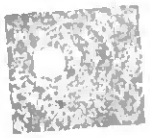
MARIA GABRIELA FIGUEIREDO DE CASTRO SCHÜTZ

RESOLUÇÃO DE PROBLEMAS DE OPTIMIZAÇÃO COMBINATÓRIA EM MÁQUINAS PARALELAS

Dezembro 1999



MARIA GABRIELA FIGUEIREDO DE CASTRO SCHÜTZ

RESOLUÇÃO DE PROBLEMAS DE OPTIMIZAÇÃO COMBINATÓRIA EM MÁQUINAS PARALELAS

Dissertação para obtenção do grau de Doutor em Matemática, na especialidade de Investigação Operacional, na Universidade do Algarve

Dezembro 1999

AGRADECIMENTOS

Quero expressar os meus sinceros agradecimentos a todos que de alguma forma me apoiaram e ajudaram a concretizar esta tese.

Agradeço principalmente aos meus orientadores Professor Doutor António Ruano e Professora Doutora Fernanda Marília Pires que tornaram possível a execução deste trabalho. Ao Professor António Ruano agradeço-lhe ter-me introduzido no domínio da Computação Paralela e supervisionado os trabalhos nesse âmbito. À Professora Marília Pires, o meu agradecimento por todos os seus sempre oportunos comentários, sugestões e correcções a propósito dos Problemas de Optimização Combinatória, feitos durante todo o desenrolar dos trabalhos. Agradeço-lhe ainda os incentivos e o apoio nas fases mais difíceis.

Agradeço também à Professora Doutora Teresa Almeida pelas suas sugestões e comentários a respeito do Problema de Optimização de Rotas de Veículos e ao Professor Doutor Joaquim Júdice pelos seus incentivos.

Ao Dr. Helder Daniel o meu agradecimento por se ter sempre prontificado a instalar e ligar os transputers, tendo ainda a paciência de me ensinar a configurar os respectivos ficheiros.

Agradeço aos colegas da Área Departamental de Engenharia Electrotécnica que de alguma forma me apoiaram nestes últimos anos. Destaco, em particular, o Professor Engenheiro José Silvestre, que infelizmente já não se encontra entre nós, e que desde o início de minha actividade na Escola Superior de Tecnologia sempre teve pelo meu trabalho uma grande consideração. A sua actuação contribuiu de forma decisiva para a minha candidatura ao Prodep e consequente dispensa por três anos para efectuar este trabalho.

Por último, mas não menos importante, agradeço à Daniela, à Alexandra e ao Ronaldo por me terem sempre apoiado emocionalmente, através de suas manifestações de amor e carinho.

NOTAÇÃO

Em cada capítulo foi utilizada a notação mais frequentemente encontrada na literatura para o tema tratado. Como foram estudados dois Problemas de Optimização Combinatória que usualmente não são tratados conjuntamente, uma mesma letra pode representar objectos diferentes conforme o contexto. No entanto, em cada capítulo, a primeira vez que se utiliza uma letra é explicitado o significado que tem nesse capítulo. Objectos gerais e comuns a todos os capítulos tiveram sempre a mesma notação.

Utilizou-se em toda a tese a notação usualmente adoptada para espaços vectoriais, matrizes, vectores, conjuntos, funções, etc.. Assim, menciona-se apenas a notação que, eventualmente, possa levantar alguma dúvida:

A^T	Matriz transposta da matriz A
G^c	Grafo complementar do grafo G
F^c	Conjunto complementar do conjunto F
x^c	É igual a $1-x$, quando $x \in \{0, 1\}$
$\|\alpha\|$	Valor absoluto do número real α
$\|F\|$	Cardinal do conjunto F

Utilizaram-se também as seguintes abreviaturas:

f.o.	função objectivo
dp	diferença percentual
top4	topologia para uma rede com 4 processadores

top8	topologia para uma rede com 8 processadores
top4c	topologia para uma rede com 4 processadores e mais comunicações entre os processadores
top8c	topologia para uma rede com 8 processadores e mais comunicações entre os processadores
ADP	Aumentar a Diversidade da População
AG	Algoritmo Genético
APA	Algoritmo de Pesquisa em Árvore proposto neste trabalho
APR	Algoritmo de Pesquisa em Árvore de Pardalos e Rodgers
DTE	Diminuir o Tempo de Execução
PCV	Problema do Caixeiro Viajante
PL	Programa Linear
POC	Problema de Optimização Combinatória
PQ 0-1	Programa Quadrático 0-1
PQL	Programa Quadrático com Limites nos valores das variáveis
PT	Pesquisa Tabu

Alguns conceitos utilizados neste trabalho são normalmente transmitidos em inglês não estando ainda a sua tradução em língua portuguesa amplamente aceite. Por isso, a par das abreviaturas utilizadas, indica-se também a nomenclatura inglesa original.

AE	Algoritmo Evolutivo	(Evolutionary Algorithm)
AS	Arrefecimento Simulado	(Simulated Annealing)
CSP	Comunicação de Processos Sequenciais	(Communicating Sequential Processes)

MIMD	Várias sequências de instruções várias sequências de dados	(Multiple Instruction stream Multiple Data stream)
MISD	Várias sequências de instruções uma sequência de dados	(Multiple Instruction stream Single Data stream)
PA	Pesquisa em Árvore	(Branch-and Bound)
PE	Processador	(Processor Element)
PMA	Programação com Memória Adaptável	(Adaptive Memory Programming)
PORV	Problema de Optimização de Rotas de Veículos	(Vehicle Routing Problem)
PRAM	Máquina Paralela de Acesso Aleatório	(Parallel Random Access Machine)
SCF	Sistema de Colónia de Formigas	(Ant Colony System)
SIMD	Uma sequência de instruções várias sequências de dados	(Single Instruction stream Multiple Data stream)
SISD	Uma sequência de instruções uma sequência de dados	(Single Instruction stream Single Data stream)

ÍNDICE

CAPÍTULO 1

INTRODUÇÃO

Em muitas e variadas áreas, tais como, transportes, logística, telecomunicações, informática, gestão e planeamento, etc., depara-se com problemas em que se tem como objectivo encontrar a melhor solução entre um número finito ou infinito numerável de alternativas discretas. São os chamados Problemas de Optimização Combinatória. A enorme quantidade de aplicações concretas com que se defrontam a generalidade das empresas e instituições levou, nas últimas décadas, a uma intensa actividade no sentido de desenvolver teorias e técnicas eficientes para a resolução destes problemas de Matemática Discreta.

A resolução deste tipo de problemas pode ser abordada de três formas distintas. O único processo de obter a solução óptima, sabendo reconhecer que efectivamente se está em presença de uma solução óptima, é através de algoritmos enumerativos. Trata-se de algoritmos exactos que obtêm garantidamente uma solução óptima examinando explicita ou implicitamente todas as soluções possíveis. Como, normalmente, os problemas com interesse real têm um grande número de variáveis e o número de

soluções a examinar tem uma relação exponencial com esse número, é óbvio que a utilização deste tipo de métodos não é, quase sempre, exequível. Recorre-se, então, a métodos ε-aproximados que são capazes de encontrar uma boa solução e garantir uma margem de erro menor ou igual a ε, ou a heurísticas ou meta-heurísticas que, normalmente, obtêm uma solução razoavelmente aceitável, embora não se tendo, na generalidade dos casos, qualquer conhecimento sobre a sua qualidade.

Qualquer método enumerativo pode levar horas, ou até dias, mesmo recorrendo a técnicas e meios computacionais eficientes. De um método aproximado é esperado que encontre uma solução próxima da óptima em tempo polinomial. As heurísticas são métodos que procuram uma boa solução num tempo de execução muito reduzido.

A formalização de problemas da vida real cai, quase sempre, na classe dos problemas NP-difíceis [Garey e Johnson (1979)], de média ou grande dimensão, sendo, por isso, pouco provável a sua resolução exacta em tempo aceitável.

Nas últimas décadas tem havido grande actividade na investigação de métodos e técnicas heurísticas, quer sejam métodos construtivos quer sejam métodos de pesquisa local (no sentido de pesquisa de uma vizinhança). Os primeiros algoritmos de pesquisa local começaram a ser estudados nos fins dos anos 50, quando foram introduzidos algoritmos de troca de arestas para o Problema do Caixeiro Viajante [Croes (1958)]. Nos anos seguintes, este conceito de trocas foi estendido e aplicado, com sucesso, a muitos outros problemas. Inicialmente estes algoritmos eram desenhados para um problema específico, tirando partido das suas características, não sendo muitas vezes considerados verdadeiros métodos de optimização. Mais recentemente, surgiram algumas abordagens de pesquisa local, baseadas em analogias com processos naturais, desenhadas num sentido genérico, com grande flexibilidade, o que permite a sua aplicação a uma variedade de problemas complexos e reais. Os exemplos mais conhecidos destas heurísticas de pesquisa local são: Arrefecimento Simulado, Algoritmos Genéticos e Pesquisa Tabu. A estes algoritmos também é usual chamar meta-heurísticas por poderem ser encarados como abordagens que, mais do que realizar, conduzem a pesquisa local executada por outros procedimentos heurísticos.

A grande evolução dos meios computacionais, que tem ocorrido na última década, tem permitido encarar a implementação de técnicas até aí impensáveis de executar, quer pela elevada capacidade de memória requerida, quer pelo elevado tempo de execução. O aparecimento dos primeiros computadores paralelos e a sua rápida evolução e generalização potenciou o desenvolvimento de muitos algoritmos paralelos. Inicialmente, só alguns foram realmente implementados e aplicados [Lootsma e Ragsdell (1988), Kindervater e Trienekens (1988)] devido às limitações das primeiras máquinas paralelas. Nos últimos anos, tem começado a ser possível resolver problemas que, devido aos elevados tempos de execução e/ou altos custos computacionais, eram na prática intratáveis.

O processamento paralelo, mesmo utilizando poucos processadores, acelera significativamente muitos algoritmos, permitindo a resolução em tempo razoável de problemas de maiores dimensões. É também, quase sempre, a única forma de obter soluções para aplicações em tempo real.

Se, por um lado, os algoritmos de Pesquisa Tabu e de Arrefecimento Simulado são processos eminentemente sequenciais, por outro lado, os Algoritmos Genéticos são naturalmente paralelizáveis, o mesmo acontecendo com os Algoritmos Exactos de Pesquisa em Árvore.

Neste trabalho pretendeu-se estudar o desempenho de algoritmos paralelos para Problemas de Optimização Combinatória pertencentes à classe dos problemas NP-difíceis.

Escolheram-se a Programação Quadrática 0-1 sem restrições e o Problema de Optimização de Rotas de Veículos. Estes problemas são muito interessantes devido a serem aplicáveis a uma grande quantidade e variedade de situações reais. Este aspecto permitiu que se aplicassem os algoritmos desenvolvidos para o Problema de Optimização de Rotas de Veículos a um caso real. Desta forma, além de resultados computacionais com dados gerados aleatoriamente e conhecidos da literatura obtiveram-se soluções para o problema real que foram depois testadas no terreno.

Desenvolveram-se as versões sequenciais e paralelas de um algoritmo exacto para a Programação Quadrática 0-1 e de Algoritmos Genéticos para os dois problemas

A aplicação de operadores genéticos ao Problema de Optimização de Rotas de Veículos exige um tratamento adequado, diferente do habitual. Tal já não acontece com a Programação Quadrática 0-1 onde os Algoritmos Genéticos se aplicam naturalmente. Assim, a escolha destes dois problemas é adequada ao estudo do comportamento de Algoritmos Genéticos e da sua paralelização para problemas que, do ponto de vista da região admissível, se podem considerar em extremos opostos. Com efeito, enquanto que para a Programação Quadrática 0-1 qualquer solução booleana é admissível, para o Problema de Optimização de Rotas de Veículos há um grande número de restrições que fazem com que dificilmente uma solução produzida aleatoriamente seja admissível.

No capítulo 2 introduzem-se os conceitos básicos referentes ao paralelismo a nível de hardware, faz-se uma resenha sobre arquitecturas e topologias mais usuais e apresentam-se os aspectos mais importantes na avaliação do desempenho dos algoritmos paralelos. Por sua vez o capítulo 3 descreve as meta-heurísticas mais usuais e, mais detalhadamente, os fundamentos e conceitos subjacentes aos Algoritmos Genéticos e abordagens que têm vindo a ser utilizadas, principalmente, na sua aplicação a Problemas de Optimização Combinatória, bem como as várias formas de paralelismo inerentes a este tipo de algoritmos.

Nos capítulos 4 e 5 descrevem-se os problemas que foram objecto de estudo. O capítulo 4 é dedicado à Programação Quadrática 0–1 sem restrições. Começa-se por introduzir o problema fazendo seguidamente referência a algumas relações com outros tipos de problemas de optimização e aos métodos de resolução existentes. Entre esses métodos dá-se especial atenção aos de Pesquisa em Árvore por serem, na maior parte dos casos, as únicas alternativas existentes para a obtenção garantida do óptimo global. Sendo um dos objectivos desta Tese o estudo de algoritmos de optimização em máquinas paralelas, faz-se um breve apontamento sobre algoritmos paralelos de Pesquisa em Árvore. Descreve-se então, com maior pormenor, um algoritmo de Pesquisa em Árvore do qual se fez uma implementação sequencial e também paralela. Apresentam-se e analisam-se os resultados obtidos na experiência computacional efectuada que põem

em evidência a vantagem da utilização de máquinas paralelas para este tipo de algoritmos. Expõem-se também as abordagens utilizadas no desenvolvimento de um Algoritmo Genético e a sua implementação sequencial e paralela. Usando os mesmos dados da experiência computacional anterior e outros referenciados num trabalho publicado recentemente, testaram-se as várias versões desenvolvidas deste tipo de algoritmo obtendo-se resultados muito satisfatórios, tanto nas versões sequenciais como na paralela. Termina-se este capítulo com algumas conclusões e comentários.

No capítulo 5 apresenta-se o Problema de Optimização de Rotas de Veículos. Detalha-se a forma utilizada para adaptar os operadores genéticos a este problema e descreve-se a implementação sequencial e paralela do Algoritmo Genético assim desenvolvido. Apresentam-se e analisam-se os resultados obtidos na experiência computacional efectuada com um conjunto de dados conhecidos da literatura. No capítulo 6 descreve-se a aplicação das técnicas descritas no capítulo 5 ao problema concreto da definição das melhores rotas de recolha de resíduos sólidos na região rural de Faro. Este problema real é também aí descrito pormenorizadamente Apresentam-se e analisam-se os resultados obtidos. Esboçam-se algumas conclusões e comentários.

Finalmente, no capítulo 7, sintetizam-se todas as conclusões que se puderam extrair ao longo das inúmeras experiências realizadas (das quais só são relatadas as mais importantes) e da pesquisa bibliográfica efectuada. Enumeram-se, também, vários trabalhos que se gostaria de poder vir a realizar futuramente, decorrentes do trabalho apresentado nesta Tese, mas que, por diversos motivos, não puderam ser concretizados.

CAPÍTULO 2

COMPUTAÇÃO PARALELA

2.1 INTRODUÇÃO

Os enormes avanços tecnológicos ao nível dos computadores ocorridos nas últimas décadas, nomeadamente, o aparecimento de máquinas paralelas, têm vindo a modificar significativamente a computação científica. Passou a ser viável a resolução de muitos problemas que, devido aos seus custos computacionais e/ou a restrições nos tempos de execução, eram intratáveis.

O desenvolvimento dos meios computacionais, que são cada vez mais rápidos e com maiores capacidades, tem tido um forte impacto na área da Optimização estimulando a pesquisa e desenvolvimento de algoritmos e métodos de optimização mais eficientes.

De facto, aplicações reais de outras áreas técnicas/científicas (engenharia, economia, etc.) são formalizadas como problemas de Optimização. Em muitos casos são problemas de grandes dimensões, chegando a ter centenas de variáveis e de restrições, difíceis ou mesmo impossíveis de resolver em tempo razoável, até nos computadores sequenciais mais rápidos. Assim, o processamento paralelo surge como uma hipótese de viabilizar a resolução deste tipo de problemas. Alguns dos algoritmos sequenciais existentes têm sido paralelizados, e outros têm sido desenvolvidos especificamente para serem implementados em paralelo, visando uma determinada máquina paralela ou de uma forma genérica. Encontram-se, na literatura, vários trabalhos sobre algoritmos paralelos para problemas de Optimização [Lootsma e Ragsdell (1988), Bertsekas e Tsitsiklis (1989), Pardalos et al (1992), Damberg et al (1997), Holmqvist et al (1997), etc.].

Quando se fala em computação paralela, é necessário distinguir o que se passa ao nível do software e ao nível do hardware, ou, dito de outro modo, ao nível do algoritmo e ao nível da arquitectura. Pode-se utilizar a expressão tarefas múltiplas ("multitasking") para referir a estruturação de um programa (software) em duas ou mais partes (tarefas) que podem ser executadas ao mesmo tempo (paralelamente), e a expressão multiprocessador em relação a um sistema computacional (hardware) em que existem dois ou mais processadores. Resumindo, fala-se em computação paralela quando se tem um programa com múltiplas tarefas sendo executado numa máquina com múltiplos processadores, em que duas ou mais partes do programa correm simultaneamente em dois ou mais processadores da máquina.

Na secção 2 deste capítulo vão-se enunciar alguns termos e conceitos básicos referentes ao paralelismo a nível de hardware. Depois, na secção 3, apresenta-se uma classificação das arquitecturas paralelas. As topologias mais usuais para as redes de comunicação são descritas na secção 4. Na secção 5 resumem-se as principais características dos processadores utilizados, os transputers. Expõem-se, na secção 6, os aspectos mais importantes relativos ao desempenho dos algoritmos paralelos. Finaliza-se este capítulo, na secção 7, com algumas notas sobre a complexidade dos algoritmos paralelos.

2.2 CONCEITOS BÁSICOS

Controle - uma unidade global de controle pode ou não estar presente no sistema. Distinguem-se três modelos:

- *Controle de fluxo* - Cada processador do sistema executa instruções pela ordem determinada pela unidade de controle;
- *Fluxo de dados* - Os processadores actuam de acordo com a disponibilidade dos dados de entrada;
- *Fluxo de procura* - Os processadores executam as operações pela ordem requerida pelos dados.

Sincronismo - um relógio global para sincronizar as operações pode, ou não, estar presente. Distinguem-se dois tipos de sistemas:

- *Síncronos* - Se o sistema possui um único relógio;
- *Assíncronos* - Se o sistema possui vários relógios, geralmente um por cada processador.

Grão ("grain") - este parâmetro indica a quantidade de dados com que cada sistema pode trabalhar. Os sistemas podem ser do tipo:

- *Grão-fino* ("fine-grained") - Cada processador opera apenas com uma quantidade pequena de dados, ou seja, executa operações escalares ou operações com vectores de pequena dimensão;
- *Grão-grosseiro* ("coarse-grained") - grandes quantidades de dados podem ser tratadas simultaneamente pelos processadores.

Comunicação - Este parâmetro refere-se à forma como os processadores trocam informação entre si. Distinguem-se dois casos:

- *Memória partilhada* - Todos os processadores podem aceder a uma memória comum, para operações de escrita ou de leitura. Esta memória no caso dos sistemas fortemente acoplados ("tightly coupled") é comum a todos os processadores; nos sistemas fracamente acoplados ("loosely coupled") está dividida em várias áreas e há um mecanismo que permite a todos os processadores acederem a qualquer área;

- *Comunicação de mensagens* ("message-passing") - Os processadores têm a sua própria memória e trocam mensagens entre si, comunicando através de uma rede de interconecção.

Redimensionável ("scalabe")- Um sistema é redimensionável se quando aumenta o número de processadores o hardware continua a ser capaz de encaminhar os dados com rapidez [Sorevik (1997)]. Um sistema ser, ou não, redimensionável depende da rede de comunicações entre os processadores e da respectiva topologia.

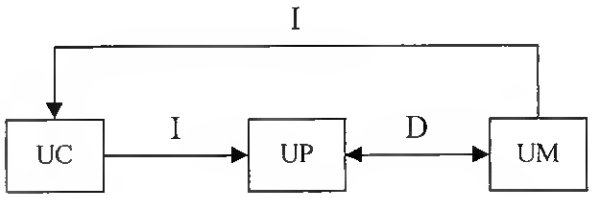
Número de processadores - Os sistemas paralelos podem ter um número de processadores na ordem das unidades ou chegarem a ter centenas ou milhares de processadores, neste último caso dizem-se massivamente paralelos.

2.3 CLASSIFICAÇÃO DE ARQUITECTURAS PARALELAS

Uma das primeiras classificações de arquitecturas paralelas e também das mais utilizadas deve-se a Flynn (1966). Baseia-se no hardware disponível para processar dados e instruções e agrupa os computadores em quatro classes diferentes, que são:

SISD ("Single Instruction stream Single Data stream") : Uma única sequência de instruções opera sobre uma única sequência de dados. Corresponde aos

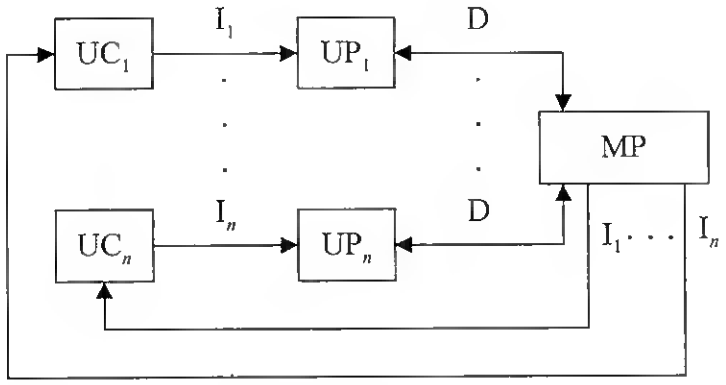
processadores sequenciais clássicos do modelo de Von Neumann. Representa-se esquematicamente esta arquitectura na figura 2.1.



UC - unidade de controlo UP - unidade de processamento UM - unidade de memória
D - sequência de dados I - sequência de instruções

Figura 2.1 – Arquitectura SISD

MISD ("Multiple Instruction stream Single Data stream") : Executam várias instruções simultaneamente sobre a mesma sequência de dados, armazenada em memória partilhada. Representa-se esquematicamente esta arquitectura na figura 2.2.,

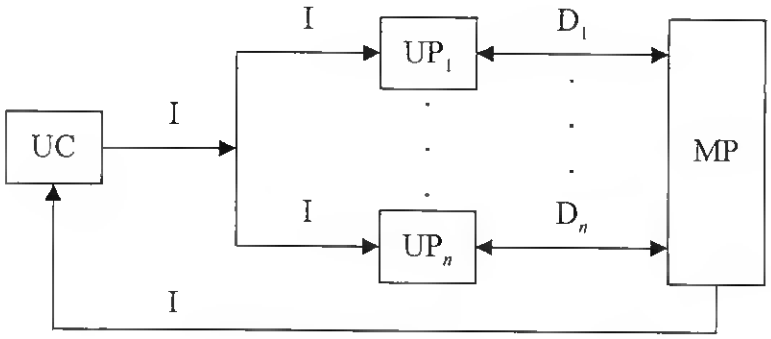


UC_i - unidade de controlo i UP_i - unidade de processamento i MP - memória partilhada
D - sequência de dados I_i - sequência de instruções i

Figura 2.2 – Arquitectura MISD

SIMD ("Single Instruction stream Multiple Data stream") : Executam uma só sequência de instruções sobre múltiplos dados, e têm memória partilhada.

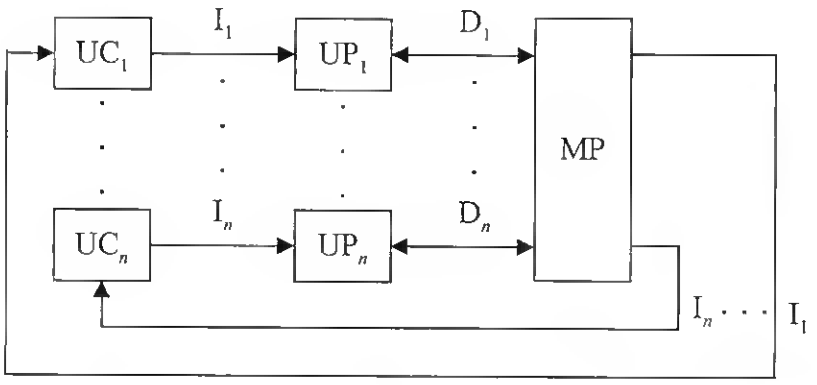
Incluem-se neste tipo, os computadores vectoriais e os computadores matriciais. Representa-se esquematicamente a arquitectura SIMD na figura 2.3.



UC - unidade de controlo UP_i - unidade de processamento i MP - memória partilhada
D_i - sequência de dados i I - sequência de instruções

Figura 2.3 – Arquitectura SIMD

MIMD ("Multiple Instruction stream Multiple Data stream") : Executam diversas instruções com diferentes dados. A maior parte das máquinas paralelas têm esta arquitectura, como por exemplo o hipercubo e as redes de transputers. Representa-se esquematicamente a arquitectura MIMD na figura 2.4.



UC_i - unidade de controlo i UP_i - unidade de processamento i MP - memória partilhada
D_i - sequência de dados i I_i - sequência de instruções i

Figura 2.4 – Arquitectura MIMD

Esta classificação é insuficiente para incluir claramente todas as arquitecturas paralelas que têm surgido desde 1966. Outras classificações foram propostas em Duncan (1990), Hockney e Jesshope (1990), Dowd (1993) e Kumar (1994).

Dowd (1993) estabelece uma classificação com base na utilização em computação científica, distinguindo os seguintes tipos de computadores:

Processadores Sequenciais - correspondem aos computadores de Von Neumann. As instruções executam-se sequencialmente sobre uma única sequência de dados. Este tipo de computadores tende a desaparecer, pois actualmente, mesmo os mais simples já executam diversas etapas de uma instrução simultaneamente.

Processadores Vectoriais - têm no seu código máquina instruções apropriadas para a manipulação de vectores. As operações realizam-se em unidades aritméticas segmentadas, capazes de dividir as instruções em várias etapas diferentes e executá-las simultaneamente. Este tipo de processadores é indicado para a execução de códigos sequenciais com um grande número de operações aritméticas sobre os dados. Como exemplos, destes computadores, conhecem-se os Cray Y-MP, Cray X-MP, Alliant FX/80, etc.

Processadores Superescalares - estão especialmente vocacionados para executar eficientemente qualquer tipo de código. Dispõem de várias unidades funcionais independentes capazes de realizar em paralelo diferentes operações de diversos tipos (aritméticas, lógicas, condicionais, de entrada e saída, etc.). Exemplos, destes computadores, são o RS/6000 da IBM, o DEC Alpha AXP, o MIPS R8000 da Silicon Graphics, etc.

Multiprocessadores e Multicomputadores - são compostos por um conjunto de processadores completos ("processing elements", PE) que colaboram na execução de uma determinada tarefa. Os PE podem ser de qualquer um dos tipos anteriores (sequenciais, vectoriais, superescalares).

No caso dos multiprocessadores (ver figura 2.5) a memória é partilhada por todos os PE do sistema e armazena tanto código como dados. O acesso à memória é feito através de uma rede de interconecção. Aqui reside o maior problema deste tipo de

computadores devido aos problemas de colisão e contenção. Para evitar, tanto quanto possível, estes problemas, a memória está estruturada em diferentes níveis e é necessária geri-la eficientemente de forma a manter a consistência dos dados e minimizar, ao mesmo tempo, o número de vezes em que eles são acedidos [Ortí (1996)]. Isto justifica que este tipo de máquinas contenham um número relativamente baixo de processadores, em geral têm de 4 a 32 PE, e que não sejam redimensionáveis. O Alliant FX/80 (com 8 PE vectoriais), o SGI Power Challenge (com 16 PE superescalares) são exemplos, entre outros, deste tipo de computadores.

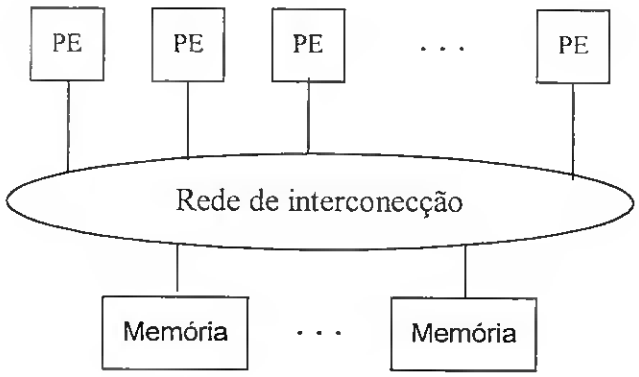


Figura 2.5 - Esquema de um multiprocessador

No caso dos multicomputadores, (ver figura 2.6) a memória é distribuída tendo portanto cada PE armazenado na sua memória o seu código e os seus dados. Os PE comunicam entre si através de uma rede de interconecções o que pode provocar "engarrafamentos" [Ortí (1996)]. Neste caso o acesso a um dado local é muito rápido, sendo o acesso a um dado remoto (armazenado na memória de outro processador) bastante mais lento. Ao contrário dos multiprocessadores, estas máquinas podem dispor de milhares de PE e são facilmente redimensionáveis. Como exemplos deste tipo de computadores têm-se o Cray T3D (com 256 PE), o IBM SP2 (com 128 PE), o CM-5 (com 1024 PE), etc.

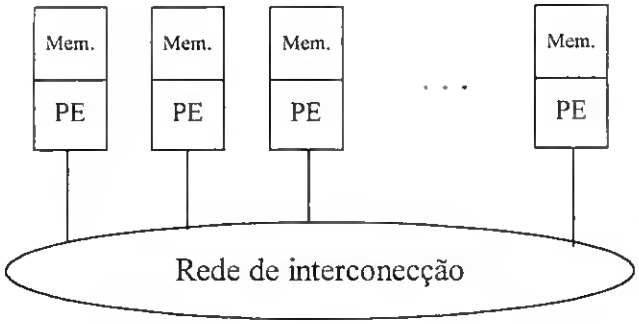


Figura 2.6 - Esquema de um multicomputador

2.4 COMUNICAÇÕES

Como se referiu anteriormente os multicomputadores fazem a transferência de informação entre PE, através de mensagens que passam na rede de interconecção. Assim, as redes devem satisfazer o melhor possível os seguintes objectivos:

- *Conectividade* - as ligações estabelecidas devem ter o menor número de pontos intermédios;
- *Simultaneidade* - as comunicações devem limitar minimamente os cálculos e a execução paralela;
- *Redimensionamento* ("scalability") - deve ser possível alterar o número de PE sem ter que redesenhar a rede de comunicações,

mas tendo em consideração os seguintes factores restritivos:

- número de fios e de pinos nos circuitos integrados de comunicação
- custo dos conectores
- custo dos PE
- custo das comunicações em função das distâncias entre PE.

As comunicações dependem do nível físico de conectividade da máquina. Para comparar os vários tipos de redes de interconecção [Fleming (1997)], utilizam-se os seguintes parâmetros:

L - número de ligações em cada PE

d - distância máxima entre dois PE

As topologias das redes mais utilizadas são:

• Barramento (Bus)

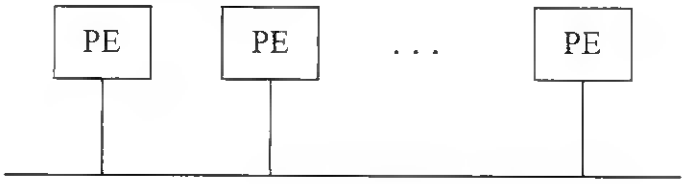


Figura 2.7 – Rede do tipo barramento

Em geral é utilizada em computadores sequenciais e só permite uma comunicação de cada vez. Neste caso tem-se $L = 1$ e $d =$ comprimento do barramento.

• Anel

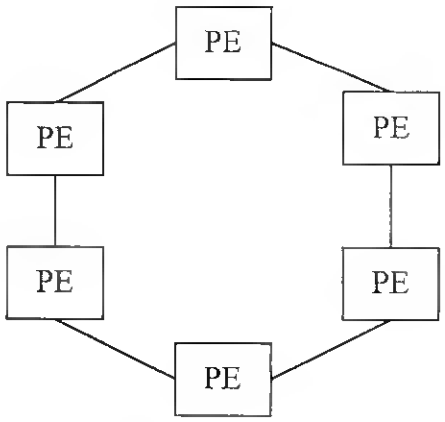


Figura 2.8 – Rede do tipo anel com 6 PE

Neste caso tem-se $L = 2$ e $d = \frac{n}{2}$.

• Grelha

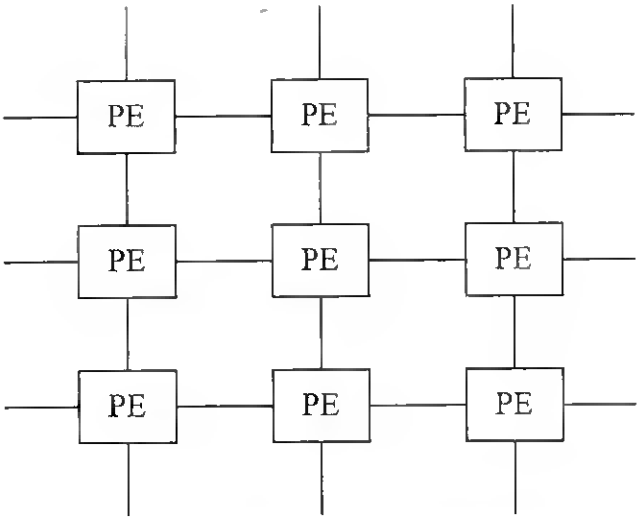


Figura 2.9 – Rede do tipo Grelha de 3x3 PE

Numa rede em grelha com n PE tem-se $L = 4$ e $d = 2\sqrt{n} - 2 \approx 2\sqrt{n}$

• Torus

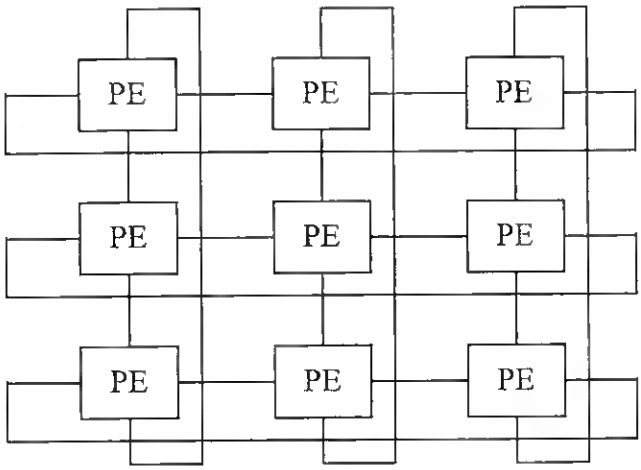


Figura 2.10 – Rede do tipo Torus de 3x3 PE

Neste caso, para um torus com n PE tem-se $L = 4$ e $d = \sqrt{n}$.

• Hipercubo

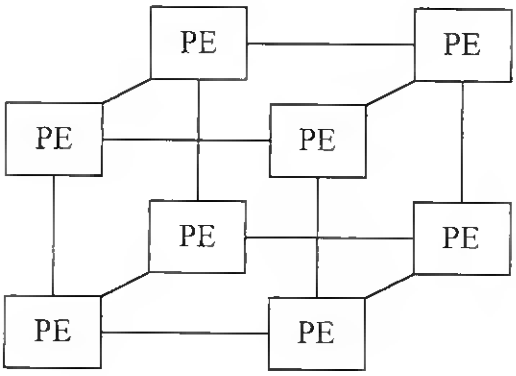


Figura 2.11 – Rede do tipo hipercubo de dimensão 3

No caso do hipercubo de dimensão m tem-se $n = 2^m$ PE, $L = log_2 n$ e $d = log_2 n$.

Dependendo da aplicação, algumas topologias são preferíveis a outras, mas nem sempre bons valores para L e para d levam a bons resultados em todos os tipos de aplicação. Por exemplo, uma rede em grelha é muito apropriada aos casos em que os PE só comunicam com os PE vizinhos. Mas quando é necessário que cada PE comunique os seus dados para todos os outros PE, este tipo de rede já não é adequado, sendo mesmo impraticável para sistemas com um grande número de PE.

Infelizmente, muitas vezes, só se consegue saber qual a topologia mais adequada a uma aplicação depois de experimentar algumas, não sendo possível, à priori, conhecer qual a que irá apresentar melhores resultados.

2.5 O INMOS TRANPUTER

Especificam-se alguns detalhes dos transputers visto ter sido o equipamento utilizado neste trabalho para testar os algoritmos paralelos desenvolvidos.

Concretamente, neste trabalho, utilizaram-se topologias com 1, 2, 4 e 8 transputers INMOS, a 25Mhz, numa plataforma TMB16, instalada num PC, onde a velocidade de comunicação entre transputers foi de 20 Mbits por segundo.

Um transputer é um circuito integrado composto essencialmente por um processador de 32 bits, uma unidade de vírgula flutuante de 64 bits, quatro canais de comunicação e memória, como representado esquematicamente na figura 2.12 [Bass (1996)].

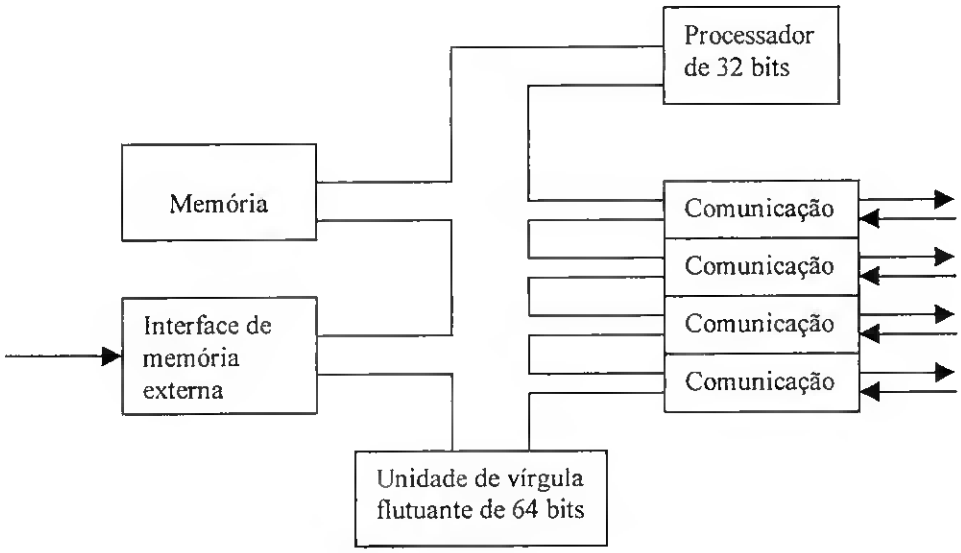


Figura 2.12 – Esquema de um transputer

Uma vez que os transputers permitem quatro ligações bidireccionais é evidente que uma toplogia adequada para um conjunto de transputers é a rede em grelha (figura 2.9). Uma característica importante destes processadores é a capacidade de processar e comunicar simultaneamente.

Em cada processador de cada transputer podem ser executados concorrentemente um ou mais processos. Para que a comutação de processos possa ser

feita rapidamente, o transputer dispõe de um "sheduler" em micro-código. Um processo pode encontrar-se num dos três estados seguintes [Bass (1996)]:

- *Pronto* ("ready") - esperando numa fila para aceder ao processador;
- *Em execução* ("running") - actualmente a executar;
- *Bloqueado* ("blocked") – à espera do fim de alguma actividade (comunicação, acesso a um recurso do sistema, etc.).

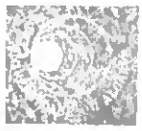
Dois níveis de prioridade são suportados por estes processadores.

Associada aos transputers, existe uma linguagem de programação especificamente desenhada para tirar partido da sua arquitectura, a linguagem Occam [Inmos,Ltd.(1989)]. Esta linguagem implementa a metodologia CSP ("Communicating Sequential Processes") [Hoare (1978)].

