº migrantes	4	8	16
1	47064	45582	45097
10	46441	45246	45626
20	47177	46557	**43775**

Tabela 4.3. Soluções obtidas pelo algoritmo; **TS**=2000; **IM**=500

Nº de ilhas / Nº migrantes	4	8	16
0	44093	44313	44450
1	45536	45215	**43222**
10	44359	44252	44532
20	44191	44125	44032

Tabela 4.4. Soluções obtidas pelo algoritmo; **TS**=4000; **IM**=100

Nº de ilhas / Nº migrantes	4	8	16
1	45537	44085	44129
10	44343	44590	43435
20	44950	45117	**42985**

Tabela 4.5. Soluções obtidas pelo algoritmo; **TS**=4000; **IM**=200

Nº de ilhas / Nº migrantes	4	8	16
1	44841	45712	44599
10	44929	44292	44298
20	44322	44265	**44185**

Tabela 4.6. Soluções obtidas pelo algoritmo; **TS**=4000; **IM**=500

Nº de ilhas / Nº migrantes	4	8	16
0	43322	43407	43900
1	43359	43640	43463
10	44358	43735	43051
20	43429	43394	**42656**

Tabela 4.7. Soluções obtidas pelo algoritmo; **TS**=8000; **IM**=100

Nº de ilhas / Nº migrantes	4	8	16
1	43311	43562	**42470**
10	43356	42829	42791
20	43416	43100	43384

Tabela 4.8. Soluções obtidas pelo algoritmo; **TS**=8000; **IM**=200

Nº de ilhas / Nº migrantes	4	8	16
1	43387	43665	42967
10	43883	43213	43450
20	43281	43188	**42508**

Tabela 4.9. Soluções obtidas pelo algoritmo; **TS**=8000; **IM**=500

Nº de ilhas / Nº migrantes	4	8	16
0	43905	43568	42688
1	42756	43033	**42246**
10	43215	42359	43320
20	43318	42829	42869

Tabela 4.10. Soluções obtidas pelo algoritmo; **TS**=16000; **IM**=100

Nº de ilhas / Nº migrantes	4	8	16
1	43196	43381	43098
10	43276	42839	**42532**
20	43376	43315	42979

Tabela 4.11. Soluções obtidas pelo algoritmo; **TS**=16000; **IM**=200

Nº de ilhas / Nº migrantes	4	8	16
1	42979	43061	42903
10 (5%)	43319	43510	**42174**
20 (10%)	43044	42466	42954

Tabela 4.12. Soluções obtidas pelo algoritmo; **TS**=16000; **IM**=500

Através da simples observação das tabelas, podemos retirar algumas conclusões:

- no caso de ilhas isoladas, isto é, o número de migrantes é zero, o aumento no número de ilhas provoca piores resultados (tabelas 4.1., 4.4. e 4.7.) ;

- no caso da existência de migrantes, com o aumento do número de ilhas, esperam-se melhores resultados, embora não o possamos afirmar de forma absoluta, pois em alguns casos isso não acontece - tabela 4.1.;
- o aumento do intervalo de migrações provoca piores resultados, no caso de subpopulações de tamanho 2000; quanto às subpopulações de tamanho 16000, os resultados não são conclusivos; assim sendo, pensamos que com o aumento do tamanho das subpopulações, o intervalo entre migrações deixa de ser muito significativo;
- quanto ao número de migrantes, no geral, aumentando o número de migrantes, com o tamanho das subpopulações e com o número de ilhas, há uma tendência para a obtenção de melhores resultados; no entanto, realizámos alguns testes suplementares com 30 migrantes e verificámos que em todos eles obtínhamos piores resultados; concluímos que o número de migrantes depende do número de ilhas e do intervalo de migrações.

Os resultados obtidos com o aumento do tamanho das subpopulações não foi tão bom como esperávamos. Assim sendo, decidimos comparar os resultados obtidos em paralelo, com os resultados obtidos com populações panmixias do mesmo tamanho, com e sem migração. Podemos ver estas comparações observando as tabelas 4.13. e 4.14.

	População Panmixia	Melhoria 4 ilhas isoladas	Melhoria 8 ilhas isoladas	Melhoria 16 ilhas isoladas
2000	**48589**	**6,5 %**	5,9 %	5,9%
4000	**46952**	6,1 %	5,6 %	5,3 %
8000	**44610**	2,9 %	2,7 %	1,6 %
16000	**43504**	piorou	piorou	1,8 %

Tabela 4.13. Comparação das populações panmixias com ilhas do mesmo tamanho, isoladas

	População Panmixia	% melhoria 4 ilhas com migração / nº de migrantes	% melhoria 8 ilhas com migração / nº de migrantes	% melhoria 16 ilhas com migração / nº de migrantes
2000	**48589**	6,9 % / 1	**10,7 %** / 1	9,9 % / 20
4000	**46952**	5,9 % / 20	6,1 % / 1	8,4 % / 20
8000	**44610**	3,0 % / 20	4,0 % / 10	4,8 % / 1
16000	**43504**	1,7 % / 1	2,6 % / 10	3,1 % / 10

Tabela 4.14. Comparação das populações panmixias com ilhas do mesmo tamanho, com migrantes

Os resultados das tabelas indicam que a melhoria é mais significativa no caso de subpopulações pequenas em paralelo, com migrações. Aumentando o tamanho das subpopulações, os resultados não melhoram significativamente, apesar de uma melhoria em relação às populações panmixias, com o mesmo tamanho, na maior parte dos casos.

Obtivemos o melhor resultado ao testar o algoritmo, no limite, com 4 ilhas de 64000 cromossomas com 10 migrantes de 100 em 100 gerações. O seu valor foi de 41800, valor que excede o óptimo em 8% e que, em relação ao melhor resultado obtido com populações panmixias, representa uma melhoria de 1,3%, bastante inferior às melhorias verificadas com subpopulações menores.

Um outro aspecto a estudar é o da convergência das várias ilhas. Para isso, estudámos a variação, ao longo de 1000 gerações, de 8 ilhas de tamanho 32000 e de 4 ilhas de tamanho 64000, com migração de 10 emigrantes de 100 em 100 gerações, em ambos os casos. Os resultados estão representados através dos gráficos das figuras 4.6. e 4.7. Através da observação dos gráficos chegámos à conclusão de que os valores das ilhas, apresentam uma convergência para um valor diferente do óptimo. Na tentativa de evitar esta situação indesejável, fizemos testes nos quais diminuímos o número de migrantes, mas esta situação manteve-se.

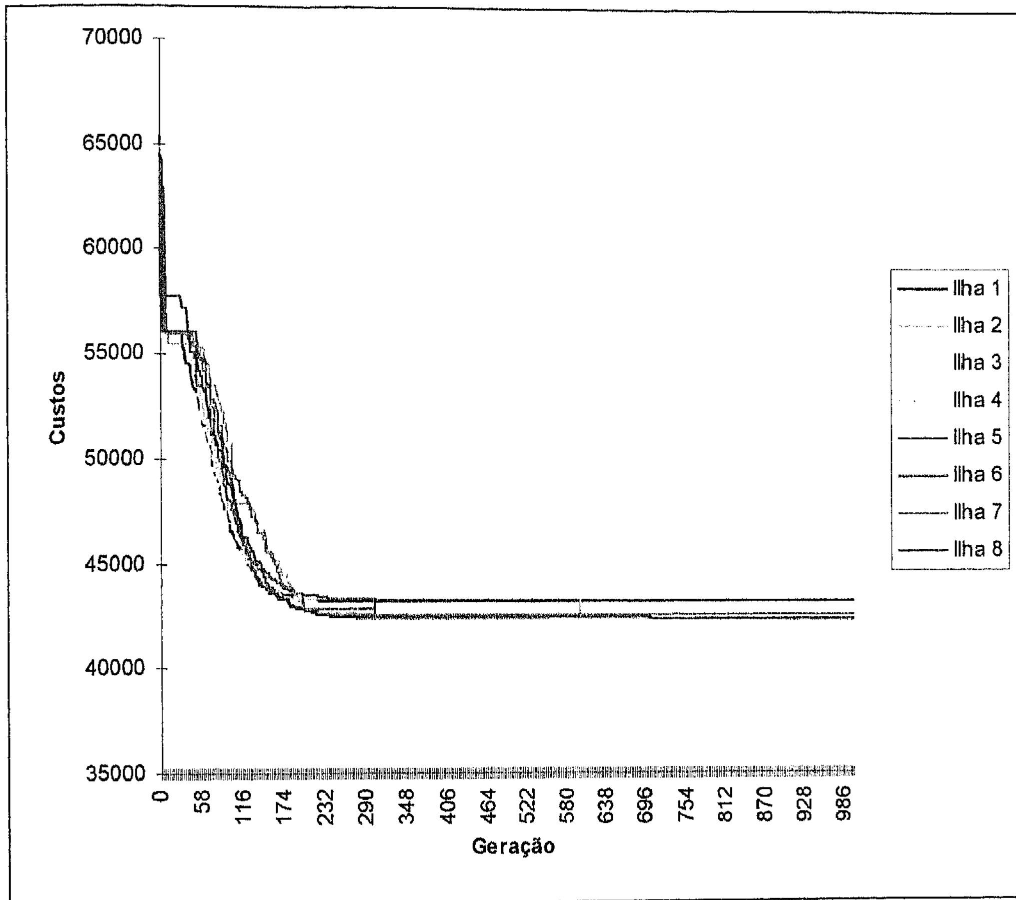


Figura 4.6. Comparação dos valores de 8 ilhas de tamanho 32000 cromossomas

Verificámos que a parametrização do Algoritmo Genético Paralelo é mais complicada do que a do Algoritmo Genético Canónico. Pudemos concluir, que os parâmetros estão relacionados entre si e quando é feita uma alteração no valor de um dos parâmetros, os outros terão de ser ajustados. Assim sendo, vimos que o número de emigrantes tem tendência a aumentar com o número de ilhas e com o tamanho das subpopulações, enquanto que o efeito do intervalo de migrações, tem tendência a diminuir com o aumento do tamanho das subpopulações.

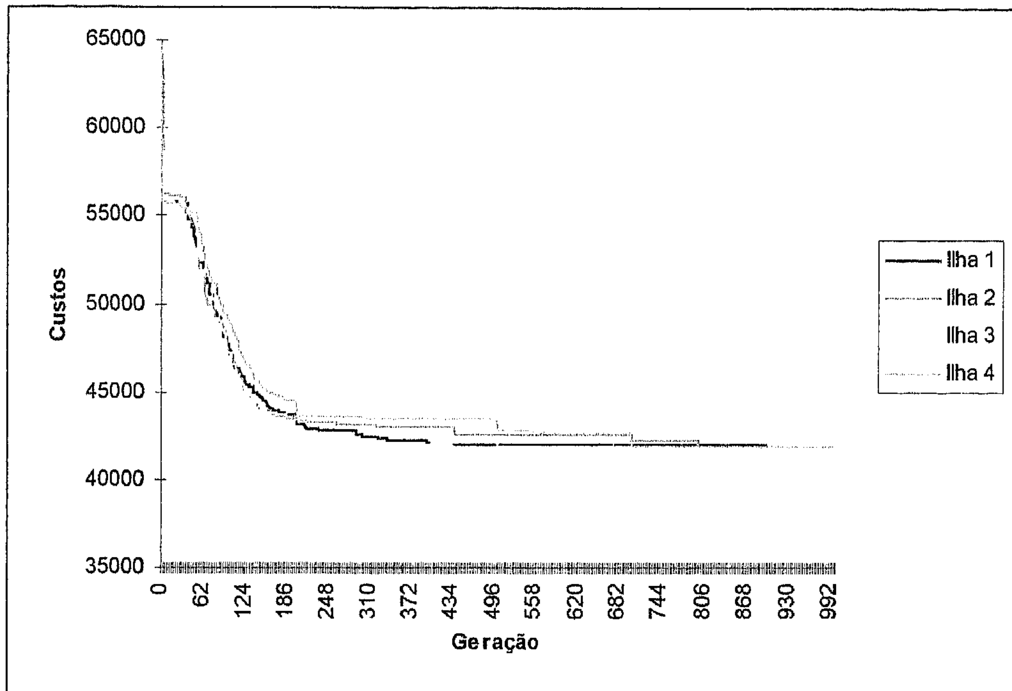


Figura 4.7. Comparação da variação dos valores de 4 ilhas de tamanho 64000

4.3.2. Tempos de Execução/Comunicação Teóricos

Consideremos T_s, o tempo de execução sequencial do algoritmo para um determinado tamanho da população, *TPOPULACAO* e T_c o tempo gasto em comunicações entre tarefas PVM. Seja *NI* o número de ilhas, *NM* o número de migrantes, *IM* o intervalo entre migrações, *M* o número de migrações, *NG* o número de gerações e *NC* o número de cidades. Então, o tempo de execução, T_p, do algoritmo paralelo é dado por

$$T_p = T_s + T_c \qquad \text{[Fórmula 4.1]}$$

Vamos elaborar um modelo teórico para o tempo de comunicação, T_c. Este é a soma do tempo gasto com as migrações, T_m, do tempo gasto no envio do melhor

cromossoma de cada ilha para o "mestre", T_i e do tempo gasto pelo "mestre" na iniciação das ilhas T_{mi}.

$$T_c = T_m + T_i + T_{mi} \qquad \text{[Fórmula 4.2]}$$

Mais uma vez, tal como em 2.2.1., vamos realizar os nossos cálculos baseados na pior máquina e no trabalho de [Guedes95].

Quanto ao tempo gasto pelo mestre na iniciação das ilhas, como não existe passagem de mensagens, é dado apenas pelo tempo de iniciação. Assim sendo, vem que

$$T_{mi} = 0,727 \times 10^{-3} \times NI \qquad \text{[Fórmula 4.3]}$$

Vamos de seguida calcular o tempo gasto com as migrações. Cada migrante é um cromossoma, representado por um vector de números inteiros, com tamanho igual ao número de cidades do Problema do Caixeiro Viajante, NC e além disso, por um número real, que é a adaptação do cromossoma. Sendo n o número de bytes ocupado por cada migrante

$$T_m = M \times (0,727 \times 10^{-3} + 0,60 \times 10^{-6}\, n \times NM) \qquad \text{[Fórmula 4.4]}$$

Cada número inteiro ocupa 2 *bytes* e cada número real ocupa 4 *bytes*. [Kernighan88]. Assim sendo,

$$n = 2 \times NC + 4 \qquad \text{[Fórmula 4.5]}$$

Sabemos que

$$M = \frac{NG}{IM} \qquad \text{[Fórmula 4.6]}$$

Obtemos

$$T_m = \frac{NG}{IM} \times \left[0{,}727 \times 10^{-3} + 0{,}60 \times 10^{-6} \times (2 \times NC + 4) \times NM\right] \quad \text{[Fórmula 4.7]}$$

Por fim, o tempo gasto no envio do cromossoma de uma ilha para o mestre é dado por

$$T_i = 0{,}727 \times 10^{-3} + 0{,}60 \times 10^{-6} \times (2 \times NCIDADES + 4) \quad \text{[Fórmula 4.8]}$$

Portanto:

$$T_c = 0{,}727 \times 10^{-3} \times (1 + \frac{NG}{IM} + NI) + 0{,}60 \times 10^{-6} \times (2 \times NC + 4) \times (1 + NM \times \frac{NG}{IM})$$

Vamos agora tomar como exemplo o Problema do Caixeiro Viajante, com 70 cidades, utilizando um intervalo entre migrações de IM=100 e NG=1000 gerações. Com o objectivo de estudar a variação do tempo de comunicação com a variação do número de ilhas e do número de migrantes, obtivemos a tabela 4.15.

Nº de Ilhas / Nº Migrantes	4	8	16
0	**$1{,}10 \times 10^{-2}$**	**$1{,}39 \times 10^{-2}$**	**$1{,}97 \times 10^{-2}$**
1	**$1{,}19 \times 10^{-2}$**	**$1{,}48 \times 10^{-2}$**	**$2{,}06 \times 10^{-2}$**
10	**$1{,}96 \times 10^{-2}$**	**$2{,}25 \times 10^{-2}$**	**$2{,}84 \times 10^{-2}$**
20	**$2{,}83 \times 10^{-2}$**	**$3{,}12 \times 10^{-2}$**	**$3{,}70 \times 10^{-2}$**

Tabela 4.15. Tempos de comunicação do Algoritmo Genético Paralelo, calculados com o modelo teórico

Dado que, mesmo no caso de populações de 2000 cromossomas, os tempos de execução do algoritmo se elevam a vários minutos (tabela 3.2.), podendo atingir algumas horas no caso de populações maiores, podemos desprezar os tempo de comunicação. Assim sendo, para tempo de execução podemos tomar apenas o tempo de

execução sequencial da pior ilha. O tempo de execução do mestre é também desprezável em relação ao tempo de execução das ilhas.

Testámos o nosso algoritmo, através da utilização dos recursos do CICA em modo de utilizador único (standalone), com quatro ilhas de tamanho 64000 durante 1000 gerações. O tempo de execução foi de 2h30mn. Ao testar o mesmo algoritmo em modo multiutilizador, o tempo de execução foi de 7 horas. Não foi possível obter subpopulações com mais de 64000 cromossomas, mesmo em modo de utilizador único

Pretendemos comparar os tempos de obtenção da melhor solução com populações panmixias - 42300 - com o tempo de o algoritmo paralelo obter o mesmo valor. Observando a tabela 4.10, vemos que esse valor é praticamente obtido com 8 ilhas de 16000 cromossomas e 10 migrantes de 100 em 100 gerações - 43359. Distribuindo as ilhas de forma adequada pelos processadores, o tempo de execução foi de 1h13mn. O tempo de execução com a população panmixia foi de 3h42mn (secção 3.3.3.3.). Dividindo este valor pelo número de processadores obtemos um tempo de 55,5 minutos, tempo este que é aproximado do valor obtido experimentalmente com a paralelização.

Considerando os dois aspectos do Algoritmo Genético Paralelo tempo de execução e soluções obtidas, podemos concluir que este modelo de paralelização, apresenta vantagens, pois além de permitir obter melhores resultados, permite que o tempo de execução sequencial para a obtenção de um determinado valor, seja dividido, aproximadamente, pelo número de processadores quando pretendemos obter o mesmo valor em paralelo.

4.4. Notas Finais

Existindo vários modelos para a paralelização de algoritmos genéticos, e depois da análise às características de cada um deles, concluímos que o melhor modelo para utilizar no sistema que está disponível no CICA, é o do Algoritmo Genético Grosseiro. Este modelo adapta-se a sistemas com poucos processadores como o utilizado ao longo deste trabalho.

Procedendo à sua implementação, e depois de realizados os testes considerados convenientes, concluímos ser vantajosa a sua implementação. No entanto, os resultados obtidos com populações muito grandes, no limite das capacidades do sistema, foram um pouco desanimadores, pois esperávamos melhores resultados. Com populações mais pequenas, longe dos limites da capacidade do sistema, as melhorias em relação às populações panmixias foram significativas, pelo que pensamos que será uma boa opção a paralelização de algoritmos genéticos, com a utilização dos recursos computacionais do CICA.

5 Conclusões e Perspectivas Futuras

No nosso trabalho pretendemos averiguar a existência, ou não, de vantagens na paralelização de algoritmos genéticos utilizando os recursos disponíveis no CICA, utilizando PVM.

Para isso começamos por, no capítulo 2, fazer a análise do software PVM. Através da implementação em paralelo do algoritmo de ordenação MergeSort, concluímos que o sistema não se adapta a modelos de paralelização que envolvam muitas rotinas, tal como esperavamos.

No capítulo 3, realizamos um estudo sobre o tema dos algoritmos genéticos sequenciais, debruçando-nos essencialmente sobre o Algoritmo Genético Canónico. Implementamos um algoritmo deste tipo, que nos permitiu resolver o Problema do Caixeiro Viajante, com resultados satisfatórios, atendendo ao grau de dificuldade do problema em causa.

Finalmente, no capítulo 4, analisamos vários modelos de paralelização, segundo critérios tais como a adaptação do modelo aos recursos disponíveis, os resultados obtidos e o tempo de execução. Quanto ao modelo de paralelização, averiguamos que o modelo mais adaptado é o modelo do Algoritmo Genético Grosseiro, visto que os outros modelos originam muitas tarefas, o que não é adequado ao sistema como verificamos no capítulo 2. Implementando este algoritmo para resolver o Problema do Caixeiro Viajante, concluímos ser vantajosa a sua implementação, obtendo melhores resultados quando o sistema se encontrou longe dos limites dos seus recursos, isto é, com a

utilização de subpopulações pequenas. Neste caso, as melhorias em relação às populações panmixias foram significativas. Verificamos que o tempo necessário à obtenção de uma determinada solução pelo algoritmo genético sequencial, é praticamente dividido pelo número de processadores, no algoritmo genético paralelo testado. Podemos concluir que será uma boa opção a paralelização de algoritmos genéticos, com a utilização dos recursos computacionais do CICA.