No caso de vários processos serem executados no mesmo transputer é necessário assegurar que o mesmo dado seja acedido em exclusão mútua pelos vários processos, de modo a impedir que uma variável tenha sido modificada por um processo e já não contenha o que era esperado ao ser acedida por outro processo. Este problema é resolvido, utilizando a metodologia CSP, não permitindo a utilização de dados partilhados (mesmo que estes residam no mesmo processador) mas implementando o acesso aos mesmos através de sistemas de mensagens. Deste modo o caso em que se tem uma rede de transputers e onde cada um executa um processo é, sob o ponto de vista de programação, análogo ao caso onde se têm vários processos a executar concorrentemente num único transputer. A troca de dados entre dois processos só é executada quando ambos os processos executam um "rendez-vous", isto é um dos processos pretende enviar um dado e o outro pretende recebê-lo.

Para que a implementação paralela seja eficiente é necessário, para além de se estabelecer uma distribuição equilibrada (*balanceamento*) da carga computacional pelos processadores, diminuir o mais possível os tempos de espera de cada processo. Estes tempos "mortos" ocorrem tanto nas comunicações estabelecidas durante a execução do processo, como na transmissão dos dados iniciais a todos os processos, como ainda na recolha dos resultados finais provenientes de cada processo.

Esta descrição mostra que os principais problemas que se levantam na implementação de um algoritmo paralelo em transputers são os mesmos que se levantariam se em vez de uma rede de transputers se estivesse perante uma rede de qualquer outro tipo de processadores com memória local (desde uma rede de PCs a uma placa com processadores de outro tipo). Os ficheiros de configuração da rede de processadores, as instruções específicas de comunicações (receber, enviar) e de atribuição de processos a processadores podem variar consoante o equipamento, mas a concepção geral de um algoritmo paralelo mantém-se. Isto levou a que, neste trabalho, se codificassem os programas paralelos em ANSI C em vez do OCCAM. Embora o OCCAM seja a linguagem desenvolvida para os transputers, consequentemente a mais apropriada, ao optar pelo ANSI C está-se a prever uma utilização, sem grandes problemas de adaptação, destes programas noutras máquinas paralelas com memória distribuída.



Nos testes computacionais realizados na fase inicial deste trabalho e comparativamente a um PC com um processador 486 a utilização de transputers para a generalidade dos algoritmos paralelos trazia melhoramentos significativos. Com o aparecimento dos processadores Pentium e a desistência, por parte das empresas fabricantes, de desenvolverem novas versões do transputer, como era esperado, este tipo de equipamento ficou superado. De facto, verificou-se, ao longo deste trabalho e da experiência computacional com os algoritmos desenvolvidos, que o tempo de execução (sequencial) num transputer é muito maior que num PC com um processador Pentium MMX a 200 Mhz.

Como não foi viável, durante o tempo de realização deste trabalho, a utilização de outro tipo de máquina paralela, continuou-se a utilizar os transputers mesmo sabendo que em termos absolutos dos tempos de execução eram ineficientes. Contudo, em termos relativos e comparando o desempenho de algoritmos paralelos em 1, 2, 4 ou 8 processadores, é possível tirar algumas conclusões. Já foi referido que o tipo de paralelização que se utilizou neste trabalho com os transputers é aplicável em quaisquer máquinas paralelas com memória distribuída. Desta forma as conclusões sobre o desempenho dos algoritmos continuam válidas se, em vez de transputers, for utilizado outro tipo de processadores mais eficientes, mas que quando ligados em rede tenham

uma relação potência de cálculo/velocidade de comunicação semelhante à da rede de transputers.

Resumindo, apesar desta limitação em termos de recursos computacionais, o trabalho realizado é válido e transponível para algumas máquinas paralelas com memória distribuída. É também de supor que, num futuro próximo, se tornem comuns os PC com placas contendo alguns processadores. Estes poderão executar, em termos absolutos, de uma forma mais eficiente, os algoritmos aqui desenvolvidos e estudados.

2.6 DESEMPENHO

Quando se desenvolve um algoritmo é necessário medir o seu desempenho de alguma forma que permita avaliá-lo e compará-lo com os outros. Para sistemas não paralelos, são normalmente utilizadas medidas de desempenho como o tempo de execução e a potência de cálculo do processador, normalmente medida em Megaflops (10^6 operações de vírgula flutuante por segundo).

Para sistemas paralelos, além destas medidas de desempenho, torna-se necessário introduzir medidas adicionais.

2.6.1 ACELERAÇÃO E EFICIÊNCIA

No que toca aos algoritmos paralelos tem havido muita controvérsia sobre qual o melhor critério a utilizar. No entanto, é geralmente aceite que o desempenho seja medido através da *aceleração* ("speedup") e da *eficiência*. São conhecidas três formas diferentes de calcular a aceleração.

A mais antiga é conhecida como a lei de Amdahl e define a aceleração [Amdahl (1967)] como o quociente entre o tempo necessário para resolver um problema

com um algoritmo paralelo num só processador e o tempo necessário para resolvê-lo com o mesmo algoritmo em N processadores do mesmo tipo.

Sejam:

N = número de processadores do sistema paralelo

T_1 = tempo de execução do programa num único processador

f = fracção do programa executada sequencialmente

p = fracção do programa executada paralelamente

com $f + p = 1$, então tem-se a lei de Amdahl:

$$\text{aceleração} = S_N = \frac{(f+p)T_1}{\left(f+\frac{p}{N}\right)T_1} = \frac{1}{f+\frac{1-f}{N}} \tag{2.1}$$

Esta relação diz que mesmo que a fracção sequencial f de computação seja pequena, a aceleração não pode exceder N, isto é, $1 \leq S_N \leq N$. Nesta afirmação está presuposto que a computação realizada pelo algoritmo num só processador é igual à realizada utilizando todos os processadores. Mas, em problemas de Optimização esta proposição nem sempre é verdadeira [Trienekens (1989), Pardalos et al. (1992)].

Sendo o grau de paralelismo dado pelo número de processadores que podem ser utilizados em paralelo em qualquer momento da execução de um programa, e considerando programas com grau de paralelização 1 ou N, pode-se equacionar a lei de Amdahl da seguinte forma [Fleming (1997)],

Sejam:

N = número de processadores do sistema paralelo

T_N = tempo de execução do programa em N processadores

f = fracção do programa executada sequencialmente

com

$$T_N = f\,T_1 + (1-f)\,\frac{T_1}{N} \tag{2.2}$$

então tem-se a aceleração dada por,

$$S_N = \frac{T_1}{T_N} = \frac{N}{1+f\,(N-1)} \tag{2.3}$$

No gráfico 2.1 pode-se ver, para esta lei, como a aceleração é influenciada pelos valores do número de processadores e da fracção sequencial do programa. Idealmente a parte sequencial do programa deveria ser zero, o que se traduziria numa aceleração linear.

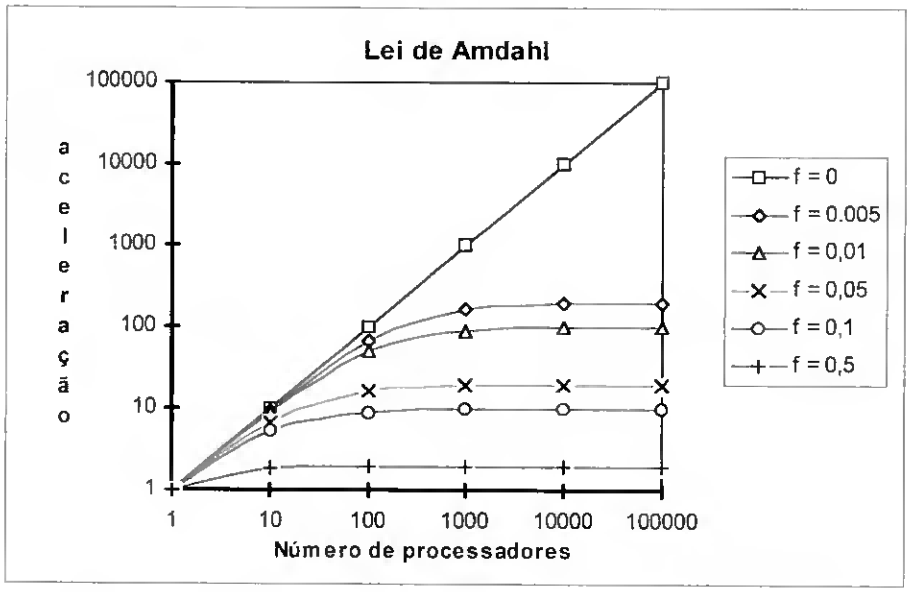


Gráfico 2.1 – Influência do nº de processadores e da fracção sequencial do programa na aceleração

A segunda forma de calcular a aceleração [Pardalos et al (1992)] é dada pelo quociente entre o tempo de execução do algoritmo sequencial mais rápido, executado num único processador, e o tempo de execução do algoritmo paralelo em estudo, para resolver o mesmo problema, usando N processadores. Neste caso os algoritmos sequencial e paralelo podem ser diferentes. Uma grande desvantagem deste método é o

facto de, na maioria dos problemas de Optimização, ser muito difícil estabelecer qual é o melhor algoritmo.

A terceira abordagem [Gustafson (1988)] considera que, num algoritmo, não é a dimensão do problema a ser resolvido que influencia o número de processadores a utilizar, mas fixado o número de processadores a utilizar, testa-se o algoritmo para problemas de diferentes dimensões. A esta forma chama-se *aceleração dimensionada* ("scaled speedup") e é dada por:

$$Sc = \frac{(f + pN)T}{(f + p)T} = N + (1 - N)f \tag{2.4}$$

onde:

N = número de processadores do sistema paralelo

T = tempo de execução do programa na máquina paralela

f = fracção do programa executada sequencialmente

p = fracção do programa executada paralelamente

com $f + p = 1$.

Quando se utiliza um algoritmo paralelo para resolver um problema e se obtém uma aceleração maior do que o número de processadores utilizados ($S_N > N$), chama-se-lhe *superlinear*.

Tem-se debatido muito a existência de aceleração superlinear [Parkinson (1986), Jansen (1987)]. A lei de Amdahl só é aplicável, quando se pressupõe que a quantidade de trabalho computacional efectuada pelo algoritmo é sempre a mesma quer se use 1 ou N processadores. Mas isto nem sempre acontece. É fácil de verificar [Pardalos et al (1992), Androulakis e Floudas (1998)] que, entre outros, algoritmos que recorrem a técnicas de pesquisa em árvore, executados sequencialmente, poderão ter comportamentos muito diferentes do que quando executados paralelamente.

Alguns trabalhos têm sido publicados sobre este assunto. Lai e Sahni (1984), Berner (1996) exploram aspectos teóricos que podem levar, ou não, à aceleração

superlinear. Li e Wah (1984), Trienekens (1989) apresentam condições suficientes que garantem não haver deterioração da aceleração quando se duplica o número de processadores, em algoritmos paralelos de pesquisa em árvore.

Outro parâmetro utilizado para medir o desempenho de um algoritmo paralelo é a eficiência, que é dada por

$$E_f = \frac{S_N}{N} \tag{2.5}$$

onde S_N é a aceleração obtida com N processadores, qualquer que seja a forma utilizada para o seu cálculo.

Se $1 \leq S_N \leq N$ então tem-se que $\frac{1}{N} \leq E_f \leq 1$.

2.6.2 OUTROS ASPECTOS IMPORTANTES

Um factor também importante para obter um algoritmo eficiente é a *granularidade* ("granularity") de um algoritmo, isto é, a relação entre o tempo de computação (R) e o tempo necessário para a comunicação (C) entre processadores. A granularidade mede-se [Fleming (1997)] através do quociente R/C.

Outro aspecto importante a atingir, especialmente nos sistemas MIMD, é o equilíbrio da carga computacional ("load balance") dos processadores.

É também necessário ter em conta que a maioria das linguagens de programação de alto nível disponíveis não possibilitam um aproveitamento eficiente de todas as características de um hardware específico [Fleming (1997)].

Verifica-se que a programação dos multiprocessadores (memória partilhada) não é muito diferente da programação sequencial. Já a programação dos multicomputadores (memória distribuída) exige o envio de mensagens entre processadores, o que, para

utilizar eficientemente os recursos da máquina, implica uma programação mais complexa [Ortí (1996)].

Uma outra dificuldade que surge quando se comparam diferentes tipos de máquinas paralelas é a inclusão dos compiladores das linguagens de programação e dos sistemas de entrada/saída nos testes do desempenho.

Quando se comparam arquitecturas e algoritmos deve-se ainda considerar que um bom algoritmo para uma arquitectura específica pode não ser apropriado para uma outra e muitas vezes ainda é menos adaptado a uma execução sequencial.

Para resumir, esboça-se na figura 2.13 um esquema representativo dos aspectos principais que diminuem a eficiência da paralelização de um problema.

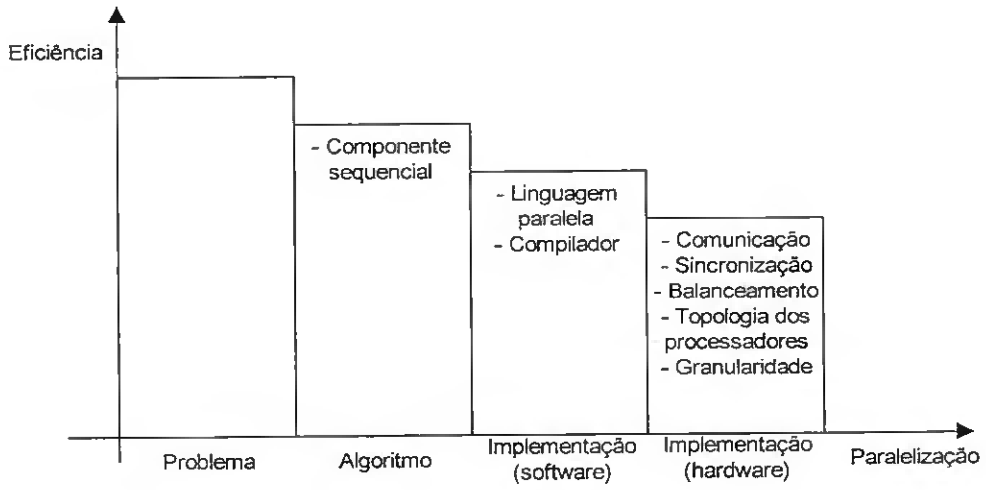


Figura 2.13 – Aspectos da paralelização de um problema e eficiência

Na passagem do problema ao algoritmo surgem partes que não se conseguem paralelizar, obtendo-se uma parte sequencial dentro do algoritmo paralelo. Ao passar à implementação, tanto a linguagem de programação com o compilador vão também reduzir a eficiência da paralelização, visto que não permitem utilizar em toda a sua potencialidade as máquinas paralelas. As comunicações entre processadores e a

respectiva sincronização, a divisão equilibrada do trabalho pelos processadores, a topologia, etc., limitam, ao nível do hardware, uma eficiente paralelização dos algoritmos. Estes aspectos impedem que se atinja o que seria ideal, isto é, que todas as partes do algoritmo fossem plenamente executadas em paralelo, que todos os processadores estivessem a trabalhar simultaneamente durante todo o tempo de execução do algoritmo, que as comunicações entre processadores não fizessem atrasar a execução e não consumissem tempo significativo, e que a fase de sincronização não implicasse em tempos de espera para os processadores.

2.7 COMPLEXIDADE DOS ALGORITMOS PARALELOS

Estudos teóricos sobre a complexidade dos algoritmos paralelos consideram duas classes, *NC* e *P* [Pardalos et al (1992)].

As definições exactas destas classes assentam no modelo PRAM ("Parallel Random Access Machine") de computação paralela [Fürer (1997)]. Neste modelo não há limites de espaço nem de tamanho do conteúdo das células de memória. Supõe-se ainda que a memória global é o único elemento requisitado por todos os processadores e que está sempre disponível. Não são também consideradas as comunicações e respectivos consumos de tempo. Estes últimos pressupostos estão longe de corresponder à realidade das máquinas paralelas existentes.

Por definição [Fürer (1997)] a classe *NC* é formada pelos problemas solúveis em tempo polilogaritmico numa PRAM, com um número polinomial de processadores. Isto significa que para alguma constante k e sendo n a dimensão do problema tem-se o tempo $T(n) = O\left((\log n)^k\right)$.

Os problemas pertencentes a *NC* têm um alto grau de paralelismo, isto é, podem ser resolvidos muito rapidamente (em tempo polilogaritmico) numa máquina paralela com memória suficiente [Fürer (1997)]. Os problemas que se supõe não terem um alto grau de paralelismo pertencem à classe *P*.

A classe P desempenha nos algoritmos paralelos um papel semelhante ao da classe NP nos algoritmos sequenciais. $NC \subseteq P$ mas não se sabe se $NC = P$ (embora não seja esperado que esta igualdade se verifique). A conjectura de que existe algum problema em P que não pertence a NC implica que todos os problemas *P-completos* não são NC.

CAPÍTULO 3

META-HEURÍSTICAS

3.1 INTRODUÇÃO

Uma das primeiras heurísticas de pesquisa local deve-se a Croes (1958) que a aplicou ao Problema do Caixeiro Viajante (PCV). Desde então, procedimentos de pesquisa local têm sido utilizados inúmeras vezes para obter boas soluções para Problemas de Optimização Combinatória (POCs).

Neste capítulo consideram-se os POCs na forma genérica:

$$\min_{x \in S} f(x)$$

onde S é um conjunto discreto de pontos admissíveis e $f : S \rightarrow \Re$ é a função objectivo (f.o.).

Inicialmente as heurísticas de pesquisa local (desenvolvidas principalmente nos anos 60 e 70) eram métodos descendentes que consistiam, basicamente, em:

(i) Obter uma solução admissível, $x \in S$, para o POC, com qualquer outro método. Considerar x a solução actual, isto é, $x_{act} = x$.

(ii) Modificar a solução actual.

O conjunto das soluções que se podem obter através de modificações da solução actual é, em geral, chamado vizinhança da solução actual. Em cada iteração, as soluções que estão na vizinhança da solução actual, $x_{act} \in S$, são geradas e calculadas uma a uma. Entre estas, uma, x_{prox}, será a candidata a ser a nova solução actual na próxima iteração.

(iii) Se $f(x_{prox}) < f(x_{act})$ então fazer $x_{act} = x_{prox}$ e voltar a (ii)

Caso contrário o algoritmo termina. (só são permitidas modificações descendentes).

O facto das heurísticas de pesquisa local descendente terminarem quando na vizinhança da solução actual não existe uma solução melhor levou a que, na maioria dos casos, estes métodos obtenham apenas um óptimo local.

Para tentar "escapar" dos óptimos locais e no intuito de atingir o óptimo global têm sido propostas, nas duas últimas décadas, algumas heurísticas genéricas de pesquisa local que, em certas condições, aceitam modificações da solução actual para uma solução pior. É usual chamar-lhes meta-heurísticas porque actuam principalmente na condução e orientação de um procedimento de pesquisa local.

As meta-heurísticas também partem de uma solução inicial e pesquisam a sua vizinhança. Mas enquanto os métodos de pesquisa local descendente exploram o espaço de soluções não permitindo modificações que piorem a solução actual (modificações ascendentes), as meta-heurísticas repetem e aleatorizam os procedimentos de pesquisa local e admitem passar, em certas condições, para uma solução pior (permitem modificações ascendentes). A deterioração da solução actual, numa iteração, pode permitir que o algoritmo não termine num óptimo local. Assim o passo (iii) da heurística de pesquisa local descendente é alterado, passando a permitir modificações ascendentes e de forma a que o método só termine quando um critério de paragem

(número de iterações ou outro) for atingindo. A solução obtida pela meta-heurística é a melhor solução entre todas as pesquisadas, podendo não ser a última solução actual.

Uma característica comum a algumas meta-heurísticas, que lhes permite por um lado executar modificações ascendentes e descendentes, e por outro não entrar em ciclo, é a memorização de soluções já examinadas ou de seus atributos. Estas informações armazenadas ajudam a conduzir a pesquisa de novas soluções.

O grande sucesso das meta-heurísticas não tem justificação a nível teórico. Os poucos resultados teóricos conhecidos são fracos e inúteis na prática [Mülhenbein (1997), Aarts et al (1997)]. Partindo de vários pressupostos, estabelecem que há uma probabilidade muito elevada de se encontrar a solução óptima, mas requisitando um tempo de execução enorme, superior ao de enumerar todas as soluções admissíveis [Taillard et al (1997)]. Ao contrário, a experiência prática demonstra que as meta-heurísticas, principalmente os métodos híbridos que combinam duas ou mais meta-heurísticas, são muito eficientes.

Neste capítulo faz-se, na secção 2, uma síntese sobre as meta-heurísticas que mais têm sido aplicadas a POCs, sendo os Algoritmos Genéticos descritos com algum detalhe, na secção 3. Na secção 4, resumem-se as formas usuais de paralelizar as meta-heurísticas detalhando mais o caso dos Algoritmos Genéticos. Na secção 5, tecem-se alguns comentários sobre os métodos descritos.

3.2 ALGUMAS META-HEURÍSTICAS

Entre as meta-heurísticas que têm vindo a ser utilizadas em POCs, descrevem-se resumidamente as que têm sido aplicadas para resolver um dos (ou os dois) POCs estudados neste trabalho.

3.2.1 ARREFECIMENTO SIMULADO (AS)

Este tipo de algoritmo baseia-se na analogia entre a pesquisa de uma boa solução aproximada, para um POC, e o comportamento de um metal quando a sua temperatura é

elevada a um nível muito alto (estado de fusão) e depois arrefecida até atingir a temperatura em que solidifica numa estrutura de energia mínima. Esta analogia e o primeiro algoritmo de AS para Optimização Combinatória foram propostos por Kirkpatrik et al (1983).

A analogia entre um sistema físico e um POC é feita com base nas equivalências da tabela 3.1.

ARREFECIMENTO SIMULADO	
SISTEMA FÍSICO	PROBLEMA DE OPTIMIZAÇÃO COMBINATÓRIA
Estados	Soluções
Energia de um estado	Valor da f.o. (f) para uma solução (x)
Temperatura (T)	Parâmetro (t)

Tabela 3.1 – Equivalências entre um Sistema físico e um Problema de Optimização Combinatória

Nos algoritmos de AS [Aarts et al (1997)], inicialmente o espaço das soluções é exaustivamente explorado, pois o algoritmo aceita a maior parte das vezes uma nova solução escolhida aleatoriamente na vizinhança da solução actual (valor grande para t que permite grandes deteriorações no valor da solução). Depois o algoritmo fica progressivamente mais selectivo na actualização da solução (a temperatura é gradualmente arrefecida). No fim (quando o valor de t tende para a temperatura mínima), quase não há modificações, só as soluções que melhoram a solução actual são aceites.

Como se referiu, na introdução, o AS difere dos métodos de pesquisa local descendente, pois aleatoriza a pesquisa e aceita, com alguma probabilidade, uma nova solução que provoque uma deterioração (mesmo que bastante grande) na solução actual.

Em Alves e Almeida (1992) é comparado o crescimento de um cristal e a pesquisa de uma solução óptima. Para efectuar o crescimento de um cristal a sua temperatura é elevada até atingir o ponto de fusão e posteriormente arrefecida. Se o arrefecimento for muito rápido a estrutura resultante é frágil com um nível de energia alto, ou seja, o sistema é levado para um estado (óptimo local) do qual não tem saída, mesmo estando muito longe de atingir o valor global de energia mínima (óptimo

global). Este processo de arrefecimento rápido é comparável às heurísticas de pesquisa local descendente que tendem a atingir óptimos locais. Por outro lado, se o arrefecimento for lento e gradual, diminuindo passo a passo e permitindo que em cada nível da temperatura seja atingido o equilíbrio térmico, obtém-se um cristal melhor estruturado e com um nível de energia mínimo ou muito baixo. Este processo cuidadoso de arrefecer o sistema compara-se aos procedimentos de pesquisa local, como os algoritmos de AS, que não actuam de uma forma ávida e aceitam soluções ascendentes com uma pequena probabilidade.

O algoritmo AS, pode ser resumido pelos Passos do algoritmo 3.1.

Passo 1:

Determinar uma solução admissível, $x_1 \in S$, e respectivo valor da f. o. $f(x_1)$

Passo 2:

Escolher: t_0 - a temperatura inicial

L - a duração do arrefecimento

$p(0)$ - a probabilidade inicial de aceitação de uma solução pior

Fazer $t = t_0 \wedge i = 1$

Definir a forma de actualizar a temperatura t e a probabilidade $p(i)$

Passo 3:

Escolher aleatoriamente uma solução x na vizinhança de x_i

Calcular $f(x)$

Se $f(x) - f(x_i) \leq 0$ fazer $x_{i+1} = x$

Caso contrário fazer $x_{i+1} = \begin{cases} x & \text{, com probabilidade } p(i) \\ x_i & \text{, com probabilidade } 1 - p(i) \end{cases}$

Passo 4:

Fazer $i = i + 1$

Actualizar t e $p(i)$ de acordo com o esquema de arrefecimento estabelecido.

Passo 5:

Se $i \leq L$ voltar ao Passo 3

Caso contrário terminar o algoritmo obtendo a solução igual a x_i e respectivo valor da f. o. ($f(x_i)$).

Algoritmo 3.1 – Algoritmo de Arrefecimento Simulado

A eficiência dos algoritmos de AS depende totalmente do esquema de arrefecimento definido. Com uma escolha apropriada dos vários parâmetros envolvidos e da forma de os actualizar o algoritmo pode chegar a uma solução óptima, ou muito próxima da óptima.

É principalmente na definição dos parâmetros que diferem os trabalhos desenvolvidos com algoritmos de AS. Uma grande parte do trabalho de implementação de um algoritmo de AS consiste na determinação dos melhores valores dos parâmetros para cada caso concreto No entanto, a maior parte deles refere obter uma boa qualidade das soluções mas em contrapartida os tempos de execução computacional são bastante elevados [Aarts et al (1997)].

3.2.2 PESQUISA TABU (PT)

O método PT foi inicialmente proposto por Glover (1986) e tem sido muito utilizado, geralmente com bons resultados, numa grande variedade de POCs [Glover (1996) e Glover e Laguna (1997)]. Uma exposição detalhada, desta meta-heurística, encontra-se em Glover (1989) e (1990) e Glover e Laguna (1997).

Tal como o AS, é um método iterativo que parte de uma solução inicial, obtida por algum outro método, e tenta "escapar" dos óptimos locais através da pesquisa da vizinhança da solução actual, permitindo também a sua deterioração em certas condições. O que fundamentalmente distingue estas duas meta-heurísticas é que, na pesquisa da vizinhança, a PT utiliza uma estratégia baseada no conceito de memória, em vez da aleatoriedade do AS, ou seja, na PT, as modificações executadas anteriormente influenciam as seguintes.

A PT começa com uma solução inicial $x_0 \in S$. Em cada iteração i uma nova solução x_{i+1} é atingida através de um *movimento* ("move") obtido a partir de uma modificação m_i da solução actual, x_i. A nova solução, x_{i+1}, é a que traz um maior melhoramento ao valor da f.o. f. Se nenhum movimento melhorar f então x_{i+1}

corresponde ao movimento que menos deteriorar o valor de $f(x_i)$, ou seja, a pesquisa não termina logo que é atingido um óptimo local, mas pode escapar dele executando o movimento que menos deteriora a função objectivo. Este procedimento poderia conduzir a uma pesquisa repetitiva (cíclica) do mesmo conjunto de soluções, mas a PT evita a execução de um movimento que retorne a soluções recentemente visitadas, proibindo os movimentos reversos dos últimos k movimentos efectuados. Estes movimentos são considerados tabu e fazem parte de uma lista tabu durante um certo número de iterações. A lista tabu é actualizada em cada iteração com a introdução de novos movimentos e a saída de outros.

Em alguns casos o estatuto tabu de um movimento poderia impedir a pesquisa de uma região promissora. A PT ultrapassa essa situação através do chamado *critério de aspiração* que permite, em certas condições, retirar o estatuto tabu a um movimento.

A PT termina quando um dado critério de paragem é atingido. Em geral, é utilizado um número máximo de iterações a executar.

Um algoritmo básico PT, pode ser resumido pelos Passos do algoritmo 3.2.

Passo 1:
Construir uma solução inicial $x_0 \in S$.
Fazer
$x^* = x_0$, onde x^* representa a melhor solução.
$T_k = \varnothing$, onde T_k representa a lista tabu (contém informação sobre as k últimas iterações).
$i = 0$.

Passo 2:
Escolher x_{i+1} entre a vizinhança de x_i, obtida de x_i por aplicação de movimentos $m_i \notin T_k$ ou que satisfaçam o critério de aspiração.
$i = i + 1$

Passo 3:
Se $f(x_i) < f(x^*)$ então $x^* = x_i$.
Actualizar T_k.

Passo 4:
Se não se verificar o critério de paragem voltar ao Passo2.
Caso contrário terminar obtendo a solução igual a x^* e respectivo valor da f.o. $f(x^*)$.

Algoritmo 3.2 – Algoritmo de Pesquisa Tabu

Em Glover (1995) são descritas várias formas de melhorar a eficiência do algoritmo PT descrito. Entre outras, são exploradas a utilização de funções de memória de curto e longo termo para intensificar e diversificar a pesquisa. A lista tabu representa uma função de memória de curto termo, uma vez que só são memorizados os movimentos mais recentes. A memória de longo termo pode, por exemplo, conter as melhores soluções obtidas ("elite solutions") e quando não existir um bom movimento na vizinhança da solução actual é possível passar para uma das soluções de elite, e pesquisar a sua vizinhança, intensificando assim a pesquisa em torno de uma boa solução. A memória de curto termo pode também ser usada para guardar atributos correspondentes a boas soluções permitindo que em cada iteração sejam primeiro examinados os elementos que têm esses atributos (outra forma de intensificar a pesquisa). Tanto a memória de curto termo como a de longo termo podem contribuir para a diversificação da pesquisa [Glover (1995), Glover e Laguna (1997)], evitando que algumas regiões do espaço das soluções não sejam minimamente exploradas. Uma outra forma de proceder à intensificação ou diversificação da pesquisa é introduzir penalidades na função objectivo, de forma a penalizar os movimentos para fora de uma região (intensificação), ou os movimentos para soluções muito próximas (diversificação).

Ainda em Glover (1995) é descrita a Pesquisa Tabu Probabilística, como uma extensão da PT, onde probabilidades substituem algumas funções de memória, na condução da pesquisa do espaço das soluções.

Outras técnicas têm sido incorporadas em algoritmos de PT para melhor intensificar e diversificar a pesquisa [Glover et al (1995), Glover e Laguna (1997)]. As mais comuns são a *Oscilação Estratégica* ("strategic oscilation") e o *Caminho de Ligação* ("path relinking"). A primeira é utilizada para mudar a direcção da pesquisa. Por exemplo, passando variáveis binárias de um valor para o outro. O Caminho de Ligação gera, a partir de duas soluções de elite, um caminho de soluções entre elas, escolhendo um movimento para passar de uma solução a outra.

Alguns dos algoritmos de PT permitem, embora de uma forma penalizada, que os movimentos conduzam à passagem para uma solução não admissível.

3.2.3 SISTEMA DE COLÓNIAS DE FORMIGAS (SCF)

A proposta inicial de algoritmos baseados no comportamento das formigas, para POCs, feita em 1991, deve-se a Colorni, Dorigo e Maniezzo [Taillard et al (1998)]. As formigas exploram a região ao redor do seu formigueiro, à procura de alimento, de uma forma aleatória. Quando encontram uma fonte de alimento, avaliam-na e transportam para o formigueiro algum alimento. Neste trajecto de regresso deixam, no chão, um rasto de feromona (substância odorosa que influencia o comportamento de outros animais da mesma espécie) na quantidade correspondente à qualidade da fonte de alimento. Após algum tempo, o caminho para uma boa fonte de alimento é indicado por um grande rasto de feromona (pois todas as formigas vão deixando o seu rasto). Entre os caminhos para boas fontes de alimento, as formigas visitam com maior frequência as mais próximas do formigueiro, para diminuírem o seu esforço, provocando um aumento maior e mais rápido destes caminhos. Ao contrário, os caminhos para fontes pouco importantes, ou para fontes mais distantes, desaparecem por evaporação da feromona, visto serem pouco visitados [Taillard et al (1997) e (1998), Taillard (1999)].

A analogia entre uma colónia de formigas e um problema de Optimização Combinatória é feita com base nas equivalências da tabela 3.2.

SISTEMA DE COLÓNIAS DE FORMIGAS	
COLÓNIA DE FORMIGAS	**PROBLEMA DE OPTIMIZAÇÃO COMBINATÓRIA**
Formiga	Solução
Formigueiro	Conjunto de soluções iniciais
Região ao redor do formigueiro	Espaço de soluções admissíveis
Quantidade de alimento de uma fonte	Função objectivo
Rasto de feromona	Memória adaptável

Tabela 3.2 – Equivalências entre uma Colónia de Formigas e um Problema de Optimização Combinatória

Neste método os rastos de feromona são utilizados, conjuntamente com a função objectivo, para conduzir a construção de novas soluções. São depois actualizados

provocando uma intensificação da pesquisa em torno das soluções mais promissoras (boas fontes). Esta actualização, que consiste em aumentar ou diminuir o rasto de acordo com a qualidade da solução obtida, é a parte crucial destes algoritmos. Nas abordagens iniciais todas as formigas, independentemente umas das outras, contribuíam para a actualização dos rastos, o que levou a que os primeiros algoritmos deste tipo não fossem muito eficientes. Nos trabalhos mais recentes, as actualizações são efectuadas sincronizando os resultados obtidos pelas formigas e intensificando a pesquisa em torno das melhores soluções. Em Taillard (1999) é descrita a evolução deste método bem como as abordagens, recentemente introduzidas, para o melhorar.

Em geral, o rasto de feromona, para cada variável, é calculado em função da sua contribuição para as soluções obtidas pelas *m* formigas e para a melhor solução. Pode, por exemplo, ser uma soma ponderada do número de vezes que essa variável entra, com um certo valor, nas soluções obtidas pelas formigas e na melhor solução. É obtido, geralmente, com uma expressão do seguinte tipo:

$$rasto(i) = \alpha\, rasto(i) + \beta\, contrsol(i) + \delta\, contrsolopt(i)$$

onde: α, β e δ são parâmetros a definir de acordo com o problema,

$1 - \alpha$ - é a percentagem de evaporação da feromona do rasto anterior

contrsol - contribuição da variável para a solução obtida

contrsolopt - contribuição da variável para a melhor solução já obtida.

Para um POC concreto torna-se necessário definir adequadamente o que é a contribuição de uma variável para uma solução e a melhor forma de utilizar a informação sobre as soluções já pesquisadas na construção de novas soluções (definição do rasto e da evaporação da feromona).

Genericamente um algoritmo baseado numa Colónia de Formigas pode ser resumido pelos Passos descritos no algoritmo 3.3.

Passo 1:

Definir o número m de formigas
Definir um critério de paragem.
Construir, para cada formiga i, uma solução x_i, $i = 1,...,m$
Inicializar o rasto de feromona a partir das soluções obtidas
Determinar x tal que $f(x) = \min_{i=1,...,m} f(x_i)$
Fazer $x^* = x$, onde x^* representa a melhor solução, e $f(x^*) = f(x)$

Passo2:

Construir, para cada formiga i, uma nova solução x_i, $i = 1,...,m$,
a partir do rasto de feromona.
Determinar x tal que $f(x) = \min_{i=1,...,m} f(x_i)$
Se $f(x) < f(x^*)$ então $x^* = x$.
Actualizar o rasto de feromona.

Passo 3:

Se não se verificar o critério de paragem voltar ao Passo 2.
Caso contrário terminar sendo a solução igual a x^* e respectivo valor da f.o. igual a $f(x^*)$.

Algoritmo 3.3 – Algoritmo de Sistemas de Colónias de Formigas

O algoritmo SCF difere principalmente das duas meta-heurísticas descritas por trabalhar simultaneamente com um conjunto de soluções, o que deve permitir uma pesquisa mais diversificada do espaço de soluções.

3.3 ALGORITMOS GENÉTICOS (AG)

Como neste trabalho se desenvolveram algoritmos baseados nos AGs descreve-se mais detalhadamente, nesta secção, este tipo de meta-heurística.

A designação Algoritmos Evolutivos (AEs) abrange um grande conjunto de métodos computacionais inspirados na evolução natural das espécies. Vários algoritmos deste tipo têm sido propostos, desde o final dos anos 60, para modelizar e resolver problemas em diversas áreas do conhecimento científico, nomeadamente na Inteligência Artificial e na Optimização Matemática.

Inicialmente os AEs, desenvolvidos por Fogel et al (1966) nos EUA e na Alemanha em 1973 por Rechenberg [Fonseca (1995)], consistiam em técnicas de pesquisa aleatória que tentavam simular a evolução natural das espécies assexuadas [Potvin (1996), Mühlenbein (1997)]. Nesse modelo novas soluções eram criados a partir de *mutações* aleatórias provocadas nas soluções já existentes.

Uma outra abordagem, os Algoritmos Genéticos (AGs), foi inicialmente desenvolvida por Holland (1975), com o objectivo de desenhar sistemas artificiais que imitassem os mecanismos mais importantes do processo de adaptação das espécies sexuadas, [Potvin (1996), Mühlenbein (1997)]. Assim, os AGs, além das mutações, utilizam *cruzamentos* entre duas soluções para gerar uma nova solução. Nalguns problemas a introdução dos cruzamentos mostrou-se fundamental para tornar mais eficiente esta técnica de pesquisa. Em Goldberg (1989) são tratados, aprofundadamente, os principais aspectos envolvidos nos AGs.

Vão-se descrever, com algum detalhe, os conceitos subjacentes aos AGs e formas de os abordar, tendo em vista a sua aplicação a POCs.

Os AGs utilizam um conjunto de soluções a partir das quais e por aplicação de operadores se vão obtendo novas e melhores soluções. Seguindo a analogia genética chama-se *população* a este conjunto de soluções, *indivíduo* a cada solução, *reprodução* ao processo de transformar a população actual numa nova população e *geração* a cada iteração.

Os operadores habitualmente utilizados na reprodução, têm como base a biologia genética e são: *selecção*, cruzamento, mutação e *avaliação* [Goldberg (1989)]. Em cada geração aplicam-se estes operadores à população com a intenção de que a nova população herde das anteriores as características que levaram a boas soluções, ficando melhor que as populações anteriores. Seleccionam-se alguns indivíduos, aos quais se chama *pais*, que se reproduzem através do seu cruzamento e mutação obtendo novos indivíduos, aos quais se chama *filhos*. Prosseguindo com a analogia genética, cada indivíduo é representado por um *cromossoma* e é avaliado através de um valor que lhe está associado e que representa a sua *aptidão* ("fitness"). A aptidão é, em geral, uma transformação (g) da função objectivo do problema (f).

No sentido de clarificar esta exposição e de mostrar como um AG é aplicado a um POC apresenta-se, na tabela 3.3, as correspondências entre os dois.

ALGORITMO GENÉTICO			
PROBLEMA DE OPTIMIZAÇÃO COMBINATÓRIA		GENÉTICA	
Designação	**Notação**	**Designação**	**Notação**
Função objectivo	$f(x)$	Função de aptidão	$A(x) = g(f(x))$
Conjunto de m soluções	$\{x^1, x^2, ..., x^m\}$	População	$\{x^1, x^2, ..., x^m\}$
Solução	x	Indivíduo	x
Vector solução	$(x_1, x_2, ..., x_n)$	Cromossoma	$[x_1\ x_2 ... x_n]$
Variável	x_i	Gene	x_i

Tabela 3.3 – Correspondências entre Genética e um Problema de Optimização Combinatória

Como já se referiu, os AGs operam numa população finita formada por um certo número (*Pop*) de indivíduos, cada um definido por um cromossoma. A carga genética do conjunto dos cromossomas é chamada *genotipo* o qual define um *fenotipo* (o indivíduo) com um certo valor de aptidão. As variáveis do problema são geralmente chamadas *genes*, os valores que podem assumir chamam-se *alelos* e à sua posição no cromossoma chama-se *locus* [Reeves (1996)].

O AG inicial, devido a Holland (1975), é caracterizado pela representação binária dos vectores solução, o que lhe permitiu enunciar o Teorema do esquema ("Schema Theorem"), considerado como sendo a fundamentação matemática dos AGs.

Um cromossoma é, então, uma sequência de 0s e 1s. Um *esquema* ("schema") é composto por 0s e 1s, como na representação binária, mas inclui também o símbolo * que significa 0 ou 1. Através deste símbolo os esquemas representam um subconjunto de cromossomas, por exemplo, o esquema 11** engloba os cromossomas [1100], [1101], [1111], [1110]. Assim, quanto maior for o número de símbolos * no esquema, maior é o subespaço de soluções por ele representado. As duas características fundamentais dos esquemas são a *ordem* e o *comprimento*. A ordem é o número de posições no cromossoma com valores fixos. O comprimento é a distância entre a

primeira e a última posição com valores fixos (no exemplo anterior a ordem é 2 e o comprimento é 1).

Um *bloco de construção* ("building block") é, por definição, um esquema de baixa ordem, pequeno comprimento e aptidão acima da média. O AG pesquisa o espaço de soluções combinando blocos de construção de dois pais. A ideia subjacente ao AG de Holland (1975) consiste na hipótese de que juntando os blocos de construção (as melhores características de dois bons cromossomas) se obtêm melhores cromossomas. Holland (1975) estimou que, em cada geração do AG, Pop^3 esquemas eram processados implicitamente, e chamou a esta propriedade paralelismo implícito, pois não exigia nenhum aumento de memória ou de processamento. No entanto, isto só é verdadeiro para alguns problemas particulares [Glover (1995), Michalewicz (1996)].

Esta teoria do esquema, algumas vezes criticada [Mühlenbein (1997)], tem sido utilizada para explicar como os AGs funcionam [Poli e Langdon (1998)]. Mas, de facto, o código binário que está fortemente ligado a esta teoria é pouco apropriado para vários POCs, sendo o PCV o exemplo mais óbvio. Assim, têm surgido AGs em que é utilizada uma representação das soluções mais adaptada ao problema [Potvin (1996), Reeves (1996), etc.].

Seguidamente, descrevem-se as abordagens que têm sido utilizadas em relação aos conceitos (representação, geração da população inicial, avaliação da aptidão, selecção, cruzamento e mutação) envolvidos num AG e ilustrados na figura 1.

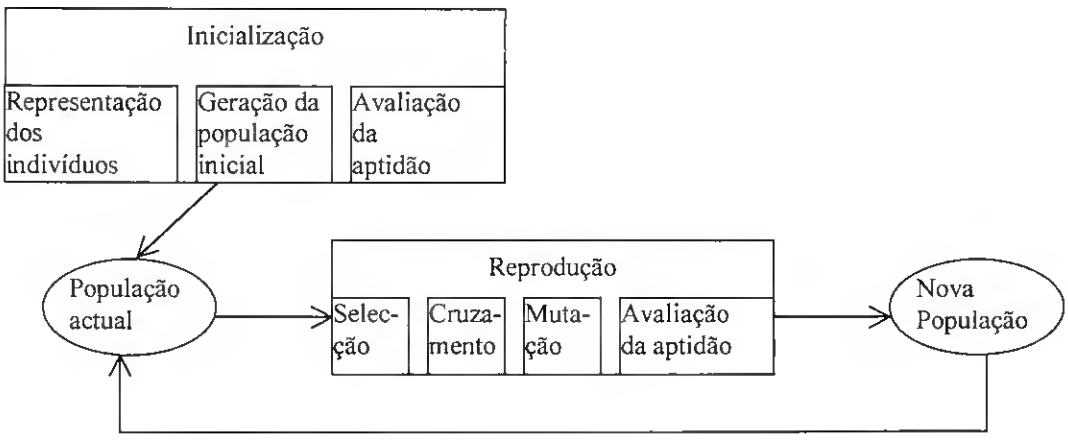


Figura 3.1 – Esquema de um Algoritmo Genético

3.3.1 REPRESENTAÇÃO E GERAÇÃO DA POPULAÇÃO INICIAL

Além da representação binária dos fenótipos, típica dos AGs [Holland (1975), Goldberg (1989)], têm surgido outras representações para Problemas de Optimização Combinatória. A mais comum, aplicável a diferentes problemas, é a utilização de inteiros [Huntley e Brown (1996), Potvin (1996), Vaz Pato e Lourenço (1997)]. Por exemplo, cada solução de um PCV, com n cidades, pode ser representada de uma forma única por uma permutação dos elementos de $\{1,2,...,n\}$; se $n = 6$ o cromossoma [1 2 5 6 3 4] pode representar a ordem (o caminho) pela qual as 6 cidades vão ser visitadas. Para o PCV, é também apropriada, por facilitar a manipulação de arestas, a representação pelas adjacências [Potvin (1996)], isto é, a cidade j ocupa a posição i do cromossoma se existir uma aresta entre i e j na solução. Para o exemplo anterior a representação seria [2 5 4 1 6 3].

A representação de um cromossoma através de um vector é a mais simples e directa, no entanto, outras estruturas como matrizes, árvores e listas são também utilizadas [Fonseca (1995)].

A inicialização da população consiste em gerar os indivíduos que a vão formar. Geralmente geram-se os indivíduos aleatoriamente, [Huntley e Brown (1996), Vaz Pato e Lourenço (1997)]. Se, para problemas sem restrições a geração aleatória não pôe qualquer obstáculo, já para problemas com restrições pode levar a soluções não admissíveis, recorrendo-se, então, a penalizações atribuídas de acordo com a gravidade da violação da admissibilidade. Alternativamente, podem-se utilizar métodos heurísticos para gerar a população inicial.

Outro aspecto importante é a definição da dimensão da população [Michalewicz (1996)]. A dificuldade reside em estabelecer qual a dimensão apropriada para cada tipo de problema, pois se a população for demasiado pequena tem menor diversidade genética, e consequentemente pode convergir muito rapidamente para uma solução de fraca qualidade. Se for demasiado grande, a diversidade genética estará assegurada, mas pode perder-se muito tempo no seu tratamento e demorar muito até obter melhoramentos. Assim, é usual estabelecer a dimensão da população num valor próximo do número de variáveis do problema [Fonseca (1998)]. No entanto, para

problemas de grande dimensão pode não ser viável, por limitações de tempo de execução e/ou de memória, trabalhar com uma população muito grande.

Michalewicz (1996) introduz, através do conceito de *idade* de um indivíduo, uma população com tamanho variável. Cada indivíduo tem definido à partida um certo limite de duração (número de gerações em que permanece na população) e quando esse valor é atingido "morre". Em cada geração são criados novos indivíduos, numa proporção λ da dimensão da população anterior, que se juntam à população (não substituem os pais). Assim, na geração t+1 a dimensão da população, $dim(t+1)$, é dada por:

$$dim(t+1) = dim(t) + \lambda\, dim(t) - D(t)$$

onde $D(t)$ é o número de indivíduos que "morreu" na geração t. Os parâmetros envolvidos necessitam de uma definição cuidadosa para evitar o crescimento exponencial da população e a "morte" prematura de indivíduos com características importantes para o melhoramento da população.

3.3.2 AVALIAÇÃO DA APTIDÃO

A função objectivo, f, de um problema fornece uma medida para avaliar a qualidade de cada indivíduo. No entanto, nos AGs o valor da função objectivo não costuma ser directamente utilizado para calcular a aptidão relativa de cada indivíduo. Em geral, recorre-se a outra função, a que se chama função de aptidão, *Apt*, que transforma o valor da função objectivo numa medida de aptidão, com valores num subconjunto dos reais não negativos [Chipperfield e Fleming (1996)]. De facto, a função de aptidão pode ser definida tendo em conta as abordagens de optimização que se vão utilizar, ao contrário da função objectivo que é parte integrante do problema [Fonseca (1995)]. Esta transformação da função objectivo é sempre necessária quando o objectivo é minimizar, para que os indivíduos com menor valor da função objectivo sejam os mais aptos.

Uma transformação usual [Chipperfield e Fleming (1996), Michalewicz (1996)] é a dada pela aptidão proporcional de cada indivíduo na população, isto é,

$$Apt\left(x^i\right) = \frac{f\left(x^i\right)}{\sum_{i=1}^{Pop} f\left(x^i\right)}, \quad i = 1,...,Pop \tag{3.1}$$

Esta forma de determinar a aptidão, garante que cada indivíduo tem uma probabilidade de se reproduzir de acordo com a sua aptidão relativa, mas não é apropriada quando a função objectivo tanto pode tomar valores negativos como positivos.

A função de aptidão deve respeitar a ordenação dos indivíduos dada pela função objectivo. Quanto à manutenção da escala dos valores relativos dos indivíduos há dois tipos de abordagem [Fonseca (1995)]:

Graduação ("scaling") – A função de aptidão respeita os valores relativos dos indivíduos atribuindo aos melhores indivíduos uma superioridade em relação aos outros.

Classificação ("ranking") – Ordena os indivíduos pelos seus valores da função objectivo e afecta-os a um conjunto de valores de aptidão de acordo com o seu lugar na ordenação.

A maior desvantagem da primeira abordagem é que a probabilidade de um indivíduo se reproduzir é proporcional ao seu desempenho, os melhores indivíduos podem tomar valores elevados em relação aos outros, levando a que esses indivíduos, mais aptos, nas primeiras gerações, dominem a reprodução e conduzam o AG a uma convergência prematura para uma solução suboptima [Chipperfield e Fleming (1996), Fonseca (1995)].

A segunda abordagem elimina esta dificuldade, visto que o melhor indivíduo tem sempre a mesma aptidão relativa, não se reproduzindo, por isso, excessivamente.

Para a classificação a função de aptidão costuma ser linear ou exponencial [Fonseca (1995)].

Sejam:

Pop = dimensão da população;

x^r = indivíduo que ocupa o lugar r na população ordenada $x^0,\ldots,x^{Pop-1}$

s = valor da aptidão a atribuir ao melhor indivíduo .

Então tem-se:

- Função de aptidão linear

$$Apt\left(x^r\right) = s - (s-1)\cdot\frac{2r}{Pop-1} \quad , \quad 1 < s \leq 2 \tag{3.2}$$

O limite superior de s garante que, para qualquer indivíduo, a aptidão é não negativa e que

$$\sum_{i=0}^{Pop-1} Apt\left(x^i\right) = Pop\,.$$

- Função de aptidão exponencial

$$Apt\left(x^r\right) = \rho^r \cdot s \quad , \quad s > 1\,;\, \rho : \sum_{i=0}^{Pop-1} \rho^i = \frac{Pop}{s} \tag{3.3}$$

Para $1 < s \leq 2$ a aptidão exponencial penaliza menos o pior indivíduo do que a linear, em contrapartida, na aptidão exponencial os indivíduos médios têm uma aptidão ligeiramente inferior à média.

Na graduação várias funções de aptidão têm sido utilizadas [Huntley e Brown (1996), Michalewicz (1996), Vaz Pato e Lourenço (1997)] de acordo com o problema e com o objectivo (minimizar ou maximizar). Quando se pretende minimizar, pode-se simplesmente passar a um problema de maximização, mudando o sinal da função objectivo [Vaz Pato e Lourenço (1997)], ou se a função objectivo não tomar valores negativos, é comum [Potvin (1996), Huntley e Brown (1996)] utilizar,

$$Apt\left(x^i\right) = \max_{j=1,\ldots,pop}\left\{f\left(x^j\right)\right\} - f\left(x^i\right) \quad , \quad i = 1,\ldots,Pop \quad .$$

3.3.3 SELECÇÃO

O operador selecção escolhe os indivíduos da população actual, com base na aptidão de cada um, para serem combinados de modo a darem origem a uma nova população. Consiste em determinar o número de vezes que um determinado indivíduo é escolhido para se reproduzir, ou seja, o número de filhos que ele vai gerar.

Para avaliar o desempenho dos algoritmos de selecção Baker (1987) introduziu três medidas: *enviesamento* ("bias"), *amplitude* ("spread") e *eficiência*. O enviesamento mede a precisão, sendo definido como a diferença, em valor absoluto, entre a probabilidade de selecção esperada e a probabilidade de selecção efectiva de um indivíduo. A amplitude mede a consistência, sendo dada pelo intervalo do número esperado de vezes que um indivíduo pode ser escolhido. A selecção é eficiente se não aumentar a complexidade do AG [Chipperfield e Fleming (1996)].

Os algoritmos de selecção mais utilizados baseiam-se na selecção aleatória de uma amostragem com reposição (SAR) ou sem reposição (SUS "Stochastic Universal Sampling"). Um dos algoritmos SAR mais conhecidos é o "método da roleta" [Goldberg (1989)] e consiste em:

(i) Calcular a soma das aptidões de todos os indivíduos da população,

$$soma = \sum_{i=1}^{Pop} Apt\left(x^i\right).$$

(ii) Gerar um número aleatório (*nal*) uniformemente distribuído no intervalo $[0, soma]$.

(iii) Seleccionar o primeiro indivíduo, x^j, tal que,

$$nal < \sum_{i=1}^{j} Apt\left(x^i\right)$$

Em (i) a *soma* também pode ser igual à adição dos valores da aptidão normalizados, ou das probabilidades esperadas da selecção dos indivíduos.

Desta forma, um indivíduo com maior valor de aptidão tem maior probabilidade de ser escolhido, visto que ocupa uma fatia maior da roleta. Este método está de acordo com os princípios defendidos por Holland (1975) pois, segundo ele, os indivíduos com maior aptidão contêm mais blocos de construção e ao serem favorecidos na selecção garante-se que as melhores soluções são transmitidas às próximas gerações. Mas, tem sido apontado [Potvin (1996)] que este método pode ter a desvantagem de conduzir rapidamente a uma solução de pouca qualidade, já que um indivíduo com um grande valor de aptidão, através deste tipo de selecção, pode predominar na população em poucas gerações. De facto, este método tem enviesamento igual a zero (como é desejável) mas a amplitude é potencialmente ilimitada o que permite a geração de uma nova população a partir de um só indivíduo.

Uma forma de minorar este problema é recorrer à amostragem com reposição parcial, isto é, cada vez que um indivíduo é escolhido a sua fatia da roleta diminui de uma unidade até um tamanho mínimo de zero [Chipperfield e Fleming (1996)]. Desta forma a amplitude é limitada superiormente, mas o limite inferior continua a ser zero e o enviesamento é superior ao anterior.

A selecção SUS pode também ser vista como uma roleta com divisões proporcionais à aptidão dos indivíduos, mas em vez de um único ponteiro utiliza k ponteiros igualmente espaçados (com k o número de elementos a seleccionar). A população é distribuída aleatoriamente (isto é, os indivíduos não estão ordenados), é calculada a *soma* como na selecção SAR, gerando-se um único número aleatório (*nal*) no intervalo $\left[0, \frac{soma}{k}\right]$. A partir dos ponteiros [*nal*, *nal*+1, ... , *nal*+*k*-1] escolhem-se os indivíduos cujo intervalo da roleta abrange a posição dos ponteiros (como em (iii)). Este método tem enviesamento igual a zero e amplitude mínima [Chipperfield e Fleming (1996)] sendo, por isso, uma boa alternativa ao SAR.

Reeves (1996) sugere que em muitas implementações é melhor utilizar uma função de aptidão baseada na classificação dos indivíduos de forma a que a selecção forneça ao melhor indivíduo uma probabilidade de ser escolhido cerca de duas vezes maior do que ao indivíduo médio.

Em [Michalewicz (1996), Mühlenbein (1997)] são descritas outras formas de selecção e referidos trabalhos com estudos sobre o desempenho de alguns algoritmos de selecção.

3.3.4 CRUZAMENTO (OU RECOMBINAÇÃO)

O cruzamento é aplicado a dois indivíduos, os pais, que produzem dois novos indivíduos, os filhos. Tal como na biologia, os filhos herdam de ambos os pais parte dos seus genes.

Resumem-se, seguidamente, as formas mais frequentes de proceder ao cruzamento de dois indivíduos.

CRUZAMENTO 1-PONTO

Dados dois cromossomas, escolhe-se aleatoriamente um ponto de cruzamento e obtêm-se dois novos cromossomas por junção de uma parte de um pai com a outra parte do outro pai. Por exemplo, se esse ponto de cruzamento for k vem:

Pais $[x_1\, x_2 \ldots x_k \ldots x_n]$ e $[y_1\, y_2 \ldots y_k \ldots y_n]$

Filhos $[x_1\, x_2 \ldots x_k\, y_{k+1}\, y_{k+2} \ldots y_n]$ e $[y_1\, y_2 \ldots y_k\, x_{k+1}\, x_{k+2} \ldots x_n]$.

CRUZAMENTO MULTI-PONTOS

Neste caso, em vez de um ponto escolhem-se aleatoriamente alguns pontos de cruzamento. Os alelos, dos dois pais, entre dois pontos sucessivos, são trocados. Os alelos entre o primeiro locus e o do primeiro ponto de cruzamento permanecem como nos pais. Por exemplo, considerando 3 pontos de cruzamento e sendo j, k, l os seus locus, então

Pais $[x_1 \ldots x_j \ldots x_k \ldots x_l \ldots x_n]$ e $[y_1\, y_2 \ldots y_j \ldots y_k \ldots y_l \ldots y_n]$

Filhos $[x_1 \ldots x_j\, y_{j+1} \ldots y_k\, x_{k+1} \ldots x_l y_{l+1} \ldots y_n]$ e

$[y_1 \ldots y_j\, x_{j+1} \ldots x_k\, y_{k+1} \ldots y_l\, x_{l+1} \ldots x_n]$.

CRUZAMENTO UNIFORME

O cruzamento uniforme generaliza o multi-ponto, visto que, neste caso, cada locus é um potencial ponto de cruzamento. Para cada gene existe uma probabilidade *p* de que o seu alelo seja herdado do *Pai* 1 e uma probabilidade 1-*p* de vir do *Pai* 2

A ideia subjacente à utilização de múltiplos pontos de cruzamento vem de se supor que as partes do cromossoma com maior contribuição para o desempenho de um indivíduo podem não estar em posições adjacentes. Michalewicz (1996) refere alguns estudos feitos sobre estes três tipos de cruzamentos concluindo que, embora não exista nenhum sistematicamente melhor, em alguns casos específicos o cruzamento 1-ponto revelou-se menos eficiente.

Para a representação dos indivíduos através de inteiros, outros operadores foram desenvolvidos no intuito de preservar a posição absoluta ou a ordem relativa dos alelos. Os mais utilizados são:

PMX ("PARTIALLY-MAPPED CROSSOVER")

Escolhe aleatoriamente dois pontos de cruzamento. Troca os alelos dos genes entre esses pontos. Define uma correspondência entre esses alelos dos pais, que é aplicada aos outros genes, para eliminar as duplicações [Potvin (1996), Michalewicz (1996)]. Por exemplo, se os pontos de cruzamento forem 3 e 5, os alelos que estão nos locus 3, 4 e 5 são trocados e definem uma correspondência, sejam os

Pais [1 2 5 6 4 3 7] e [4 3 1 2 5 7 6]

tem-se as correspondências, $5 \leftrightarrow 1$, $6 \leftrightarrow 2$, $4 \leftrightarrow 5$, então obtêm-se

Filhos [4 6 1 2 5 3 7] e [1 3 5 6 4 7 2]

Este operador tenta manter as posições absolutas dos alelos quando são copiadas para dar origem aos filhos. De facto, o número de alelos que não herdaram as suas posições de um dos pais é, no máximo, igual ao comprimento da sequência entre os dois pontos de cruzamento [Potvin (1996)]. Neste exemplo, em cada um dos filhos só dois alelos é que não têm a sua posição absoluta igual a um dos pais.

CX ("CYCLE CROSSOVER")

Este cruzamento determina um conjunto de genes que ocupa o mesmo subconjunto de locus nos dois pais e copia os seus alelos para os filhos; os outros locus são preenchidos com os alelos do outro pai [Potvin (1996), Michalewicz (1996)]. No exemplo anterior,

Pais [1 2 5 6 4 3 7] e [4 3 1 2 5 7 6]

os alelos {1, 4, 5} ocupam o subconjunto dos locus {1, 3, 5}, então tem-se,

Filhos [1 3 5 2 4 7 6] e [4 2 1 6 5 3 7]

Neste caso, os filhos herdam a posição absoluta de todos os seus genes de um dos pais, no entanto, como o subconjunto dos locus comuns pode não ser de locus adjacentes, este cruzamento pode provocar, no PCV, a quebra de arestas [Potvin 1996)].

CRUZAMENTO MODIFICADO

Esta é uma extensão do cruzamento 1-ponto para este tipo de representação (permutações de n) que tenta manter a ordem.

É escolhido aleatoriamente um ponto de cruzamento. Os alelos, dos pais, entre o início e esse ponto são copiados para os filhos. O restante é preenchido com os genes do outro pai eliminando as duplicações [Reeves (1996)]. No exemplo anterior,

Pais [1 2 5 6 4 3 7] e [4 3 1 2 5 7 6]

Seja 2 o ponto de cruzamento, então vem:

Filhos [1 2 4 3 5 7 6] e [4 3 1 2 5 6 7]

OX ("ORDERED CROSSOVER")

Este operador [Goldberg (1989)] é uma adaptação do cruzamento 2-pontos à representação por permutações. São escolhidos aleatoriamente dois pontos de cruzamento e os alelos dos genes dos pais, entre esses pontos, são copiados para os filhos. Os restantes genes são obtidos do outro pai mantendo a ordem e eliminando as duplicações a partir do segundo ponto de cruzamento. No exemplo anterior,

Pais [1 2 5 6 4 3 7] e [4 3 1 2 5 7 6]

Sejam os pontos de cruzamento 3 e 5; então a ordem dos outros genes é, respectivamente, 3 7 1 2 5 6 4 e 7 6 4 3 1 2 5, e tem-se:

Filhos [1 2 5 6 4 7 3] e [6 4 1 2 5 3 7]

Alguns outros operadores deste tipo, desenhados para manter a ordem são descritos em [Potvin (1996), Michalewicz (1996)]. No caso da representação por adjacências, os operadores tentam que os filhos herdem a maior parte das arestas dos pais. Em [Potvin (1996), Michalewicz (1996)] são descritos três operadores deste tipo e são referidos estudos para o PCV (com 30 cidades) que indicam que os operadores que preservam as arestas obtiveram os melhores resultados e os que preservam a ordem são melhores do que os que mantêm a posição. No entanto [Potvin (1996)] nenhum deles consegue, por si só, bons resultados em PCVs de maiores dimensões (50 ou mais cidades).

3.3.5 MUTAÇÃO

Na evolução natural, a mutação é um processo aleatório onde o alelo de um gene é modificado produzindo uma nova estrutura genética. Nos AGs [Chipperfield e Fleming (1996)], a mutação é aplicada aleatoriamente alterando cada alelo, independentemente dos outros, com uma probabilidade muito pequena, levando a que os filhos herdem a quase totalidade das características genéticas dos pais. O objectivo deste operador é introduzir perturbações aleatórias no processo de pesquisa, trazendo

essencialmente uma certa diversificação a uma população homogénea. É a garantia de que todos os cromossomas tem uma probabilidade maior que zero de serem pesquisados. [Chipperfield e Fleming (1996)]. Actua, também, como restaurador de material genético que se pode ter perdido com a aplicação dos operadores selecção e cruzamento [Goldberg (1989)].

Uma questão importante é a de saber qual o melhor valor da probabilidade de mutação (p_m) de um gene, isto é, a probabilidade de cada gene de um cromossoma ser alterado independentemente dos outros. Quanto menor é a aptidão de um indivíduo maior deveria ser o número dos seus genes alterados, para o aproximar dos outros; então a probabilidade de mutação dos indivíduos piores deveria ser maior do que a dos melhores. Isto levou a considerar interessante não manter constante a probabilidades de mutação ao longo do AG [Fonseca (1995)]. Várias formas de fazer variar a probabilidade de mutação têm sido propostas [Mühlenbein (1997)], como por exemplo, a diminuição da probabilidade de mutação consoante a convergência da população [Chipperfield e Fleming (1996)].

Para vários problemas tem sido demonstrado que a probabilidade de mutação óptima é igual ao inverso do comprimento do cromossoma. Este critério tem sido igualmente utilizado com bons resultados práticos numa variedade de problemas [Mühlenbein (1997)]. É também admitido que esta probabilidade alcance resultados idênticos aos que se poderia obter com uma variação da probabilidade de mutação ao longo das gerações [Fonseca (1995)].

A probabilidade de mutação também pode ser definida em função da probabilidade de um indivíduo sobreviver (p_s) sem qualquer alteração dos seus genes. Assim, e considerando exclusivamente a actuação do operador mutação, a probabilidade de um indivíduo sobreviver não deve ser inferior ao inverso do número esperado (μ) de filhos do melhor indivíduo [Fonseca (1995)], isto é,

$$p_s \geq \frac{1}{\mu} \ ,$$

$$\text{mas } p_s = (1 - p_m)^l$$

onde l é o comprimento do cromossoma, então

$$p_m \leq 1 - \mu^{-1/l} \tag{3.4}$$

Considerando que além da mutação também está presente o operador cruzamento, então para manter a mesma probabilidade de sobrevivência de um indivíduo de uma geração para a seguinte, deve-se estabelecer a probabilidade de mutação abaixo do limite dado em (3.4) [Fonseca (1995)].

No caso da representação binária a mutação altera o alelo de 0 para 1 ou de 1 para 0. Na representação por uma permutação de n, têm sido utilizadas outras formas de mutação (principalmente para o PCV). As mais comuns são:

TROCA

São escolhidos aleatoriamente dois locus de um cromossoma e trocam-se os respectivos alelos [Potvin (1996), Michalewicz (1996)]. Por exemplo, sejam 2 e 5 os dois locus e considere-se o cromossoma

[1 2 6 5 4 3] obtém-se [1 4 6 5 2 3].

Este operador produz apenas uma pequena mudança no cromossoma, tendo assim uma filosofia muito próxima da mutação aplicada aos cromossomas binários.

MELHORAMENTO LOCAL

Este tipo de mutação [Potvin (1996)] consiste na aplicação de uma heurística de troca de arestas, do tipo 2 ou 3-optimal, durante um certo número de iterações. Um caso particular desta mutação é o operador inversão [Holland (1975)], que inverte os alelos que estão entre dois locus do cromossoma. Para o cromossoma e locus do exemplo anterior, obtém-se:

[1 4 5 6 2 3].

A utilização de AGs com cruzamento heurístico de arestas e mutação do tipo melhoramento local tem obtido bons resultados para o PCV em problemas de grandes dimensões [Potvin (1996), Michalewicz (1996)].

3.3.6 NOVA POPULAÇÃO

A inserção dos filhos na população pode ser feita de diversas formas. Nos AGs mais simples, em cada geração, toda a população é substituída pelos filhos gerados, dando origem a uma nova população. Outras abordagens substituem apenas uma parte da população por novos indivíduos permitindo a competição, na população seguinte, entre alguns pais e filhos. No caso do AG *invariável* ("steady-state") é produzido, através do cruzamento e da mutação, um número mínimo de filhos (em geral um ou dois) que são depois avaliados e inseridos na população [Chipperfield e Fleming (1996), Fonseca (1995)].

Quando apenas uma parte da população é substituída, é necessário escolher quais os indivíduos a substituir. Os critérios mais comuns são:

- escolha aleatória dos indivíduos;
- os indivíduos mais *velhos* (que se mantêm há mais gerações na população);
- os respectivos pais;
- os menos aptos.

A estratégia elitista mantém os melhores indivíduos da população actual na população seguinte. Noutras, os filhos só substituem os pais (ou outros indivíduos) se forem melhores. Outros casos permitem que a população aumente inserindo os novos indivíduos na população actual [Potvin (1996), Michalewicz (1996)].

Qualquer que seja a estratégia utilizada torna-se sempre necessária a avaliação dos indivíduos na nova população.

3.3.7 OUTRAS CARACTERÍSTICAS

Ao longo da execução a população do AG sujeita à selecção aleatória tende a evoluir para uma região pequena do espaço de soluções, mesmo que existam outras regiões com boas e idênticas aptidões. A este fenómeno chama-se *tendência genética* ("genetic drift") e ocorre devido à acumulação de erros do operador selecção. A selecção SUS origina menores desvios, diminuindo a tendência genética sendo, por isso, considerada mais adequada. Populações de dimensões maiores e taxas de mutação mais elevada têm um efeito positivo na eliminação desta tendência mas trazem, como já se referiu, outros problemas. Algumas técnicas para evitar a tendência genética têm sido propostas, tais como:

- A introdução de uma pequena percentagem de *imigrantes* na população, em cada geração, pode ajudar o AG a recuperar alguma informação genética perdida com a actuação da selecção [Fonseca (1995)].

- A penalização por produzir indivíduos semelhantes, obrigando os filhos a substituir os pais que mais se lhes assemelhem [Goldberg (1989)]. Esta técnica é conhecida por "crowding".

- Partilha da aptidão dos indivíduos semelhantes [Goldberg (1989)]. A aptidão destes indivíduos fica reduzida em relação aos outros que mantêm a sua aptidão. Desta forma a probabilidade de indivíduos semelhantes serem seleccionados diminui e leva a população a formar vários grupos de indivíduos em torno dos óptimos locais.

Estudos aprofundados, teóricos e experimentais, sobre estas e outras dificuldades surgidas no desenvolvimento e aplicação de AGs têm sido desenvolvidos [Goldberg (1989), Michalewicz (1996), Mühlenbein (1997), etc.]. No entanto, não existem regras bem estabelecidas para definir os parâmetros envolvidos na actuação dos operadores, nem demonstrações matemáticas que permitam saber com exactidão como desenvolver um AG eficiente para determinado problema.

3.4 PARALELIZAÇÃO

3.4.1 META-HEURÍSTICAS

As meta-heurísticas, como se referiu, tentam fugir dos óptimos locais, e obter uma solução óptima ou próxima da óptima, sacrificando os tempos de execução computacional no processo de pesquisa do espaço de soluções. Este aspecto negativo das meta-heurísticas torna, à priori, a sua execução em máquinas paralelas particularmente interessante.

Para se passar de um algoritmo sequencial a um paralelo tem que se ter em consideração o tipo de máquina em que vai ser implementado (como já se expôs no capítulo 2) e a estrutura do problema que vai ser resolvido. Outro aspecto importante, e nem sempre fácil, é a definição da topologia mais adequada.

Entre as meta-heurísticas descritas anteriormente, tem-se, por um lado, o AS e a PT que são inerentemente sequenciais, o que dificulta a sua paralelização. Por outro, o SCF e o AG são bons candidatos à paralelização devido a operarem com um conjunto de soluções.

A maneira, mais simples e directa, de paralelizar o AS e a PT [Holmqvist et al (1997), Badeau et al (1997)] consiste em, cada processador, aplicar ao problema o algoritmo sequencial (AS ou PT), partindo de uma solução inicial diferente, ou utilizando um conjunto de valores diferentes para os parâmetros. No fim, a solução é a melhor entre as soluções obtidas por cada processador. Outra alternativa, consiste em executar, sequencialmente, o algoritmo em cada processador, por um determinado tempo (ou número de iterações). Depois, os processadores são coordenados de forma a explorarem melhor as regiões que se tenham mostrado mais promissoras [Holmqvist et al (1997)].

Outras duas estratégias usuais consistem em dividir, pelos processadores, o trabalho efectuado pelo algoritmo sequencial. Uma divide o espaço das soluções pelos processadores e cada um executa a pesquisa local apenas nessa parte. A outra divide,

pelos processadores, os procedimentos do algoritmo que consomem mais tempo. Qualquer destas duas abordagens depende totalmente da estrutura do problema e das máquinas paralelas a utilizar, podendo ser vantajosa, ou não, em relação ao algoritmo sequencial.

Uma estratégia mais específica da PT [Holmqvist et al (1997), Badeau et al (1997)] consiste na paralelização do procedimento para gerar e avaliar a vizinhança da solução actual, visto que, geralmente, a avaliação de cada potencial movimento é independente. Note-se que, em cada iteração, o numero de potenciais movimentos depende do tamanho do problema e da definição da função de vizinhança e pode ser muito grande.

Em relação ao SCF cada formiga pode construir a sua solução independentemente das outra. Assim, podem-se distribuir as formigas pelos processadores, sendo interessante que haja trocas de informação, entre processadores, sobre os rastos de feromona. No entanto, talvez seja mais apropriado utilizar os processadores para produzir mais soluções (formigas) e pesquisar melhor o espaço das soluções, executando em paralelo o SCF durante o mesmo tempo que levaria a execução sequencial.

Para problemas que possam ser formalizados com multi-objectivos existe também a alternativa de cada processador trabalhar com um conjunto de formigas (no SCF) ou com uma população de indivíduos (no AG) para atingir um objectivo diferente

.

3.4.2 ALGORITMOS GENÉTICOS

O objectivo de paralelizar um algoritmo é, em geral, torná-lo mais rápido e consequentemente permitir a resolução de problemas de maiores dimensões e, também, a resolução de alguns problemas em tempo real. No caso dos AGs, a sua paralelização permite ainda torná-lo mais semelhante aos sistemas naturais devido a ser possível introduzir estrutura e localização geográfica na população. Desta forma a utilização de

AGs em máquinas paralelas (AGPs) pode trazer outros benefícios para além do decréscimo no tempo de execução.

Para se avaliar se num problema específico deve ser utilizado um AGP é necessário analisar a relação entre a diversidade da população em termos da dimensão da população e o tempo de execução para a tratar. Uma população reduzida levará a um tempo de execução pequeno, mas certas regiões do espaço das soluções não serão pesquisadas devido à falta de diversidade genética da população. Isto pode ter como consequência que só sejam alcançados óptimos locais ou soluções de fraca qualidade. Por outro lado, grandes populações mantêm uma boa diversidade genética, mas o seu tratamento sacrifica os tempos de execução.

Atendendo às várias formas de explorar o paralelismo dos AGs, à estrutura da população e aos operadores de reprodução utilizados, podemos classificar os AGPs, descritos na literatura, em três grandes grupos: global, de migração e de difusão [Chipperfield e Fleming (1996)].

MODELO GLOBAL

No AGP global a população é vista como uma unidade e é explorado o paralelismo inerente ao AG. Este modelo destina-se a casos em que a parte mais pesada, em termos computacionais de um AG, é formada pelo cruzamento, mutação e avaliação dos novos indivíduos.

Utiliza uma topologia "Master - Slave" síncrona [Goldberg (1989), Huntley e Brown (1996)], onde o processador "Master" contém toda a população, calcula a aptidão dos indivíduos e executa a selecção. Os processadores "Slave" aplicam os operadores cruzamento e mutação aos pares de indivíduos que lhes foram enviados pelo "Master" e calculam o valor da função objectivo para os novos indivíduos, enviando os resultados para o "Master" (ver figura 3.2).

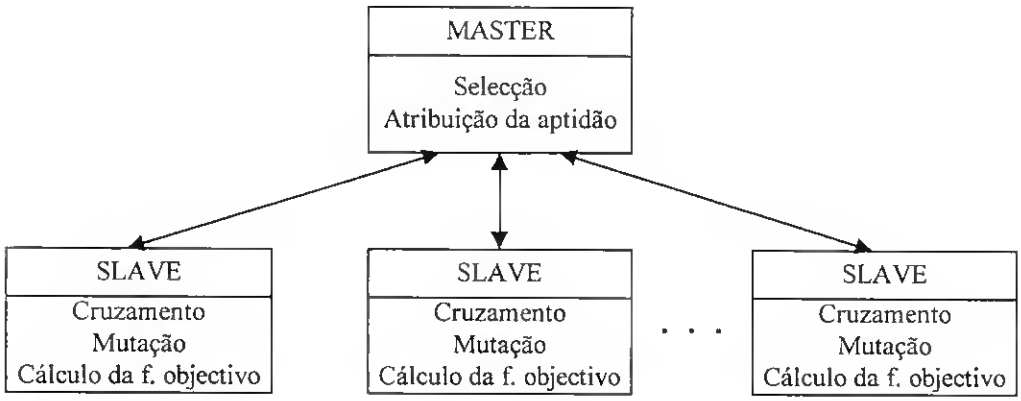


Figura 3.2 – Modelo global de um AGP

Se o cálculo do valor da função objectivo for computacionalmente rápido ou se for muito variável, o AGP global pode ficar pouco eficiente devido à ocorrência de "engarrafamentos".

Mesmo optando por tornar este modelo assíncrono [Goldberg (1989), Huntley e Brown (1996)], isto é, o "Master" enviaria os indivíduos a serem reproduzidos e actualizaria a sua população à medida que recebesse uma mensagem de um processador "Slave" informando que estaria pronto para enviar os seus resultados e receber nova tarefa, a dependência dos "Slaves" em relação ao "Master" pode comprometer a eficiência.

Uma extensão deste modelo, mais robusta, recorre a um sistema com memória partilhada, onde é armazenada a população, e implementa uma paralelização assíncrona do AG [Goldberg (1989)]. Neste caso, cada processador aplica os operadores genéticos e calcula o valor da função objectivo, independentemente dos outros processadores, a indivíduos da população armazenada na memória partilhada. Tem a desvantagem dos processadores não poderem aceder simultaneamente ao mesmo indivíduo e de ser mais difícil de implementar do que o modelo original.

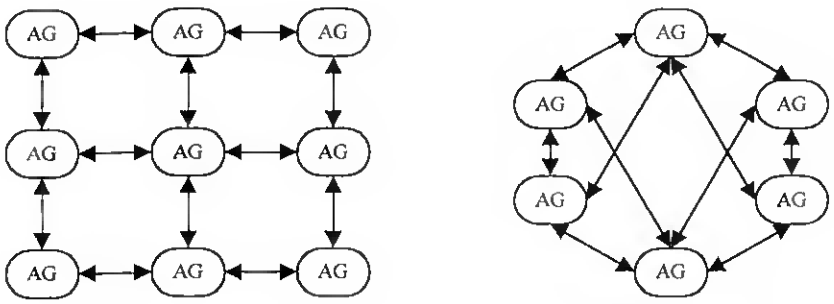
MODELO DE MIGRAÇÃO

O modelo de migração [Chipperfield e Fleming (1996), Michalewicz (1996)] ou de ilhas, como também é chamado, dos AGPs reproduz a distribuição geográfica das

populações dos sistemas naturais, dividindo uma grande população em várias subpopulações semi isoladas. Cada subpopulação forma uma unidade distinta que utiliza selecção local e regras de reprodução para ir evoluindo localmente. De tempos a tempos ocorre uma migração de indivíduos entre subpopulações que provoca a introdução de indivíduos de uma população noutra. Em geral, este modelo é do tipo grão-grosseiro.

O número de indivíduos que migram, o intervalo entre migrações e os caminhos de migração entre subpopulações vão limitar a diversidade genética da população total. A determinação destes valores bem como a escolha de quais os indivíduos que migram tem que ser definida de acordo com o problema.

Em relação ao AG sequencial este modelo introduz no código mais uma rotina para troca (migração) de indivíduos entre subpopulações de acordo com a topologia de comunicações utilizada. Por exemplo, numa topologia de rede em grelha (figura 3.3 A), o mais natural é as migrações darem-se entre processadores adjacentes, numa topologia em anel (figura 3.3 B) é comum que haja também migrações entre processadores não adjacentes.



A - Rede em grelha B - Rede em anel

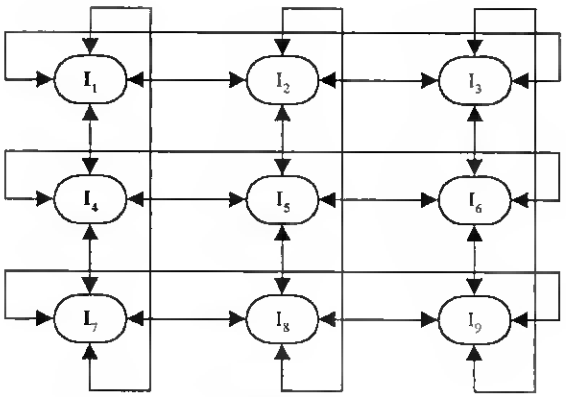
Fig. 3.3 – Modelo de Migração de um AGP

MODELO DE DIFUSÃO

A difusão [Mühlenbein (1997), Michalewicz (1996)] é um modelo alternativo que também tem uma população distribuída, mas é do tipo grão-fino (já definido no capítulo 2). Neste caso, cada indivíduo tem uma localização geográfica na superfície

ocupada pela população e pode-se combinar com indivíduos de uma vizinhança próxima, em geral os adjacentes.