Quanto a perspectivas futuras de trabalho, para o Problema do Caixeiro Viajante, penso que se poderão centrar em dois aspectos:

- reduzir a necessidade de populações muito grandes para a resolução do Problema do Caixeiro Viajante, por exemplo criando melhores populações iniciais utilizando heurísticas [Yang97];
- melhorar os resultados obtidos por cada uma das ilhas, utilizando um outro tipo de codificação e operadores genéticos mais apropriados à resolução de problemas de optimização combinatória [Davis91][Goldberg89].

Um outro tipo de trabalho futuro que poderá ser realizado é a paralelização de um algoritmo genético adaptado a outros problemas, reais ou não, utilizando o modelo do Algoritmo Genético Grosseiro. Para esses problemas pode já ser conhecido um algoritmo genético sequencial, com resultados satisfatórios, que, pelo que nos foi dado observar melhorarão.

Bibliografia

1. Paralelização e PVM

[Ahuja86] Sudhin Ahuja; Nicholas Carriero; David Gelernter. "Linda and Friends". *IEEE Computer*. August 1986.

[Alves95] Alexandre Alves; Luís Silva; João Carreira; JoãoGabriel Silva. "WPVM: Parallel Computing for the People". Departamento de Engenharia Informática da Universidade de Coimbra. Portugal.

[Alves96] Alexandre Alves. "Parallel Computing - Windows Style". *Byte*.May,1996.

[Beguelin94b] Adam Beguelin; Jack Dongarra; Al Geist; Robert Manchek; Vaidy Sunderman. HeNCE: A Users' Guide Version 2.0. http://ccis.unl.edu/lab/hence/hence-2.0-doc.html. June 15, 1994.

[Blelloch96] Guy E. Blelloch. "Programming Parallel Algorithms". *Communications of the ACM*. March 1996, Vol. 39, No3, pp 85-96.

[Buttler92] Ralph Butler; Ewing Lusk. *Users' Guide to the P4 Parallel Programming System*. Argonne National Laboratory - ANL-92/17. 1992.

[Candy91] K. Mani Chandy; Carl Kesselman. "Parallel Programming in 2001". *IEEE Software*. November 1991.

[Carriero89] Nicholas Carriero; David Gelernter. "Linda in Context". *Communications of the ACM.* April 1989, Vol. 32. Number 4.

[Carriero90] Nicholas Carriero; David Gelenter. *How to write Parallel Programs, a first course*. The MIT Press. 1990.

[Chergui95] Jalel Chergui; Juan Escobar. *Programation par Échange de Messages (PVM)*. IDRIS. Février, 1995.

[Dongarra93] Jack J.Dongarra; G. A.Geist; Robert Manchek; V.S. Sunderman . "Integrated PVM Framework Supports Heterogeneous Network Computing". *Computer in Physics*. April 1993.

[Flower91] J. Flower; A. Kolawa; S. Bharadwaj. "The Express Way to Distributed Processing". *Supercomputing Review*, pp 54-55. May, 1991

[Geist94a] Al Geist; Adam Beguelim; Jack Dongarra; Weichen Jiang; Vaidy Sunderman. *PVM3 user's Guide and Reference Manual.* Technical Report ORNL/TM-12187, Oak Ridge National Laboratory. 1994.

[Geist94b] Al Geist; Adam Beguelim; Jack Dongarra; Weichen Jiang; Robert Manchek; Vaidy Sunderman. *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing.* The MIT Press. 1994.

[Geist95] G. A. Geist; J. A. Kohl; R. J. Manchek; R. J. Papadopoulus. "New Features Of PVM 3.4 and Beyond". *EuroPVM95*. August 25, 1995.

[Harel92] Harel, David. *Algorithmic, The Spirit of Computing*. Second Edition. Addison-Wesley Publishing Company, Inc. . 1992.

[Leighton92] F. Thomson Leigton. *Introduction to Parallel Algorithms and Arquitectures: Arrays-Trees-Hypercubes*. Morgan Kaufman Publishers, Inc. 1992.

[Lester93] Bruce P. Lester. *The art of Parallel Programming*. Prentice-Hall International Editions.1993.

[MacDonald94] N. MacDonald; E. Minty; T. Harding; S. Brown. *Writing Message Passing Parallel Programs with MPI - Course Notes of a Two-Day Course*. Edinburgh Parallel Computing Center. The University of Edinburgh. 1994.

[Peres95] Maria Joana C. Peres. *Paralelização de um método de refinamento de valores próprios numa arquitectura de memória distribuída*. Faculdade de Engenharia da Universidade do Porto. Portugal. 1995.

[Preston96] Martin Preston. *Introduction to MPI - Student Notes*. Manchester and North HPCT&EC. Jannuary, 1996.

[Quinn87] Michael J. Quinn. *Designing Efficient Algorithms for Parallel Computers*. McGraw-Hill Series in Artificial Intelligence. 1987.

[Bianchini93] Ricardo Bianchini; Christopher Brown. "Parallel Genetic Algorithms on Distributed-Memory Architectures". Tecnhical Report 436. Computer Science Department. The University of Rochester. 1993

[Cantú-Paz95a] Erick Cantú-Paz. "A Summary of Research on Parrallel Genetic Algorithms". IlliGAL Report No. 95007. July 1995.

[Cantú-Paz95b] Erick Cantú-Paz; David E. Goldberg. "Modeling Idealized Bounding Cases of Parallel Genetic Algoritmhs". IlliGAL Report No. 96007. December 1996.

[Collins91] Robert J. Collins; David R. Jefferson. "Selection in Massively Parallel Genetic Algorithms". *Proceedings of the Fourth International Conference on Genetic algorithms*. 1991.

[Davis91] Lawrence Davis. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold. 1991.

[Dias95] Maria Teresa Galvão Dias. *Aplicação de algoritmos Genéticos ao problema da Geração de Serviços de tripulação*. Faculdade de Engenharia da Universidade do Porto. DEEC. 1995.

[Forrest92] Forrest Stephanie; Melanie Mitchell. "Relative Building Block Fitness and the Building Block Hypothesis". *Foundations of Genetic Algorithms 2*.1992.

[Goldberg89] David E. Goldberg. *Genetic Algorithms in Search, Optimisation and Machine Learning*. Addison-Wesley. 1989

[Gordon93] V. Scott Gordon; Darrell Whitley. "Serial and Parallel Genetic Algorithms as Function Optimizers". Technical Report CS-93-114. 1993.

[Grefenstette90a] John J. Grefenstette. "Conditions for Implicit Parallelism". *Proceedings of the 1990 Workshop on Foundations of Genetic Algorithms*, Morgan Kaufmann.

[Grefenstetteb] John J. Grefenstette. "Deception Considered Harmfull". Navy Center for Applied Research in Artificial Intelligence. Navy Research Laboratory. Washington.

[Gruau94] F. Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithms*. PhD thesis, École Normale Superieur de Lyon. 1994

[Harik96] Georges Harik; Erick Cantú-Paz; David E. Goldberg; Brad Miller. "The Gambler's Ruin Problem, Genetic Algorithms and the Sizing of Populations". IlliGAL Report nº 96004. July, 1996.

[Herrmann93] Jeffrey W. Herrmann; Chung Yee Lee. "Solving a Class Scheduling Problem With a Genetic Algorithm". *ORSA Journal on Computing*, Vol. 7, Nº 4, 1995.

[Horn92] Jeffrey Horn; Davis E. Goldberg; Kalyanmog Deb. "Research Notes: Long Path Problems for Mutation-Based Algorithms". IlliGAL Report nº 92011. October 1992.

[Horn93] Jeffrey Horn. "Finite Markov Chain Analysis of Genetic Algorithms with Niching". IlliGAL Report nº 93002. February, 1993.

[Khuri93] Sami Khuri. "Walsh and Harr Functions in Genetic Algorithms". *ACM Press*. 1993.

[Khuri93] Sami Khuri; Thomas Bäck; Jörg Heitkötter. "The Zero/One Multiple Knapsack Problem and Genetic Algorithms". *ACM Press*. 1993.

[Kraft95] P. Kraft; M. Nölle; G. Schreiber; S. Marshall; H. Burkhardt. "A Parallel Genetic Algorithm for Optimizing Morphological Filters on Inhomogeneous Workstation Clusters". *Proceedings of the Fourth IWPIA.1995.*

[Koza92] John R. Koza. *Genetic Programming: on the programming of computers by means of natural selection.* Massachusetts Institute of Technology. 1992.

[Levine94] David Levine. *A Parallel Genetic Algorithm for the Set Partitioning Problem.* ANL-94/93. Argonne National Laboratory. 1994

[Lobo96] Fernando G. Lobo; David E. Goldberg. "Decision Making in a Hybrid Genetic Algorithm". IlliGAL Report nº 96009. September, 1996.

[Mahfoud93] Samir W. Mahfoud. "Finite Markov Chain Models of an Alternative Selection Strategy for the Genetic Algorithm". IlliGAL Report nº 91007. May, 1993.

[Michalewicz91] Zbigniew Michalewicz; George A. Vignaux; Matthew Hobbs. "A non standard Genetic Algorithm for the Nonlinear

Transformation Problem". *ORSA Journal on Computing*, Vol. 3, Nº 4, 1991.

[Neto95] João Neto. *Algoritmos Genéticos Distribuídos e suas aplicações.* Departamento de Engenharia Electotécnica e de Computadores. Faculdade de Engenharia da Universidade do Porto. 1995.

[Potvin96] Jean-Yves Potvin; Samy Bengo. "The Vehicle Routing Problem with Time Windows- Part II: Genetic Search". *INFORMS Journal on Computing*, Vol 8, nº 2. 1996.

[Talbi91] E. G. Talbi; P. Bessière. "A Parallel Genetic Algorithm for the Graph Partitioning Problem". Laboratoire de Génie Informatique / Institut IMAG. University of Grenoble. 1991

[Tanase90] Reiko Tanese. "Distributed Genetic Algorithms". Department of Electrical Engineering and Computer Science. University of Michigan. 1990.