Este modelo pode ser apropriadamente implementado numa topologia de rede toros [Chipperfield e Fleming (1996)], onde cada indivíduo é afecto a um processador da rede (figura 3.4). Cada processador envia o seu indivíduo (por exemplo o $\mathbf{I_4}$) para os processadores adjacentes e recebe destes os seus indivíduos ($\mathbf{I_1}$, $\mathbf{I_5}$, $\mathbf{I_6}$, $\mathbf{I_7}$). A selecção é feita em cada processador entre o seu indivíduo e os vizinhos recebidos, a reprodução é feita como habitualmente através dos operadores cruzamento e mutação e origina, em cada processador, um único indivíduo que vai, se for suficientemente apto, substituir o indivíduo residente nesse processador



I_i - indíviduo i

Figura 3.4 – Modelo de difusão de um AGP

Este modelo tem a característica de todas as decisões serem tomadas pelos próprios indivíduos sem nenhum controle centralizado, sendo totalmente distribuído, tal como acontece nos sistemas naturais auto-organizados [Mühlenbein (1997)].

No início a distribuição genética da população pela rede é aleatória. Após algumas gerações surgem, em geral, grupos de indivíduos com material genético semelhante, formando muitas manchas na rede que, ao longo das gerações, vão diminuindo em número e aumentando em tamanho, pois os indivíduos mais aptos difundem as suas características (de vizinho em vizinho) a toda a população.

Têm surgido algumas variantes destes modelos adaptadas ao problema a resolver, às diferentes máquinas paralelas e às topologias de comunicação em que têm sido implementadas [Chipperfield e Fleming (1996)].

3.5 CONCLUSÕES

As meta-heurísticas descritas têm sido combinadas entre si e com outro tipo de algoritmos (procedimentos de melhoramento, técnicas ávidas, etc.) de diferentes formas dando origem a variados algoritmos híbridos [Glover et al (1995)]. De facto, se no AS forem incorporadas funções de memória, do tipo das utilizadas na PT, obtém-se um algoritmo híbrido que provavelmente será mais eficiente que o AS. Também o SCF deverá ficar mais eficiente se o rasto de feromona for utilizado para produzir soluções e estas forem optimizadas aplicando-lhes procedimentos melhorativos. Da mesma forma, procedimentos melhorativos aplicados aos indivíduos de um AG devem produzir melhores soluções. Muitas outras combinações de técnicas envolvidas nestes métodos podem originar bons algoritmos híbridos.

Tendo em conta que subjacente à maioria das meta-heurísticas e algoritmos híbridos está a utilização da memória ou de funções de memória, quer seja para armazenar várias soluções que vão sendo trabalhadas, como no AG e no SCF, quer seja para intensificar e diversificar a pesquisa, como na PT, e com o intuito de agrupar, numa única terminologia mais simples e significativa, estas meta-heurísticas baseadas na utilização da memória, Taillard et al (1998) propõem a designação de Programação com Memória Adaptável ("Adaptive Memory Programming", PMA).

Neste trabalho pretende-se estudar um algoritmo PMA e a sua paralelização para o problema de Programação Quadrática 0-1 e para o Problema de Optimização de Rotas de Veículos. Escolheram-se os AGs por, de acordo com o que foi exposto, se ter concluído que:

- Os algoritmos PMA mais apropriados a serem paralelizados são os que trabalham com várias soluções simultaneamente (AG e SCF).

- As meta-heurísticas AS, PT e SCF envolvem um grande número de parâmetros que é necessário afinar para cada caso concreto. Os AGs envolvem menos parâmetros e permitem, de uma forma fácil, definições diferentes dos operadores mais apropriadas a cada problema específico, bem como a inclusão de outras técnicas melhorativas próprias das heurísticas.

- Os Algoritmos Genéticos são directamente aplicáveis ao problema de Programação Quadrática 0-1. Contudo, para o Problema de Optimização de Rotas de Veículos é necessário encontrar uma forma adequada de aplicar os operadores genéticos. Assim, pareceu interessante, estudar e comparar o comportamento desta meta-heurística em problemas tão distintos.

O facto de praticamente não existirem AGs para qualquer um destes problemas, ao contrário do que acontece, principalmente, com a Pesquisa Tabu, embora pudesse ser um indício de que nestes casos não seriam eficientes, também contribuiu para a decisão de os estudar.

CAPÍTULO 4

PROGRAMAÇÃO QUADRÁTICA 0-1

4.1 INTRODUÇÃO

Um problema de Programação Quadrática 0-1 sem restrições (PQ 0-1) é um problema de optimização discreta, que pode ser formalizado como:

$$\begin{aligned} & min\ f(x) = c^{\mathrm{T}} x + \tfrac{1}{2}\, x^{\mathrm{T}} Q\, x \\ & x \in \{0,1\}^n \end{aligned} \tag{4.1}$$

onde $Q \in \Re^{n \times n}$ e $c \in \Re^n$.

Este problema tem muitas aplicações importantes. Problemas de análise financeira [McBride e Yomark (1980), Chardaire e Sutter (1995)], de desenho assistido por computador (CAD) [Krarup e Pruzan (1978)], desenho de placas de circuitos impressos e redes de telecomunicações [Boros e Hammer (1991)], afectação de módulos

de programas a processadores em redes distribuídas [Boros e Hammer (1991), Chardaire e Sutter (1995)], modelos de gestão de tráfego de mensagens [Gallo et al. (1980)], entre outros, podem ser resolvidos através de problemas PQ 0-1. É também comum resolver problemas em redes, NP-difíceis, formalizando-os como PQ 0-1.

Neste capítulo indica-se, na secção 2, como alguns problemas de optimização em redes podem ser formalizados através de um PQ 0-1. Mencionam-se, na secção 3, as várias relações entre alguns Problemas Quadráticos e apresentam-se algumas propriedades. Descrevem-se, na secção 4, as principais abordagens que têm sido adoptadas para resolver o PQ 0-1, incluindo um resumo sobre os aspectos fundamentais dos algoritmos paralelos de Pesquisa em Árvore. Expõem-se, na secção 5, os algoritmos desenvolvidos e utilizados neste trabalho e a forma como foram paralelizados. Apresenta-se, na secção 6, o estudo computacional desenvolvido, os resultados obtidos e respectiva análise. Termina-se este capítulo com algumas conclusões e comentários, na secção 7.

4.2 PROBLEMAS DE OPTIMIZAÇÃO E O PQ 0-1

Existe uma estreita relação entre alguns problemas de grafos NP-difíceis e o PQ 0-1. Considere-se que, o grafo $G = (V, E)$, onde V é o conjunto dos vértices e E é o conjunto das arestas, é não orientado, que A_G é a matriz de adjacências de G e que a matriz identidade se representa por I.

Um *Conjunto Independente*, *CI*, de G, é um subconjunto de vértices do grafo G que são não adjacentes dois a dois, ou seja,

$$\forall i, j \in CI \Rightarrow (i, j) \notin E .$$

O *Conjunto Independente Máximo CIM*, de G, é o Conjunto Independente de cardinalidade máxima. Pode ser determinado a partir da solução óptima, x^*, do seguinte PQ 0-1:

$$\begin{aligned} min\ & f(x) = x^{\mathrm{T}}(A_G - I)\, x \\ & x \in \{0, 1\}^n \end{aligned} \tag{4.2}$$

onde n é o número de vértices do grafo G e. É fácil mostrar [Pardalos e Rodgers (1992)] que o conjunto

$$\left\{ i \in V : x_i^* = 1 \right\}$$

é o *CIM* e que

$$|CIM| = -f\left(x^*\right).$$

A formalização (4.2) também é válida para determinar o Conjunto Independente Máximo com pesos, w_i, $i = 1,\ldots, n$, nos vértices do grafo G. Basta incorporar estes pesos na matriz A, fazendo:

$$\begin{aligned} a_{ii} &= -w_i && , \quad i = 1,\ldots,n \\ a_{ij} &= \frac{1}{2}\left(w_i + w_j\right) && , \quad \forall\,(i,j) \in E \\ a_{ij} &= 0 && , \quad \forall\,(i,j) \notin E \end{aligned}$$

Neste caso $-f\left(x^*\right)$ é o peso do Conjunto Independente Máximo.

Um subconjunto *SP* de vértices do grafo G diz-se um *Suporte* no grafo G se toda a aresta de E tem um vértice em *SP*, ou seja,

$$\text{Se } (i,j) \in E \Rightarrow i \text{ ou } j \in SP$$

O *Suporte Mínimo*, *SPM*, é um Suporte de cardinalidade mínima.

Verifica-se, pelas definições, que o Suporte Mínimo e o Conjunto Independente Máximo do grafo G são conjuntos complementares, então

$$SPM = V - CIM,$$

ou seja,

$$SPM = \left\{ i \in V : x_i^* = 0 \right\} \text{ e } |SPM| = n + f\left(x^*\right)$$

sendo x^* a solução óptima de (4.2).

Uma *Clique*, C, do grafo G é um subconjunto de V cujo subgrafo gerado por C, $G(C) = (C, E \cap C \times C)$, é um subgrafo completo de G, ou seja,

$$\forall i, j \in C \Rightarrow (i, j) \in E \cap C \times C$$

Uma *Clique Máxima* do grafo G é uma Clique de cardinalidade máxima.

Resolver o problema da Clique Máxima do grafo G é equivalente a determinar a solução do Conjunto Independente Máximo do grafo complementar $G^c = (V, E^c)$, onde

$$E^c = \{(i, j) : i, j \in V, i \neq j \wedge (i, j) \notin E\}.$$

Assim, também o problema da Clique Máxima do grafo G [Pardalos e Rodgers (1992)] pode ser resolvido pelo PQ 0-1:

$$\begin{aligned} \min f(x) &= x^T \left(A_{G^c} - I\right) x \\ x &\in \{0,1\}^n \end{aligned} \tag{4.3}$$

onde n é o número de vértices e A_{G^c} é a matriz de adjacências do grafo complementar G^c. As variáveis x_i^*, cujo valor é 1 na solução óptima de (4.3), correspondem aos vértices que pertencem à Clique Máxima cuja cardinalidade é igual a $-f(x^*)$.

O problema da *k-Clique* consiste em determinar se um grafo G tem uma clique com k vértices. Este problema é equivalente ao seguinte PQ 0-1 [Pardalos e Rodgers (1992)]:

$$\begin{aligned} \min f(x) &= x^T A x \\ x &\in \{0,1\}^n \end{aligned} \tag{4.4}$$

onde:

$$\begin{aligned} a_{ii} &= -|E|(2k+1), \quad i = 1,\ldots,n \\ a_{ij} &= |E| - 1, \quad \forall (i, j) \in E \\ a_{ij} &= |E|, \quad \forall (i, j) \notin E \end{aligned}$$

Se x^* é solução óptima de (4.4) então o grafo G tem uma k-Clique se e só se $f(x^*)+|E|k^2=-k(k-1)$[Pardalos e Rodgers (1992)]. Nesse caso, a k-Clique, C_k, é dada por:

$$C_k=\{i\in V: x_i^*=1\}.$$

Dado um grafo $G=(V,E)$, com $V=\{0,1,...,n,n+1\}$ o conjunto dos vértices e c_{ij} as capacidades dos arcos $(i,j)\in E$, um *Corte* (V_1,V_1^c) separando os vértices 0 e n+1 é uma partição dos vértices de V tal que $0\in V_1$ e $n+1\in V_1^c$. A capacidade de um Corte (V_1,V_1^c) é igual à soma das capacidades dos arcos que unem os vértices de V_1 aos vértices de V_1^c. O *Corte Mínimo* (*Máximo*) é o Corte (V_1,V_1^c) de capacidade mínima (máxima).

O problema do Corte Mínimo (Máximo) também pode, ser formalizado como um PQ 0-1 [Faustino (1992), Boros e Hammer (1991)]. De facto, sejam $x_0,\ldots,x_n,x_{n+1}$ variáveis 0-1 associadas aos vértices do grafo G. Qualquer afectação dos valores 0 e 1 a estas variáveis define uma partição dos vértices em dois conjuntos $V_1=\{i:x_i=1\}$ e $V_0=\{i:x_i=0\}$. Sendo $x_0=1$ e $x_{n+1}=0$, os arcos $(i,j)\in E$ que ligam os vértices de V_1 aos de V_0 formam um corte (V_1,V_1^c) no grafo G e a sua capacidade pode ser expressa pela seguinte função quadrática:

$$f(x)=\sum_{(i,j)\in E}c_{ij}\left(x_i x_j^c+x_i^c x_j\right)\ \text{com}\ \ x_i^c=1-x_i \tag{4.5}$$

Como qualquer corte do grafo G corresponde a alguma afectação 0-1 das variáveis x_i e $f(x)=f(x^c)$ em (4.5), o problema do Corte Mínimo (Máximo) é equivalente ao seguinte PQ 0-1

$$\begin{aligned} &min\ (max)\ f(x)\\ &x\in\{0,1\}^n\\ &x_0\ =1\\ &x_{n+1}=0 \end{aligned}$$

onde $f(x)$ é a função (4.5).

4.3 PROBLEMAS QUADRÁTICOS

4.3.1 PROBLEMAS QUADRÁTICOS COM LIMITES NOS VALORES DAS VARIÁVEIS

Considere-se o seguinte Problema Quadrático com Limites nos valores das variáveis (PQL)

$$\begin{aligned} \min f(x) &= q^{\mathrm{T}} x + \tfrac{1}{2} x^{\mathrm{T}} M x \\ a &\le x \le b \end{aligned} \tag{4.6}$$

com $x, q, a, b \in \Re^n$ e $M \in \Re^{n \times n}$.

Se a função f for convexa existem vários algoritmos polinomiais bastante eficientes para resolver o problema (4.6) [Pires (1994)]. Se a função f for côncava a solução óptima de (4.6) é um ponto extremo da região admissível, [Hansen et al (1993)].

Um algoritmo de pesquisa em árvore para o problema (4.6) em que a função f é indefinida, é descrito em Hansen et al. (1993).

Em muitas aplicações o vector a é nulo obtendo-se assim a forma:

$$\begin{aligned} \min f(x) &= q^{\mathrm{T}} x + \tfrac{1}{2} x^{\mathrm{T}} M x \\ 0 &\le x \le b \end{aligned} \tag{4.7}$$

Além disso em (4.7)

$$f(x) = \sum_{i=1}^{n} q_i x_i + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} m_{ij} x_i x_j \quad , \text{ com } 0 \le x_i \le b_i$$

e, sendo $0 < b_i < \infty$, $i = 1,...,n$ pode-se fazer a mudança de variável $y_i = \dfrac{x_i}{b_i}$, $i = 1,...,n$, obtendo-se a função:

$$g(y) = c^{\mathrm{T}} y + \tfrac{1}{2} y^{\mathrm{T}} Q y, \text{ com } c_i = q_i b_i \text{ e } q_{ij} = m_{ij} b_i b_j, \; i, j = 1,...,n.$$

Conclui-se, assim, que o PQL, da forma dada em (4.7), é equivalente a um PQ com variáveis limitadas entre 0 e 1,

$$\begin{array}{l} min\ f(x) = q^{\mathrm{T}} x + \frac{1}{2} x^{\mathrm{T}} M x \\ \quad 0 \le x \le b \end{array} \quad \Leftrightarrow \quad \begin{array}{l} min\ g(y) = c^{\mathrm{T}} y + \frac{1}{2} y^{\mathrm{T}} Q y \\ \quad 0 \le y \le e \\ \quad com\ e = (1,\ldots,1)^{\mathrm{T}} \end{array}$$

Se no problema (4.7), M for uma matriz negativa semi-definida então o minímo global, x^*, de f ocorre num ponto extremo do hiper-rectângulo [Faustino (1992)],

$$\Omega = \left\{ x \in \Re^n : 0 \le x \le b \right\}, \tag{4.8}$$

isto é, as variáveis x_i tomam, na solução óptima, o valor 0 ou b_i , logo as variáveis y_i tomam o valor 0 ou 1. Assim, tem-se:

$$\begin{array}{l} min\ f(x) = q^{\mathrm{T}} x + \frac{1}{2} x^{\mathrm{T}} M x \\ \quad 0 \le x \le b \\ \quad \text{com } M \text{ uma matriz negatva} \\ \quad \text{semi - definida} \end{array} \quad \Leftrightarrow \quad \begin{array}{l} min\ g(y) = c^{\mathrm{T}} y + \frac{1}{2} y^{\mathrm{T}} Q y \\ \quad y \in \{0,1\}^n \end{array}$$

e, nestas condições, a solução óptima do problema (4.7) pode ser obtida a partir da resolução de um PQ 0-1.

No entanto, partindo do problema

$$\begin{array}{l} min\ g(y) = c^{\mathrm{T}} y + \frac{1}{2} y^{\mathrm{T}} Q y \\ \quad y \in \{0,1\}^n \end{array}$$

e fazendo as transformações inversas não se obtém necessariamente um PQL com matriz negativa semi-definida. Contudo, fazendo uma perturbação da diagonal da matriz Q, qualquer PQ 0-1 pode ser transformado num PQL côncavo da forma

$$\begin{array}{l} min f(y) = c^{\mathrm{T}} y + \frac{1}{2} y^{\mathrm{T}} Q y + \tau y^{\mathrm{T}} (e - y) \\ \quad 0 \le y \le e \end{array} \tag{4.9}$$

com τ suficientemente grande de modo a que (Q - 2τ I) seja negativa semi-definida.

Em alguns casos pretende-se resolver o seguinte PQ

$$\begin{aligned} min\ & f(z) = q^{\mathrm{T}} z + \tfrac{1}{2} z^{\mathrm{T}} M z \\ & z \in \{-1, 1\}^n \end{aligned} \tag{4.10}$$

com $M \in \Re^{n \times n}$ e $q \in \Re^n$.

Fazendo em (4.10) a transformação $z = e - 2x$, com $e = (1,...,1)^{\mathrm{T}}$, obtém-se o PQ 0-1 de (4.1) onde

$$Q = 4M \quad \text{e} \quad c = \frac{-4q - Qe}{2} \ .$$

Portanto, podem-se utilizar os métodos de resolução do PQ 0-1, para encontrar a solução óptima de (4.10).

4.3.2 PROPRIEDADES DOS PROBLEMAS QUADRÁTICOS

Usando certas características específicas dos PQL (de (4.7) fazendo $b = e$) e propriedades da função gradiente é possível desenvolver técnicas que permitem fixar definitivamente o valor que certas variáveis vão tomar na solução óptima. Nesse sentido, em Pardalos (1989) é demonstrado o seguinte teorema:

Teorema 4.1: Se x^* é o mínimo global de um PQ com variáveis limitadas entre 0 e 1 então x^* é também a solução óptima do problema linear

$$\begin{aligned} min\ & (\nabla f(x^*))^{\mathrm{T}} x \\ & 0 \le x \le e \end{aligned} \quad \text{, onde } e = (1,...,1)^{\mathrm{T}} .$$

Isto permite afirmar que as variáveis x_i, cujas derivadas parciais têm sinal constante no hipercubo unitário, podem ser fixas no valor 0 ou 1 de acordo com esse sinal. Assim, Pardalos (1989) estabelece o corolário seguinte:

Corolário 4.1: Suponha-se que $l_i \leq \dfrac{\partial f(x)}{\partial x_i} \leq L_i$ para todo o $x \in [0, 1]^n$, então:

(i) Se $l_i \geq 0 \Rightarrow x_i^* = 0$;

(ii) Se $L_i \leq 0 \Rightarrow x_i^* = 1$.

A partir deste corolário é possível concluir que o conhecimento de boas estimativas para os valores de l_i e L_i é bastante importante na resolução deste tipo de problemas. Como é fácil constatar, há todo o interesse em encontrar técnicas que permitam determinar limites l_i e L_i de forma que o intervalo $[l_i, L_i]$ da variação das derivadas parciais $\dfrac{\partial f(x)}{\partial x_i}$ tenha a menor amplitude possível..

Considere-se o PQ 0-1 de (4.1). Como, neste problema, as componentes do vector x só podem tomar os valores 0 ou 1, então $x_i = x_i^2$, $i = 1,...,n$. Este facto permite incorporar os termos lineares na diagonal da matriz da forma quadrática, obtendo uma função objectivo equivalente. Isto leva a que se utilize indiferentemente as expressões "termos diagonais" e "termos lineares", e se chame termos quadráticos aos termos não diagonais.

A partir desta propriedade é possível encontrar muitas funções $\tilde{f}$ que sejam equivalentes a f definida por $f(x) = c^{\mathrm{T}} x + \frac{1}{2} x^{\mathrm{T}} Q x$, desde que se respeite o princípio de manter constante a soma do coeficiente linear com o elemento da diagonal. Assim, por exemplo, os problemas

$$\begin{aligned} & min\, \hat{f}(x) = \hat{c}^{\mathrm{T}} x + \frac{1}{2} x^{\mathrm{T}} \hat{Q}\, x \\ & x \in \{0,1\}^n \end{aligned} \tag{4.11}$$

$$\text{com } \hat{c}_i = c_i + \frac{1}{2} q_{ii} \quad \text{e} \quad \hat{q}_{ij} = \begin{cases} q_{ij} & \text{se } i \neq j \\ 0 & \text{se } i = j \end{cases}$$

e

$$\begin{aligned} &min\, f(x) = x^{\mathrm{T}} A x \\ &x \in \{0,1\}^n \end{aligned} \tag{4.12}$$

$$\text{onde } a_{ii} = c_i + \frac{q_{ii}}{2} \text{ e } a_{ij} = \frac{q_{ij}}{2} \text{ para } i \neq j$$

são equivalentes ao problema (4.1).

Em Pardalos (1989) é demonstrado que a utilização da formalização (4.11) conduz aos menores intervalos de $[l_i, L_i]$ para todas as funções equivalentes a f. Então, para determinar esses limites podem-se usar as expressões

$$l_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} \hat{q}_{ij}^{-} + \hat{c}_i \quad \text{e } L_i = \sum_{\substack{j=1 \\ j \neq i}}^{n} \hat{q}_{ij}^{+} + \hat{c}_i \tag{4.13}$$

$$\text{onde } \hat{q}_{ij}^{-} = min\left\{0, \hat{q}_{ij}\right\} \text{ e } \hat{q}_{ij}^{+} = max\left\{0, \hat{q}_{ij}\right\}$$

Por outro lado, a formulação (4.12) é a mais simples de todas e muitos algoritmos são desenvolvidos a partir dela. Neste trabalho, a forma (4.12) foi utilizada no desenvolvimento dos algoritmos. Nesta formalização, sem qualquer perda de generalidade, pode-se considerar a matriz A simétrica. Assim, trabalha-se só com os elementos da parte triangular superior (ou inferior) da matriz e tem-se:

$$\begin{aligned} f(x) &= \sum_{i=1}^{n}\sum_{j=1}^{n} a_{ij} x_i x_j \\ &= \sum_{i=1}^{n} a_{ii} x_i x_i + \sum_{i=1}^{n}\sum_{j=i+1}^{n} \left(a_{ij} + a_{ji}\right) x_i x_j \\ &= \sum_{i=1}^{n} a_{ii} x_i + 2\sum_{i=1}^{n}\sum_{j=i+1}^{n} a_{ij} x_i x_j \end{aligned} \tag{4.14}$$

4.4 MÉTODOS DE RESOLUÇÃO PARA O PQ 0-1

O problema formalizado em (4.1) resolve-se em tempo polinomial, se os elementos não diagonais da matriz Q forem todos não positivos [Picard e Ratliff (1974)], ou para alguns tipos específicos de matrizes [Barahona (1986)]. Mas, em geral, este problema pertence à classe dos problemas NP-difíceis [Garey e Johnson (1979)], sendo, portanto, de difícil resolução.

Hansen (1979) faz uma síntese dos métodos conhecidos, até então, para resolver o PQ 0-1. Posteriormente, muitos trabalhos [Carter (1984), Gulati et al. (1984), Barahona et al. (1989), Pardalos e Rodgers (1990), Chardaire e Sutter (1995), Glover et al (1998), etc.] têm sido apresentados para resolver este problema.

4.4.1 ALGORITMOS EXACTOS

Os algoritmos exactos recorrem à enumeração (explícita ou implícita) de todas as soluções admissíveis, em geral, através da pesquisa em árvore. Como a enumeração exaustiva, na generalidade dos casos, não é exequível, utilizam-se técnicas de corte para reduzir o número de soluções pesquisadas. Por isso, muitos trabalhos centram-se na pesquisa de bons limites, inferiores ou superiores consoante se pretenda minimizar ou maximizar a função objectivo [Hui (1984), Boros et al (1990), Poljak e Wolkowicz (1993), Luz (1998)] e em métodos de escolha das variáveis a ramificar [Körner (1983)].

No entanto, mesmo utilizando bons limites os métodos exactos requerem, geralmente, tempos de execução computacional proibitivos. Uma forma óbvia de tentar reduzi-los é a utilização de máquinas paralelas

Assim, descrevem-se, em seguida, os principais tipos de paralelismo que se podem aplicar a um algoritmo de Pesquisa em Árvore. Para distinguir entre um algoritmo de Pesquisa em Árvore a ser executado em máquinas sequenciais de outro a

ser executado em máquinas paralelas, vamos chamar ao primeiro algoritmo sequencial e ao segundo algoritmo paralelo. Seguidamente, resumem-se algumas abordagens utilizadas para a obtenção de limites da função objectivo.

ALGORITMOS PARALELOS DE PESQUISA EM ÁRVORE

Um algoritmo de Pesquisa em Árvore (PA) é um método enumerativo para resolver exactamente um problema de optimização

$$P_0: \qquad Z = \min_{x \in S} f(x)$$

onde f é uma função real e S a região admissível. Represente-se por x^* uma solução óptima de P_0 e por $Z(P_0)$ o valor óptimo.

Adicionando uma ou mais restrições ao problema inicial P_0 obtém-se um subproblema P_i. Cada subproblema P_i pode ser definido por:

$$P_i: \qquad Z = \min_{x \in S_i} f(x)$$

em que o conjunto de restrições S_i é tal que $S_i \subset S$ e $\bigcup_i S_i \subseteq S$. Sejam $Z(P_i)$ o valor óptimo de P_i, g uma função limite inferior ($g \leq f$) e $\underline{Z}(P_i)$ o valor da solução óptima de P_i, obtido com a função g.

Considere-se a árvore de pesquisa $G = (V, E)$ onde V é o conjunto dos vértices e E o conjunto das arestas. A raiz de G, denotada por P_0, corresponde ao problema inicial P_0 e os outros vértices $P_i \in V$ correspondem aos subproblemas P_i. A aresta $(P_i, P_j) \in E$ se e só se P_j for obtido por ramificação de P_i. Represente-se por T o conjunto dos subproblemas resolvidos de G, isto é, os subproblemas P_i para os quais $g = f$, e B o conjunto formado pelo problema inicial e pelos vários subproblemas. O conjunto dos vértices activos (que ainda não foram divididos nem testados) é denotado por Act.

Durante a PA chame-se solução incumbente à melhor solução do problema P_0 já obtida e represente-se por $\bar{x}$ sendo $\bar{Z}(P_0)$ o seu valor ($Z(P_0) \leq \bar{Z}(P_0)$).

De acordo com as notações anteriores, em cada vértice tem-se:

- $\underline{Z}(P_i) \leq Z(P_i)$ para $P_i \in V$
- $\underline{Z}(P_i) = Z(P_i)$ para $P_i \in T$
- $\underline{Z}(P_i) \leq \underline{Z}(P_j)$ se P_j for um descendente de P_i

Em termos gerais descreve-se um algoritmo de PA através dos seguintes passos [Ibaraki (1987)] (algoritmo 4.1):

Passo 1:
Fazer $Act = \{P_0\}$, $B = \{P_0\}$, $\overline{Z}(P_0) = \infty$ e $T = \varnothing$.
Passo 2:
Se $Act = \varnothing$ ir para o Passo 7,
Caso contrário seleccionar um vértice $P_i \in Act$ e ir para o Passo 3.
Passo 3:
Se $g = f$, para P_i
fazer $T = T \cup \{P_i\}$
e se $\underline{Z}(P_i) < \overline{Z}(P_0)$
fazer $\overline{Z}(P_0) = \underline{Z}(P_i)$ e $\bar{x} = x$
Ir para o Passo 6.
Caso contrário ir para o Passo 4.
Passo 4:
Se $\overline{Z}(P_0) \leq \underline{Z}(P_i)$ ir para o Passo 6,
Caso contrário ir para o Passo 5.
Passo 5:
Dividir P_i em k subproblemas $P_{i_1}, \ldots, P_{i_k}$ e fazer:
$Act = Act \cup \{P_{i_1}, \ldots, P_{i_k}\} - \{P_i\}$
$B = B \cup \{P_{i_1}, \ldots, P_{i_k}\}$.
Voltar para o Passo 2.
Passo 6:
Fazer $Act = Act - \{P_i\}$ e voltar para o Passo 2.
Passo 7:
Terminar a execução.
Se $\overline{Z}(P_0) < \infty$
então $Z(P_0) = \overline{Z}(P_0)$ e $x^* = \bar{x}$ é a solução óptima,
Se $\overline{Z}(P_0) = \infty$ então P_0 não tem solução admissível.

Algoritmo 4.1 – Algoritmo sequencial de Pesquisa em Árvore

Quanto menor for $\overline{Z}(P_0)$ mais cedo os ramos da árvore poderão ser podados (Passo 4). Assim, é comum utilizar um método heurístico eficiente para obter uma solução admissível, $x \in S$, com valor $z = f(x)$ próximo do óptimo. Faz-se então, no Passo 1, $\overline{Z}(P_0) = z$ e $\overline{x} = x$.

Ao contrário doutras aplicações, o paralelismo de um algoritmo de PA não é tão evidente. As estratégias de pesquisa da árvore mais utilizadas, melhor limite e pesquisa em profundidade, são, por definição, sequenciais e levam a uma certa ordenação dos vértices da árvore.

No entanto, dois tipos de paralelismo podem, obviamente, ser aplicados aos algoritmos paralelos de PA. Um consiste em efectuar as operações, nos subproblemas ($P_i \in B$), em paralelo. O outro, constrói paralelamente árvores de pesquisa $G = (V, E)$ distintas, cada uma baseada em diferentes critérios de ramificação e de pesquisa. A informação relevante obtida numa é transmitida às outras.

A nível dos dados, existe uma forte dependência apenas entre um vértice e os seus descendentes, isto é, entre os vértices de cada ramo da árvore, mas não há dependência entre ramos distintos. Como os vértices activos ($P_i \in Act$) não têm, entre si, uma relação de ascendência-descendência, podem ser subdivididos simultaneamente. Assim, os diferentes ramos da árvore de pesquisa constituem subproblemas independentes e podem ser tratados paralelamente. Este é o tipo de paralelismo que mais tem sido explorado nos algoritmos paralelos de PA. Esta abordagem é mais apropriada para sistemas MIMD assíncronos e grão-grosseiro (já mencionados no capítulo 2) [Gendron e Crainic (1994)], embora também seja executada em sistemas SIMD [Pardalos e Li (1990)]. As árvores obtidas paralelamente são, em geral, diferentes das obtidas sequencialmente e podem, por isso, ocasionar eficiência superior a um ou ser muito pouco eficientes. Em Gendron e Crainic (1994), é feita uma compilação detalhada dos trabalhos publicados, quer a nível de implementações de algoritmos paralelos de PA, quer a nível de desenvolvimentos teóricos sobre as acelerações superlineares e anomalias, utilizando este tipo de paralelismo em sistemas MIMD. Em Corrêa et al (1998) e em Androulakis e Floudas (1998), são apresentados vários aspectos

relacionados com a paralelização de algoritmos de PA em máquinas com memória partilhada e com memória distribuída.

Estes três tipos de paralelismo podem ser combinados e utilizados de uma forma conjunta, em diferentes partes de um algoritmo paralelo de PA [Pekny e Miller(1992)].

Como já foi referido no capítulo 2, quando se procede à paralelização de um algoritmo, é necessário ter em consideração alguns aspectos que contribuem para a sua eficiência. No caso do algoritmo de PA, utilizando o terceiro tipo de paralelismo descrito (vários ramos em paralelo), os principais aspectos a considerar são:

- Divisão da árvore pelos processadores;
- Ocupação equilibrada dos processadores;
- Frequência e sincronização das comunicações entre processadores.

A forma de os abordar depende do tipo de máquina paralela que se pretende utilizar (memória distribuída, memória partilhada, número de processadores disponíveis, arquitectura a empregar, etc.). As maiores diferenças surgem entre os algoritmos para sistemas com memória partilhada e para sistemas com memória distribuída.

Considere-se um sistema com $N + 1$ processadores, T_0, T_1, ... , T_N , em que cada processador T_i se encontra no estado livre ($st(T_i) = 0$), ou no estado ocupado ($st(T_i) = 1$).

Se o sistema tiver memória partilhada [Gendron e Crainic (1994)], esta é acedida por todos os processadores. É pois comum, esta conter todos os dados globais, isto é,

- a lista dos problemas activos $P_i \in Act$;
- a solução incumbente, $\bar{x}$, e o seu valor $\bar{Z}(P_0)$.

Os processadores estão sincronizados por ciclos. Em cada ciclo, cada processador executa os Passos 2 a 7 do algoritmo sequencial. Retira da memória, no Passo 2, um dos subproblemas do conjunto *Act*. Coloca na memória, no Passo 5, os

subproblemas obtidos, actualizando os conjuntos *Act* e *B*. Actualiza na memória, no Passo 3, a solução incumbente, $\bar{x}$,e o seu valor $\overline{Z}(P_0)$.

Inicialmente cada processador está no estado livre. Quando executa o Passo 2 muda para o estado ocupado. Ao atingir o Passo 7 fica no estado livre. O algoritmo termina quando todos os processadores estiverem no estado livre o que corresponde a ter-se na memória o conjunto *Act* vazio.

Nos sistemas com memória distribuída, cada processador tem a sua memória e executa o seu próprio código, o que torna o processo de troca de informação mais complicado. Em geral, têm sido consideradas duas abordagens para proceder à paralelização de um algoritmo de PA nestes sistemas.

A primeira [Pardalos e Li (1990)] executa o algoritmo de uma forma muito semelhante à descrita para os sistemas com memória partilhada. Recorre a uma lista centralizada (LC) dos problemas activos ($P_i \in Act$), a um processador "master" que gere o processo e a N processadores "slaves". O processador "master" contém os dados globais (LC, $\overline{Z}(P_0)$, $\bar{x}$), selecciona os *N* melhores subproblemas do conjunto *Act* e envia-os para os *N* processadores "slaves" (um para cada processador). Estes, por sua vez, executam as operações necessárias para gerar novos subproblemas, calcular os limites e testar os critérios de eliminação e de paragem, ou seja, executam os Passos 2 a 7 do algoritmo sequencial. Enviam os resultados, $P_i \in Act$, $\overline{Z}(P_0)$, $\bar{x}$, para o processador "master", que os insere na LC e procede às actualizações dos dados. O algoritmo termina quando a LC está vazia e os N processadores "slaves" estão livres.

Neste método o processador "master" tem o papel da memória partilhada, o que tem três desvantagens consideráveis:

- Necessita de um grande espaço de memória (do processador "master") para armazenar a LC ;
- Requer a comunicação de duas mensagens para cada subdivisão de um subproblema (uma para receber o subproblema activo, outra para enviar os subproblemas gerados);

- A comunicação dos N processadores "slaves" com o processador "master" vai originar, obviamente, estrangulamentos que degradam o desempenho do algoritmo.

Na segunda abordagem [Pardalos e Li (1990)], tenta-se aproveitar melhor os recursos deste tipo de sistema (memória distribuída), repartindo os dados globais por todos os processadores. Ao contrário de uma lista centralizada passa-se a ter uma lista distribuída (LD). Cada processador, T_j, tem a sua lista de subproblemas activos, $P_i \in Act_j$, a sua solução incumbente, $\bar{x}_j$, e respectivo valor, $\bar{Z}_j(P_0)$. A um dos processadores, seja T_0, é afecto o problema inicial. A execução começa subdividindo esse problema e enviando os subproblemas gerados, $P_i \in B$, para os outros processadores, até todos os processadores estarem ocupados (com um ramo da árvore para tratar). Cada processador executa a pesquisa no seu ramo da árvore, expandindo-a, testando os critérios de eliminação e paragem, actualizando a solução incumbente, $\bar{x}_j$, e respectivo valor, $\bar{Z}_j(P_0)$. Insere todos os resultados na sua lista local e quando termina a sua pesquisa envia a solução incumbente para o processador T_0 e passa ao estado livre. Quando todos os processadores estiverem livres o algoritmo termina. A solução óptima é a melhor de todas as soluções incumbentes.

As principais desvantagens deste método são:

- O trabalho não estar distribuído de uma forma equilibrada pelos processadores;

- Cada processador ter apenas o incumbente referente ao seu ramo.

A distribuição não equilibrada do trabalho pelos processadores pode acontecer em duas situações: no início, quando ainda não foram enviados subproblemas para todos os processadores e durante a execução, quando alguns processadores ficam livres e outros ainda estão ocupados.

Inicialmente o paralelismo não é completamente conseguido, porque, por um lado a subdivisão do problema P_0 gera poucos subproblemas, por outro lado o processador T_0 centraliza esta operação e também o envio dos subproblemas para todos

os outros processadores, o que torna o processo de difusão mais demorado fazendo com que alguns processadores permaneçam livres durante algum tempo.

Sendo um algoritmo de PA imprevisível, não se pode saber qual o comprimento de cada ramo, nem qual o tempo de execução de cada subproblema (uns são mais demorados do que outros), assim, o tempo de execução da pesquisa difere de um ramo para o outro, o que leva a que alguns processadores possam ficar livres mais cedo do que outros.

O facto de, em cada iteração, o melhor incumbente não ser conhecido por todos os processadores (cada processador tem apenas o incumbente actualizado no seu ramo), como acontece no algoritmo sequencial ou no caso da memória partilhada, faz com que alguns subproblemas não sejam podados, podendo levar inclusive, em certas instâncias, a que o algoritmo paralelo seja mais lento do que o sequencial. Têm sido publicados trabalhos com estudos teóricos sobre esta anomalia [Li e Wah (1984), Androulakis e Floudas (1998)], conhecida como anomalia detrimental.

Várias estratégias para resolver estas desvantagens (ocupação equilibrada, sincronização e granularidade) têm sido propostas e estudadas. Todas dependem da arquitectura escolhida e das comunicações estabelecidas de acordo com a topologia utilizada.

A primeira desvantagem pode ser ultrapassada fazendo com que o processador que recebe o problema inicial o subdivida num número de subproblemas igual ao número de processadores vizinhos, e transmita um subproblema apenas a cada um dos seus vizinhos. Estes por sua vez dividem e transmitem também novos subproblemas aos vizinhos e assim sucessivamente.

Durante a execução a estratégia mais utilizada, para tentar equilibrar o trabalho pelos processadores, consiste em permitir que os processadores livres possam interrogar os ocupados de forma a que estes repartam o seu trabalho e lhes enviem novos subproblemas. Os processadores ocupados podem inclusive estimar o trabalho que lhes falta e verificar se devem ou não transferir uma parte dele para um processador livre. A

forma de executar esta alteração depende, mais uma vez, da topologia utilizada [Pardalos e Li (1990)].

Em relação à segunda desvantagem, é necessário que os incumbentes sejam enviados de uns processadores para os outros de forma a que todos conheçam o melhor valor, o mais cedo possível. A forma de comunicar os incumbentes entre os processadores, depende da topologia utilizada. Em certas implementações estas comunicações são feitas só entre os processadores vizinhos, noutras cada processador comunica com todos os outros. Em qualquer caso, o número de vezes em que os incumbentes são trocados num algoritmo depende da sobrecarga do tempo de execução que essas comunicações acarretem.

No algoritmo paralelo de PA desenvolvido neste trabalho utilizaram-se máquinas paralelas com memória distribuída. Descreve-se, seguidamente em termos genéricos, um algoritmo paralelo de PA, para este tipo de arquitectura (algoritmo 4.2):

Nos Passos 4 e 5 é feita a sincronização das trocas de mensagens entre os processadores. Como se referiu, esta fase pode-se tornar mais eficiente permitindo que os processadores que vão ficando livres comuniquem com os que ainda estão ocupados e recebam destes um subproblema, ou que esperem só até que os processadores vizinhos (poucos) estejam livres.

Passo 1:

Pôr todos os processadores no estado livre, $st(T_i) = 0$, $i = 0,1, \ldots, N$.
Afectar o problema inicial P_0 ao processador T_0,
passá-lo ao estado activo, $st(T_0) = 1$

Passo 2:

Enquanto houver um processador, $T_i : st(T_i) = 0$ (livre)
dividir um subproblema ainda não resolvido ($P_i \in Act$) e
afectar um dos novos subproblemas a esse processador,
passando-o ao estado activo, $st(T_i) = 1$.

Passo 3:

Em cada processador T_i:
Executar a pesquisa em árvore (Passos 2 a 6 do Algoritmo 4.1)
até atingir:
um limite fixado de tempo (*maxt*) e/ou de iterações (*maxn*),
ou o Passo 7 do Algoritmo 4.1 e neste caso fazer $st(T_i) = 0$.

Passo 4:

Esperar até todos os processadores terem terminado os seus
subproblemas (terem passado ao estado livre), ou terem atingido
o limite de tempo e/ou iterações

Passo 5:

Obter a melhor solução , $\bar{x}_i$ e $\overline{Z}_i(P_0)$, $i = 0,1,\ldots,N$.
e transmiti-la a todos os processadores como a nova melhor solução
(solução incumbente actualizada).

Passo 6:

Se todos os processadores estiverem no estado activo,
($st(T_i) = 1$, para $i = 0,1, \ldots, N$)
ir para o Passo 3 (recomeçar a pesquisa).
Caso contrário ir para o Passo 7.

Passo 7:

Se algum dos processadores ainda estiver activo
ir para o Passo 2.
Se todos os processadores estiverem livres
($st(T_i) = 0$, para $i = 0,1,\ldots,N$) terminar.
A solução óptima é a melhor solução incumbente.

Algoritmo 4.2 – Algoritmo paralelo de Pesquisa em Árvore para Sistemas com Memória Distribuída

LIMITES INFERIORES

Os limites inferiores utilizados nos algoritmos de PA para o PQ 0-1 têm sido tipicamente obtidos de dois modos:

(a) Relaxação da condição de integralidade das variáveis;

(b) Linearização dos termos quadráticos.

No caso (a) o limite inferior é dado pelo valor da solução de

$$\begin{aligned} \min f(x) &= c^{\mathrm{T}} x + \tfrac{1}{2} x^{\mathrm{T}} Q x \\ 0 &\le x \le e \end{aligned} \tag{4.15}$$

onde $Q \in \Re^{n \times n}$, $c \in \Re^n$ e $e = (1,...,1)^{\mathrm{T}}$

A desvantagem desta abordagem é que se a matriz Q for indefinida, a função contínua é não convexa e por isso difícil de minimizar. É possível torná-la convexa fazendo a transformação [Carter (1984)]:

$$\tilde{q}_{ii} = q_{ii} + 2w_i \text{ e } \tilde{c}_i = c_i - w_i$$

sendo o vector $w \in \Re^n$ suficientemente grande para que a matriz obtida, $\tilde{Q}$, fique diagonal dominante. O inconveniente de escolher w muito grande é que a solução obtida vai estar próxima do ponto $(0{,}5,...,0{,}5)$, o que não favorece a pesquisa em árvore. Carter (1984) apresenta uma transformação que obtém uma solução contínua entre 0 e 1.

Em Luz (1998) é apresentado um limite inferior para o PQ 0-1, de (4.1), a partir da aplicação de propriedades da dualidade Lagrangeana a um PQ com variáveis contínuas em [0, 1], equivalente ao PQ 0-1. Este limite pode ser calculado através da resolução de um problema quadrático convexo.

No caso (b) é possível obter a partir do PQ um problema de programação linear 0-1 (PL 0-1) com restrições lineares, introduzindo novas variáveis $y_{ij} = x_i x_j$ para

cada termo quadrático de coeficiente não nulo [Pardalos (1989), Boros e Hammer (1991)]. Obtém-se assim, a partir da formalização (4.12), o seguinte PL 0-1:

$$
\begin{aligned}
& min\, f(x) = \sum_{i=1}^{n} a_{ii} x_i + \sum_{i \neq j} a_{ij} y_{ij} \\
\text{s.a} \quad & y_{ij} \geq x_i + x_j - 1 \\
& y_{ij} \leq x_i \\
& y_{ij} \leq x_j \\
& y_{ij} \geq 0 \\
& x \in \{0,1\}^n
\end{aligned}
$$

Estas quatro restrições impostas às variáveis y_{ij} garantem que elas tomam os valores 0 ou 1 e que se tem sempre $y_{ij} = x_i x_j$.

Um limite inferior para este PL 0-1 pode ser obtido relaxando as condições de integralidade, obtendo-se assim, um problema fácil de resolver com técnicas de Programação Linear. Note-se que para cada termo quadrático de coeficiente não nulo é introduzida uma variável y_{ij} . Além disso, para cada uma destas variáveis são geradas quatro restrições, o que faz com que esta abordagem tenha o inconveniente de gerar um grande número de restrições, principalmente se o PQ for moderadamente denso.

Os limites baseados na complementação das variáveis e na utilização de funções lineares que majoram a função quadrática, no caso do problema ser de maximização, conduzem ao mesmo limite obtido através da linearização dos termos quadráticos [Hammer et al (1984), Boros et al (1990)]

Seguindo uma outra abordagem, Chardaire e Sutter (1995) decompõem a função quadrática de (4.1) numa soma de um tipo específico de funções quadráticas, a que chamam pseudo-bilineares, pois são a soma de uma função quadrática com uma função bilinear. Provam que a melhor decomposição é obtida pelo método da decomposição Lagrangeana (um caso particular da relaxação Lagrangeana) e que o valor exacto do dual Lagrangeano fornece um limite inferior para o PQ 0-1.

Em Pardalos e Rodgers (1990) e Pardalos et al (1992) descreve-se um algoritmo eficiente de PA para o PQ 0-1, que utiliza as propriedades deste problema, expostas em 4.3.2, para fixar as variáveis e calcular limites inferiores da função objectivo.

De facto, o corolário 4.1 permanece verdadeiro para o caso discreto, isto é quando $x \in \{0,1\}^n$. Aplicando-o ao PQ 0-1, obtém-se uma forma muito fácil de fixar variáveis nos valores 0 ou 1, tanto no problema inicial como nos subproblemas dos vértices da árvore de pesquisa. Considerando o PQ 0-1 na forma dada em (4.12) determina-se o seguinte intervalo para o valor das derivadas parciais de $f(x)$:

$$l_i \leq \frac{\partial f(x)}{\partial x_i} \leq L_i$$

onde

$$l_i = 2 \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}^{-} + a_{ii} \quad \text{e} \quad L_i = 2 \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}^{+} + a_{ii}$$

com

$$a_{ij}^{+} = max\{0, a_{ij}\} \quad \text{e} \quad a_{ij}^{-} = min\{0, a_{ij}\}$$

e é possível fixar as variáveis nos valores 0 ou 1 nos seguintes casos:

(i) Se $l_i \geq 0 \Rightarrow x_i^* = 0$;

(ii) Se $L_i \leq 0 \Rightarrow x_i^* = 1$.

Nesta abordagem, como se pode ver pelas expressões de l_i e L_i, quantas mais linhas (colunas) da matriz A forem diagonal dominantes, mais fácil se torna a resolução do problema. Se a linha (coluna) i da matriz A, correspondente à variável x_i, for diagonal dominante então a variável x_i pode ser fixa (a 0 ou a 1), de acordo com o sinal (positivo ou negativo) do elemento da diagonal. É claro que, se a matriz A tiver um grande número de elementos diagonal dominantes, todas as correspondentes variáveis são fixas definitivamente no vértice inicial, sendo a pesquisa em árvore executada apenas para as variáveis não fixas, o que origina árvores com poucos vértices.

Reciprocamente, quanto menos linhas da matriz forem diagonal dominantes, maiores deverão ser as árvores de pesquisa. Nestes casos, este algoritmo só consegue resolver eficientemente problemas de muito pequena dimensão.

Demonstra-se [Pardalos e Rodgers (1990)] que problemas quadráticos do tipo:

$$\min f(x) = -(n-1)\sum_{i=1}^{n} x_i - \frac{1}{n}\sum_{i=1}^{n/2} x_i + 2\sum_{i<j} x_i x_j \qquad (4.16)$$
$$x \in \{0,1\}^n$$

têm um número exponencial de mínimos locais discretos e são difíceis para este algoritmo. No entanto, tal já não é verdade se outros algoritmos forem utilizados. Por exemplo, Körner (1992) apresenta uma abordagem baseada na função Lagrangeana e no seu dual, que resolve este tipo específico de problemas, no vértice inicial, sem necessitar de proceder à pesquisa da árvore. Também, Luz (1998) prova que o limite inferior que propõe, nesse trabalho, encontra a solução óptima para a família de problemas de (4.16).

O algoritmo de PA para o PQ 0-1 descrito em [Pardalos e Rodgers (1990)] também foi implementado neste trabalho. Referem-se, em seguida, os seus passos principais (algoritmo 4.3), considerando o PQ 0-1 na forma (4.12):

No algoritmo 4.3 sejam:

x^* - vector incumbente

opt - valor da solução incumbente

niv - nível da árvore de pesquisa

l_i - limite inferior do gradiente

L_i - limite superior do gradiente

$p_1, .., p_{niv}$ - índices das variáveis fixas até ao nível *niv*

$p_{niv+1}, .., p_n$ - índices das variáveis livres

g - valor do limite inferior da função objectivo

Passo 1:

Obter a partir de uma heurística a solução inicial (x^* e opt)

Fazer $niv = 0$

Calcular o intervalo do gradiente, para cada variável x_i, de acordo com:

$$l_{p_i} = 2\sum_{j=1}^{niv} a_{p_i p_j} x_{p_j} + 2 \sum_{\substack{j=niv+1 \\ j \neq i}}^{n} a^-_{p_i p_j} + a_{p_i p_i} \quad , \quad i = niv+1,\ldots,n$$

$$L_{p_i} = 2\sum_{j=1}^{niv} a_{p_i p_j} x_{p_j} + 2 \sum_{\substack{j=niv+1 \\ j \neq i}}^{n} a^+_{p_i p_j} + a_{p_i p_i} \quad , \quad i = niv+1,\ldots,n \tag{4.17}$$

Calcular o valor de g a partir de:

$$g = \sum_{i=1}^{n}\sum_{j=1}^{n} a^-_{ij} - \left(2\sum_{i=1}^{niv}\sum_{j=i+1}^{n} a^-_{p_i p_j}\left(1 - x_{p_i}\right) + \sum_{i=1}^{niv} a^-_{p_i p_i}\left(1 - x_{p_i}\right) \right) + \sum_{i=1}^{niv}\sum_{j=1}^{niv} a^+_{p_i p_j} x_{p_i p_j} \tag{4.18}$$

Passo 2:

Para $i = 1,\ldots, n$ fazer:

Se $l_{p_i} \geq 0$ então $x^*_{p_i} = 0$

Se $L_{p_i} \leq 0$ então $x^*_{p_i} = 1$ (4.19)

actualizar os valores de g , niv , $l_{p_{i+1}}$, $L_{p_{i+1}}$

Se $niv = -1$ (todas as variáveis estão fixas) ir para o Passo 4.

Passo 3

Enquanto a árvore não tiver sido toda pesquisada fazer:

Escolher como variável a ramificar em cada vértice, aquela a que corresponder o máximo dos valores de δ, com

$\delta_k = min\,(-l_k, L_k)$ para $k = p_{niv+1},\ldots,p_n$. (4.20)

Fixá-la a 0 ou 1 consoante o que aumentar menos o valor de g.

Actualizar os valores de g , niv , l_i e L_i para $i = p_{niv+1},\ldots,p_n$

Fixar as variáveis em cada vértice de acordo com (4.17).

Se $g < opt$ e $niv = n$

actualizar a solução incumbente ($opt = g$ e $x^* = x$)

Se $g \geq opt$

subir na árvore

Passo 4:

O algoritmo termina e a solução óptima é a solução incumbente.

Algoritmo 4.3 – Algoritmo de Pesquisa em Árvore para o PQ 0-1

4.4.2 ALGORITMOS HEURÍSTICOS

Os algoritmos heurísticos baseiam-se nas características e propriedades do problema e utilizam formas expeditas para obter uma boa solução. Em alguns casos obtém-se um óptimo local, ou mesmo global, mas com este tipo de algoritmos não se pode saber se a solução encontrada é o óptimo global. Na maioria dos casos não se pode mesmo avaliar a qualidade da solução obtida. Assim, a utilização de métodos heurísticos é aconselhável quando os problemas são de grande ou média dimensão, ou quando o factor tempo de execução é determinante para uma aplicação.

Como se expôs em 4.3.2, o intervalo do gradiente é muito importante para o PQ 0-1. É claro que, se podem desenvolver muitas heurísticas a partir do seu exame, baseando-se apenas num dos seus pontos (extremo, médio, etc.), em todo o intervalo, na sua amplitude, etc.

Uma heurística deste tipo, para o PQ 0-1, baseada no ponto médio do gradiente, é, por exemplo, utilizada em Pardalos e Rodgers (1992), para encontrar uma boa solução inicial para o algoritmo de pesquisa em árvore. Esta heurística, calcula o ponto médio do intervalo do gradiente no hipercubo unitário para cada variável x_i. A variável é então fixa a 0 ou a 1, consoante o ponto médio for positivo ou negativo, respectivamente.

Heurísticas do tipo ávido são frequentemente utilizadas em POCs. Em geral, nestes métodos, as variáveis vão sendo escolhidas segundo a ordem definida por um critério baseado no contributo de cada uma, independentemente das outras, para a função objectivo, de forma que se fixam primeiro as que mais contribuem. Pardalos e Rodgers (1992), utilizam um critério baseado nos valores dos extremos do gradiente, para escolher a variável a fixar e fixam-na a 0 ou a 1 consoante o que aumentar menos o valor da função objectivo.

Em Faustino (1992) é descrito um algoritmo heurístico para o PQ formalizado em (4.11). Note-se que para o caso do PQ 0-1, este método é equivalente ao apresentado em Gulati et al. (1984). Este algoritmo encontra uma solução, $\bar{x}$, que é um ponto extremo do conjunto Ω definido em (4.8) e é, em geral, um mínimo local do PQ

dado em (4.7). Pesquisa depois se existe algum outro ponto extremo de Ω, adjacente a $\bar{x}$, para o qual o valor da função objectivo seja menor. Assim, este método permite obter uma solução básica $\bar{x}$, tal que $f(\bar{x}) \leq f(x)$ para todo o ponto extremo $x \in \Omega$, adjacente de $\bar{x}$. Neste trabalho, utilizou-se, também, este algoritmo heurístico, tendo sido necessário fazer algumas adaptações para o tornar adequado à determinação de uma boa solução para o PQ 0-1. Assim, seguidamente (algoritmo 4.4), resume-se por Passos o algoritmo adaptado à formalização dada em (4.12)

Passo 1:

Fazer

$$vf = 0,\ F = \varnothing,\ F^c = \{1,...,n\},\ u = (u_1,...,u_n)^t = 0$$

Passo 2:

Para $i = 1,...,n$

Se $2u_i + a_{ii} < 0$ fazer

$vf = vf + 1$

Actualizar F e F^c a partir de

$$F = \begin{cases} F - \{i\} & \text{se } i \in F \\ F \cup \{i\} & \text{se } i \notin F \end{cases}$$

$$F^c = \{1,...,n\} - F$$

Actualizar u a partir de

$$u_i = \begin{cases} -\left(u_i + \sum_{\substack{j \in F \\ j \neq i}} a_{ij} \right) & \text{se } i \in F \\ u_i + \sum_{j \in F} a_{ij} & \text{se } i \in F^c \end{cases}$$

Passo 3:

Se $vf = 0$ terminar.

A solução é um mínimo local dado por

$$x_i = \begin{cases} 1 & \text{se } i \in F \\ 0 & \text{se } i \in F^c \end{cases}$$

Caso contrário fazer $vf = 0$ e voltar para o Passo 2

Algoritmo 4.4 – Heurística para o PQ 0-1

Em Luz (1998) é descrita uma heurística, para o PQ 0-1 na forma (4.1), que, em cada iteração, obtém uma solução do problema dual de um subproblema obtido do inicial, mas com menos variáveis.

Em Chardaire e Sutter (1995) é proposta uma heurística que recorre a um algoritmo de PA para calcular o valor mínimo de funções pseudo-bilineares que compõem a função quadrática.

Glover et al (1998) propõem uma heurística de Pesquisa Tabu com memória adaptável para o PQ 0-1, no caso de maximização da função quadrática.

4.5 DESCRIÇÃO DAS IMPLEMENTAÇÕES DE ALGORITMOS PARA O PQ 0-1

Nesta secção descrevem-se as implementações, sequenciais e paralelas, que foram desenvolvidas, de algoritmos de PA e de Algoritmos Genéticos para o PQ 0-1. Considerou-se a formalização dada em (4.12).

4.5.1 IMPLEMENTAÇÃO SEQUENCIAL DE UM ALGORITMO DE PA PARA O PQ 0-1

Implementou-se um algoritmo de PA baseado no de Pardalos e Rodgers (1990) (algoritmo 4.3). No Passo 1 utilizou-se o algoritmo 4.4 para obter heuristicamente uma solução inicial.

OUTRO LIMITE INFERIOR PARA A FUNÇÃO OBJECTIVO

Como já foi referido em 4.4.1, para que o número de vértices da árvore de pesquisa seja reduzido é necessário que o valor do limite inferior seja sempre o mais próximo possível do valor do incumbente. Para tornar o algoritmo de PA mais eficiente o cálculo desse limite deve ser computacionalmente pouco exigente.

Ora, o valor do limite inferior, g, da função objectivo utilizado em Pardalos e Rodgers (1990) dado em (4.18), em geral, só estará próximo do valor do incumbente, quando a maior parte das variáveis estiver fixa, o que corresponde aos vértices da árvore com nível perto de n. Inicialmente este limite corresponde à soma de todos os elementos negativos da matriz A. Vai sendo, depois, actualizado à medida que as variáveis vão sendo fixadas. Assim, o valor de g só se aproxima do incumbente quando grande parte das variáveis estiver fixa. Isto leva a que, na generalidade dos casos, os ramos da árvore sejam podados após muita pesquisa (em níveis próximos de n).

Optou-se, então, por calcular este limite, de uma outra forma mais eficiente, explorando o facto de que uma variável só deve ser fixa a 1 se contribuir negativamente para a função objectivo, como também foi feito em Faustino (1992) para o PQ de (4.7). No entanto, o cálculo e actualização do limite inferior da função objectivo, utilizada em Faustino (1992), é algo trabalhoso, e por isso [Faustino (1998)] a sua actualização foi aí feita apenas algumas vezes durante a execução da PA. A forma como se procedeu, neste trabalho, para calcular e actualizar esse limite inferior, que se vai descrever nesta secção, permitiu actualizá-lo todas as vezes que se fixa uma variável o que contribuiu para que alguns ramos possam ser podados mais cedo tornando o algoritmo de PA mais eficiente.

Neste trabalho definiu-se, para o PQ 0-1, o limite inferior da função f dada em (4.12) da seguinte forma:

$$g_1(x) = \sum_{i=1}^{niv} \left(\sum_{\substack{j=1 \\ j\neq i}}^{niv} a_{p_i p_j} x_{p_j} + \sum_{j=niv+1}^{n} a^{-}_{p_i p_j} + a_{p_i p_i} \right) x_i + \sum_{i=niv+1}^{n} \left(\sum_{j=1}^{niv} a_{p_i p_j} x_{p_j} + \sum_{\substack{j=niv+1 \\ j\neq i}}^{n} a^{-}_{p_i p_j} + a_{p_i p_i} \right)^{-} \tag{4.21}$$

onde (como no algoritmo 4.3):

niv indica o nível da árvore de pesquisa;

$p_1, .., p_{niv}$ são os índices das variáveis fixas até ao nível niv

$p_{niv+1}, .., p_n$ são os índices das variáveis livres

Teorema 4.2: Considerando as funções g_1 de (4.21) e f de (4.12) tem-se $g_1(x) \leq f(x)$, $\forall x \in \{0,1\}^n$.

Demonstração:

Tem-se que:

$$f(x) = x^{\mathrm{T}} A x = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}\, x_i x_j$$

$$= \sum_{i=1}^{n} \left(\sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij}\, x_j + a_{ii} \right) x_i$$

Sem perda de generalidade pode-se considerar, para simplificação da escrita, que as variáveis fixas, até ao nível *niv*, foram as *niv* primeiras. Então $g_1(x)$ pode ser escrito como:

$$g_1(x) = \sum_{i=1}^{niv} \left(\sum_{\substack{j=1 \\ j \neq i}}^{niv} a_{ij} x_j + \sum_{j=niv+1}^{n} a_{ij}^{-} + a_{ii} \right) x_i \qquad (4.22)$$

$$+ \sum_{i=niv+1}^{n} \left(\sum_{j=1}^{niv} a_{ij} x_j + \sum_{\substack{j=niv+1 \\ j \neq i}}^{n} a_{ij}^{-} + a_{ii} \right)^{-} \qquad (4.23)$$

Do mesmo modo,

$$f(x) = \sum_{i=1}^{niv} \left(\sum_{\substack{j=1 \\ j \neq i}}^{niv} a_{ij} x_j + \sum_{j=niv+1}^{n} a_{ij} x_j + a_{ii} \right) x_i \qquad (4.24)$$

$$+ \sum_{i=niv+1}^{n} \left(\sum_{j=1}^{niv} a_{ij} x_j + \sum_{\substack{j=niv+1 \\ j \neq i}}^{n} a_{ij} x_j + a_{ii} \right) x_i \qquad (4.25)$$

Se $x_i = 0$ então a parcela correspondente de (4.24) e de (4.25) fica igual a 0. A parcela correspondente de (4.22) fica também igual a 0 , mas a de (4.23) fica menor ou igual a 0

Se $x_i = 1$ as parcelas correspondentes de (4.24) ou de (4.25) ficam iguais a

$$\sum_{\substack{j=1 \\ j\neq i}}^{niv} a_{ij}\, x_j + \sum_{\substack{j=niv+1 \\ j\neq i}}^{n} a_{ij}\, x_j + a_{ii} \tag{4.26}$$

Como $a_{ij}^{-} = min\{0, a_{ij}\} \leq a_{ij}\, x_j$, para $x_j = 0$ ou 1, tem-se que as parcelas correspondentes de (4.22) e de (4.23) são menores ou iguais que (4.26), logo vem $g_1(x) \leq f(x)$, $\forall x \in \{0,1\}^n$.

Os cálculos de $g_1(x)$ podem ser muito simplificados tendo em conta que o limite inferior do gradiente, utilizado no algoritmo 4.3 para fixar as variáveis (equações (4.17)), pode ser calculado através de,

$$li_{p_i} = \begin{cases} \left(\sum_{\substack{j=1 \\ j\neq i}}^{niv} a_{p_i p_j} x_{p_j} + \sum_{j=niv+1}^{n} a_{p_i p_j}^{-} + \frac{1}{2} a_{p_i p_i} \right) x_{p_i} & \text{para } i = 1,\ldots,niv \\ \sum_{j=1}^{niv} a_{p_i p_j} x_{p_j} + \sum_{\substack{j=niv+1 \\ j\neq i}}^{n} a_{p_i p_j}^{-} + \frac{1}{2} a_{p_i p_i} & \text{para } i = niv+1,\ldots,n \end{cases} \tag{4.27}$$

Desta forma tem-se (4.21) igual a

$$g_1 = \sum_{i=1}^{niv} \left(li_{p_i} + \frac{1}{2} a_{p_i p_i} x_{p_i} \right) + \sum_{i=niv+1}^{n} \left(li_{p_i} + \frac{1}{2} a_{p_i p_i} \right)^{-} \tag{4.28}$$

Nesta implementação manteve-se sempre o limite inferior (*li*) do gradiente, dado em (4.27), actualizado para todas as variáveis, e com pouco esforço computacional, actualizou-se o valor do limite inferior (g_1) da função objectivo em todas as iterações, utilizando (4.28).

Teorema 4.3: Considerando as funções g_1 de (4.21) e g de (4.18) tem-se $g(x) \leq g_1(x)$, $\forall x \in \{0,1\}^n$.

Demonstração:

Como na demonstração anterior considere-se que $p_j = j$, $j = 1,...,n$.

Sejam:

$$(1) = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^{-}$$

$$(2) = 2 \sum_{i=1}^{niv} \sum_{j=i+1}^{n} a_{ij}^{-} (1 - x_i)$$

$$(3) = \sum_{i=1}^{niv} a_{ii}^{-} (1 - x_i)$$

$$(4) = \sum_{i=1}^{niv} \sum_{j=1}^{niv} a_{ij}^{+} x_i x_j$$

Então,

$$g = (1) - (2) - (3) + (4)$$

Inicialmente ($niv = 0$) g é igual a (1), ou seja, é igual à soma de todos os elementos negativos da matriz A. A subtracção de (2) e (3) e a adição de (4) provocam a actualização de g para as variáveis que vão sendo fixas a 0 ou a 1. Assim, a subtracção de (2) elimina de (1) os coeficientes negativos dos termos quadráticos que tenham pelo menos uma variável fixa a 0; a subtracção de (3) elimina de (1) os coeficientes dos termos lineares cujas variáveis tenham sido fixas a 0; a adição de (4) soma a g os coeficientes positivos, dos termos lineares das variáveis fixas a 1 e dos termos quadráticos cujas duas variáveis foram fixas a 1.

Então pode-se escrever:

$$g = \sum_{i=1}^{niv}\left(\sum_{j=1}^{niv} a_{ij}x_i x_j + \sum_{j=niv+1}^{n} a_{ij}^{-}x_i\right) \tag{4.29}$$

$$+ \sum_{i=niv+1}^{n}\left(\sum_{j=1}^{niv} a_{ij}^{-}x_j + \sum_{\substack{j=niv+1\\ j\neq i}}^{n} a_{ij}^{-} + a_{ii}^{-}\right) \tag{4.30}$$

$$g_1 = \sum_{i=1}^{niv}\left(\sum_{\substack{j=1\\ j\neq i}}^{niv} a_{ij}x_j + \sum_{j=niv+1}^{n} a_{ij}^{-} + a_{ii}\right)x_i \tag{4.31}$$

$$+ \sum_{i=niv+1}^{n}\left(\sum_{j=1}^{niv} a_{ij}x_j + \sum_{\substack{j=niv+1\\ j\neq i}}^{n} a_{ij}^{-} + a_{ii}\right)^{-} \tag{4.32}$$

As expressões dadas em (4.29) e em (4.31) são iguais e tem-se (4.30) ≤ (4.32), visto que,

$$\sum_{j=1}^{niv} a_{ij}^{-}x_j \leq \sum_{j=1}^{niv} a_{ij}x_j \quad ; \quad a_{ii}^{-} \leq a_{ii}$$

e como as parcelas de (4.30) são sempre negativas então (4.30) é menor ou igual a 0.

Então $g \leq g_1$ como queríamos provar.

Desta forma mostrou-se que o limite inferior, g_1, é melhor que o limite g proposto por Pardalos e Rodgers (1990).

Se no algoritmo 4.1, a função g for calculada a partir de (4.28) em vez de (4.18) e as equações (4.17) forem substituídas por (4.27) e por:

$$ls_{p_i} = \begin{cases} \left(\sum_{\substack{j=1\\ j\neq i}}^{niv} a_{p_i p_j}x_{p_j} + \sum_{j=niv+1}^{n} a_{p_i p_j}^{+} + \frac{1}{2}a_{p_i p_i}\right)x_{p_i} & \text{para } i = 1,\ldots,niv \\ \sum_{j=1}^{niv} a_{p_i p_j}x_{p_j} + \sum_{\substack{j=niv+1\\ j\neq i}}^{n} a_{p_i p_j}^{+} + \frac{1}{2}a_{p_i p_i} & \text{para } i = niv+1,\ldots,n \end{cases} \tag{4.33}$$

obtém-se um algoritmo de PA para o PQ 0-1, que deverá ser mais eficiente.

A actualização destes limites, quando se fixa uma variável, é muito simples, pois consiste em adicionar ou subtrair um elemento da matriz da forma quadrática. Assim, tem-se:

$$\text{Se } x_{p_{niv}} = 1 \Rightarrow \begin{cases} li_{p_j} = li_{p_j} + a^{+}_{p_{niv}p_j} x_{p_j} & , \quad j = 1,\ldots,niv-1 \\ li_{p_j} = li_{p_j} + a^{+}_{p_{niv}p_j} & , \quad j = niv+1,\ldots,n \\ ls_{p_j} = ls_{p_j} + a^{-}_{p_{niv}p_j} & , \quad j = niv+1,\ldots,n \end{cases} \tag{4.34}$$

$$\text{Se } x_{p_{niv}} = 0 \Rightarrow \begin{cases} li_{p_j} = li_{p_j} - a^{-}_{p_{niv}p_j} x_{p_j} & , \quad j = 1,\ldots,niv-1 \\ li_{p_j} = li_{p_j} - a^{-}_{p_{niv}p_j} & , \quad j = niv+1,\ldots,n \\ ls_{p_j} = ls_{p_j} - a^{+}_{p_{niv}p_j} & , \quad j = niv+1,\ldots,n \end{cases} \tag{4.35}$$

Depois de actualizar os limites a partir de (4.34) ou (4.35) o valor de g_1 é facilmente actualizado recorrendo a (4.28).

4.5.2 Implementação Paralela de um Algoritmo de PA para o PQ 0-1

Adaptou-se o algoritmo sequencial de PA, na versão que se acabou de descrever, à execução em máquinas paralelas com memória distribuída, com base no exposto em 4.4.1 e no algoritmo 4.2.

O problema inicial e a solução heurística são afectos ao processador zero, o qual calcula os intervalos do gradiente, de acordo com (4.27) e (4.33), e fixa definitivamente todas as variáveis possíveis, conforme o exposto no corolário 4.1. Em seguida, se o problema não tiver ficado resolvido no vértice inicial, utiliza-se o mesmo critério (4.20) do algoritmo sequencial para escolher uma variável a partir da qual se ramifica a árvore. Assim, essa variável fixa-se a δ, com $\delta = 0$ ou $\delta = 1$. Para cada valor de δ obtém-se um subproblema. Um deles fica afecto ao processador zero (um ramo da árvore) e o

outro (o outro ramo), é por ele enviado para um processador adjacente, que passa ao estado activo.

Repete-se este processo de fixar variáveis e ramificar, de forma a que os processadores livres recebam subproblemas e, juntamente com os processadores que os enviaram, criem novos subproblemas e os enviem para os processadores livres que lhes estão adjacentes, até todos os processadores estarem no estado activo.

Quando esta operação de divisão da árvore de pesquisa pelos processadores terminar, todos os processadores estão no estado activo. Nesse momento, começam paralelamente a pesquisar o ramo da árvore que lhes pertence. Cada processador executa a pesquisa do seu ramo (Passos 2 e 3 do algoritmo 4.3), até que ocorra uma das duas situações seguintes (Passo 3 do algoritmo 4.2):

(i) atinja um número especificado (*maxn*) de subproblemas a resolver, mantendo-se o processador no estado ocupado;

(ii) o seu ramo fique completamente pesquisado, passando o processador ao estado livre.

Ao atingirem qualquer uma destas situações comunicam com os processadores adjacentes, trocando os respectivos estados, os incumbentes e actualizando a solução incumbente e respectivo valor da f.o. se for caso disso (Passo 5 do algoritmo 4.2). Se um estiver livre e o outro ocupado este repete o processo de ramificação enviando um novo subproblema para o processador livre (Passos 6 e 7 do algoritmo 4.2).

O algoritmo termina quando todos os processadores estiverem livres, o que significa que a pesquisa da árvore está completa, como é demonstrado em Pardalos (1989).

4.5.2.1 TOPOLOGIA UTILIZADA

O algoritmo paralelo de PA foi executado utilizando 2 , 4 e 8 transputers além do transputer *raiz* ("root"), que é utilizado apenas na comunicação com o PC.

As topologias utilizadas para 4 e para 8 transputers (no caso de 2 transputers foi feita, obviamente, a ligação de um ao outro), estão esquematizadas, respectivamente nas figuras 4.1 e 4.2 .

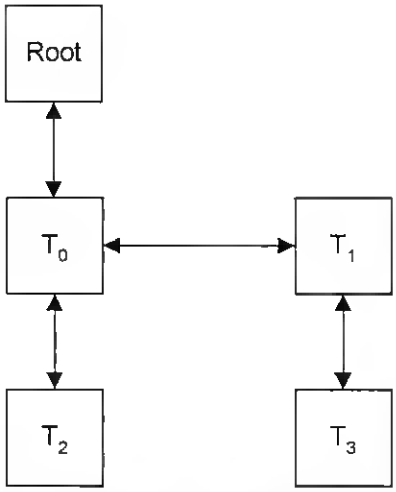


Figura 4.1 - Topologia para uma rede de 4 transputers

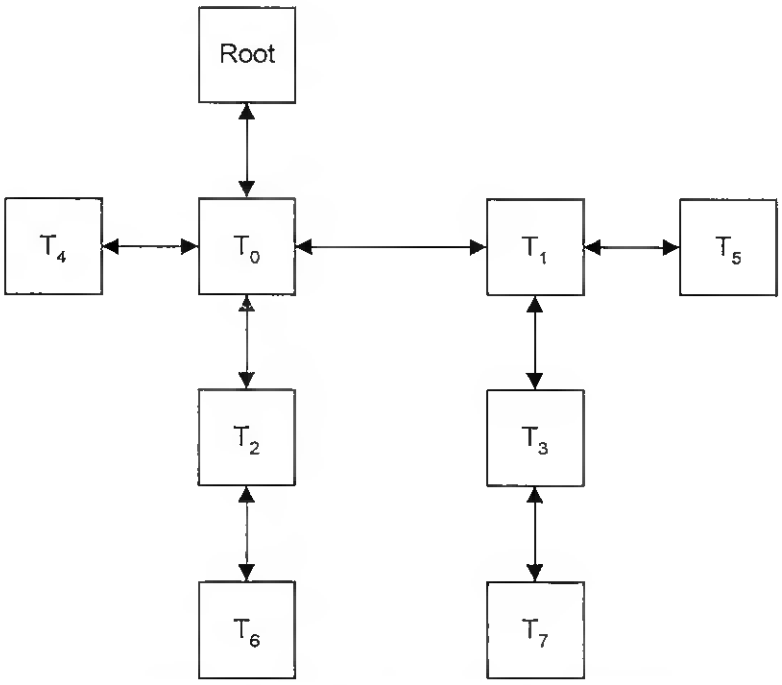


Figura 4.2 - Topologia para uma rede de 8 transputers

Como já foi exposto, a divisão inicial do problema é feita escolhendo uma variável para ramificar, de acordo com o critério (4.20), ficando num ramo essa variável

com o valor δ e no outro com o valor $1-\delta$ (para $\delta \in \{0,1\}$). As figuras 4.3 e 4.4 ilustram esquematicamente a distribuição inicial dos ramos da árvore de pesquisa pelos processadores (transputers).

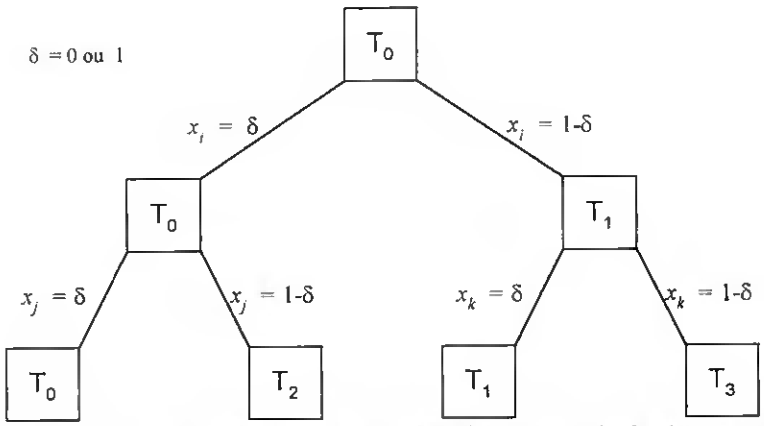


Figura 4.3 – Distribuição da árvore de pesquisa numa rede de 4 transputers

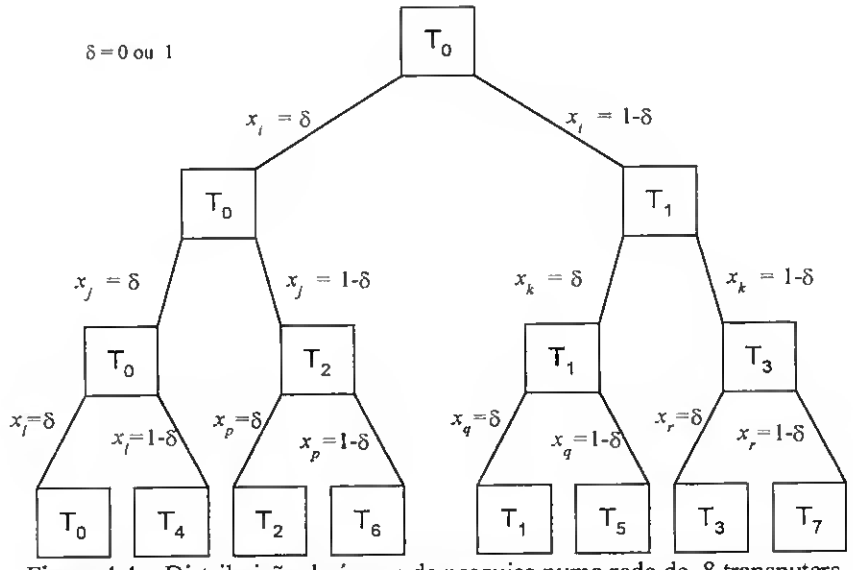


Figura 4.4 – Distribuição da árvore de pesquisa numa rede de 8 transputers

As comunicações entre processadores, nas topologias utilizadas neste trabalho, são sempre feitas só entre processadores adjacentes. Por exemplo, no caso de 8 processadores, T_0 comunica com T_1, T_2 e T_4. O T_2 troca primeiro informações com o T_6, actualiza os seus dados se for caso disso e só depois comunica com o T_0, assim T_0 só

recebe indirectamente e implicitamente alguma informação de T_6. Da mesma forma, quando T_1 comunica com T_0, envia informações actualizadas com as já recebidas de T_3, T_5 e T_7 (via T_3) e recebe-as de T_0 também já influenciadas por T_2, T_4 e T_6.

Utilizando esta topologia e esta forma de dividir a árvore pelos processadores consegue-se que esta seja pesquisada simultaneamente em profundidade e em largura.

4.5.2.2 IMPLEMENTAÇÃO COM "ROUTER"

Além da implementação anterior, fez-se também uma implementação do algoritmo paralelo, utilizando as topologias descritas, mas tirando partido das características dos transputers, que permitem que um transputer execute simultaneamente mais do que um processo.

Do ponto de vista da pesquisa em árvore o ideal seria que o melhor incumbente fosse conhecido de todos os processadores sempre que um deles o actualizasse (Passo 5 do algoritmo 4.2), ou seja, seria conveniente que a variável incumbente fosse uma variável global. Para transformar uma variável local de cada processador numa global acessível a todos, num sistema com memória distribuída, teria que se atribuir a *maxn* o valor 1 na implementação paralela. Mas, do ponto de vista das comunicações entre processadores, isto seria desaconselhável, pois o que se pouparia no tamanho da árvore seria largamente suplantado pelas comunicações entre processadores.

Para se conseguir que o melhor incumbente fosse enviado a todos os processadores o mais cedo possível, criaram-se processos virtuais e executaram-se dois processos em cada transputer. Um, o "router", encarregado de estabelecer as comunicações (enviar e receber mensagens), o outro, o "worker", destinado a executar a pesquisa em árvore. Assim, enquanto o "router" comunicava, o "worker" podia continuar a processar (ver figura 4.5).