[Taylor95] Jeffrey S. Taylor. "A Parallel Genetic Algorithm that Utilizes the PVM libraries". Department of Biophysics. University of Pennsylvania. 1995.

[Whitley93] Darrel Whitley. "A Genetic Algorithm Tutorial". Technical Report CS-93-103. Department of Computer Science. Colorado State University. 1993.

[Yang97] Rong Yang. "Solving Large Travelling Salesman Problems with Small Populations". *Preceedings of GALESIA97*. 1997.

[Zeigler91] Bernard P. Zeigler; Jinwoo Kim. "Characteristics of an Asynchronous Parallel Genetic Algorithm". Department of Electrical and Computer Engineering. University of Arizona. 1991.

3. Outros Assuntos

[Anderson94] Edward J. Anderson; Michael C. Ferris. "Genetic Algorithms for Combinatorial Optimization: The Assembly Line Balancing Problem". *ORSA Journal on Computing*, Vol. 6, Nº 2, 1994.

[Balakrishnan] V. K. Balakrishnan. *Introductory Discrete Mathematics*. Prentice-Hall International, Inc.

[Bondy78] J. A. Bondy; U.S.R.Murty. *GraphTheory with Applications*. The MacMillan Press, Ltd. 1978

[Dubois95] Paul Dubois. *Using csh and tcsh*. O'Reilly & Associates, Inc. 1995.

[Fischetti96] Matteo Fischetti; Paolo Toth. "A polyedral approach to the Asymetric Traveling Salesman Problem". University of Udine. 1996.

[ILTEC93] ILTEC - Intituto de Linguistica teórica e computacional. *Diccionário de Termos Informáticos*. Edição Cosmos. 1993.

[Kernighan88] Brian Kernigham; Dennis M. Ritchie. *The C Programming Language*. Second Edition. Prentice Hall Software Series. 1988.

[Knuth73] Donald E. Knuth. *The art of Computer Programming.* Second Edition. Addison-Wesley Publishing Company. 1973.

[Lawler85] E. L. Lawaler; J. K. Lenstra; A. H. G. Rinnoy kan; D. B. Shmoys. *The Traveling Salesman Problem - A Guided Tour to Combinatorial Optimization.* Wiley - Interscience series in discrete mathematfics. 1985.

[Ross92] Kenneth A. Ross; Charles R. B. Wright. *Discrete Mathematics.* Third Edition. Prentice-Hall International, Inc. 1992.

[Tanembaum90] A. S. Tanembaum. *Distributed Operating Systems.* Prentice Hall. 1990.

[Tucker80] Alan Tucker. *Applied Combinatorics.* John Wiley and sons, Inc. 1980.

[Waite90] Mitchell Waite; Donald Martin; Stephen Prata. *The Waite's Groups UNIX Primer Plus.* Second Edition. Howard W. Sams & Company. 1990.

Anexo A

1. Problema do Caixeiro Viajante de tamanho 17

NAME: br17
TYPE: ATSP
COMMENT: 17 city problem (Repetto)
DIMENSION: 17
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_SECTION

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	--	3	5	48	48	8	8	5	5	3	3	0	3	5	8	8	5
2	3	--	3	48	48	8	8	5	5	0	0	3	0	3	8	8	5
3	5	3	--	72	72	48	48	24	24	3	3	5	3	0	48	48	24
4	48	48	74	--	0	6	6	12	12	48	48	48	48	74	6	6	12
5	48	48	74	0	0	6	6	12	12	48	48	48	48	74	6	6	12
6	8	8	50	6	6	--	0	8	8	8	8	8	8	50	0	0	8
7	8	8	50	6	6	0	--	8	8	8	8	8	8	50	0	0	8
8	5	5	26	12	12	8	8	--	0	5	5	5	5	26	8	8	0
9	5	5	26	12	12	8	8	0	--	5	5	5	5	26	8	8	0
10	3	0	3	48	48	8	8	5	5	--	0	3	0	3	8	8	5
11	3	0	3	48	48	8	8	5	5	0	--	3	0	3	8	8	5
12	0	3	5	48	48	8	8	5	5	3	3	--	3	5	8	8	5
13	3	0	3	48	48	8	8	5	5	0	0	3	--	3	8	8	5
14	5	3	0	72	72	48	48	24	24	3	3	5	3	--	48	48	24
15	8	8	50	6	6	0	0	8	8	8	8	8	8	50	--	0	8
16	8	8	50	6	6	0	0	8	8	8	8	8	8	50	0	--	8
17	5	5	26	12	12	8	8	0	0	5	5	5	5	26	8	8	--

2. Problema do Caixeiro Viajante de tamanho 70

NAME: ft70
TYPE: ATSP
COMMENT: Asymmetric TSP (Fischetti)
DIMENSION: 70
EDGE_WEIGHT_TYPE: EXPLICIT
EDGE_WEIGHT_FORMAT: FULL_MATRIX
EDGE_WEIGHT_SECTION