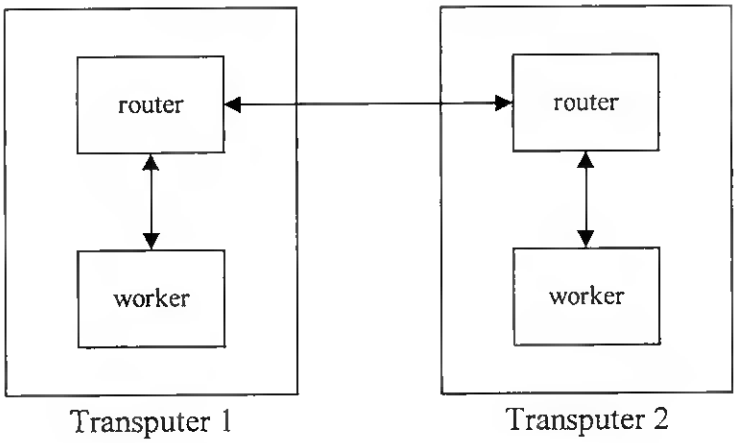


Figura 4.5 – implementação com "router"

Esta forma de utilização dos transputers, tem sido eficiente [Ruano (1992)] nos algoritmos em que o tipo de dados a comunicar é sempre o mesmo (por exemplo uma coluna de uma matriz) e em que a própria topologia adoptada (por exemplo a rede BUS) obriga a uma ordem nas comunicações entre processos, de forma a que cada um sabe sempre de quem vai receber os dados ou para quem vai enviar e que tipo de informação vai receber ou enviar.

No entanto, neste caso, os dados a comunicar não são sempre os mesmos, distinguindo-se três situações diferentes:

- estado do processador (livre ou activo);
- solução incumbente;
- subproblema.

Cada processador pode enviar (receber) estes dados para (de) qualquer outro que lhe seja adjacente. Resumindo, por um lado, o processo de enviar ou receber e o tipo de informação a comunicar é dinâmico e não está pré-estabelecido. Por outro lado a informação a comunicar entre "routers" e entre "router" e "worker" é em geral diferente.

Foi, por isso, necessário criar uma ordem na comunicação entre os "routers" para que não surgissem conflitos (simultaneamente dois "routers" enviando dados um

para o outro, ou esperando receber), isto é, durante um certo número de iterações um dos "routers" só recebia mensagens e o outro só enviava, e iam alternando o seu papel de receptor e de mensageiro.

4.5.2.2 ALGORITMO ε-APROXIMADO

O algoritmo paralelo de PA, foi também utilizado para obter uma solução ε-aproximada. O critério para podar um ramo da árvore passou a ser:

$$g \geq (g^* - \varepsilon)$$

onde:

g é o valor da função limite inferior da função objectivo, calculado como em (4.28),

g^* é o valor do incumbente,

e, para que o ε nunca fosse superior a 5% da melhor solução conhecida e diminuísse à medida que se aprofunda a pesquisa da árvore, definiu-se

$$\varepsilon = min \left\{ \left| g^* - g^*_{ant} \right|, 0.05 \left| g^* \right| \right\}$$

onde g^*_{ant} é a solução incumbente anterior.

Assim, sempre que se obtém um novo incumbente o ε é actualizado.

4.5.3 ALGORITMO GENÉTICO SEQUENCIAL PARA O PROBLEMA PQ 0 - 1

Como o PQ 0-1 é um problema de variáveis binárias, sem restrições, enquadra-se directamente no tipo de problemas que têm sido resolvidos com sucesso através de

Algoritmos Genéticos. Apesar disso, não se conhecem, da literatura, abordagens genéticas para este problema. Nesta secção, expõem-se as formas como se aplicaram os operadores genéticos a este problema, que deram origem a cinco versões de um algoritmo genético.

4.5.3.1 CODIFICAÇÃO

Tirando partido do PQ 0-1 ser um problema com variáveis binárias, codificou-se, como é usual, cada indivíduo x^i, $i = 1,...,Pop$, por um cromossoma com l genes $x_j^i, j = 1,...,l$, onde l é o número de variáveis livres do problema, e cada gene $x_j^i \in \{0,1\}$.

Neste caso, determinaram-se inicialmente os valores dos limites dos gradientes das variáveis e fixaram-se definitivamente a 0 ou a 1 as variáveis cujos dois limites tinham o mesmo sinal, de uma forma análoga ao que foi feito nos Passo 1 e 2 do algoritmo 4.3 descrito em 4.4.1 (de acordo com (4.28), (4.27) e (4.33) fazendo *niv* = 0). Assim, cada indivíduo é composto apenas pelas variáveis livres. Adicionalmente tem-se um vector com o valor das variáveis fixas.

Por exemplo, para um problema com 10 variáveis, $x_1, x_2, ..., x_{10}$ em que inicialmente x_2, x_3, x_7 e x_9 ficam definitivamente fixas, trabalha-se com os vectores:

Variáveis fixas	[1	0	0	1]						
	↑									
Índices das variáveis	[2	3	7	9	1	4	5	6	8	10]
					↓					
Indivíduo *i*					[0	0	1	1	1	0]

O valor da função objectivo, para o indivíduo x^i, que se representa por $f(x^i)$ é calculado através da expressão:

$$f(x^i) = \sum_{j=1}^{n} a_{jj} x_j^i + 2 \sum_{j=1}^{n} \sum_{k=j+1}^{n} a_{jk} x_j^i x_k^i \quad , i = 1,...,Pop \tag{4.36}$$

4.5.3.2 POPULAÇÃO INICIAL

A população inicial é gerada aleatoriamente, como habitualmente é feito nos AGs. No entanto, optou-se por gerar, aproximadamente, um terço dos indivíduos com 30% de zeros, um terço com 50% de zeros e o outro terço com 70 % de zeros, com o intuito de obter logo à partida uma população mais diversificada.

Testou-se também a introdução de um indivíduo correspondente a uma solução heurística (supostamente melhor) na população inicial, com o intuito de, por um lado estudar a influência de uma solução não aleatória na qualidade da solução óptima e, por outro, verificar se havia uma convergência das soluções aleatórias para a solução não aleatória. Assim, a introdução deste indivíduo foi feita de duas formas:

a) na população inicial;

b) após terem sido executadas, aproximadamente, dois terços do número de gerações previstas.

Obteve-se esta solução heurística recorrendo ao algoritmo 4.4 descrito em 4.4.2 e já utilizado para obter um majorante no algoritmo de pesquisa em árvore.

4.5.3.3 FUNÇÃO DE APTIDÃO E SELECÇÃO

De acordo com o exposto, em 3.2 e 3.3, sobre as formas de seleccionar a população, optou-se por utilizar a selecção SUS por ser referida como a que tem enviesamento nulo e menor amplitude.

Ordenaram-se os indivíduos, por ordem decrescente do valor da função objectivo. Atribuiu-se uma aptidão 0 ao pior indivíduo (maior valor) e uma aptidão 2 ao melhor. Aos outros indivíduos atribuiram-se aptidões entre 0 e 2 de acordo com a sua classificação e igualmente espaçadas, isto é, fez-se $s = 2$ na expressão da função de avaliação dada em (3.2). Tem-se, então, a função de aptidão dada por:

$$Apt\left(x^i\right) = 2 - \frac{2\ perm(i)}{Pop-1}\ ,\ i = 1,...,Pop \tag{4.37}$$

onde *perm* (*i*) é o vector que indica a classificação (de 0 a *Pop*-1) do indivíduo *i* na população.

Construiu-se a roleta de acordo com:

$$\begin{aligned} &roleta\ (0) = 0\ , \\ &roleta\ (i) = roleta\ (i-1) + Apt\left(x^i\right)\ \ ,\ i = 1,...,Pop\ . \end{aligned} \tag{4.38}$$

Gerou-se um número aleatório uniformemente distribuído no intervalo [0 , 1] e formou-se um ponteiro (com a dimensão da população) adicionando sucessivamente uma unidade.

De acordo com o exposto em 3.3.7, se existirem vários indivíduos com o mesmo valor da função objectivo, um deles tem a aptidão correspondente (de acordo com (4.37)) e os outros são sucessivamente penalizados em 0,5 até atingirem, no mínimo, uma aptidão 0.

Por exemplo, para uma população de 10 indivíduos com a seguinte função de aptidão

Indiv.	1	2	3	4	5	6	7	8	9	10
Apt	1,78	0,89	1,33	0,22	2,00	0,44	0,00	1,56	1,11	0,67
roleta	1,78	2,67	4,00	4,22	6,22	6,66	6,66	8,22	9,33	10,00

Tabela 4.1 – Exemplo para uma população de 10 indivíduos

Supondo que 0.4 era o número gerado aleatoriamente no intervalo [0 , 1], obtinha-se o ponteiro:

pont.	0,4	1,4	2,4	3,4	4,4	5,4	6,4	7,4	8,4	9,4

que leva à selecção dos indivíduos:

selec.	1	1	2	3	5	5	6	8	9	10

Na figura 4.6 representa-se a roleta e o ponteiro.

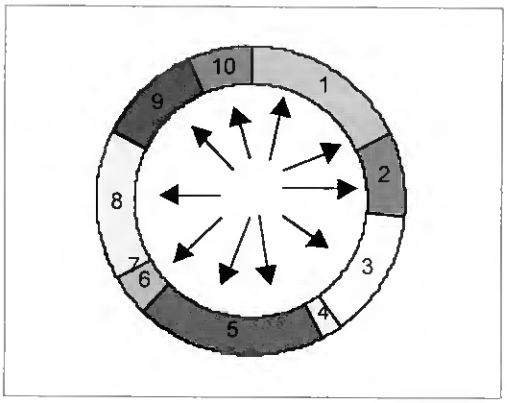


Figura 4.6 – Roleta e ponteiro

4.5.3.4 CRUZAMENTO

Em geral é conveniente que, após o cruzamento, os filhos tenham sempre os seus genes herdados (locus e alelos iguais) de pelo menos um dos pais.

Testaram-se os três tipos de cruzamento mais usuais (1-ponto, 2-pontos e uniforme) e duas formas para escolher os indivíduos que fariam parte da nova população (referidas em 3.3.6):

a) os dois indivíduos que têm melhor valor da função objectivo (dos quatro envolvidos, pais e filhos). Desta forma a qualidade global da população nunca piora e a nova população é, em geral, formada por alguns indivíduos da anterior (pais) e por alguns novos indivíduos (filhos);

b) os dois filhos independentemente dos valores da aptidão (dos pais e dos filhos). Assim a nova população é formada pelos filhos e a solução final é a melhor encontrada durante todas as gerações, podendo não pertencer à última geração.

4.5.3.5 MUTAÇÃO

Como foi referido em 3.3.5 aplicou-se o operador mutação a todos os indivíduos e cada gene foi alterado (de 1 para 0 ou vice-versa) com uma probabilidade igual a $0.7/l$, onde l é o número de variáveis livres do problema.

Fixou-se este valor em vez de $1/l$ porque se aplica, em cada geração, o operador cruzamento que contribui, também, para produzir alterações nos indivíduos (de acordo com o exposto em 3.3.5).

Após esta operação foram também testadas as duas alternativas, mencionadas no cruzamento, em relação aos indivíduos que fariam parte da nova população.

4.5.3.6 PÓS-OPTIMIZAÇÃO

Utilizou-se um procedimento de pós-optimização como um outro tipo (não aleatório) de mutação. Este procedimento foi aplicado de duas formas:

a) a cada indivíduo que após o cruzamento ou a mutação tivesse sido alterado;

b) a todos os indivíduos substituindo o anterior operador mutação.

A pós-optimização a um indivíduo consistiu em:

(i) Calcular para cada gene (variável), x_j^i , $j = 1,...,l$, do indivíduo x^i, $i = 1,...,Pop$, o valor da sua contribuição para a função objectivo (s_j) supondo-o fixo a 1, isto é,

$$s_j = \sum_{\substack{k=1 \\ k \neq j}}^{l} a_{jk} x_k^i + a_{jj} \ , \ j = 1,...,l \ ; \qquad (4.39)$$

(ii) Ordenar, em termos absolutos, os valores de s_j por ordem decrescente;

(iii) Para cada gene x_j^i , $j = 1,...,l$ (pela ordem obtida) fazer:

$$\text{se } s_j > 0 \wedge x_j^i = 1$$

$$\text{então fazer } x_j^i = 0 \text{ se melhorar o valor de } f(x^i) \tag{4.40}$$

$$\text{se } s_j < 0 \wedge x_j^i = 0$$

$$\text{então fazer } x_j^i = 1 \text{ se melhorar o valor de } f(x^i)$$

Por exemplo, considere-se o PQ 0-1 dado em (4.12) com a seguinte matriz:

$$A = \begin{bmatrix} 1 & 1 & -1 & 1 \\ 1 & -1 & 0 & -1 \\ -1 & 0 & -1 & 0 \\ 1 & -1 & 0 & 0 \end{bmatrix}$$

Como foi exposto, o AG começa por fixar definitivamente as variáveis que verificam (4.19), calculando os limites a partir de (4.27) e (4.33). Neste exemplo tem-se:

$$li_1 = -1 \text{ e } ls_1 = 5 \quad ; \quad li_2 = -3 \text{ e } ls_2 = 1$$
$$li_3 = -3 \text{ e } ls_3 = -1 \; ; \quad li_4 = -2 \text{ e } ls_4 = 2$$

logo, a variável x_3 fica definitivamente fixa a 1 ($x_3 = 1$). Assim, cada indivíduo é do tipo $[x_1 \quad x_2 \quad x_4]$. Considere-se o indivíduo x^i dado por [1 0 1] ao qual corresponde a solução $x^i = (1,0,1,1)$ do PQ 0-1 cujo valor da função objectivo é igual a zero ($f(x^i) = 0$).

Aplicando o operador pós-optimização a este indivíduo vêm, já ordenados:

$$s_1 = 2 \text{ , } s_2 = -1 \text{ e } s_4 = 1 \text{ ,}$$

então como

$$s_1 > 0 \text{ , } x_1^i = 1 \text{ e } f(0,0,1,1) = -1 \Rightarrow x_1^i = 0$$

$$s_2 < 0 \text{ , } x_2^i = 0 \text{ e } f(0,1,1,1) = -4 \Rightarrow x_2^i = 1$$

$$s_4 > 0 \text{ , } x_4^i = 1 \text{ e } f(0,1,1,0) = -2 \Rightarrow x_4^i = 1$$

Desta forma, mudam-se dois genes do indivíduo que passa a ser [0 1 1] correspondente à solução $x^i = (0,1,1,1)$ com $f(x^i) = -4$.

Neste procedimento vão-se alterando os valores das variáveis de δ para 1-δ, com $\delta \in \{0,1\}$, sequencialmente, de acordo com o maior acréscimo que trazem ao valor da função objectivo e desde que essa alteração melhore o valor da solução.

O AG com este procedimento de pós-optimização passa da aleatoriedade do operador mutação geralmente utilizado a uma forma de alteração das variáveis sistemática e baseada no problema. Neste sentido fica mais semelhante a outras meta-heurísticas que utilizam informação sobre o problema na pesquisa da vizinhança de uma solução, nomeadamente a PT.

De facto, a passagem de uma solução a outra neste procedimento pode produzir o mesmo resultado que os procedimentos do algoritmo baseado em PT descrito em Glover et al (1998).

O algoritmo de Glover et al (1998) parte de uma solução inicial qualquer que sofre alterações provocadas pela actuação de dois procedimentos que vão alternando. Num as variáveis que estão a 0 passam a 1 (fase construtiva) no outro as que estão a 1 passam a 0 (fase destrutiva). Quais e quantas variáveis mudam de valor depende de listas tabu, de limites máximos no número de alterações em cada fase e da inclusão de penalidades na avaliação de cada alteração [Glover et al (1998)].

As duas fases do algoritmo de PT aplicadas à solução correspondente ao indivíduo $x^i = (1,0,1,1)$ do exemplo anterior, poderiam originar, supondo que no máximo são feitas duas alterações em cada fase, as seguintes soluções e respectivos valores da função objectivo:

$$x = (1,1,1,1), \quad f(x) = -1 \text{ (fase construtiva)}$$

$$x = (0,1,1,1), \quad f(x) = -4 \text{ (fase destrutiva)}$$

$$x = (0,1,1,0), \quad f(x) = -2 \text{ (fase destrutiva).}$$

O AG, aqui descrito, tem a vantagem de trabalhar com um conjunto de soluções às quais aplica o operador de pós-optimização, o que pode aumentar a probabilidade de encontrar a solução óptima e de a encontrar mais rapidamente.

O algoritmo de Glover et al (1998) tem a vantagem de basear a pesquisa de novas soluções em informações (listas tabu) armazenadas na memória sobre soluções obtidas nas iterações anteriores (memória de curto prazo) e também em todas as soluções já obtidas (memória de longo prazo), o que pode contribuir para escapar dos óptimos locais.

De acordo com a descrição da forma de aplicar os operadores genéticos, feita nesta secção, implementaram-se cinco versões, ligeiramente diferentes umas das outras, deste AG. São as seguintes:

AG1 - população inicial aleatória, cruzamento e mutação;

AG2 - população inicial aleatória, cruzamento, mutação e pós-optimização a);

AG3 - população inicial aleatória, cruzamento, mutação e inserção de um indivíduo obtido heuristicamente;

AG4 - população inicial aleatória, cruzamento, mutação, inserção de um indivíduo obtido heuristicamente e pós-optimização a);

AG5 - população inicial aleatória, cruzamento, inserção de um indivíduo obtido heuristicamente e pós-optimização b).

Podem-se resumir estas versões do AG pelos Passos do algoritmo 4.5, onde:

geração = nº da geração corrente

Maxger = nº máximo de gerações a executar

x^* = solução (indivíduo) incumbente

sol^* = valor da f.o. para x^*

Pop = dimensão da população

l = nº de variáveis livres

niv = nº de variáveis fixas

Passo 1:

Aplicar os Passo 1 e Passo 2 do algoritmo 4.3, substituindo (4.17) por (4.27) e (4.33) e (4.18) por (4.28).

Passo 2:

Gerar aleatoriamente $Pop \times l$ números aleatórios uniformemente distribuídos em [0,1] obtendo Pop indivíduos $x^1, ..., x^{Pop}$;
Calcular $f(x^i)$ de acordo com (4.36);
Fazer

$$sol^* = \min_i f(x^i) \text{ e}$$

$$x^* = \{x^i : f(x^i) = sol^*\} ;$$

$geração = 0$.

Passo 3:

Escolher os indivíduos a cruzar de acordo com (4.37) e (4.38).
Aplicar-lhes o operador cruzamento;
Aplicar pós-optimização a) (AG2 e AG4) .

Passo 4:

Aplicar o operador mutação aleatório (AG1, AG2, AG3, AG4);
Aplicar pós-optimização a) (AG2 e AG4) ;
Aplicar pós-optimização b) (AG5) ;
$geração = geração + 1$.

Passo 5:

Se $geração < Maxger$
Actualizar $f(x^i)$, $i = 1,...,Pop$, sol^* e x^*;
Se $geração = \alpha \times Maxger$ $(\alpha \in [0,1[)$
substituir pior indivíduo por indivíduo obtido heuristicamente (AG3, AG4, AG5);
Voltar ao Passo2.
Se $geração = Maxger$
Terminar o algoritmo obtendo a solução igual a sol^* e o vector solução igual a x^*.

Algoritmo 4.5 – Algoritmo Genético para o PQ 0-1

4.5.4 ALGORIMO GENÉTICO PARALELO PARA O PQ 0-1

A versão sequencial do AG que obteve nos testes computacionais melhores resultados foi paralelizada utilizando 2, 4 e 8 transputers. Optou-se pelo modelo de migração, descrito em 3.4.2, por ser o mais adequado aos transputers, visto que, o

modelo global é apropriado para máquinas paralelas com memória partilhada e com o modelo de difusão obter-se-ia uma população muito reduzida por se dispor de poucos transputers.

Como anteriormente (em 4.5.2) a comunicação dos dados e dos resultados entre o PC e os transputers é feita através do processador "Root" que por sua vez comunica com o transputer zero. As comunicações entre os transputers são feitas de acordo com a topologia utilizada. Assim, quer o problema inicial quer uma solução heurística são afectos ao transputer zero que os transmite a outros e assim sucessivamente. Cada processador executa completamente o AG, isto é, gera uma população inicial e aplica-lhe os operadores genéticos durante um número previamente fixo de gerações (Passos 2 a 5 do algoritmo 4.5). Durante a execução são efectuadas trocas de alguns indivíduos entre processadores. O algoritmo termina quando todos os processadores tiverem terminado a execução do seu AG e transmitido ao processador zero a melhor solução obtida.

Para executar o AG nos transputers fizeram-se as adaptações necessárias, nomeadamente a introdução das instruções que possibilitam as comunicações entre os processadores e a atribuição do trabalho a cada procesador.

TOPOLOGIA

Para 4 e também para 8 transputers utilizaram-se duas topologias, uma com mais comunicações do que a outra. Assim numa, em cada execução só são trocados indivíduos entre dois processadores uma vez. Na outra são trocados várias vezes entre diferentes processadores. Pretendeu-se, com estas 2 topologias, verificar se uma maior troca de indivíduos entre populações distintas melhorava a solução.

As duas topologias (top4 e top4c) utilizadas para 4 transputers estão esquematizadas nas figuras 4.7 e 4.8

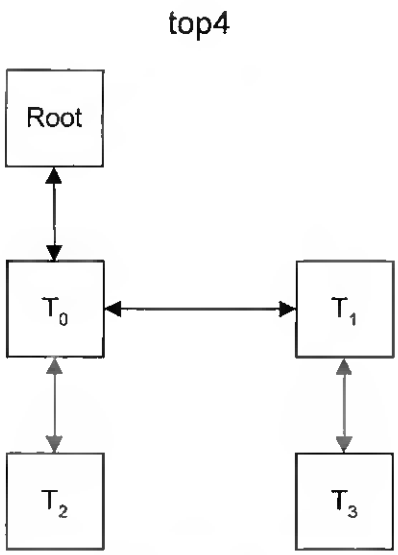


Figura 4.7 – topologia para uma rede de 4 transputers

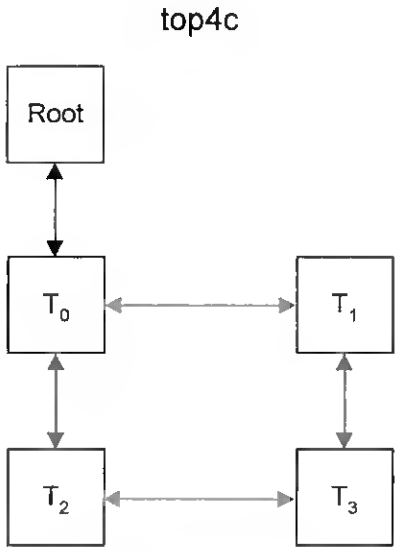


Figura 4.8 – topologia para uma rede de 4 transputers com mais comunicações

Os dados são encaminhados do mesmo modo nas duas topologias, o T_0 recebe-os do "Root" e envia-os para o T_1, depois, simultaneamente, o T_0 envia-os para o T_2 e o T_1 para o T_3 (a azul).

Na primeira topologia (top4) os transputers T_0, T_2 e T_1, T_3 (a azul) trocam alguns indivíduos escolhidos aleatoriamente após um terço do total de gerações que devem executar. Na segunda (top4c), além de fazerem esta troca de indivíduos, os transputers T_0, T_1 e T_2, T_3 (a vermelho) trocam também mais alguns indivíduos após terem executado dois terços do total de gerações. Então o Passo 5 do algoritmo 4.5 passa a ser:

Passo 5:
Se *geração* < *Maxger*
Actualizar $f(x^i)$, $i = 1,...,Pop$, sol^* e x^*;
Se *geração* = $\frac{1}{3}$ *Maxger* (top4 e top4c)
Trocar indivíduos
Se *geração* = $\frac{2}{3}$ *Maxger*
Trocar indivíduos (top4c)
Se *geração* = $\alpha \times$ *Maxger* ($\alpha \in [0,1[$)
substituir pior indivíduo por indivíduo obtido heuristicamente (AG3, AG4, AG5);
Voltar ao Passo2.
Se *geração* = *Maxger*
Terminar o algoritmo obtendo sol^* e x^*.

As figuras 4.9 e 4.10 representam esquematicamente as duas topologias para 8 transputers (top8 e top8c)

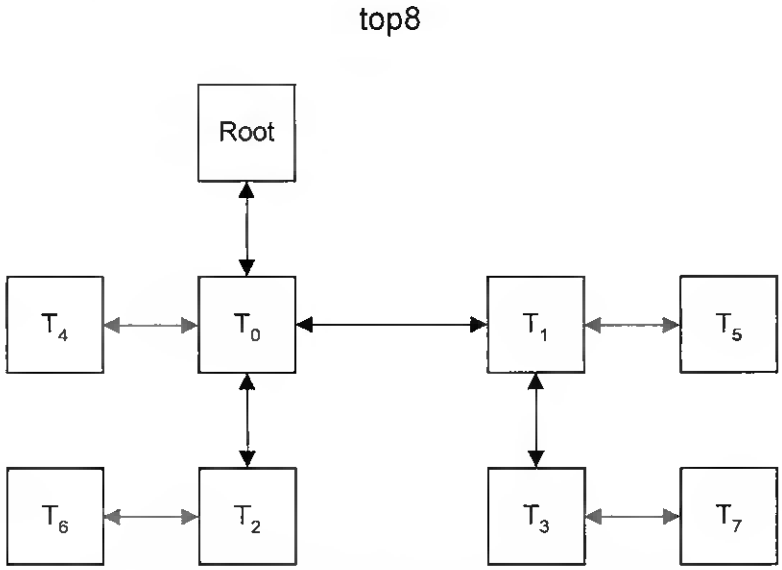


Figura 4.9 – topologia para uma rede de 8 Transputers

Na topologia para 8 transputers com uma única troca de indivíduos (top8) os transputers T_0, T_4 ; T_2, T_6 ; T_1, T_5 e T_3, T_7 (a azul) trocam alguns indivíduos escolhidos aleatoriamente após um quarto do total de gerações que devem executar.

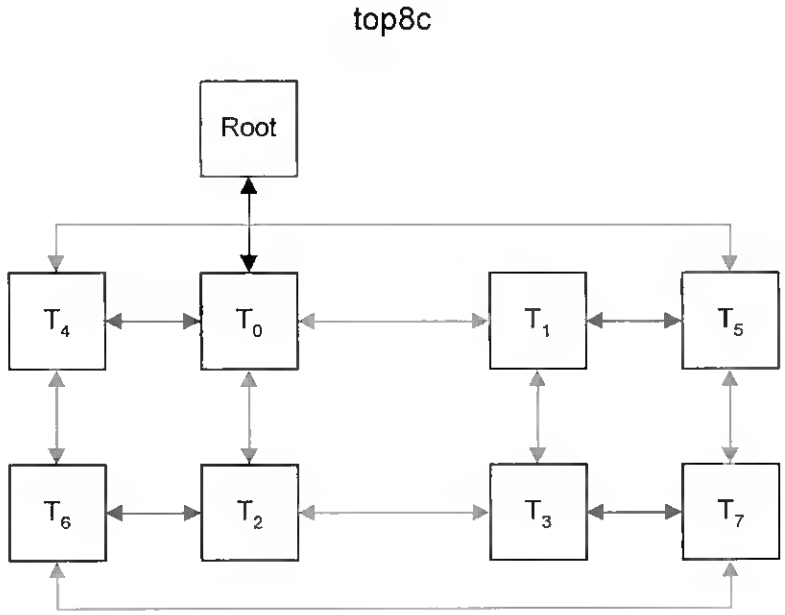


Figura 4.10 – topologia para uma rede de 8 Transputers com mais comunicações

Na topologia com mais trocas de indivíduos entre processadores (top8c), cada troca é feita de um quarto em um quarto das gerações a efectuar. Na primeira vez as trocas são feitas entre os transputers T_0, T_4; T_2, T_6; T_1, T_5 e T_3, T_7 (a azul), na segunda entre T_4, T_6; T_0, T_2; T_1, T_3 e T_5, T_7 (a vermelho) e na última entre T_0, T_1; T_2, T_3; T_4, T_5 e T_6, T_7 (a verde).

Então o Passo 4 do algoritmo 4.5, passa a ser:

Passo 5:
Se *geração* < *Maxger*
Actualizar $f(x^i)$, $i = 1,...,Pop$, sol^* e x^*;
Se *geração* = $\frac{1}{4}$ *Maxger* (top8 e top 8c)
Trocar indivíduos
Se *geração* = $\frac{\alpha}{4}$ *Maxger*, com $\alpha = 2$ ou 3 (top8c)
Trocar indivíduos
Se *geração* = $\alpha \times$ *Maxger* ($\alpha \in [0,1[$)
substituir pior indivíduo por indivíduo obtido heuristicamente (AG3, AG4, AG5);
Voltar ao Passo2.
Se *geração* = *Maxger*
Terminar o algoritmo obtendo a solução igual a sol^* e o vector solução igual a x^*.

Nestas duas topologias para 8 transputers os dados são enviados como descrito para 4 transputers e depois, em paralelo, são enviados de T_0 para T_4, de T_1 para T_5, de T_2 para T_6 e de T_3 para T_7 (a azul).

4.6 EXPERIÊNCIA COMPUTACIONAL

Como foi exposto anteriormente, pretende-se, com a experiência computacional efectuada, estudar o comportamento dos algoritmos desenvolvidos e da sua paralelização. Nesta secção, expõe-se em primeiro lugar a experiência computacional com os algoritmos exactos e, seguidamente a experiência computacional com os AGs. Em cada uma das partes começa-se pela descrição dos testes com os algoritmos

sequenciais, num PC, e prossegue-se com os testes da paralelização, em transputers, do algoritmo de pesquisa em árvore e da versão do algoritmo genético que obtiveram, sequencialmente, melhores resultados.

4.6.1 Experiência Computacional com os Algoritmos de Pesquisa em Árvore Sequenciais

Começou por se testar as duas versões sequenciais do Algoritmo de Pesquisa em Árvore:

APR – Algoritmo de Pardalos e Rodgers descrito em 4.4.1 (algoritmo 4.3);

APA – Algoritmo de Pesquisa em Árvore descrito em 4.5.1. Utiliza outro limite para a função objectivo, (4.28), e outra forma de calcular os limites do gradiente, (4.27) e (4.33).

Estes algoritmos e a heurística (algoritmo 4.4) para obtenção de um majorante foram implementados em Turbo C e executados num PC com um processador Pentium MMX 200 MHz e com 16 MB de RAM.

Fez-se também uma implementação, em Turbo C, do gerador descrito em Pardalos e Rodgers (1990) para obter vários problemas–teste gerados aleatoriamente.

Geraram-se matrizes com dimensões 20, 40, 60, 80, 100 150 e 200, de diferentes esparsidades. Os elementos não diagonais foram gerados no intervalo $[-5,5]$, enquanto os elementos diagonais foram gerados em intervalos diferentes, alguns simétricos do tipo $[-\alpha,\alpha]$ e outros não simétricos do tipo $[-\alpha,\beta]$ com α muito menor do que β e com α e β variando entre 1 e 100. Em Pardalos e Rodgers (1990) o valor de α é muito

superior aos que se utilizaram neste estudo. De facto, como se referiu em 4.4.1, quantas mais linhas (colunas), da matriz da forma quadrática, forem diagonal dominantes mais fácil é resolver o PQ 0-1, sendo mesmo na maioria dos casos resolvido no vértice inicial. Como, o que interessa ao presente trabalho é o estudo do desempenho dos algoritmos paralelos quando é necessário recorrer à pesquisa em árvore, não foram considerados os casos em que o problema ou era resolvido no vértice inicial ou conduzia a árvores de muito pequena dimensão. Assim, para cada dimensão e esparsidade foram experimentados vários valores de α e de β de modo a conseguir obter problemas de mais difícil resolução, correspondentes a matrizes com poucas linhas diagonal dominantes.

A tabela 4.2 resume as características dos dados gerados.

Probl.	nº variáveis	densi-dade	elementos (1) > 0		elementos (1) < 0		diagonal (2) > 0		diagonal (2) < 0		número (3) total		variáveis (4) fixas	
			min.	max.	min.	max.	min.	max.	min.	max.	min.	max.	min.	max.
1 - 10	20	1	96	113	77	94	0	17	3	20	210	210	0	0
11 - 20	40	1	433	441	339	347	0	32	8	40	820	820	0	2
21 - 30	60	0,3	279	288	243	258	0	37	23	60	591	597	0	6
31 - 40	60	0,5	467	487	408	414	0	35	25	60	882	914	0	3
41 - 50	80	0,3	516	519	446	452	0	47	33	80	1045	1048	2	32
51 - 60	80	0,5	876	883	718	720	0	47	33	80	1674	1683	4	14
61 - 70	100	0,1	246	264	220	248	46	76	24	54	584	594	6	56
71 - 80	100	0,3	790	796	681	684	0	58	42	100	1571	1580	2	25
81 - 90	150	0,1	580	611	509	518	0	91	59	150	1248	1269	15	90
91 - 100	200	0,1	1050	1097	875	887	0	133	67	200	1937	1972	23	138

(1) = número de elementos da matriz triangular (sem a diagonal) diferentes de zero

(2) = número de elementos diferentes de zero da diagonal

(3) = número de elementos da matriz triangular diferentes de zero

(4) = número de variáveis fixas definitivamente no vértice inicial

Tabela 4.2 – Características das matrizes geradas aleatoriamente

Os algoritmos não conseguiram obter a solução óptima, em menos de 16 horas de tempo de execução, para 8 (com o APR) e para 4 (com o APA) destes 100 problemas gerados.

Como já foi exposto anteriormente, para reduzir o número de vértices da árvore é fundamental encontrar uma "boa" solução inicial. A heurística utilizada, algoritmo 4.4, cumpriu este objectivo, pois em 22 dos 96 problemas que foram resolvidos exactamente, a solução por ela obtida era já a solução óptima e a pesquisa em árvore apenas serviu para confirmar a optimalidade. Além disso em mais 52 problemas a solução heurística estava a menos de 1% do óptimo. O tempo de execução gasto com a heurística não está contabilizado nos tempos de execução dos algoritmos exactos, pois não é significativo sendo sempre inferior a 0,3 segundos.

Na tabela 4.3 expõem-se resumidamente os resultados obtidos (o melhor, a média e o pior para cada conjunto de problemas) em relação aos problemas para os quais se encontrou a solução óptima com os dois algoritmos.

Verifica-se, na tabela 4.3, que:

- O número de vértices da árvore de pesquisa é sempre menor com o segundo algoritmo (APA). Isto confirma que o limite inferior dado em (4.28) é mais eficiente pois permite podar os ramos da árvore mais cedo;
- O número de vértices pesquisados e o tempo de execução para o APR encontrar uma solução óptima são sempre consideravelmente superiores aos do APA. Como os tempos de execução do APR são em média 4,8 vezes maiores do que os do APA e os vértices pesquisados pelo APR são em média 1,6 vezes mais do que os pesquisados pelo APA (ver tabela 4.4), conclui-se que o tempo de execução gasto no cálculo e actualização do limite dado em (4.28) é menor do que o gasto com o limite de (4.17). Assim, o limite de (4.28) também é computacionalmente mais eficiente.

n	*d*	Pr.		APR		APA	
				Vért.	Tempo	Vért.	Tempo
20	1	10	Melhor	702	0,5	379	0,11
			Média	**1399**	**0,93**	**950**	**0,32**
			Pior	2073	1,32	1538	0,5
40	1	10	Melhor	888	3,297	769	1,48
			Média	**72158**	**273,94**	**65084**	**152,06**
			Pior	609646	2340,17	565223	1396,32
60	0,3	10	Melhor	2295	14,07	1448	4,45
			Média	**38095**	**231,52**	**21162**	**58,98**
			Pior	160079	965	75614	186,15
	0,5	10	Melhor	672	5,39	524	1,92
			Média	**203919**	**1666,39**	**153709**	**600,23**
			Pior	1127100	8917,86	757890	2759,23
80	0,3	10	Melhor	218	3,02	201	1,98
			Média	**312424**	**3977,14**	**216514**	**1233,62**
			Pior	2851351	36259	1967126	11157,69
	0,5	10	Melhor	141	2,2	127	0,77
			Média	**2597**	**42,61**	**1932**	**15,04**
			Pior	12184	201,87	8511	69,89
100	0,1	9	Melhor	3072	32,86	1306	2,09
			Média	**44659**	**484,27**	**12160**	**45,08**
			Pior	157207	1733,35	36724	171,15
	0,3	8	Melhor	576	10,71	476	2,2
			Média	**62912**	**1274,36**	**42300**	**326,82**
			Pior	286832	5504,84	190095	1156,04
150	0,1	8	Melhor	17	0,55	17	0,17
			Média	**447887**	**12580,53**	**151271**	**1534,24**
			Pior	1982274	54927	548167	4033,08
200	0,1	7	Melhor	1267	69,12	719	9,23
			Média	**128448**	**7141,4**	**50475**	**543,62**
			Pior	501950	28077,69	164651	1292,31

n = Número de variáveis

d = Densidade da matriz

Pr. = Número de problemas resolvidos exactamente (em menos de 16 horas)

Vért. = Número de vértices pesquisados

Tempo = tempo, em segundos, de execução do algoritmo

Tabela 4.3 – Resultados obtidos para os problemas gerados aleatoriamente

- A variação obtida em relação ao número de vértices necessários à pesquisa da árvore, para cada dimensão, é muito grande, o que confirma o comportamento imprevisível dos algoritmos de Pesquisa em Árvore. A única relação que foi possível tirar sobre as características dos problemas e o

comportamento do algoritmo, tem a ver com a dominância dos elementos diagonais da matriz da forma quadrática. Verificou-se que, quantos mais elementos da diagonal fossem dominantes, mais variáveis eram fixas definitivamente no vértice inicial, formando-se árvores menores. Observou-se, também, que os problemas cuja diagonal tinha sido gerada nos intervalos não simétricos $[-\alpha, \beta]$, resolviam-se com árvores menores

Na tabela 4.4 e no gráfico 4.1 mostram-se as médias dos quocientes entre os valores obtidos com o APR e o APA, para o número de vértices pesquisado e tempo de execução, isto é, considerando os valores obtidos para cada conjunto de problemas têm-se na coluna Vértices as média dos quocientes:

$$\frac{\text{nº de vértices pesquisado pelo APR}}{\text{nº de vértices pesquisado pelo APA}}$$

e na coluna Tempo as média dos quocientes:

$$\frac{\text{tempo de execução do APR}}{\text{tempo de execução do APA}}$$

n	*d*	Pr.	Rácio	
			Vértices	Tempo
20	1	10	1,5	3,04
40	1	10	1,2	3,05
60	0,3	10	1,7	3,51
	0,5	10	1,3	2,49
80	0,3	10	1,4	2,95
	0,5	10	1,2	2,67
100	0,1	9	2,96	11,71
	0,3	8	1,37	3,48
150	0,1	8	2,3	8,13
200	0,1	7	1,98	10,21
média			1,62	4,75

n = Número de variáveis
d = Densidade da matriz
Pr. = Número de problemas resolvidos (em menos de 16 horas)

Tabela 4.4 – valores médios dos rácios para o nº de vértices e do tempo de execução

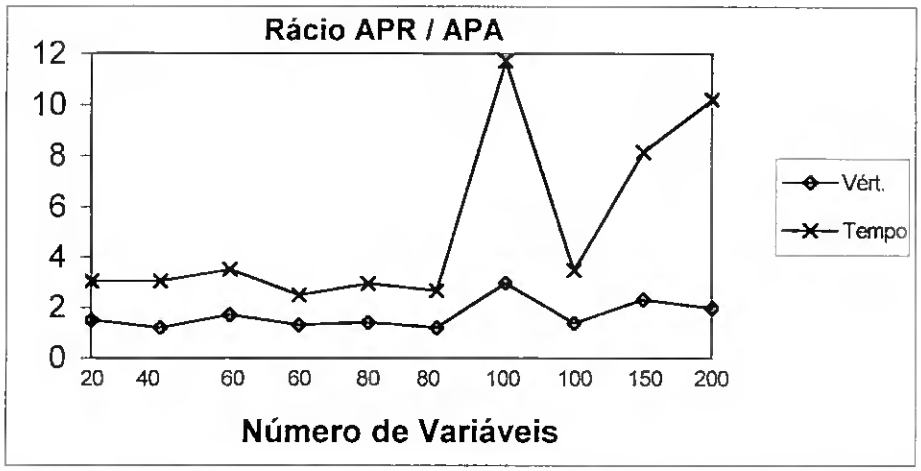


Gráfico 4.1 – Valores médios dos rácios para o nº de vértices e tempo de execução

Estes resultados mostram, claramente, que, enquanto o número médio de vértices pesquisados pelo APR varia entre 1,2 e 3 vezes o do APA, o tempo médio de execução do APR varia entre 2,7 e 11,7 vezes o do APA. Isto significa que o APA é muito mais eficiente do que o APR não só porque, em média, as árvores de pesquisa são cerca de duas vezes menores, como também o tempo de execução é, em média, cerca de cinco vezes menor.