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
1	----	375	1000	1011	853	950	936	1027	1101	1235	1279	1109	1141	976	1389	858	1368	1110	1263	1435	1091	936	1047	962	1079	1083	826	926	1004	1203	1037	1105	1229	986	1227
2	609	----	1068	980	1029	976	1068	1144	1100	1342	1262	1097	1304	1217	1494	960	1214	1233	1040	1175	1122	1237	1133	1031	1192	1004	1005	1087	1119	1109	1019	1087	1248	1257	1196
3	669	419	----	888	528	955	989	989	794	1185	1055	987	933	1009	1022	709	1124	1137	1203	1048	984	845	657	600	880	998	874	739	603	1042	939	1024	986	813	1136
4	665	645	664	----	559	491	528	592	505	769	680	576	793	724	996	796	898	755	525	1002	639	629	676	635	694	854	562	543	524	790	495	632	968	715	923
5	632	606	506	676	----	659	742	781	582	926	778	865	868	620	798	499	996	787	883	792	858	615	438	483	671	523	527	557	562	661	544	609	871	587	837
6	712	373	621	441	416	----	441	666	565	524	541	507	803	574	680	613	739	875	860	1018	699	697	576	691	801	761	606	511	597	805	619	790	808	798	965
7	412	702	566	534	473	744	----	623	690	765	770	526	699	752	986	492	577	596	733	746	598	672	681	715	676	762	608	590	522	782	694	769	976	723	700
8	826	666	682	535	668	660	705	----	688	861	755	706	766	495	693	913	1001	768	727	1046	724	894	748	927	1051	816	863	743	654	822	872	952	1094	786	839
9	874	915	804	786	736	873	725	491	----	729	749	578	641	757	747	942	943	1023	937	1170	746	1066	943	808	943	893	1064	909	1005	1095	803	1046	1041	1079	933
10	806	919	992	826	940	683	913	470	456	----	728	555	653	586	802	962	896	1137	811	950	972	1005	797	934	1070	975	857	887	791	1055	852	946	1049	983	1037
11	773	960	1030	780	702	970	966	697	559	472	----	803	609	743	612	1100	974	970	970	1162	825	989	910	954	1156	1107	974	1092	1059	908	944	1118	1050	1023	1062
12	805	663	762	640	787	632	870	475	381	585	501	----	628	577	806	846	909	1021	880	1087	1009	934	1024	932	905	990	1032	994	847	1121	830	911	1138	1018	1134
13	910	777	916	662	851	937	745	695	527	528	402	533	----	909	669	911	1044	1129	938	1099	873	1187	888	956	1193	1126	1172	1026	800	979	1119	1041	1235	1154	1195
14	1058	766	1030	850	962	822	865	655	471	625	580	655	608	----	703	975	1238	1063	991	1213	885	987	1023	942	1137	956	889	923	935	1214	1023	1063	1339	983	1046
15	935	773	823	884	766	684	735	663	571	613	479	685	682	643	----	901	1005	902	877	1117	842	878	963	1038	1156	914	984	1052	999	1176	963	1175	1245	942	1114
16	647	593	614	715	510	570	622	492	696	670	681	532	710	716	1007	----	1025	786	989	812	936	662	825	588	842	916	765	563	573	985	908	794	825	759	1062
17	675	703	683	693	611	725	910	776	662	870	943	557	856	907	878	375	----	699	582	560	728	970	601	719	799	831	843	696	625	848	718	681	907	695	1095
18	852	654	736	652	626	608	767	795	635	851	827	876	673	663	692	567	555	----	760	648	682	734	458	790	645	480	679	676	665	805	767	812	845	861	1080
19	676	602	822	793	723	660	862	636	719	828	885	742	942	880	967	492	606	587	----	551	710	955	743	877	672	582	785	746	707	821	477	502	898	734	958
20	820	868	737	766	816	800	744	861	631	819	966	775	1015	755	855	709	774	617	494	----	854	677	563	725	799	482	799	770	742	816	506	755	866	710	997
21	844	767	714	878	815	561	758	717	752	704	821	642	944	600	808	651	662	398	377	793	----	780	726	868	582	686	768	937	914	680	628	688	762	749	931
22	894	676	988	739	891	721	818	957	878	880	788	849	981	721	828	806	1065	982	793	913	824	----	797	747	545	706	498	582	690	912	904	985	972	680	969
23	721	892	856	716	760	758	739	640	635	787	908	803	876	716	717	727	1099	1018	1015	1038	798	492	----	574	651	395	666	707	827	927	674	995	1005	811	870
24	811	753	647	617	776	593	724	835	588	768	645	641	601	745	927	434	997	749	658	990	798	550	777	----	632	594	483	744	646	644	519	609	787	674	909
25	678	604	575	573	490	579	453	828	776	847	727	786	622	630	925	672	757	817	646	794	700	630	627	515	----	652	611	486	514	598	468	775	798	501	727
26	731	732	725	489	589	589	680	626	799	712	803	556	656	438	737	666	872	817	972	1107	711	656	552	602	514	----	765	697	666	670	595	745	834	849	855
27	772	738	682	764	816	664	570	905	717	697	860	848	948	547	1005	756	991	1006	723	900	799	666	656	646	624	610	----	775	651	764	687	734	833	626	1018
28	910	887	809	662	731	716	593	859	803	630	677	712	760	547	708	628	1054	891	742	867	856	515	331	512	470	531	496	----	669	660	795	662	777	783	801
29	908	545	674	582	724	560	725	639	563	628	676	630	602	733	902	440	949	750	837	902	801	497	619	561	750	630	492	616	----	804	476	746	844	730	781
30	1085	780	885	770	888	874	926	884	638	611	717	693	577	669	766	852	1054	1168	1098	1000	1039	613	663	612	673	748	758	622	712	----	787	986	879	775	928
31	878	696	874	663	770	717	884	655	731	732	732	902	803	960	869	500	778	784	861	857	792	885	630	762	761	613	752	690	885	590	----	838	742	805	716
32	813	784	877	869	876	781	1000	859	706	634	754	805	707	746	843	553	822	959	704	833	949	938	822	920	966	816	808	793	784	648	370	----	670	692	782
33	651	673	934	789	845	801	764	715	548	747	612	704	451	772	786	642	1139	871	941	960	797	811	566	708	687	632	597	458	945	437	530	661	----	633	664
34	929	854	813	724	841	643	752	877	605	730	710	834	647	690	1005	492	1060	804	895	938	718	972	667	849	704	740	877	645	746	514	475	423	475	----	589
35	755	447	1127	804	1000	723	994	736	822	832	875	733	776	786	766	778	957	916	992	1118	932	801	826	680	994	749	756	787	999	623	636	564	458	684	----
36	1858	1976	1887	1686	1781	1843	1899	1634	1777	1688	1743	1598	1732	1402	1622	1612	2196	2109	2183	2035	1905	2180	1912	1921	1990	2028	2069	1981	1926	1987	2098	2050	1948	2016	1931
37	1023	1198	1259	1228	1416	1113	1340	1334	1089	1213	1244	1135	1397	914	1167	1038	1399	1487	1368	1555	1444	1164	1179	1341	1207	1362	1318	1267	1166	1195	1360	1142	1318	1300	1313
38	1203	1180	1235	1260	1198	1296	1196	1297	1266	1350	1316	1346	1498	1353	1361	1120	1501	1386	1508	1532	1483	1369	1348	1271	1430	1305	1405	1210	1393	1494	1353	1596	1381	1423	1319
39	1137	1280	1415	1140	1466	1182	1287	1125	1146	1204	1134	1012	1215	788	1310	1290	1483	1543	1527	1605	1548	1426	1418	1461	1494	1605	1516	1484	1438	1644	1347	1455	1452	1311	1481
40	1044	962	1287	1247	1101	1285	1134	1081	1249	1282	1241	1225	1197	1129	1382	1128	1326	1264	1422	1308	1409	1197	1215	1096	1038	1190	1330	1210	1147	1424	1229	1126	1325	1315	1466
41	1071	1028	1447	1400	1224	1301	1366	1154	1187	1060	1056	1278	1224	1152	1141	1110	1471	1481	1432	1546	1213	1158	1123	1050	1148	1047	1159	1127	1214	959	1067	1112	925	1170	1050
42	1050	882	1342	1161	1325	1217	1275	1168	1197	1192	1178	1299	1322	1062	1302	1131	1501	1478	1236	1330	1422	1063	1244	931	954	1228	1073	1092	1210	1338	1333	1161	1191	1241	1280
43	1118	1166	1415	1307	1108	1232	1357	1272	1258	1393	1412	1346	1249	1158	1211	1082	1498	1420	1245	1451	1139	1208	1146	1007	965	1377	1144	1251	1092	1159	1167	1169	1378	1072	1209
44	1120	1026	1104	1029	1039	1144	875	1148	1176	1140	1117	944	1065	1279	1132	1079	1329	1250	1270	1470	1094	1095	973	1079	1243	1148	1235	988	1039	807	953	912	1069	1152	879
45	1369	1236	1472	1205	1340	1223	1195	1303	1266	1524	1284	1353	1294	1243	1534	1132	1252	1399	1376	1400	1504	1185	1081	1080	1336	1235	1127	965	1366	1145	1379	1342	1244	1257	1159
46	1562	1364	1372	1207	1311	1293	1237	1450	1295	1368	1284	1359	1349	1471	1459	1602	1502	1487	1531	1802	1496	1493	1363	1448	1562	1356	1530	1300	1568	1290	1311	1225	1346	1614	1272
47	1663	1689	1725	1516	1698	1648	1597	1522	1702	1575	1471	1725	1775	1839	1556	1778	1900	1901	1845	1982	1753	1932	1593	1793	1768	1808	1694	1732	1700	1705	1616	1638	1589	1964	1692
48	1207	1142	1416	1333	1350	1096	1469	1331	1341	1332	1410	1418	1320	1257	1638	1404	1293	1419	1272	1332	1318	1518	1288	1272	1403	1328	1482	1452	1402	1086	1186	1338	1364	978	1246
49	1080	1222	1165	1152	1249	1021	974	1077	1082	1231	1338	1172	1072	1394	1483	1154	915	1295	1308	1203	1272	1143	1187	1058	1309	1164	1058	1304	1248	1046	1127	1034	1160	1227	1127
50	1220	1228	1212	1160	1249	1025	1072	943	843	1070	913	1120	1022	1014	857	1410	1417	1291	1131	1561	1193	1271	1241	1199	1450	1411	1352	1226	1209	986	1194	1152	1041	1327	1072
51	885	859	1096	981	946	697	832	762	749	852	779	875	995	769	838	781	697	536	491	743	655	870	615	796	746	719	891	983	881	947	847	700	1057	627	1040
52	1035	921	1037	989	943	922	1038	886	858	1008	1056	852	1085	931	1017	725	863	857	642	877	723	996	781	970	926	850	977	935	1041	855	973	806	880	731	1173
53	769	779	788	757	686	749	800	793	930	793	913	690	965	526	837	679	953	882	639	983	847	587	624	657	708	503	807	888	738	929	676	888	859	798	818
54	723	663	638	515	582	691	689	738	744	828	936	626	842	698	1059	598	829	756	656	981	636	640	627	606	774	550	601	554	784	745	610	873	974	648	893
55	753	653	755	855	820	700	852	884	903	878	562	874	759	772	759	960	848	930	839	1024	661	873	664	707	629	764	954	943	691	939	839	861	1012	748	907
56	649	622	595	759	775	682	844	678	630	823	826	676	898	573	1009	761	669	738	681	888	641	840	580	710	777	511	688	716	808	523	433	658	742	641	899
57	739	602	661	823	656	690	849	719	676	486	494	695	487	755	758	562	719	687	789	756	635	743	648	762	765	562	779	833	788	637	490	661	641	395	724
58	807	776	908	1020	667	1005	973	912	861	1184	1074	989	1111	1025	1102	833	1235	1076	1128	1235	1068	959	677	682	981	952	977	660	793	882	758	867	1157	946	1085
59	1047	1017	869	1092	940	1142	985	1163	1165	1249	1064	1303	1003	857	1263	843	1229	1183	1188	1192	1164	917	847	861	898	840	862	801	834	1051	1035	961	1283	1104	1152
60	1081	904	825	949	716	1083	990	847	762	760	496	877	949	1036	930	838	909	1029	844	1127	1095	825	914	1019	977	1011	881	861	880	1128	1026	833	977	1025	887
61	1006	952	726	975	904	1099	970	718	724	717	792	865	1091	947	799	1019	986	1153	817	962	1142	821	753	947	1049	854	1037	771	990	849	997	762	876	887	706
62	1050	835	996	835	678	941	962	816	678	689	674	941	1082	1102	939	940	806	980	730	1033	897	948	938	744	874	771	851	887	816	1099	854	946	871	890	877
63	962	1085	996	1061	875	1138	910	891	667	805	532	810	936	1081	835	872	885	1005	724	1038	1104	932	725	961	1082	869	985	773	796	1099	1085	877	1078	1011	778
64	1927	1983	1997	1959	2157	2170	2045	2221	2159	2159	2264	2231	2046	1910	2335	2056	1966	1987	2199	1989	2150	1950	2147	2255	2087	1995	2160	2226	2154	2039	2188	2019	1977	2218	2043
65	2129	2282	2240	2197	2273	2185	2270	2203	2402	2253	2413	2255	2321	2107	2259	2100	2034	2205	2269	2219	2272	1814	2239	2166	2161	2217	2002	2092	1996	2253	2289	2170	2251	2071	2307
66	1923	1990	2082	1882	1829	2080	1929	1977	2138	2100	2001	1977	2120	1947	2141	1944	1927	1964	2176	2114	2267	1893	1869	2108	2138	1996	1960	1994	2081	2130	2058	1978	2068	2200	2045
67	2271	2110	2196	2019	2099	2085	2033	2056	2015	2050	2189	2107	1949	2157	2134	2041	2197	2053	2072	1750	2205	2042	2120	2107	1977	1940	2022	2106	2040	1906	1854	2049	1808	2076	2052
68	2170	2021	2292	1997	2202	2012	2037	2216	2244	2226	2018	2067	2008	2096	2295	2040	2053	2053	2003	2067	1978	2182	2142	2157	2151	2089	2183	1900	2175	1963	2121	2044	2065	1883	2122
69	2243	2278	2112	2059	2217	2099	2293	2166	2311	2106	2136	2348	1952	1959	2380	2041	2253	1954	2053	1801	2215	2009	2104	2283	2228	2008	1935	2167	2264	2064	2026	2138	1856	2037	1875
70	2058	2030	2049	2062	2021	2048	2033	2126	2328	2299	2170	2276	2164	1985	2236	2245	2137	1972	2182	1902	2327	2130	2046	2237	2246	2273	2163	2031	2267	1981	2073	2057	1955	2224	2130

36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	
1096	1235	973	753	1165	1518	1064	1285	1149	1312	914	1161	1349	1418	1515	1170	966	1112	1049	1064	912	901	1040	1086	1177	1118	1239	1103	1203	1094	1237	1143	1195	1293	1229	1
1061	1284	1142	951	1117	1329	1320	1284	1340	1282	934	1207	1200	1559	1717	1190	1168	1213	1144	1196	1317	1112	1048	1259	1008	871	1326	1455	977	1139	1335	974	1352	1468	1057	2
872	963	1013	710	801	1189	1126	977	1102	1069	683	1102	1253	1316	1373	978	1046	881	998	1001	948	751	982	1092	881	829	1335	1082	1071	1053	941	889	1100	1164	970	3
665	809	811	496	794	1031	759	714	829	764	609	504	1040	903	986	544	703	543	942	814	712	794	546	741	737	430	926	1047	660	686	558	632	944	808	890	4
541	787	571	558	594	1038	910	761	873	765	648	651	970	1026	1307	811	669	749	621	548	748	649	697	923	652	707	879	968	617	879	706	773	1035	944	863	5
654	698	636	676	842	986	886	910	690	897	464	556	1062	1178	1097	697	830	869	627	821	661	784	848	868	562	623	927	823	544	475	852	674	803	814	786	6
704	923	692	467	751	964	813	806	818	538	485	700	865	996	1234	548	645	766	800	759	606	786	681	651	549	482	901	792	702	544	726	456	902	925	827	7
968	1126	1039	662	789	1152	1080	798	884	852	986	622	1202	1255	1072	813	980	841	984	929	872	868	893	810	876	791	1090	1030	708	822	879	577	1058	896	1107	8
958	1078	1100	820	845	1225	975	971	755	1020	956	690	1241	1129	939	820	922	981	1105	961	1136	922	905	956	1001	785	1087	1012	905	839	921	753	913	755	981	9
1094	1111	1122	807	1019	1342	1160	883	814	1119	870	932	1124	1364	910	782	1104	1072	1063	938	929	939	926	1084	831	864	1151	1137	618	757	828	481	786	740	980	10
975	1172	1154	833	988	1312	1045	1154	592	1182	879	815	1097	1368	1193	1001	1106	1003	1026	1130	1007	967	1085	973	922	1006	1271	1285	505	737	875	689	1027	842	1123	11
745	854	851	740	818	965	740	868	514	883	968	735	1046	1307	948	694	847	1014	1023	1016	949	1047	882	1138	744	844	1307	1154	657	721	856	664	927	730	954	12
981	1185	1111	858	1143	1435	1001	1012	833	1205	1129	1027	1087	1332	1057	968	1110	849	1089	1207	1026	1111	993	902	1075	893	1112	1073	474	624	684	773	1074	750	1060	13
1040	1025	1180	956	932	1412	906	1006	853	1155	871	836	1155	1480	1172	1011	1157	1050	1274	1125	975	950	1092	1001	858	795	1295	1039	615	856	948	721	886	799	1149	14
997	1091	1096	802	990	1247	1022	876	799	949	948	890	1137	1274	912	759	898	1094	1040	1175	969	1116	1076	1045	808	837	1244	1071	398	656	702	757	998	786	1207	15
882	1084	794	743	918	1318	931	1050	611	913	809	582	742	1357	1068	731	839	816	718	828	975	707	743	830	869	897	1194	1054	918	832	723	785	942	910	985	16
688	946	811	699	969	1136	716	788	958	656	802	709	627	1002	1098	690	559	685	752	689	784	926	790	864	776	936	1195	1008	699	845	833	687	940	972	688	17
723	838	713	675	777	1154	830	628	974	706	608	855	783	976	1245	920	684	781	727	678	600	809	806	817	766	871	1144	1058	616	862	762	836	1124	979	729	18
782	853	577	621	666	1088	883	770	921	868	973	877	626	1113	1233	654	709	888	781	941	740	1006	685	675	864	902	926	677	808	800	874	931	991	1050	781	19
639	856	556	800	771	879	492	692	931	626	779	553	526	1047	1112	528	455	754	744	755	483	843	527	636	725	836	837	800	932	917	650	831	1038	864	836	20
745	890	786	747	555	902	872	694	975	763	609	816	756	1192	1166	912	868	809	929	994	917	819	654	871	691	921	906	796	568	558	711	688	1092	990	912	21
620	937	763	775	755	1315	1000	787	988	1119	645	770	1092	1354	1337	784	713	829	888	704	664	712	824	932	872	707	1022	922	611	867	553	749	1007	773	932	22
544	932	926	914	738	1173	918	984	1028	1123	885	685	1145	1252	1354	954	954	765	748	755	742	738	895	836	869	975	1179	961	644	737	691	666	1073	743	1067	23
646	758	609	701	740	1067	657	780	852	788	773	612	1035	1194	1225	586	644	514	633	625	505	581	728	848	725	917	771	644	536	707	754	640	793	848	876	24
793	827	811	726	802	1006	692	697	696	943	586	675	922	1063	1286	579	707	717	737	682	559	456	710	923	515	621	894	792	556	683	409	512	751	647	847	25
432	1002	871	869	682	1258	747	712	946	993	563	846	1123	1189	1187	835	860	688	846	856	780	678	809	878	712	871	1082	1030	756	798	668	589	839	642	790	26
519	1110	936	871	695	1008	913	950	969	1059	776	615	1105	1347	1195	622	733	632	741	717	581	724	762	772	788	764	948	879	738	841	774	827	1001	663	938	27
656	996	716	750	724	1176	908	752	872	770	600	712	882	1081	1311	616	698	615	799	760	764	676	643	640	545	858	1006	901	542	685	582	663	1021	936	702	28
512	864	683	834	623	1008	800	712	703	822	452	805	986	977	1197	633	616	575	690	463	555	588	608	675	596	978	1029	696	670	782	846	884	838	730	616	29
787	1129	823	987	775	1338	1010	997	845	862	892	916	1174	1191	1161	879	720	973	841	680	698	888	907	801	827	998	998	986	519	624	615	780	628	575	596	30
519	574	406	632	501	925	509	627	924	671	716	547	874	1240	1251	516	662	700	809	812	852	788	1006	882	842	881	1154	849	557	899	756	683	906	745	646	31
511	626	580	693	589	991	506	698	870	649	842	557	1083	945	1276	576	587	698	957	904	1055	972	744	529	728	644	854	526	676	903	695	669	858	668	70	32
617	743	643	636	596	960	572	791	635	543	830	621	957	801	1228	802	802	874	825	886	944	873	728	593	677	976	922	664	742	608	676	755	600	693	421	33
543	730	578	609	667	978	586	662	795	697	778	750	982	886	1100	759	535	802	813	871	921	1022	809	605	602	707	873	678	732	903	753	683	683	656	671	34
824	757	611	868	726	1103	815	615	992	711	845	768	1137	853	1159	891	792	862	842	834	895	875	962	788	602	850	810	801	778	879	898	664	874	878	686	35
----	1262	1205	1130	1200	1605	1232	1167	1689	1926	2031	1923	2276	2160	2197	1858	1820	1924	1965	1932	1846	2129	1716	1877	1731	1769	2022	2157	1179	1230	1366	1386	1452	1320	1302	36
642	----	570	692	492	845	626	451	1350	1235	1232	1202	1404	1647	1647	1182	1040	1242	1281	1253	1205	1376	1280	1191	1120	1173	1432	1339	727	734	847	820	822	718	557	37
742	864	----	786	711	1009	841	876	1334	1284	1408	1321	1587	1572	1730	1453	1323	1424	1514	1339	1376	1518	1095	1103	1095	947	1540	1562	515	819	601	716	649	648	526	38
530	787	731	----	558	1148	886	719	1126	1300	1458	1332	1705	1571	1622	1521	1289	1388	1339	1421	1326	1425	1185	1349	1149	1148	1646	1506	874	974	697	789	812	965	653	39
650	702	332	576	----	712	403	448	1354	1125	1256	1268	1409	1514	1535	1239	1018	1029	1219	1127	1140	1347	1217	1202	1277	944	1469	1354	614	648	572	675	754	838	514	40
671	548	682	583	547	----	747	650	1060	861	708	1233	1670	1196	1380	1110	1214	1230	1144	1313	1159	1207	1138	1069	1158	1045	1409	1022	645	951	700	897	718	810	751	41
811	515	464	601	721	673	----	580	1322	1111	1156	1295	1583	1473	1836	1096	1289	1372	1242	1128	1113	1277	1091	1258	1215	928	1601	1381	553	604	827	763	752	607	602	42
509	648	652	700	699	722	349	----	1288	1274	1108	1056	1375	1471	1755	1239	1157	1095	1104	1176	1096	1378	1110	1231	1050	1253	1566	1279	684	600	690	753	896	824	490	43
989	1326	1181	1080	1152	1508	1097	1254	----	1029	1282	967	1259	1103	762	1172	1106	1289	1155	1229	1137	1257	1157	1211	1087	1035	1381	1100	713	582	646	679	705	638	814	44
1023	1543	1256	1457	1439	1619	1341	1464	765	----	1251	937	1100	770	937	1185	1213	1297	1468	1344	1258	1387	1377	1352	1221	1439	1551	1341	783	794	837	971	896	818	1223	45
1113	1509	1429	1372	1459	1608	1528	1451	821	1126	----	1261	1616	1183	1039	1341	1539	1586	1500	1561	1585	1420	1221	1005	1231	1323	1135	1045	883	937	1058	981	1022	842	1098	46
1593	1727	1624	1691	1833	1787	1547	1663	935	1285	639	----	1559	1194	1452	1645	1659	1879	1792	1793	1757	1775	1413	1283	1377	1409	1456	1211	1301	1260	1246	1203	1182	1215	1502	47
1238	1384	1239	1298	1172	1521	1398	1248	967	1429	1175	1042	----	1055	1124	1039	1181	1235	1126	1238	971	1373	1390	1259	1411	1277	1448	1192	1210	1120	1181	1138	1335	1176	1215	48
931	1474	1106	1127	1234	1597	1225	1238	398	1065	906	791	1011	----	841	1112	1254	1240	1325	1411	1294	1387	1147	1209	1187	1076	1350	1169	829	685	838	634	827	601	1021	49
550	1437	1123	1260	1166	1555	1361	1309	571	1249	1292	1123	765	791	----	1043	1133	1215	1467	1472	1387	1274	1205	1152	1302	1155	1467	1234	839	531	806	698	680	720	1023	50
606	785	628	672	821	1151	777	781	1053	945	939	799	880	1149	1217	----	734	612	641	787	762	731	787	939	995	830	1077	821	579	903	717	875	1145	870	684	51
786	988	844	765	951	1120	936	831	1147	1073	709	513	1082	1269	1397	455	----	761	682	911	859	822	1071	949	914	1180	928	992	866	810	891	685	1151	1144	785	52
703	931	594	631	866	1043	984	669	996	976	753	578	1144	1078	1291	535	525	----	716	586	573	600	810	813	776	927	1135	916	831	811	864	958	1063	835	578	53
618	866	585	471	807	1138	803	742	684	1011	779	707	938	1154	1056	617	540	506	----	686	788	914	877	965	762	831	1078	997	752	856	694	717	898	908	690	54
853	891	792	660	741	1217	1039	912	945	591	868	773	1071	994	1228	605	566	381	563	----	759	686	482	736	513	752	999	810	851	957	967	725	1131	909	685	55
662	727	656	690	648	824	707	642	954	653	907	676	818	936	1102	407	390	645	482	560	----	865	701	774	776	911	746	846	672	626	794	693	805	829	585	56
711	673	754	619	568	1014	559	536	802	656	647	654	986	867	1145	360	545	561	408	457	417	----	669	678	637	866	901	713	466	590	890	546	945	772	572	57
956	1118	984	715	892	1172	1190	1005	713	476	873	1042	1351	790	1078	1069	783	978	1030	1009	798	764	----	675	992	1150	1146	1048	859	997	947	789	978	798	1035	58
1047	1259	1083	1020	971	1529	1078	1268	889	760	1074	1244	1282	939	1191	1053	1097	1154	1020	1117	1191	897	676	----	909	1120	1207	1040	937	965	927	965	1041	871	1276	59
1122	1063	1136	939	918	1408	911	1040	800	621	1072	1012	1237	1070	1103	914	934	990	1063	900	1165	1056	496	584	----	698	963	854	749	800	797	695	1126	906	1063	60
1021	1048	951	815	1010	1139	945	849	735	540	830	1132	1224	873	1185	1030	1111	914	997	1145	1084	1019	657	701	503	----	778	818	489	568	766	516	1044	925	782	61
921	763	792	777	763	974	649	786	836	649	926	988	1233	1064	1136	964	878	911	720	668	945	1075	536	538	521	718	----	890	679	736	761	574	1004	940	774	62
957	867	762	795	935	1018	826	816	782	577	1090	1033	1010	839	1271	1019	1040	1051	968	862	1025	967	433	389	404	679	498	----	686	850	801	816	1138	902	821	63
1606	1494	1604	1558	1461	1832	1398	1580	1490	1921	1875	1859	1955	1418	1962	1990	2011	2198	2146	2312	2146	2251	2255	2335	2273	1991	2373	2181	----	486	752	415	1018	923	916	64
1728	1627	1569	1622	1533	1960	1795	1579	1787	2181	2233	1937	1875	1408	1993	2240	2289	2217	2316	2263	2346	2263	2179	2266	2332	2320	2588	2275	691	----	978	787	1209	881	963	65
1592	1451	1525	1531	1474	1911	1520	1660	1526	2080	1956	1578	1999	1565	1918	2259	2279	2017	2238	2231	2088	2038	2067	1866	2106	2195	2452	2069	471	566	----	433	645	552	719	66
1577	1404	1441	1507	1539	1976	1675	1460	1687	2010	2212	1746	1956	1659	2021	1959	1995	1995	1916	2095	1960	2185	1916	1993	2040	2181	2270	2134	463	443	618	----	909	816	871	67
1473	1527	1532	1468	1613	1794	1644	1634	1441	1840	2122	1932	2006	1403	1761	1915	1889	1926	1994	1983	1999	2031	1926	1662	2008	2318	2310	2285	386	627	627	759	----	712	749	68
1612	1581	1757	1682	1713	1918	1738	1632	1726	2061	2041	1934	2100	1601	2052	2200	2048	2068	1979	2066	1989	2081	1967	1715	2213	2352	2395	2113	588	640	641	461	682	----	757	69
1673	1669	1563	1658	1645	1853	1785	1729	1714	1988	2230	1766	1910	1507	1817	2175	2111	2024	2289	2177	2214	2080	1984	1770	2042	2173	2271	2101	689	522	613	514	589	663	---	70