O comportamento do APR relativamente ao do APA foi sistematicamente pior para todos os problemas. A diferença entre a eficiência dos dois algoritmos é ainda maior nos problemas de maior dimensão (100 com densidade 0,1; 150 e 200).

Como o APA foi sistematicamente o algoritmo mais eficiente procedeu-se à sua paralelização.

4.6.2 Experiência Computacional com o Algoritmos de Pesquisa em Árvore (APA) Paralelo

O algoritmo paralelo foi executado em 2 , 4 e 8 transputers INMOS 25 MHz numa plataforma TMB16, instalada num PC, com velocidade de comunicação de 20 Mbits por segundo. Para se poder avaliar o desempenho do algoritmo paralelo é

necessário obter valores para a aceleração e eficiência (já definidas em2.6), o que obriga a executar o algoritmo sequencial num único transputer.

Não se executou nenhum dos algoritmos, sequencial ou paralelo, no transputer raiz ("root"), para que os tempos de execução fossem correctamente comparáveis.

Como se referiu na secção anterior, codificou-se o algoritmo sequencial em Turbo C. Como esta linguagem não é compatível com os transputers, foi necessário fazer algumas modificações ao código do programa, bem como adaptá-lo à execução em paralelo, de acordo com as topologias utilizadas (consoante se utilizasse 2, 4 ou 8 transputers). Isso implicou incluir no programa o código para ser executado por cada processador e as instruções de comunicação entre os processadores, para cada caso.

Para que o estudo desta paralelização não exigisse um tempo excessivo optou-se por limitar o tempo de execução do algoritmo sequencial (num transputer) para cada problema a cinco minutos.

Para determinar como a granularidade afecta a aceleração e o tamanho da árvore de pesquisa resolveram-se, previamente, dois problemas de características diferentes, variando o valor atribuído a *maxn*. Como se referiu, nas secções 4.4 e 4.5, *maxn* denota o número máximo de vértices que cada processador pode pesquisar no seu ramo da árvore independentemente dos outros, isto é, sem comunicar com os outros (algoritmo 4.2).

Para este estudo, escolheram-se, entre os problemas gerados, um com dimensão 40 e densidade 1 e outro com dimensão 60 e densidade 0,3 . A escolha recaiu nestes dois problemas, por serem de média dimensão e por num a árvore de pesquisa ser relativamente pequena e no outro ser bastante maior. As tabelas 4.5 e 4.6 e os gráficos 4.2 a 4.5 mostram os resultados obtidos.

	Processadores										
	1		2			4			8		
maxn	vért.	tempo	vért.	tempo	acel.	vért.	tempo	acel.	vért.	tempo	acel.
10	7820	287,78	7797	164,22	1,75	7791	90,15	3,19	7747	48,86	5,89
50	"	"	7813	162,55	1,77	7801	87,86	3,28	7800	47,82	6,02
100	"	"	7816	160,36	1,79	7814	84,71	3,40	7804	48,88	5,89
1000	"	"	7818	155,41	1,85	7816	77,52	3,71	7809	47,75	6,03
10000	"	"	7819	146,85	1,96	7817	77,57	3,71	7813	47,80	6,02

vért = número de vértices da árvore de pesquisa
tempo = tempo de execução em segundos
acel. = aceleração

Tabela 4.5 – Resultados da granularidade para $n = 40$

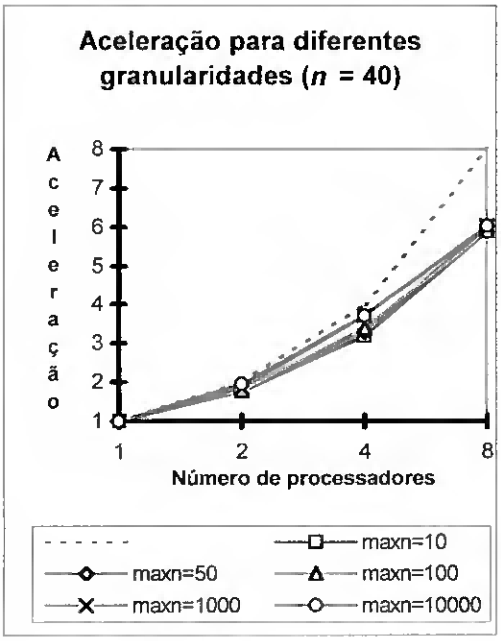


Gráfico 4.2 – Granularidade vs aceleração

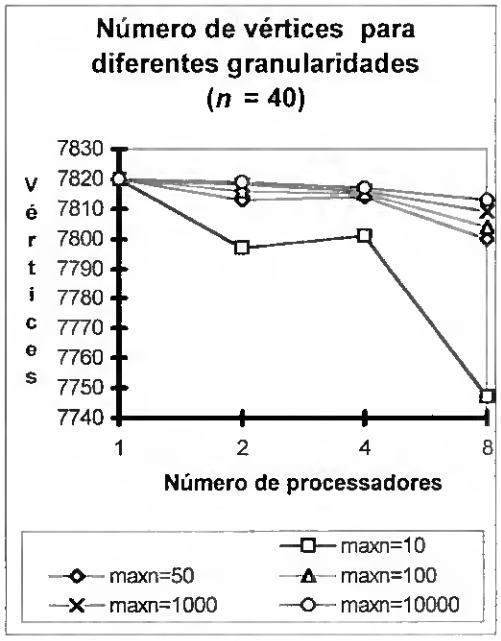


Gráfico 4.3 – Granularidade vs nº de vértices

	Processadores										
	1		2			4			8		
maxn	vért.	tempo	vért.	tempo	acel.	vért.	tempo	acel.	vért.	tempo	acel.
10	2753	152,56	2680	95,13	1,60	2604	64,79	2,35	2583	41,99	3,63
100	"	"	2721	87,03	1,75	2695	77,53	1,97	2686	42,18	3,62
500	"	"	2725	92,46	1,65	2719	75,74	2,01	2737	43,64	3,50
1000	"	"	2727	92,11	1,66	2737	76,27	2,00	2744	43,98	3,47
3000	"	"	2752	93,01	1,64	2750	77,08	1,98	2744	43,98	3,47

vért = número de vértices da árvore de pesquisa
tempo = tempo de execução em segundos
acel. = aceleração

Tabela 4.6 – Resultados da granularidade para $n = 60$

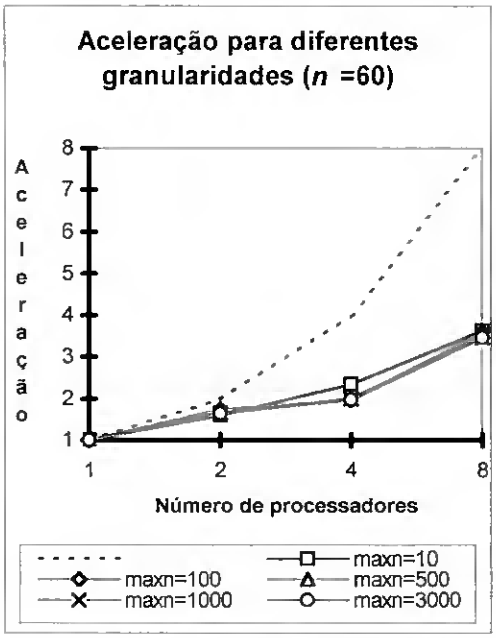


Gráfico 4.4 – Granularidade vs aceleração

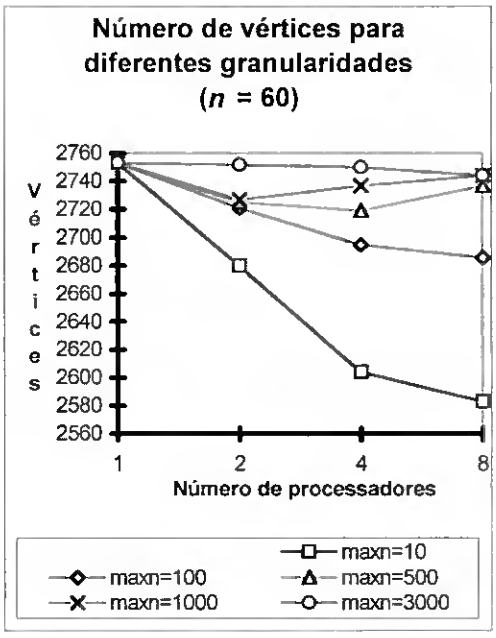


Gráfico 4.5 – Granularidade vs nº de vértices

Estas tabelas e gráficos mostram, por um lado, que quanto menor for o valor de *maxn*, menor é também o número de vértices das árvores de pesquisa. Por outro lado, no que respeita à aceleração, os dois problemas tiveram comportamentos diferentes, verificando-se que para árvores maiores é melhor um valor grande para *maxn*, enquanto

que para árvores com menos vértices um menor valor para *maxn* conduz a melhores acelerações. No entanto, a variação da aceleração em função dos valores de *maxn* é sempre pouco significativa, embora os seus valores variem de um problema para o outro. De facto, se a comunicação frequente entre processadores provoca uma diminuição no tamanho da árvore, também acarreta um aumento do tempo de execução. Assim, enquanto que para árvores de dimensão reduzida o aumento de tempo pode ser equilibrado com a diminuição do número de vértices a pesquisar, isto já não acontece para as árvores maiores, em que o aumento com os tempos de comunicações frequentes não contrabalança a diminuição do número de vértices.

Tendo em conta que não se sabe, à priori, se o problema a resolver vai necessitar de pesquisar uma árvore com muitos ou poucos vértices e também que neste estudo da paralelização se estabeleceu o limite de tempo de execução do algoritmo sequencial em 5 minutos, definiu-se *maxn* = 100 como um valor de compromisso.

Conseguiram-se resolver sequencialmente, em 1 transputer, 47 dos 100 problemas-teste em menos de 5 minutos. Estes foram depois resolvidos paralelamente em 2, 4 e 8 transputers. Apresenta-se, na tabela 4.7, o resumo dos resultados obtidos.

Na tabela 4.7 os resultados "melhor" e "pior" referem-se, no caso sequencial, ao melhor e ao pior tempo de execução e no caso paralelo à melhor e à pior aceleração obtida com 2, 4 e 8 transputers, por isso, os valores de uma mesma linha podem corresponder a problemas diferentes. Assim, o estudo dos resultados obtidos vai ser feito, principalmente, em função dos valores médios.

Não se incluíram, nesta tabela, os resultados relativos aos problemas gerados (10 para cada dimensão e densidade) com dimensão 200, 150 e 100 com densidade 0,3, porque só foi possível resolver sequencialmente em menos de 5 minutos, 3 dos de dimensão 100, 2 de dimensão 150 e 1 de dimensão 200, o que não permitiu fazer um estudo adequado da aceleração e eficiência do algoritmo paralelo para estas dimensões.

n	*d*	Pr.		Processadores											
				1		2				4				8			
				vért.	tempo	vért.	tempo	acel.	efic.	vért.	tempo	acel.	efic.	vért.	tempo	acel.	efic.
			B(a)	379	2,39	428	1,46	1,83	0,92	685	1,36	3,10	0,78	983	1,05	5,64	0,71
20	1	10	A	**950**	**5,60**	**966**	**3,18**	**1,76**	**0,88**	**969**	**1,94**	**2,89**	**0,72**	**1028**	**1,15**	**4,86**	**0,61**
			W(a)	1538	8,93	377	1,46	1,64	0,82	1164	2,55	2,48	0,62	1238	1,56	4,05	0,51
			B(a)	769	31,22	7816	160,36	1,80	0,90	2505	29,86	3,41	0,85	2490	17,07	5,97	0,75
40	1	7	A	**2629**	**89,65**	**2623**	**50,80**	**1,74**	**0,87**	**2623**	**28,23**	**3,08**	**0,77**	**2620**	**16,42**	**5,21**	**0,65**
			W(a)	7820	287,78	1740	21,57	1,66	0,83	3301	42,63	2,70	0,68	1288	7,83	4,49	0,56
			B	684	39,09	684	21,29	1,84	0,92	680	13,31	2,94	0,74	674	9,39	4,16	0,52
	0,3	7	A	**2646**	**154,84**	**2632**	**101,51**	**1,56**	**0,78**	**1219**	**72,39**	**2,24**	**0,56**	**2625**	**41,77**	**3,76**	**0,47**
			W(a)	4444	249,39	1419	75,93	1,22	0,61	2695	77,53	1,97	0,49	3101	43,37	3,45	0,43
60			B(a)	524	41,97	1010	40,75	1,87	0,94	852	34,16	3,41	0,85	847	18,58	6,28	0,79
	0,5	6	A	**1229**	**111,47**	**1216**	**62,41**	**1,81**	**0,91**	**1219**	**35,49**	**3,09**	**0,77**	**1222**	**19,46**	**5,74**	**0,72**
			W(a)	3004	290,97	2999	167,24	1,74	0,87	522	15,33	2,74	0,69	687	10,17	5,17	0,65
			B(a)	201	43,31	540	30,14	1,81	0,91	539	17,56	3,11	0,78	535	8,56	6,38	0,80
	0,3	4	A	**1038**	**129,83**	**1037**	**86,39**	**1,56**	**0,78**	**1008**	**49,84**	**2,63**	**0,66**	**1030**	**31,84**	**4,41**	**0,55**
			W(a)	2372	246,35	1039	141,10	1,24	0,62	198	19,34	2,24	0,56	194	13,71	3,16	0,40
80			B(b)	219	54,84	1342	97,53	1,81	0,91	1332	54,14	3,26	0,82	1328	31,26	5,65	0,71
	0,5	4	A	**758**	**120,68**	**754**	**76,75**	**1,56**	**0,78**	**751**	**48,22**	**2,67**	**0,67**	**748**	**29,48**	**4,15**	**0,52**
			W(b)	1345	176,58	516	80,59	1,37	0,69	514	65,28	1,69	0,42	512	37,91	2,92	0,37
			B(b)	1305	40,70	3501	58,76	2,42	1,21	3853	37,33	3,81	0,95	4012	23,04	6,17	0,77
100	0,1	5	A	**2961**	**125,26**	**2517**	**72,89**	**1,72**	**0,86**	**2573**	**43,99**	**2,86**	**0,72**	**2707**	**30,35**	**4,20**	**0,53**
			W(b)	5507	199,34	1374	84,86	1,43	0,72	1378	59,82	2,02	0,51	1388	41,17	2,94	0,37

n = dimensão *d* = densidade da matriz

Pr. = número de problemas-teste que correram sequencialmente em menos de 5 minutos

vért. = número de vértices da árvore de pesquisa

acel. = aceleração efic. = eficiência

B = melhor resultado A = média W = pior resultado

tempo = tempo de execução em segundos

(a) os resultados para 1, 2 ,4 e 8 processadores correspondem a problemas diferentes

(b) os resultados para 2 ,4 e 8 processadores correspondem ao mesmo problema

Tabela 4.7 – Resultados obtidos com o algoritmo paralelo de PA

Os valores obtidos para a aceleração são bastante bons, principalmente no caso de 2 processadores como se pode ver nas colunas da eficiência, tendo-se obtido inclusive aceleração superlinear (com valor 2.42) para um problema-teste. Pode-se explicar esta aceleração pelo facto de o número de vértices da árvore, obtido com 2 processadores, ser muito menor do que o obtido sequencialmente. De facto, na versão sequencial explora-se primeiro um dos ramos da árvore completamente e só depois se passa ao

outro, enquanto que na versão paralela os dois ramos da árvore vão sendo explorados simultaneamente. Como o incumbente em cada ramo é comunicado ao outro, com alguma frequência, e actualizado, se for caso disso, é possível podar os dois ramos mais cedo.

Na tabela 4.8 apresentam-se o valor médio e o desvio-padrão da aceleração, para os casos de 2, 4 e 8 processadores, em relação a todos os problemas resolvidos.

	Aceleração		
	Processadores		
	2	4	8
Média	1,68	2,78	4,62
Desvio-padrão	0,20	0,47	0,98

Tabela 4.8 – Estatísticas referentes à tabela 4.7

Apesar do número de vértices da árvore de pesquisa variar bastante de problema para problema (tabela 4.7), verifica-se, pelos valores do desvio-padrão, que o comportamento do algoritmo paralelo não variou muito de problema para problema. No entanto é mais homogéneo para 2 processadores e menos para 8. A grande disparidade no tamanho das árvores de pesquisa para os problemas de cada dimensão, não se reflectiu de uma forma significativa na aceleração.

Os gráficos 4.6 a 4.10 mostram, para cada dimensão da matriz da forma quadrática, as acelerações médias obtidas. Nestes gráficos d denota a densidade da matriz e a linha tracejada representa a aceleração linear (ideal).

Estes gráficos confirmam as afirmações anteriores sobre a homogeneidade do comportamento do algoritmo em relação à aceleração, tendo as linhas obtidas para a aceleração um comportamento semelhante para as dimensões estudadas. Os valores da aceleração são bons, estando mesmo muito próximos dos valores lineares para 2 processadores, distanciando-se deles principalmente para 8 processadores (como era de se esperar devido ao aumento das comunicações).

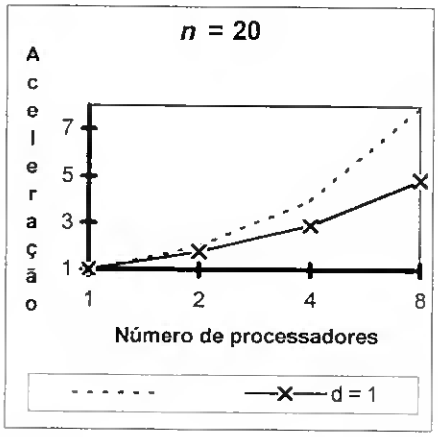


Gráfico 4.6 – Aceleração para $n = 20$

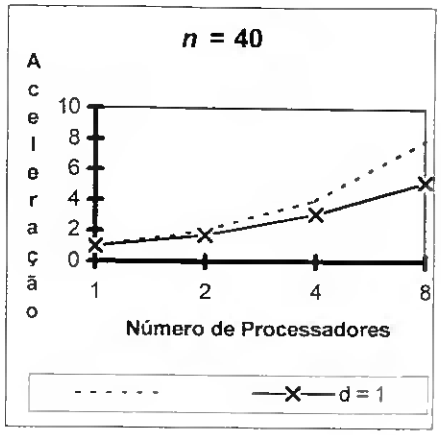


Gráfico 4.7 – Aceleração para $n = 40$

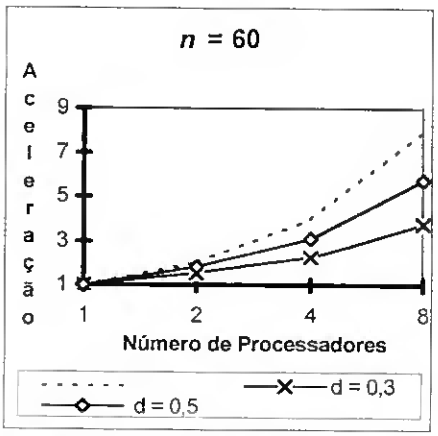


Gráfico 4.8 – Aceleração para $n = 60$

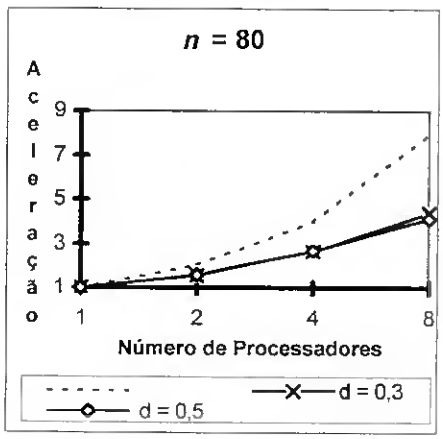


Gráfico 4.9 – Aceleração para $n = 80$

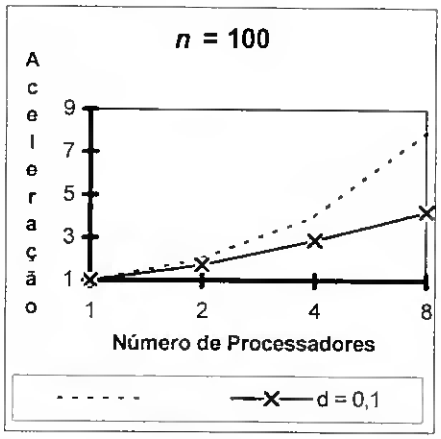


Gráfico 4.10 – Aceleração para N = 100

4.6.2.1 COMENTÁRIOS SOBRE A IMPLEMENTAÇÃO COM "ROUTER"

Embora a utilização do "router" tenha surgido com a intenção de acelerar a execução em paralelo, tal não aconteceu.

Como foi explicado em 4.5.2.2 a utilização de um processo virtual incumbido apenas das comunicações ("router" - "router" e "router" - "worker") obrigou, neste caso, à criação de uma ordem de comunicação. A implementação deste método foi feita primeiro para 2 transputers e como os resultados obtidos, para os problemas que tinham sido resolvidos com o algoritmo anterior, foram sistematicamente piores, abandonou-se esta ideia e já não se fez a sua implementação para 4 e 8 transputers.

Esta necessidade de cada "router" só comunicar um melhor incumbente quando era a "sua vez" acarretou numa menor eficiência, porque implicou que nalguns casos os "worker" tivessem que ficar parados à espera de receber ou enviar dados para o respectivo "router", levando sempre a maiores tempos de execução, embora, em geral, o número de subproblemas fosse um pouco menor.

4.6.2.2 RESULTADOS COM A VERSÃO ε - APROXIMADA

Para se testar o funcionamento deste algoritmo duma forma aproximada, tentaram-se resolver, utilizando a versão ε-aproximada descrita em 4.5.2.3, para 8 processadores, os problemas para os quais não se tinha encontrado a solução óptima no tempo limite. Optou-se por executar este algoritmo só em 8 processadores porque é mais rápido e, neste caso, o objectivo era estudar a utilização do algoritmo como um método aproximado e não a aceleração. Para se poder avaliar o tempo de execução do algoritmo ε-aproximado, comparou-se com o tempo de execução, também em 8 processadores, do algoritmo exacto (APA) para os mesmos problemas. Apresentam-se os resultados obtidos na tabela 4.9.

n	*d*	8 Processadores						
		Solução Exacta		solução ε-aproximada				
		vértices	tempo	vértices	tempo	(a)	(b)	(c)
40	1	16975	85,36	14273	74,21	0,00	0,00	0,00
		50150	297,32	45881	263,02	1,47	0,55	0,55
60	0,3		(*)	15322	255,12	0,00	0,00	0,00
		8157	165,15	7083	148,40	0,00	0,00	0,00
60	0,5	11460	210,72	9136	186,43	3,70	0,52	0,52
80	0,3	13300	180,95	9172	145,63	0,00	0,00	0,00
		3771	104,71	3087	91,07	1,40	0,19	0,19
80	0,5	4281	146,90	2841	135,60	0,00	0,00	0,00
		1981	132,63	1577	96,62	0,00	0,00	0,00
100	0,1	17111	131,90	12324	88,47	3,90	1,57	11,31
		9849	103,75	5317	55,59	3,60	1,36	10,95
200	0,1		(*)	3369	157,52	0,04	0,00	0,00
		3046	156,14	849	78,41	1,00	0,07	0,07

n = dimensão da matriz
d = densidade da matriz
vértices = número de vértices da árvore de pesquisa
tempo = tempo de execução em segundos
(a) = |opt. - ε-sol.|
(b) = erro relativo da solução ε-aproximada
(c) = erro relativo da solução heurística inicial
(*) = mais de 300 segundos

Tabela 4.9 – Resultados obtidos com a versão ε-aproximada

Dos 43 problemas, ficaram resolvidos 11, com o algoritmo exacto, e 13, com o algoritmo ε-aproximado, em menos de 5 minutos. A solução ε-aproximada foi a óptima em todos os problemas em que a solução heurística inicial também já era a óptima. Nos outros casos, embora a solução não seja a óptima está muito próxima da óptima e o erro relativo foi sempre muito menor do que o erro máximo permitido pelo algoritmo. Os tempos de execução são sempre menores que os obtidos com o algoritmo exacto e como a solução está sempre muito próxima do óptimo e o erro máximo possível é conhecido, este algoritmo é uma alternativa para problemas muito demorados. No entanto, verifica-se que só em dois casos a solução obtida pelo algoritmo ε-aproximado foi

consideravelmente melhor que a da heurística inicial, sendo que esta última é muito rápida (menos de 0,1 segundos).

4.6.2.3 RESULTADOS COM DADOS OBTIDOS DA LITERATURA

Completou-se este estudo da paralelização do Algoritmo de Pesquisa em Árvore, testando-o num conjunto de 8 matrizes da Harwell-Boeing Collection, já utilizados em trabalhos anteriores de Faustino ainda não publicados, e disponíveis na Internet no endereço http://gams.nist.gov/MatrixMarket provenientes de diferentes tipos de problemas (Engenharia Estrutural, Equações Diferenciais Parciais, Redes de Sistemas de Energia). Estas matrizes são simétricas, reais e não são diagonal dominantes. Apresentam-se na tabela 4.10 o número de variáveis (n) e o número de elementos diferentes de zero (m) da matriz triangular (sem contar os elementos da diagonal).

Matriz	n	m
bcsstk02	66	2145
bcsstk03	112	264
bcsstm07	420	3416
gr3030	900	3422
nos1	237	390
nos2	957	1590
nos6	675	1290
nos7	729	1944
494bus	494	586

Tabela 4.10 – Características das matrizes

Para se obterem algumas instâncias de cada matriz geraram-se aleatoriamente diferentes vectores c (o termo independente de (4.1)). Cada vector foi depois incluído na diagonal da matriz, passando a instância a ter a forma (4.12). No caso dos problemas gr3030, nos6, nos7, obteve-se a solução óptima no vértice inicial para os muitos vectores c gerados. Nestes problemas as variáveis foram todas fixas no vértice inicial a partir dos

limites dos respectivos gradientes ((4.27) e (4.33)) não sendo necessária qualquer pesquisa da árvore. Para cada uma das outras matrizes obtiveram-se 5 instâncias diferentes que foram utilizadas para testar o algoritmo paralelo. Resumem-se, na tabela 4.11, as características destes problemas-teste.

Probl. nº	matriz	nº variáveis	elementos (1)		diagonal (2)				número (3)		variáveis (4)	
			> 0	< 0	> 0		< 0		total		fixas	
					min.	max.	min.	max.	min.	max.	min.	max.
1 - 5	bcsstk02	66	987	1158	46	49	17	20	2211	2211	12	16
6 - 10	bcsstk03	112	115	149	59	89	23	53	376	376	83	89
11 - 15	nos1	237	156	234	132	141	81	83	606	614	170	185
16 - 20	bcsstm07	420	2146	1270	256	257	163	164	3836	3836	382	394
21 - 25	494bus	494	0	586	286	310	184	208	1080	1080	438	463

(1) = número de elementos da matriz triangular (sem a diagonal) diferentes de zero
(2) = número de elementos diferentes de zero da soma do termo independente com a diagonal
(3) = número de elementos da matriz triangular diferentes de zero
(4) = número de variáveis fixas definitivamente no vértice inicial

Tabela 4.11 – Características dos problemas-teste

Nestes problemas-teste, também se utilizou a heurística descrita em 4.4.2 (algoritmo 4.4) para se obter um bom limite do valor da solução óptima e, mais uma vez, numa grande parte dos problemas, a solução obtida por esta heurística ou é a solução óptima ou está muito próxima. Os tempos de execução da heurística também não foram incluídos por terem sido obtidos sequencialmente e serem sempre inferiores a 0,1 segundos.

Neste caso, dada a experiência anterior com os dados gerados aleatoriamente, não se procedeu ao estudo da granularidade para determinar o valor do número máximo de subproblemas (*maxn*) a executar antes das comunicações entre processadores. Optou-se por estabelecê-lo de acordo com o número de variáveis livres no vértice inicial, visto que quanto maior for o número de variáveis livres provavelmente maior é a árvore de pesquisa. A tabela 4.12 mostra os valores utilizados para *maxn*.

Nº de variáveis livres	Número de Processadores		
	2	4	8
menos de 50	50	25	15
mais de 50	100	50	25

Tabela 4.12 – Valores de Maxn

Os resultados obtidos sequencialmente e com 2, 4 e 8 processadores estão resumidos na tabela 4.13. Como anteriormente, também nesta tabela os resultados "melhor" e "pior" referem-se, no caso sequencial, ao melhor e ao pior tempo de execução e no caso paralelo à melhor e à pior aceleração obtida com 2, 4 e 8 transputers, por isso, os valores de uma mesma linha podem corresponder a problemas diferentes. Assim, a análise dos resultados obtidos vai ser feita, principalmente, em função dos valores médios.

Observa-se, na tabela 4.13, que os três primeiros conjuntos de problemas têm um comportamento semelhante com bons valores da aceleração e consequentemente da eficiência. O quarto conjunto de problemas obteve valores muito bons para a aceleração e eficiência, atingindo em alguns casos aceleração superlinear (em 1, 3 e 2 casos com 2, 4 e 8 processadores respectivamente). Pelo contrário, o quinto conjunto de problemas obteve acelerações detrimentais em todos os casos.

Os resultados obtidos para estes dois últimos conjuntos de problemas vêm confirmar o comportamento imprevisível dos algoritmos (principalmente dos paralelos) de pesquisa em árvore. Este comportamento deve-se à forma diferente de pesquisar a árvore executada pelo algorimo sequencial e pelos paralelos. De facto, a versão sequencial realiza uma pesquisa em profundidade. O ramo da direita só é pesquisado depois de terminada toda a pesquisa do ramo esquerdo. Na versão paralela, com a topologia utilizada, explora-se simultaneamente os dois ramos, cada um em profundidade. Para o mesmo problema, estratégias diferentes de pesquisar a árvore podem provocar uma grande disparidade no número de vértices pesquisado por cada uma.

		Processadores													
n		1		2				4				8			
m		vért.	tempo	vért.	tempo	acel.	efic.	vért.	tempo	acel.	efic.	vért.	tempo	acel.	efic.
66	B(b)	310	30,43	308	17,51	1,74	0,87	306	10,02	3,04	0,76	290	11,87	5,03	0,63
2211	A	**821**	**77,48**	**815**	**55,21**	**1,51**	**0,76**	**807**	**34,62**	**2,43**	**0,61**	**794**	**20,08**	**4,22**	**0,53**
	W(a)	1600	149,30	1590	114,49	1,30	0,65	446	20,72	2,04	0,51	1564	42,83	3,49	0,44
112	B(b)	259	11,46	764	27,67	1,95	0,98	764	14,74	3,67	0,92	758	7,73	7,00	0,88
376	A	**689**	**40,53**	**690**	**24,48**	**1,60**	**0,80**	**684**	**14,75**	**2,72**	**0,68**	**671**	**9,08**	**4,66**	**0,58**
	W(b)	1023	60,38	250	9,91	1,16	0,58	247	6,14	1,86	0,47	239	3,78	3,03	0,38
237	B	597	174,87	3192	399,89	1,65	0,83	588	54,07	3,23	0,81	1642	56,11	7,70	0,96
627	A	**1467**	**346,82**	**1914**	**247,46**	**1,38**	**0,69**	**1815**	**125,13**	**2,87**	**0,72**	**1688**	**63,11**	**5,33**	**0,67**
	W(a)	2701	658,76	3056	358,17	1,21	0,61	3862	257,35	2,56	0,64	1691	61,74	3,79	0,47
420	B(b)	135	135,89	265	168,06	3,27	1,64	260	91,73	5,99	1,50	248	63,30	8,67	1,08
3836	A	**385**	**381,46**	**434**	**248,28**	**1,62**	**0,81**	**380**	**140,29**	**3,58**	**0,90**	**378**	**101,11**	**5,50**	**0,69**
	W(b)	737	715,51	265	168,06	1,06	0,53	246	95,19	1,43	0,36	319	59,85	2,27	0,28
494	B(b)	67	60,77	267	133,38	1,43	0,72	357	97,84	1,95	0,49	586	88,49	2,16	0,27
1080	A	**254**	**237,79**	**415**	**207,35**	**1,16**	**0,58**	**478**	**159,79**	**1,47**	**0,37**	**1092**	**155,22**	**1,59**	**0,20**
	W(b)	377	339,75	118	59,80	1,02	0,51	199	56,53	1,08	0,27	325	49,36	1,23	0,15

n = dimensão *m* = número de elementos da matriz
vért. = número de vértices da árvore de pesquisa
acel. = aceleração efic. = eficiência
B = melhor resultado A = média W = pior resultado
tempo = tempo de execução em segundos
(a) os resultados para 1, 2, 4 e 8 processadores não correspondem ao mesmo problema
(b) os resultados para 2, 4 e 8 processadores correspondem ao mesmo problema

Tabela 4.13 – Resultados obtidos com o Algoritmo paralelo de Pesquisa em Árvore

No quarto conjunto de problemas verificou-se que o número de vértices pesquisado pelo algoritmo paralelo é muito menor que o pesquisado pelo sequencial. Nestes problemas ocorreu uma frequente actualização do incumbente que associada à pesquisa simultânea dos dois ramos, no algoritmo paralelo, permitiu podar mais cedo a árvore, principalmente o ramo esquerdo, enquanto que a versão sequencial executou uma pesquisa muito maior no ramo esquerdo.

Com o último conjunto de problemas acontece praticamente o oposto. O algoritmo sequencial obtém o óptimo no fim da pesquisa do ramo esquerdo e esse valor permite-lhe podar o ramo direito logo no início (após a ramificação de uma ou duas variáveis), enquanto o algoritmo paralelo vai actualizando o incumbente e pesquisando

ambos os ramos, podando-os só depois de terem sido os dois bastante pesquisados, o que obviamente se traduz em aproximadamente pesquisar o dobro de vértices.

O número de elementos positivos e negativos da matriz triangular (sem a diagonal) pode ser uma das causas destas anomalias de comportamento. Repare-se que (ver tabela 4.11), no caso da matriz bcsstm07 (quarto conjunto de problemas) o número de elementos positivos é praticamente o dobro dos negativos, enquanto que a matriz 494bus (quinto conjunto) não tem elementos positivos. As restantes matrizes, que apresentam um comportamento normal, têm um número semelhante de elementos positivos e negativos.

Os gráficos 4.11 a 4.15 mostram para cada conjunto de problemas, as acelerações médias obtidas. Nestes gráficos a linha tracejada representa a aceleração linear.

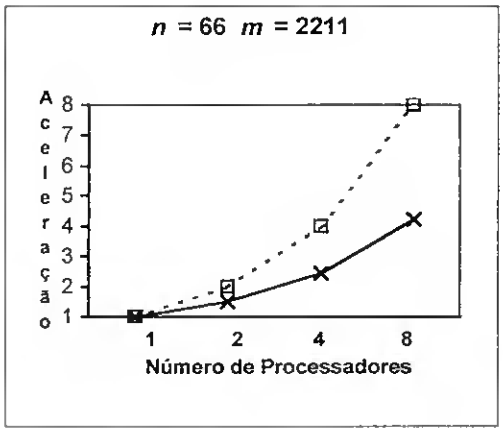


Gráfico 4.11 – Aceleração para $n = 66$

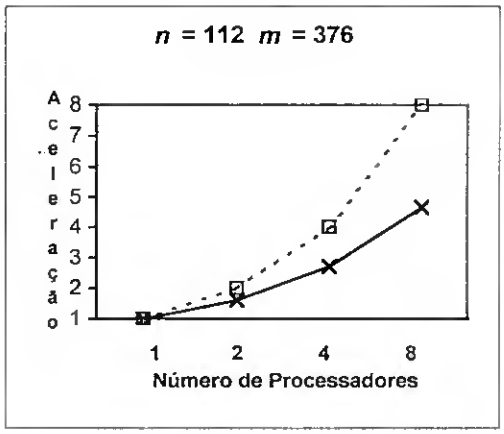


Gráfico 4.12 - Aceleração para $n = 112$

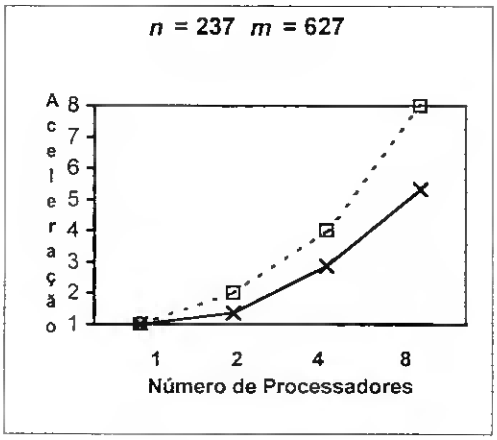


Gráfico 4.13 - Aceleração para $n = 237$

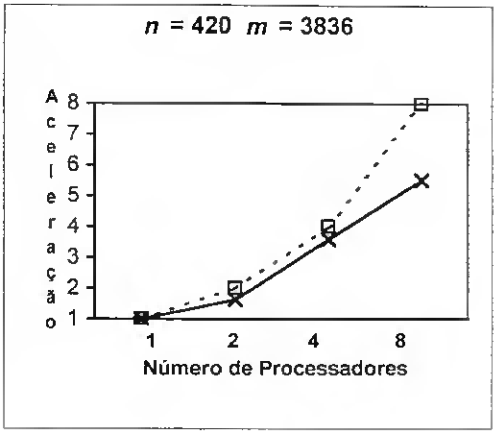


Gráfico 4.14 - Aceleração para $n = 420$

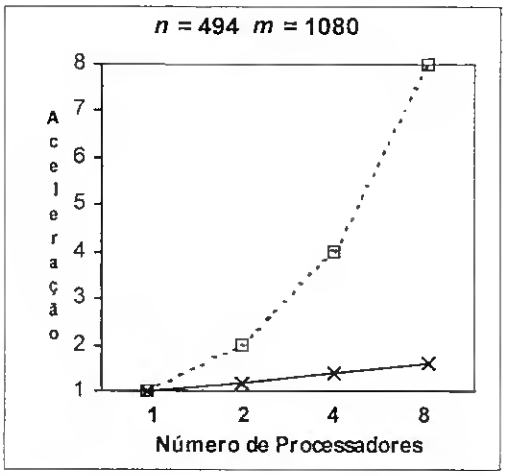


Gráfico 4.15 - Aceleração para $n = 494$

Os gráficos 4.11, 4.12 e 4.13 mostram um comportamento homogéneo em relação à aceleração, com linhas de aceleração semelhantes. No gráfico 4.14 a linha da aceleração está muito próxima da linear. No gráfico 4.15 a aceleração é sempre aproximadamente 1.

O algoritmo paralelo, como se pode ver pelas colunas da eficiência e nos gráficos, tem um desempenho melhor com 2 processadores e piora para 8 processadores. Como já acontecia com os dados gerados, a utilização de um maior número de processadores acarreta um aumento do tempo de comunicação que não chega a ser contrabalançado com o decréscimo do número de vértices pesquisado.

Para relacionar o número de vértices pesquisado pelo algoritmo sequencial e pelos paralelos, com o comportamento (bom ou mau) do algoritmo paralelo, separaram-se os problemas, que apresentavam o mesmo comportamento para qualquer número de processadores, em dois grupos. Um grupo contém 8 problemas que atingiram boas ou muito boas eficiências (maiores ou iguais a 0,6) e o outro com 6 problemas que atingiram más eficiências (menores que 0,6). Para estes dois grupos representa-se no gráfico 4.16 o número médio de vértices pesquisado sequencialmente e em paralelo para 2, 4 e 8 processadores.