Anexo B

1. Código do mestre

1.1. Ficheiro de dados

```
#include <stdio.h>
#include <stdlib.h>
#include "pvm3.h"

#define NCIDADES 70
#define NILHAS 32
#define TAG 0

int custos[NCIDADES*NCIDADES];
```

1.2. Programa Principal

```
#include "dados_mestre.h"

main()
{
        int i,j,k,l;

        float melhores[NILHAS];

        char linha[10];
        FILE *fp2;

        extern double treal(void);

        int numt,bufid,info,meu_tid,tid_enviar,tid_receber;
    int tids_tom[NILHAS/4],tids_crazy[NILHAS/4],tids_jerry[NILHAS/4],tids_riff[NILHAS/4];

        int cadeia[NCIDADES];

        int alea_0_tam(int tam);

        meu_tid=pvm_mytid();
```

```
    /*********************************/
    /* Inicializa a matriz de custos         */
    /*********************************/
    fp2=fopen("doc70.txt","r");
    for(j=0; j<NCIDADES*NCIDADES; j++)
            custos[j]=atoi(fgets(linha,10,fp2));
    fclose(fp2);


    /*************************************************************/
    /* Inicializa as varias ilhas e envia-lhes a matriz de custos          */
    /* e a parte correspondente da populacao                                */
    /*************************************************************/
    numt=pvm_spawn("ilha2",0,1,"tom.fe.up.pt",NILHAS/4,tids_tom);
    numt=pvm_spawn("ilha2",0,1,"crazy.fe.up.pt",NILHAS/4,tids_crazy);
    numt=pvm_spawn("ilha2",0,1,"jerry.fe.up.pt",NILHAS/4,tids_jerry);
    numt=pvm_spawn("ilha_riff2",0,1,"riff.fe.up.pt",NILHAS/4,tids_riff);


    /**********************************/
    /* Envia os dados das ilhas do tom          */
    /**********************************/
    for(i=0;i<NILHAS/4;i++){
            if(NILHAS==4){
                    tid_enviar=tids_crazy[0];
                    tid_receber=tids_riff[0];
            }
            else{
                    tid_enviar=(i==NILHAS/4-1? tids_crazy[0] : tids_tom[i+1]);
                    tid_receber=(i==0?tids_riff[NILHAS/4-1]:tids_tom[i-1]);
            }
            bufid=pvm_initsend(PvmDataDefault);
            info=pvm_pkint(custos,NCIDADES*NCIDADES,1);
            info=pvm_pkint(&tid_enviar,1,1);
            info=pvm_pkint(&tid_receber,1,1);
            info=pvm_send(tids_tom[i],TAG);
    }


    /************************************/
    /* Envia os dados das ilhas do crazy          */
    /************************************/
    for(i=0;i<NILHAS/4;i++){
            if(NILHAS==4){
                    tid_enviar=tids_jerry[0];
                    tid_receber=tids_tom[0];
            }
            else{
                    tid_enviar=(i==NILHAS/4-1?tids_jerry[0]:tids_crazy[i+1]);
                    tid_receber=(i==0?tids_tom[NILHAS/4-1]:tids_crazy[i-1]);
            }
            bufid=pvm_initsend(PvmDataDefault);
            info=pvm_pkint(custos,NCIDADES*NCIDADES,1);
            info=pvm_pkint(&tid_enviar,1,1);
            info=pvm_pkint(&tid_receber,1,1);
            info=pvm_send(tids_crazy[i],TAG);
}
```

```
/*************************************/
/* Envia os dados das ilhas do jerry           */
/*************************************/
for(i=0;i<NILHAS/4;i++){
            if(NILHAS==4){
                    tid_enviar=tids_riff[0];
                    tid_receber=tids_crazy[0];
            }
            else{
                    tid_enviar=(i==NILHAS/4-1?tids_riff[0]:tids_jerry[i+1]);
                    tid_receber=(i==0?tids_crazy[NILHAS/4-1]:tids_jerry[i-1]);
            }
            bufid=pvm_initsend(PvmDataDefault);
            info=pvm_pkint(custos,NCIDADES*NCIDADES,1);
            info=pvm_pkint(&tid_enviar,1,1);
            info=pvm_pkint(&tid_receber,1,1);
            info=pvm_send(tids_jerry[i],TAG);
}

/*************************************/
/* Envia os dados das ilhas do riff            */
/*************************************/
for(i=0;i<NILHAS/4;i++){
            if(NILHAS==4){
                    tid_enviar=tids_tom[0];
                    tid_receber=tids_jerry[0];
            }
            else{
                    tid_enviar=(i==NILHAS/4-1? tids_tom[0] : tids_riff[i+1]);
                    tid_receber=(i==0?tids_jerry[NILHAS/4-1]:tids_riff[i-1]);
            }
            bufid=pvm_initsend(PvmDataDefault);
            info=pvm_pkint(custos,NCIDADES*NCIDADES,1);
            info=pvm_pkint(&tid_enviar,1,1);
            info=pvm_pkint(&tid_receber,1,1);
            info=pvm_send(tids_riff[i],TAG);
}


/*****************************************************************/
/* Recebe os quatro melhores cromossomas de cada uma das ilhas          */
/* e escolhe o melhor                                                     */
/*****************************************************************/
printf("Vou receber\n");

for(i=0;i<NILHAS;i++){
            info=pvm_recv(-1,TAG);
            info=pvm_upkfloat(&melhores[i],1,1);
}

printf("Melhores:");
for(i=0;i<NILHAS;i++)
            printf("%f ",melhores[i]);
printf("\n");

pvm_exit();

}
```

```
int alea_0_tam(int tam){
/*********************************************************/
/* gera um numero inteiro aleatorio entre o e tamanho tam          */
/*********************************************************/

    return ( tam * rand()/32767);
}
```

2. Código das ilhas

2.1. Ficheiro de dados

```
#include <stdio.h>
#include <stdlib.h>
#include "pvm3.h"

#define NCIDADES 70
#define TAM_POPULACAO 16000
#define N_GERACOES 1000
#define P_MUTACAO 0.04
#define P_CRUZAMENTO 0.8
#define P_TORNEIO 0.8

#define TAG 0
#define TAG1 1
#define NILHAS 32
#define NEMIGRANTES 20
#define TEMIGRACAO 500

int custos[NCIDADES*NCIDADES];

struct cromossoma{
        int cadeia[NCIDADES];
        float avaliacao;
};

struct cromossoma subpop_corrente[TAM_POPULACAO];
```

2.2. Programa Principal

```
#include "dados.h"

main()
{
        int i,j,k;

        float envio;

        struct cromossoma melhor;

        int meu_tid,meu_pai,bufid,info,tid_enviar,tid_receber,alea;

        void genetic(int tam_pop, float p_cruzamento , float p_mutacao , float p_torneio);
```

```
float avalia(struct cromossoma cromossoma1);

extern int alea_0_tam(int);


meu_tid=pvm_mytid();
meu_pai=pvm_parent();

srand(meu_tid);
/*****************************/
/* recebe a matriz de custos          */
/*****************************/
bufid=pvm_recv(meu_pai,TAG);
info=pvm_upkint(custos,NCIDADES*NCIDADES,1);
info=pvm_upkint(&tid_enviar,1,1);
info=pvm_upkint(&tid_receber,1,1);

/**************************************/
/* Cria geracao inicial aleatoriamente                */
/**************************************/
for(i=0; i<TAM_POPULACAO; i++)
        for(j=0;j<NCIDADES;j++)
        subpop_corrente[i].cadeia[j]=alea_0_tam(NCIDADES-j-1);

/****************************************************************/
/* avalia um a um os cromossomas, colocando o resultado da avaliacao          */
/* na populacao corrente                                                      */
/****************************************************************/
for(i=0;i<TAM_POPULACAO;i++)
        subpop_corrente[i].avaliacao=avalia(subpop_corrente[i]);

/****************************************************************/
/* imprime os resultados, para a geracao indicada: a geracao, o melhor e          */
/* a media                                                                        */
/****************************************************************/
melhor=subpop_corrente[0];
for(i=0;i<TAM_POPULACAO;i++){
        if(melhor.avaliacao>subpop_corrente[i].avaliacao)
                        melhor=subpop_corrente[i];
}
printf("%d            %f\n",0,melhor.avaliacao);

/****************************************************************/
/* Vai realizar todas as operacoes geneticas para um numero de geracoes           */
/* N_GERACOES                                                                     */
/****************************************************************/
for(i=1;i<N_GERACOES;i++){
        /**********************************************/
        /* Gera a nova populacao realizando os tres operadores        */
        /* geneticos de seleccao, cruzamento e mutacao, utilizando    */
        /* como metodo de seleccao o torneio estocastico de ordem 1 */
        /**********************************************/
      genetic(TAM_POPULACAO,P_MUTACAO,P_CRUZAMENTO,P_TORNEIO);

        /**********************************************/
        /* Vamos emigrar: depois de TEMIGRACAO geracoes        */
        /* emigramos em numero de NEMIGRANTES                  */
        /**********************************************/
        if(i!=0 && i%TEMIGRACAO==0){
```

```
                        /************************/
                        /* Envia os emigrantes      */
                        /************************/
                        bufid=pvm_initsend(PvmDataDefault);
                        for(j=0;j<NEMIGRANTES;j++){
                                alea=alea_0_tam(TAM_POPULACAO);
                              info=pvm_pkint(subpop_corrente[alea].cadeia,NCIDADES,1);
                        }
                        info=pvm_send(tid_enviar,TAG1);

                        /************************/
                        /* Recebe os imigrantes      */
                        /************************/
                        bufid=pvm_recv(tid_receber,TAG1);
                        for(j=0;j<NEMIGRANTES;j++){
                                alea=alea_0_tam(TAM_POPULACAO);
                              info=pvm_upkint(subpop_corrente[alea].cadeia,NCIDADES,1);
                        }
                }

                /************************/
                /* Avaliacao ........           */
                /************************/
                for(j=0;j<TAM_POPULACAO;j++)
                        subpop_corrente[j].avaliacao=avalia(subpop_corrente[j]);

                /***********************************/
                /* Imprime resultados desta geracao       */
                /***********************************/
                for(j=0;j<TAM_POPULACAO;j++){
                        if(melhor.avaliacao>subpop_corrente[j].avaliacao)
                        melhor=subpop_corrente[j];
                }

                printf("%d            %f\n",i,melhor.avaliacao);getchar();
        }

        envio=melhor.avaliacao;
        bufid=pvm_initsend(PvmDataDefault);
        info=pvm_pkfloat(&envio,1,1);
        info=pvm_send(meu_pai,TAG);

pvm_exit();
}
```

2.3. Função de avaliação

```
#include "dados.h"

float avalia(struct cromossoma cromossoma1)
{

        /* converte a tabela de permutacoes cromossoma1 numa permutacao ciclo1 */

        int i,j,k,numero;
        int ciclo1[NCIDADES];
        int v, vs;
```

```
	float avali=0;

	/*******************************************************/
	/* Descodifica  a cadeia contendo a tabela de inversoes      */
	/* num ciclo contendo o as cidades do problema               */
	/*******************************************************/
	for(i=NCIDADES-1;i>=0;i--){
		j=cromossoma1.cadeia[i];
		for(k=NCIDADES-i-1;k>j;k--)
			ciclo1[k]=ciclo1[k-1];
		ciclo1[k]=i;
	}

	for(i=0; i<NCIDADES;i++){
		v=ciclo1[i];
		vs=((i==NCIDADES-1)?ciclo1[0]:ciclo1[i+1]);
		/******************************************/
		/* A posicao correspondente a matriz[i][j]      */
		/* se a transformarmos num vector sera          */
		/* dada por NCIDADES*i+j                         */
		/******************************************/
		avali=avali+custos[NCIDADES*v+vs];
	}

return avali;

}
```

2.4. Rotina para criar uma nova geração

```
#include "dados.h"

struct cromossoma pop_intermedia[TAM_POPULACAO];

void genetic
(int tam_populacao,float p_mutacao, float p_cruzamento, float p_torneio)
{
	int i,j,k;
	int alea;
	float alea_o;

	struct cromossoma torneio(float p_torneio);
	void cruzamento(int i);
	void mutacao(int i);
	float alea_0_1(void);
	int alea_0_tam(int tam);

	/******************************************/
	/* Selecciona cromossomas da populacao corrente     */
	/* atraves de um torneio estocastico de grau um      */
	/******************************************/
	for(i=0;i<TAM_POPULACAO;i++)
		pop_intermedia[i]=torneio(p_torneio);
```

```
        /**************************************************/
        /*  Realiza os operadores geneticos de cruzamento,        */
        /*  mutacao e copia                                       */
        /**************************************************/
        for(i=1;i<TAM_POPULACAO;i++){
                alea_o=alea_0_1( );
                if(alea_o<=p_cruzamento){
                        cruzamento(i);
                        i++;
                }
                if(alea_o<=p_cruzamento+p_mutacao && alea_o>p_cruzamento){
                        mutacao(i);
                }
                if(alea_o>p_cruzamento+p_mutacao){
                        alea=alea_0_tam(TAM_POPULACAO);
                        subpop_corrente[i]=pop_intermedia[alea];
                }
        }

return;
}


float alea_0_1(void){
/****************************************/
/* gera um numero aleatorio entre 0 e 1         */
/****************************************/

    return ( rand()/(float)32767);
}


int alea_0_tam(int tam){
/********************************************************/
/* gera um numero inteiro aleatorio entre o e tamanho tam        */
/********************************************************/

    return ( tam * rand()/32767);
}


struct cromossoma torneio(float p_torneio)
{
/**************************************************************/
/* selecciona um cromossoma atraves de um torneio estocastico de ordem 1,  */
/* com probabilidade p_torneio de selecionar o de menor custo              */
/**************************************************************/

        int alea1;
        int alea2;
        float alea;

        struct cromossoma retorno;

        alea1=alea_0_tam(TAM_POPULACAO);
        alea2=alea_0_tam(TAM_POPULACAO);

        alea=alea_0_1();
```

```
        if(p_torneio<=alea)          /* retorna o maior */

                retorno=subpop_corrente[alea1].avaliacao>subpop_corrente[alea2].avaliacao?s
                ubpop_corrente[alea1]:subpop_co  rrente[alea2];
        else                         /* retorna o menor*/

                retorno=subpop_corrente[alea1].avaliacao>subpop_corrente[alea2].avaliacao?s
                ubpop_corrente[alea2]:subpop_corrente[alea1];

return retorno;
}


void cruzamento(int i)
{

/***********************************************************************/
/* Realiza o cruzamento simples de um so ponto, de dois cromossomas pai      */
/* gerando dois cromossomas filho                                             */
/***********************************************************************/
        int alea1,alea2;
        int ponto_cruzamento;
        int j;

        alea1=alea_0_tam(TAM_POPULACAO);
        alea2=alea_0_tam(TAM_POPULACAO);
        ponto_cruzamento=alea_0_tam(NCIDADES);

        for(j=0;j<ponto_cruzamento;j++)
                subpop_corrente[i].cadeia[j]=pop_intermedia[alea1].cadeia[j];
        for(j=ponto_cruzamento;j<NCIDADES;j++)
                subpop_corrente[i].cadeia[j]=pop_intermedia[alea2].cadeia[j];
        i++;
        for(j=0;j<ponto_cruzamento;j++)
                subpop_corrente[i].cadeia[j]=pop_intermedia[alea2].cadeia[j];
        for(j=ponto_cruzamento;j<NCIDADES;j++)
                subpop_corrente[i].cadeia[j]=pop_intermedia[alea1].cadeia[j];

        return;
}


void mutacao(int i)
{

/***********************************************************************/
/* Realiza uma mutacao num ponto do cromossoma, com a restricao de que        */
/* o novo cromossoma nao deve ser maior do que o ponto de mutacao             */
/***********************************************************************/
        int alea1;
        int ponto_mutacao;
        int j;

        alea1=alea_0_tam(TAM_POPULACAO);
        ponto_mutacao=alea_0_tam(NCIDADES);

        j=alea_0_tam(NCIDADES-ponto_mutacao);
```

```
        pop_intermedia[alea1].cadeia[ponto_mutacao]=j;
        subpop_corrente[i]=pop_intermedia[alea1];
return;
}
```

2. Algoritmos Genéticos e Genéticos Paralelos

[Aarts94] E. H. L. Aarts; P. J. M. van Laarhoven; J. K. Lenstra; N. L. J. Ulder. "A Computational Study of Local Search Algorithms for Job Shop Scheduling". *ORSA Journal on Computing*, Vol. 6, Nº 2, 1994.

[Bäcka] Thomas Bäck. "Optimal Mutation Rates in Genetic Search". Department of Computer Science. University of Dortmund. Germany.

[Bäckb] Thomas Bäck. "Self-Adaptation in Genetic Algorithms". Department of Computer Science. University of Dortmund. Germany.

[Bäckc] Thomas Bäck. "Optimisation by means of Genetic Algorithms". Department of Computer Science. University of Dortmund. Germany.

[Bäckd] Thomas Bäck. "The Interaction of Mutation Rate, Selection and Self-Adaptation Within a Genetic Algorithm". Department of Computer Science. University of Dortmund. Germany.

[Baluja92] Shumeet Baluja. "A massively Distributed Parallel Genetic Algorithm". CMU-CS-92-196R. School of Computer Science. Carnegie Mellon University.

[Bean94] James C. Bean. "Genetic Algorithms and Random Keys for Sequencing and Optimisation". *ORSA Journal on Computing*, Vol. 6, Nº 2, 1994.