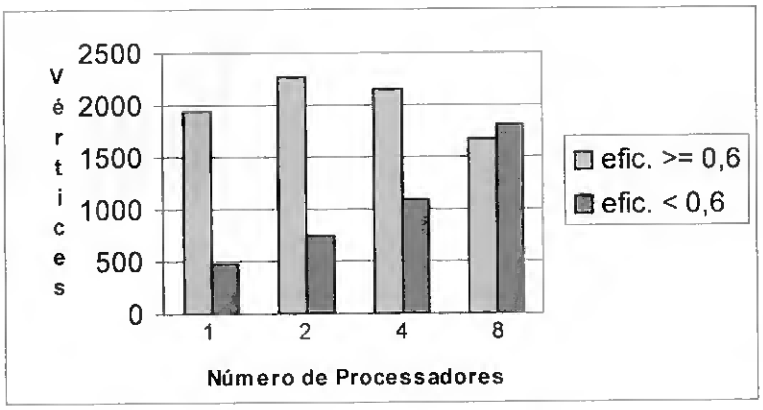


Gráfico 4.16 – Número de vértices vs eficiência

Este gráfico ilustra o que foi exposto sobre as causas das anomalias nos valores da aceleração. Para os problemas mais eficientes o número de vértices pesquisado é semelhante qualquer que seja o número de processadores utilizado. Ao contrário, para os problemas menos eficientes o número de vértices pesquisado aumenta significativamente com o aumento do número de processadores.

DISTRIBUIÇÃO DO TRABALHO PELOS PROCESSADORES

Uma vez que, para estes problemas, o comportamento do algoritmo paralelo abrangeu todas as situações (boas acelerações, aceleração superlinear, aceleração reduzidíssima) resolveu-se verificar se a distribuição de trabalho pelos processadores variava significativamente, de uns para outros, podendo contribuir para a justificação dos resultados. Voltaram-se a usar os dois grupos anteriores para comparar também a

distribuição de trabalho pelos processadores em problemas com boas e com más eficiências.

A média do número de vértices pesquisado por cada processador para cada grupo de problemas está representada nos gráficos 4.17 e 4.18. No gráfico 4.19 representam-se os valores médios considerando todos os (25) problemas.

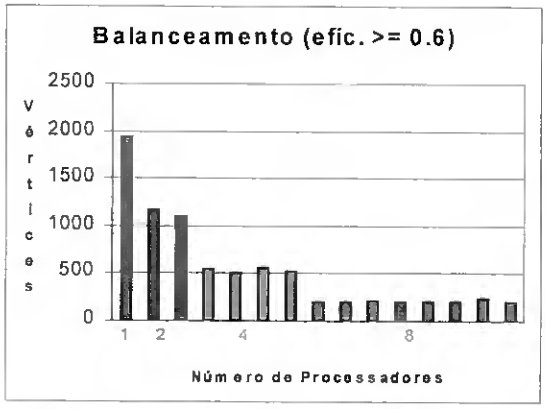


Gráfico 4.17 – Distribuição do trabalho pelos Processadores para boas eficiências

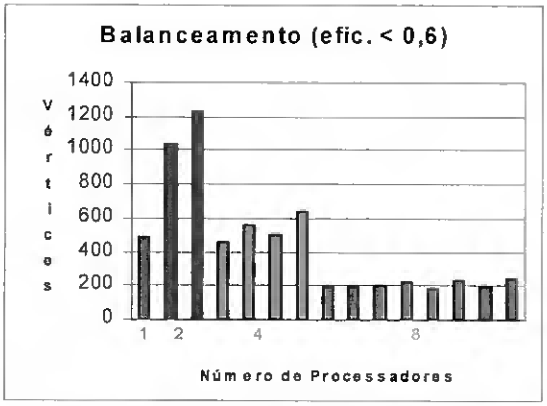


Gráfico 4.18 – Distribuição do trabalho pelos Processadores para eficiências fracas

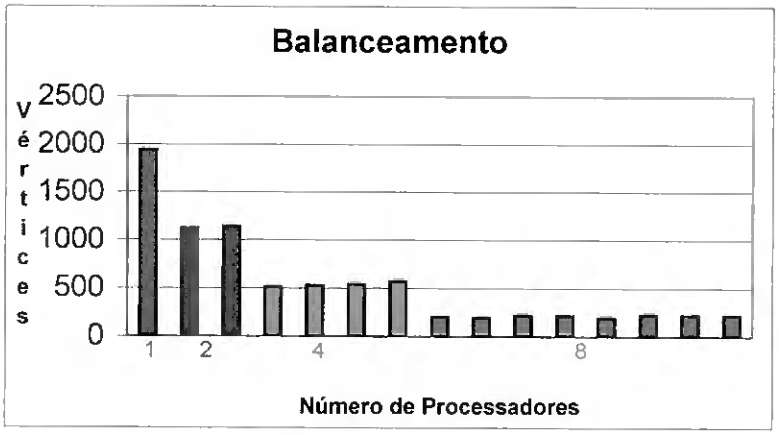


Gráfico 4.19 – Distribuição do trabalho pelos processadores

Verifica-se, nestes gráficos, que há uma distribuição equilibrada do trabalho pelos processadores, não sendo significativas as diferenças existentes entre o número médio de vértices pesquisado por cada processador, em cada caso (2, 4 ou 8 processadores). No entanto, nota-se uma maior discrepância no trabalho realizado por cada processador, principalmente para 2 e 4 processadores, no caso dos problemas que obtiveram baixas eficiências (gráfico 4.18).

4.6.3 EXPERIÊNCIA COMPUTACIONAL COM O ALGORITMO GENÉTICO SEQUENCIAL

As várias versões deste AG foram testadas nos conjuntos de problemas (gerados aleatoriamente e obtidos da Internet) utilizados em 4.6.1 e 4.6.2 e resumidos nas tabelas 4.2 e 4.11.

Costuma ser uma boa prática nos AG [Fonseca (1998)] gerar a população com uma dimensão próxima do número de variáveis do problema. Neste caso, devido a alguns problemas serem de grande dimensão (200 a 500 variáveis) e ao número máximo de variáveis livres dos problemas ser determinado inicialmente, estabeleceu-se a dimensão da população em função do número de variáveis livres. A inserção do indivíduo correspondente à solução fornecida pela heurística (algoritmo 4.4) na população inicial revelou-se ineficiente. Com efeito, rapidamente se obtinha convergência para esse indivíduo. Assim, optou-se por só considerar a sua inserção após $\frac{2}{3}$ do número total de gerações ter sido tratado. Os valores utilizados para a dimensão da população (*Pop*), número máximo de gerações a efectuar (*Maxger*) e número da geração em que é introduzida a solução heurística (*ger*) nos AG3, AG4 e AG5, encontram-se nas tabelas 4.14 e 4.15.

Probl. nº	Dados gerados aleatoriamente		
	Pop	*Maxger*	*ger*
1 - 10	20	30	21
11 - 20	40	40	27
21 - 30	60	50	34
31 - 40	60	50	34
41 - 50	80	70	47
51 - 60	80	70	47
61 - 70	100	80	54
71 - 80	100	80	54
81 - 90	100	100	67
91 - 100	100	100	67

Tabela 4.14 – Parâmetros utilizados no AG para os problemas gerados

Probl. nº	Dados da literatura		
	Pop	*Maxger*	*ger*
1 - 5	50	50	34
6 - 10	50	50	34
11 - 15	70	50	34
16 - 20	50	50	34
21 - 25	50	50	34

Tabela 4.15 – Parâmetros utilizados no AG para os problemas da literatura

Os testes computacionais que se descrevem, seguidamente, fazem um estudo comparativo das cinco versões do AG descritas em 4.5.3 e destas com a heurística (algoritmo 4.4). As diferentes versões dos AG foram implementados em Turbo C e executadas num PC com um processador Pentium MMX 200 MHz e com 16 MB de RAM.

Nas tabelas seguintes tem-se:

$$dp = \left| \frac{sol - sol^*}{sol^*} \right| \times 100 \quad , \quad \text{onde } sol \text{ é a solução obtida}$$

e sol^* é a solução óptima

n = número de variáveis

d = densidade da matriz

Pr. = número de problemas considerados

As tabelas 4.16 e 4.17 comparam, respectivamente, para os dados gerados aleatoriamente e para os dados da literatura, os valores médios obtidos para a diferença percentual, *dp*, com as várias versões do AG e com a heurística (algoritmo 4.4).

n	*d*	Pr.	Valor médio da diferença percentual					
			heur	AG1	AG2	AG3	AG4	AG5
20	1	10	15,19	4,08	1,27	2,58	1,25	1,30
40	1	10	1,42	1,63	1,56	0,06	0,91	0,06
60	0,3	10	0,45	3,95	1,66	0,40	0,34	0,20
60	0,5	10	1,04	2,14	0,85	0,33	0,23	0,09
80	0,3	10	0,48	1,76	0,64	0,31	0,12	0,11
80	0,5	10	0,05	1,15	0,51	0,05	0,01	0,00
100	0,1	10	5,94	3,90	3,70	2,10	2,70	0,75
100	0,3	9	0,49	4,00	1,10	0,30	0,30	0,00
150	0,1	9	0,45	2,59	2,01	0,22	0,26	0,06
200	0,1	8	0,33	3,30	1,52	0,29	0,31	0,08
média			2,58	2,85	1,48	0,66	0,64	0,27

Tabela 4.16 – Resultados da *dp* obtidos para os dados gerados aleatoriamente

Verifica-se, a partir de tabela 4.16, que apenas o AG1 obtém, em média, resultados piores do que a heurística, todas as outras versões mostram-se mais eficientes, no que diz respeito à qualidade do resultado. Note-se que mesmo a heurística, no geral (excepto para dois conjuntos de problemas), obtém resultados muito bons. Comparando os resultados do AG1 com os do AG2 e os resultados do AG1 com os do AG3 verifica-se que a introdução de procedimentos baseados em optimização num AG puramente aleatório provoca melhoramentos significativos. No primeiro caso trata-se da aplicação a alguns indivíduos de um operador de mutação baseado em técnicas de optimização. No segundo, é inserido na população um indivíduo obtido heuristicamente. Dos resultados obtidos com o AG5 (principalmente em relação ao AG3) ressalta que a substituição da mutação aleatória pela mutação com base em métodos de optimização também é proveitosa.

No geral, e como era esperado devido a este problema encaixar-se perfeitamente nos casos tratados com sucesso com AGs, pode-se afirmar que, para este tipo de problemas, o AG é uma meta-heurística eficiente principalmente se incluir nos seus operadores técnicas utilizadas no desenvolvimento de algoritmos exactos ou heurísticos.

n	*d*	Pr.	Valor médio da diferença percentual					
			heur	AG1	AG2	AG3	AG4	AG5
66	1	5	0,06	2,77	1,96	0,00	0,00	0,00
112	0,05	5	0,04	0,00	0,00	0,00	0,00	0,00
237	0,02	5	24,67	0,71	4,18	1,35	4,18	0,03
420	0,04	5	8,43	0,99	1,80	0,99	1,66	2,10
494	0,01	5	4,25	1,46	1,43	1,46	1,43	1,46
média			7,49	1,18	1,87	0,76	1,45	0,72

Tabela 4.17 – Resultados da *dp* obtidos para as matrizes da literatura

Nos resultados com as matrizes da literatura (tabela 4.17) fica ainda mais claro que os AG são eficientes para este tipo de problemas, mesmo o puramente aleatório. Não houve, nestes dados, vantagem em aplicar a alguns indivíduos o procedimento de pós-optimização, mas quando este substituiu sistematicamente (AG5) a mutação aleatória já os resultados foram melhores.

Nas tabelas 4.18 e 4.19 apresenta-se o número médio de gerações dos AGs em que houve melhoramento da solução incumbente.

n	*d*	Pr.	Nº médio de gerações com melhoramento da solução				
			AG1	AG2	AG3	AG4	AG5
20	1	10	14	10	15	11	4
40	1	10	23	11	25	16	9
60	0,3	10	28	13	34	32	15
60	0,5	10	28	12	32	25	12
80	0,3	10	39	17	39	21	6
80	0,5	10	30	14	38	26	3
100	0,1	10	38	25	42	32	11
100	0,3	10	37	21	46	29	4
150	0,1	10	48	17	49	30	22
200	0,1	10	42	22	48	29	6
média			33	16	37	25	9

Tabela 4.18 – Resultados para os dados gerados aleatoriamente

n	*d*	Pr.	Nº médio de gerações com melhoramento da solução				
			AG1	AG2	AG3	AG4	AG5
66	1	5	28	27	26	25	8
112	0,05	5	16	16	16	21	3
237	0,02	5	39	26	39	20	20
420	0,04	5	21	14	21	20	9
494	0,01	5	25	31	24	27	17
média			26	23	25	23	11

Tabela 4.19 – Resultados para os dados da literatura

Verifica-se, nestas tabelas, que o número médio de gerações necessárias para os AGs convergirem é reduzido e foi sempre bastante inferior ao máximo estipulado (ver tabelas 4.14 e 4.15). Isto significa que para alguns problemas poder-se-ia ter obtido os mesmos resultados num tempo inferior, se tivessem sido estabelecidos valores menores para *Maxger*. Nota-se também que este número é consideravelmente reduzido no caso do AG5 o que significa que a pós-optimização acelera a convergência do AG e ainda conduz a uma melhoria dos resultados (tabelas 4.16 e 4.17).

Nas tabelas 4.20 e 4.21, apresentam-se os tempos médios de execução obtidos.

n	*d*	Pr.	Tempo médio de execução					
			heur	AG1	AG2	AG3	AG4	AG5
20	1	10	0,01	0,10	0,09	0,10	0,09	0,18
40	1	10	0,01	0,95	0,91	0,96	0,95	1,47
60	0,3	10	0,02	3,89	3,49	4,06	3,71	6,23
60	0,5	10	0,02	4,89	4,36	5,08	4,57	7,63
80	0,3	10	0,07	16,23	15,00	17,22	15,34	24,39
80	0,5	10	0,08	16,75	15,12	17,64	15,65	23,74
100	0,1	10	0,03	15,71	14,04	16,41	14,82	26,66
100	0,3	10	0,06	32,81	28,85	35,61	31,34	48,16
150	0,1	10	0,11	60,32	52,58	61,13	52,89	92,11
200	0,1	10	0,16	109,24	98,88	110,72	99,09	163,13
média			0,06	26,09	23,33	26,89	23,84	39,37

Tabela 4.20 – Tempos médios, em segundos, para os dados gerados aleatoriamente

n	*d*	Pr.	Tempo médio de execução					
			heur	AG1	AG2	AG3	AG4	AG5
66	1	5	0,08	5,53	5,12	5,68	5,26	6,50
112	0,05	5	0,04	4,91	4,32	5,00	4,39	8,58
237	0,02	5	0,08	29,15	23,56	29,59	23,87	57,71
420	0,04	5	0,56	63,46	58,25	64,11	59,31	150,66
494	0,01	5	0,33	64,76	68,18	65,52	69,02	154,01
média			0,22	33,56	31,89	33,98	32,37	75,49

Tabela 4.21 – Tempos médios, em segundos, para as matrizes da literatura

A heurística funciona sempre com grande rapidez, o que, como é usual, não acontece com os AGs. Tendo em conta as características dos problemas, os AGs são mais demorados quando o número de variáveis aumenta e não dependem tanto do número de elementos da matriz. Isto nota-se comparando os tempos de execução dos problemas em que o número de variáveis e o número de elementos da matriz variam inversamente, como por exemplo, nos problemas dos dois últimos conjuntos da literatura. Embora o problema que tem menos variáveis tenha quase o triplo de elementos da matriz, a sua resolução demora sempre menos tempo do que a do outro (ambos têm um número semelhante de variáveis livres).

É claro que o tempo de execução do AG também depende da dimensão da população. Isto só reforça o que foi dito no parágrafo anterior, visto que, em geral, ao aumentar o número de variáveis também se aumentou a dimensão da população.

Os tempos de execução das várias versões do AG são semelhantes excepto os do AG5. Este último é mais demorado que os outros devido ao procedimento de pós-optimização, que substitui a mutação aleatória, ser mais demorado. No entanto, mesmo os tempos de execução obtidos com esta versão mais demorada do AG são bons, demorando os piores casos (problemas com 494 variáveis) menos de 3 minutos.

O tempo necessário para gerar a população inicial não variou ou variou muito pouco para os vários problemas da mesma dimensão. Em média representou 5% do

tempo total de execução do AG, variando entre 0%, apenas para os problemas de dimensão 20, e 9 % do tempo total de execução.

Na tabela 4.22 comparam-se os tempos de execução do AG5 (a melhor versão dos AGs mas também a menos rápida) com os tempos de execução do algoritmo exacto de pesquisa em árvore (APA) descrito em 4.5.1 cujos resultados se encontram em 4.6.1. Inclui-se, nesta tabela, a coluna rácio que contem o quociente entre os tempos de execução do APA e do AG5, indicando quantas vezes o APA é mais demorado do que o AG5.

Fica claro que o AG5 é, para a generalidade dos casos, muito mais rápido do que o APA.

Na tabela 4.22 verifica-se também que os tempos de execução do algoritmo exacto têm uma variação enorme, ao contrário do que se passa com o AG em que as diferenças de tempo de execução de problema para problema não são tão diferentes e têm a ver, principalmente, com o número de variáveis e de variáveis livres do problema. O APA mesmo para problemas com a mesma dimensão e um número semelhante de variáveis fixas obtém nalguns casos tempos muito bons (menos de um minuto) e noutros muito maus (algumas horas). Assim, se não for absolutamente necessário o conhecimento da solução óptima é mais interessante aplicar o AG que obteve sempre soluções óptimas ou quase óptimas num tempo de execução bastante aceitável.

n	*d*	Pr.		Tempo		
				APA	AG5	racio
			Melhor(a)	0,11	0,11	1,00
20	1	10	Média	**0,32**	**0,18**	**1,78**
			Pior(a)	0,50	0,33	1,52
			Melhor(b)	0,99	0,66	1,50
40	1	10	Média	**152,06**	**1,47**	**103,44**
			Pior(b)	1396,32	4,01	348,21
			Melhor(a)	4,45	4,53	0,98
	0,3	10	Média	**58,98**	**6,23**	**9,47**
60			Pior(a)	186,15	13,19	14,11
			Melhor(a)	1,92	3,24	0,59
	0,5	10	Média	**600,23**	**7,63**	**78,67**
			Pior(a)	2759,23	15,99	172,56
			Melhor(a)	1,98	16,26	0,12
	0,3	10	Média	**1233,62**	**24,38**	**50,60**
80			Pior(a)	11157,69	52,31	213,30
			Melhor(a)	0,77	15,88	0,05
	0,5	10	Média	**15,04**	**23,74**	**0,63**
			Pior(a)	69,89	62,09	1,13
			Melhor(b)	2,09	10,77	0,19
	0,1	10	Média	471,60	26,66	**17,69**
100			Pior(a)	4310,28	40,77	105,72
			Melhor(a)	2,20	29,07	0,08
	0,3	9	Média	850,96	48,16	**17,67**
			Pior(a)	5044,07	120,61	41,82
			Melhor(b)	2,97	62,64	0,05
150	0,1	9	Média	**5236,55**	**92,10**	**56,86**
			Pior(a)	34855,00	239,51	145,53
			Melhor(b)	9,23	78,52	0,12
200	0,1	8	Média	**3792,29**	**163,13**	**23,25**
			Pior(b)	26533,20	505,66	52,47

Tempo = tempo de execução em segundos

(a) os resultados para os dois algoritmos não correspondem ao mesmo problema

(b) os resultados para os dois algoritmos correspondem ao mesmo problema

Tabela 4.22 – Comparação entre os tempos de execução do algoritmo de PA e do AG5

No gráfico 4.20 representam-se os valores da coluna rácio.

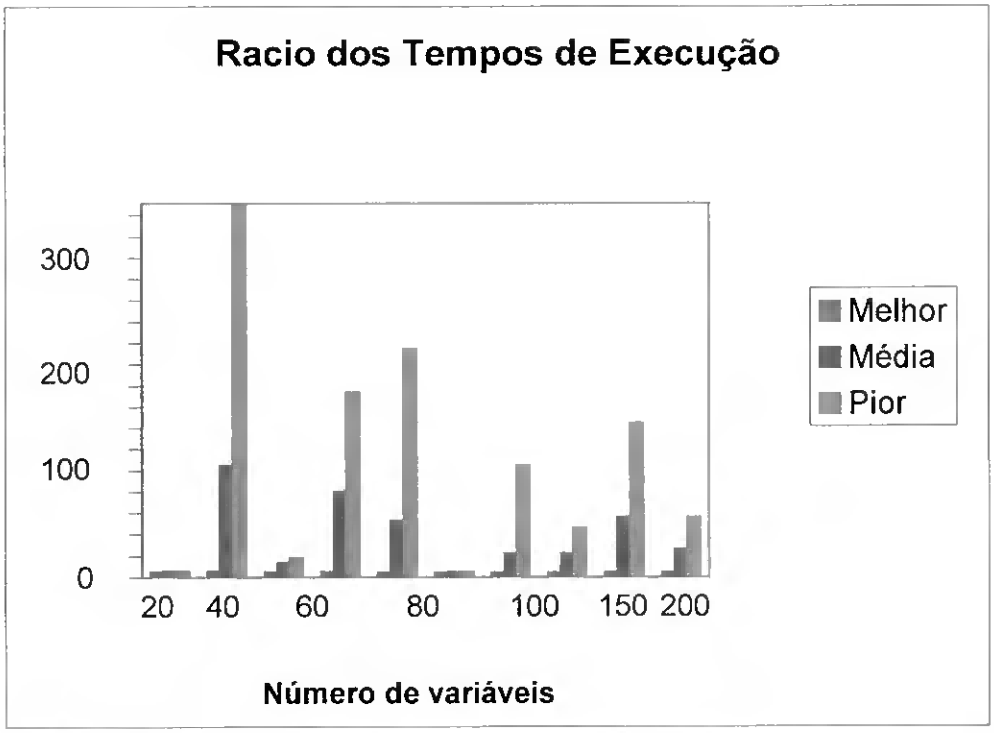


Gráfico 4.20 – Comparação entre os tempos de execução do algoritmo de PA e do AG5

Enquanto que, para os melhores valores obtidos o tempo de execução do APA é semelhante ou menor do que o do AG5, para os valores médios e piores é bastante maior. A única excepção ocorreu nos problemas de dimensão 80 e densidade 0,5 que têm a particularidade de terem sido resolvidos exactamente com árvores de pequena dimensão. Esta característica é também comum aos problemas que obtiveram o melhor tempo de execução.

4.6.4 EXPERIÊNCIA COMPUTACIONAL COM O ALGORITMO GENÉTICO PARALELO

Como se expôs, no capítulo 3, a paralelização de um AG pode ser feita visando um de dois objectivos:

1 - Diminuir o tempo de execução;

2 - Aumentar a diversidade da população.

Para atingir o objectivo 1 repartiu-se a dimensão da população (*Pop*) do AG5 sequencial pelos processadores e como o número máximo de gerações (*Maxger*) a efectuar deve estar de acordo com a dimensão da população também se dividiu esse número utilizado no sequencial pelo número de processadores. Assim, cada processador gerou aleatoriamente Pop/p indivíduos e efectuou $Maxger/p$ gerações, com $p = 2, 4 \text{ ou } 8$.

Para alcançar o objectivo 2 o AG5, efectuou, em cada processador, o mesmo número de gerações do sequencial e trabalhou com uma população da mesma dimensão do sequencial, sabendo-se à priori que desta forma o tempo de execução aumentaria.

Os valores das sementes que inicializam, em cada processador, a sequência de números aleatórios, são diferentes para que as várias populações iniciais sejam distintas.

Para o objectivo 2 estabeleceu-se em cinco o número de indivíduos a trocar entre processadores. Para o objectivo 1 estabeleceu-se em cinco e em três indivíduos consoante fossem utilizados 2 ou 4 e 8 processadores. Estes indivíduos são escolhidos aleatoriamente entre as populações de cada processador.

Analisam-se separadamente os resultados obtidos para cada objectivo. Nas tabelas e gráficos destes resultados, de acordo com as topologias descritas em 4.5.4.1, as colunas de 4 e 8 processadores referem-se às topologias top4 e top8 respectivamente e as colunas de 4c e 8c processadores referem-se respectivamente às topologias top4c e top8c, que são as que executam mais comunicações entre processadores.

4.6.4.1 DIMINUIR O TEMPO DE EXECUÇÃO (DTE)

Fixou-se em 10 indivíduos a dimensão mínima da população, por processador, por isso, os problemas menores só foram testados em 2 ou em 2 e 4 processadores. Nas tabelas 4.23 e 4.24 apresentam-se os resultados obtidos em relação à *dp*.

Diferença percentual										
n	*d*	Pr.		Processadores					
				1	2	4	8	4c	8c
20	1	10	Melhor	0	0				
			Média	**1,3**	**1,72**				
			Pior	7,79	7,79				
40	1	10	Melhor	0	0	0		0	
			Média	**0,06**	**0,16**	**0,06**		**0,06**	
			Pior	0,56	0,98	0,56		0,56	
60	0,3	10	Melhor	0	0	0		0	
			Média	**0,2**	**0,12**	**0,07**		**0,09**	
			Pior	1,38	0,42	0,4		0,4	
	0,5	10	Melhor	0	0	0		0	
			Média	**0,09**	**0,08**	**0,1**		**0,13**	
			Pior	0,51	0,43	0,51		0,51	
80	0,3	10	Melhor	0	0	0	0	0	0
			Média	**0,11**	**0,11**	**0,01**	**0,13**	**0,11**	**0,11**
			Pior	1,03	1,03	0,11	1,22	1,03	1,11
	0,5	10	Melhor	0	0	0	0	0	0
			Média	**0**	**0,02**	**0**	**0**	**0**	**0**
			Pior	0	0,16	0	0	0	0
100	0,1	10	Melhor	0	0	0	0	0	0
			Média	**0,75**	**0,48**	**0,78**	**1,54**	**0,73**	**0,88**
			Pior	3,75	2,17	2,02	4,51	2,02	3,79
	0,3	9	Melhor	0	0	0	0	0	0
			Média	**0**	**0,11**	**0,05**	**0,02**	**0**	**0,05**
			Pior	0	0,59	0,42	0,19	0	0,42
150	0,1	9	Melhor	0	0	0	0	0	0
			Média	**0,06**	**0,08**	**0,06**	**0,1**	**0,01**	**0,1**
			Pior	0,28	0,26	0,26	0,29	0,09	0,37
200	0,1	8	Melhor	0	0	0	0	0	0
			Média	**0,08**	**0,02**	**0,09**	**0,13**	**0,05**	**0,09**
			Pior	0,51	0,16	0,42	0,66	0,42	0,42

Tabela 4.23 – Resultados da *dp* com o AG paralelo para os dados gerados

A paralelização não apresenta o mesmo tipo de resultado para todas as instâncias geradas aleatoriamente (tabela 4.23). Para as de menor dimensão (até 40) a utilização de 2 processadores piorou os resultados. Para as restantes, em geral, melhorou, embora para os problemas de dimensão 80 e 100 de maior densidade também tenha obtido resultados ligeiramente piores. Com 4 processadores os resultados foram quase todos melhorados, ou em média ou pelo menos o pior valor, excepto para os problemas de dimensão 100 e densidade 0,3 . Como é natural, a introdução de mais comunicações com troca de indivíduos entre os 4 processadores só trouxe vantagem para os problemas de maior dimensão (mais de 100 variáveis). A utilização de 8 processadores com mais ou menos comunicações entre eles não resultou em melhoramentos dos valores obtidos sequencialmente.

Os gráficos 4.21 a 4.24 mostram os valores médios obtidos para a *dp*.

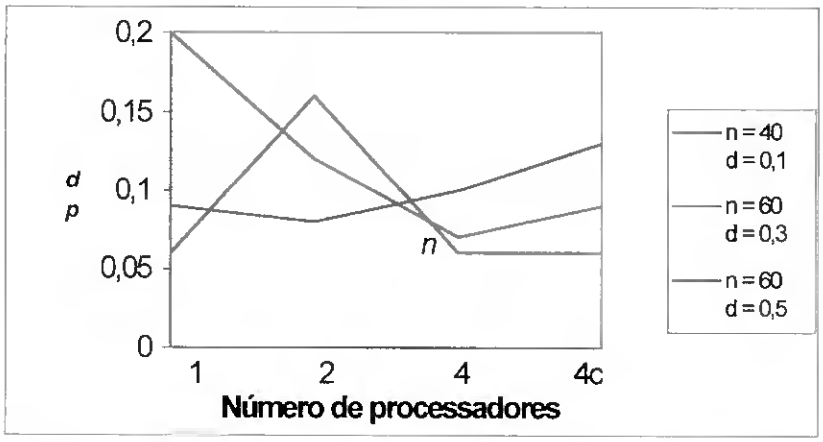


Gráfico 4.21 – Valores médios da *dp* para dimensão inferior ou igual a 60

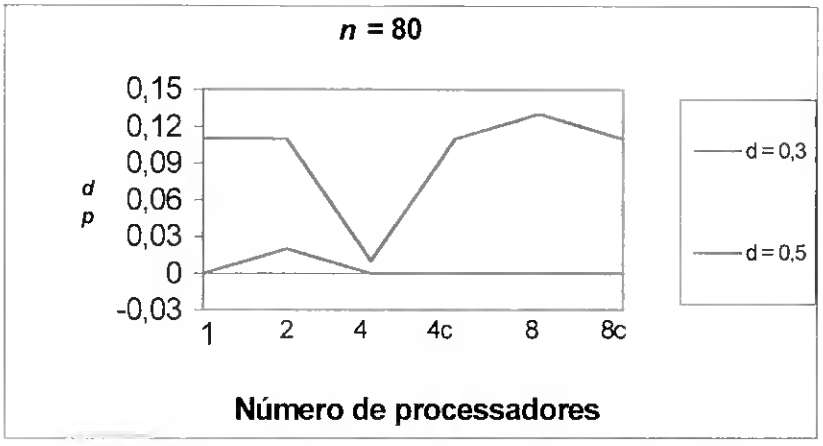


Gráfico 4.22 – Valores médios da *dp* para dimensão igual a 80

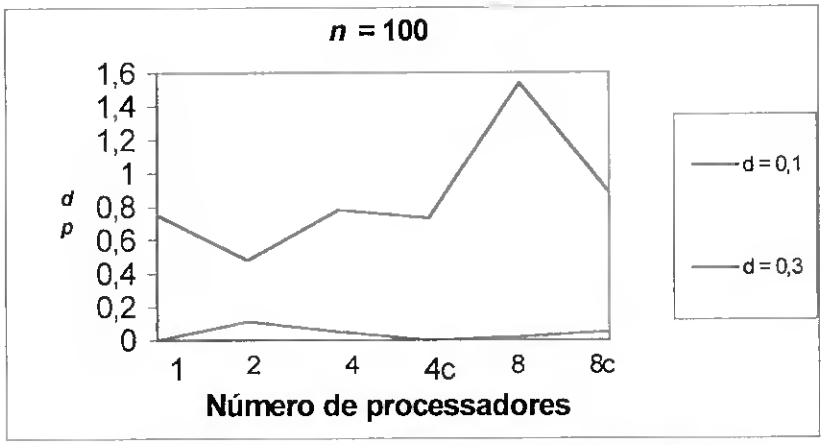


Gráfico 4.23 – Valores médios da *dp* para dimensão igual a 100

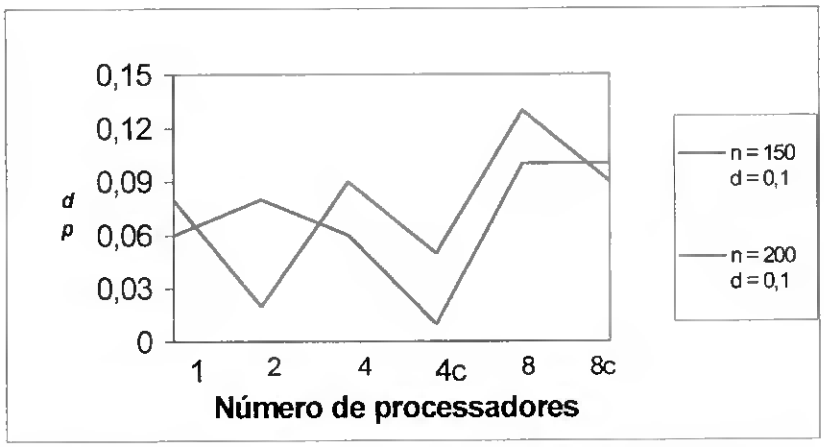


Gráfico 4.24 – Valores médios da *dp* para dimensão igual a 150 e 200

Destes gráficos ressalta sobretudo que os resultados da paralelização não são homogéneos para os vários problemas.

Para o conjunto de matrizes da literatura (tabela 4.24) os resultados obtidos com os vários processadores apresentaram a mesma tendência, tiveram resultados melhores e piores para os mesmos problemas, com um desempenho ligeiramente pior para 4 processadores. Por outro lado para o último conjunto de problemas ($n = 494$) a utilização de 2 processadores trouxe bons melhoramentos ao valor da *dp*.

Diferença percentual							
n	*d*	Pr.		Processadores			
				1	2	4	4c
66	1	5	Melhor	0	0	0	0
			Média	**0**	**0**	**0**	**0**
			Pior	0	0	0	0
112	0,05	5	Melhor	0	0	0	0
			Média	**0**	**0**	**0**	**0**
			Pior	0,01	0,01	0	0,01
237	0,02	5	Melhor	0	0	0,02	0,02
			Média	**0,03**	**0,73**	**1,41**	**0,08**
			Pior	0,09	3,48	3,55	0,13
420	0,04	5	Melhor	1,38	0,69	0,69	0,71
			Média	**2,1**	**2,11**	**2,35**	**2,17**
			Pior	2,62	3,84	3,63	3,95
494	0,01	5	Melhor	1,11	0,29	1,14	0,92
			Média	**1,46**	**0,93**	**1,56**	**1,45**
			Pior	2,13	1,31	1,93	1,85

Tabela 4.24 – Resultados do AG paralelo para as matrizes da literatura

No gráfico 4.25 apresentam-se os valores médios obtidos para a *dp* com a paralelização.

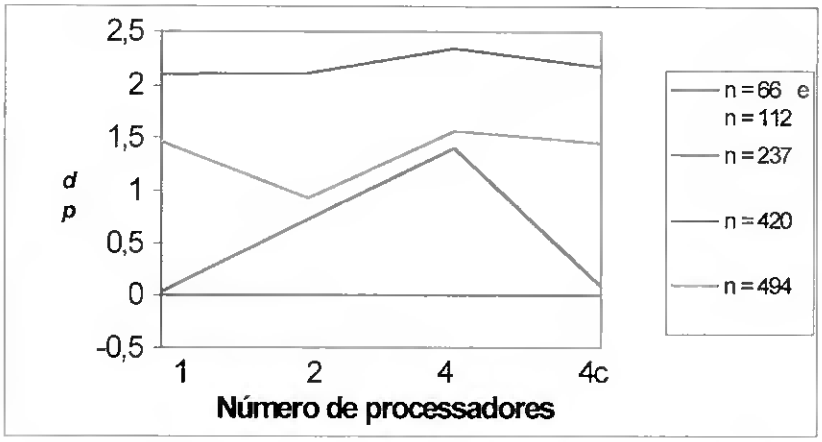


Gráfico 4.25 – Valores médios da *dp* para as matrizes da literatura

Neste gráfico nota-se a mesma tendência para 4 processadores, quer se tenha mais ou menos comunicações, piorou com uma comunicação e melhorou com duas comunicações por processador. Os resultados com 2 processadores dependeram do problema, em alguns melhorou e noutros piorou.

Nas tabelas 4.25 e 4.26 apresentam-se os resultados obtidos com esta paralelização em relação à aceleração.

É evidente que, neste tipo de paralelização, devido ao facto de cada processador tratar um número de gerações bastante inferior ao algoritmo sequencial (cerca de metade, um quarto ou um oitavo consoante sejam 2, 4 ou 8 processadores), quanto maior é o número de processadores menor é o trabalho executado, provocando maiores acelerações e muito superiores à aceleração linear. Assim, as acelerações obtidas com 2, 4 e 8 processadores oscilam entre 2 e 3 ; 7 e 12 ; 25 e 35 respectivamente (tabelas 4.25 e 4.26). Em geral as acelerações obtidas para os dados da literatura (tabela 4.26) foram ligeiramente inferiores às obtidas com os dados gerados aleatoriamente (tabela 4.25). Mesmo assim, pode-se considerar o comportamento do algoritmo paralelo bastante homogéneo e pouco dependente do problema, principalmente para 2 e 4 processadores. As melhores e piores acelerações obtidas não dependeram nem da dimensão nem da densidade dos problemas. Também não houve aumento considerável de tempo de execução quando se passaram a fazer mais comunicações com troca de indivíduos entre os processadores (4c e 8c).

Aceleração									
n	*d*	Pr.		Processadores					
				2	4	8	4c	8c	
20	1	10	Melhor	3,33					
			Média	**2,92**					
			Pior	2,5					
40	1	10	Melhor	3,46	11,26		10,95		
			Média	**3,16**	**10,47**		**10,47**		
			Pior	2,98	9,82		10,17		
60	0,3	10	Melhor	2,99	10,4		10,04		
			Média	**2,82**	**9,55**		**9,49**		
			Pior	2,65	9,18		9,3		
	0,5	10	Melhor	2,94	10,14		10,09		
			Média	**2,76**	**9,13**		**9,36**		
			Pior	2,58	8,58		8,84		
80	0,3	10	Melhor	2,69	10,04	29,24	9,93	31,3	
			Média	**2,55**	**9,41**	**28,07**	**9,45**	**30,04**	
			Pior	2,25	8,67	26,82	8,73	28,97	
	0,5	10	Melhor	2,75	10,01	29,28	9,81	31,46	
			Média	**2,56**	**9,26**	**27,41**	**9,25**	**29,38**	
			Pior	2,33	8,87	25,05	8,68	26,74	
100	0,1	10	Melhor	2,83	11,31	34,37	11,23	34,01	
			Média	**2,68**	**10,52**	**31,29**	**10,19**	**30,6**	
			Pior	2,55	9,81	28,72	9,49	27,32	
	0,3	10	Melhor	2,91	12,34	35,03	11,6	33,33	
			Média	**2,66**	**10,16**	**29,64**	**10,1**	**29,32**	
			Pior	2,42	8,97	26,9	9,08	26,21	
150	0,1	10	Melhor	2,61	10,08	30,98	10,02	30,18	
			Média	**2,52**	**9,54**	**28,5**	**9,35**	**27,83**	
			Pior	2,37	8,84	26,75	8,55	26,45	
200	0,1	10	Melhor	2,7	10,68	31,5	10,5	32,54	
			Média	**2,55**	**9,67**	**28,43**	**9,59**	**28,58**	
			Pior	2,39	8,73	26,83	8,7	26,74	

Tabela 4.25 – Resultados da aceleração para os dados gerados aleatoriamente

Aceleração						
n	*d*	Pr.		Processadores		
				2	4	4c
66	1	5	Melhor	2,33	8,75	8,4
			Média	**2,22**	**8,24**	**7,84**
			Pior	2,02	7,83	7,47
112	0,05	5	Melhor	2,98	11,56	11,7
			Média	**2,62**	**9,73**	**9,77**
			Pior	2,41	8,26	7,87
237	0,02	5	Melhor	2,62	9,49	9,41
			Média	**2,46**	**9**	**8,91**
			Pior	2,24	8	8,06
420	0,04	5	Melhor	2,29	9,02	9,09
			Média	**2,11**	**8,58**	**8,49**
			Pior	1,98	8,26	8,01
494	0,01	5	Melhor	2,23	8,83	9
			Média	**2,19**	**8,56**	**8,57**
			Pior	2,14	8,33	8,18

Tabela 4.26 – Resultados da aceleração para as matrizes da literatura

Os gráficos 4.26 a 4.28 ilustram claramente as considerações anteriores.

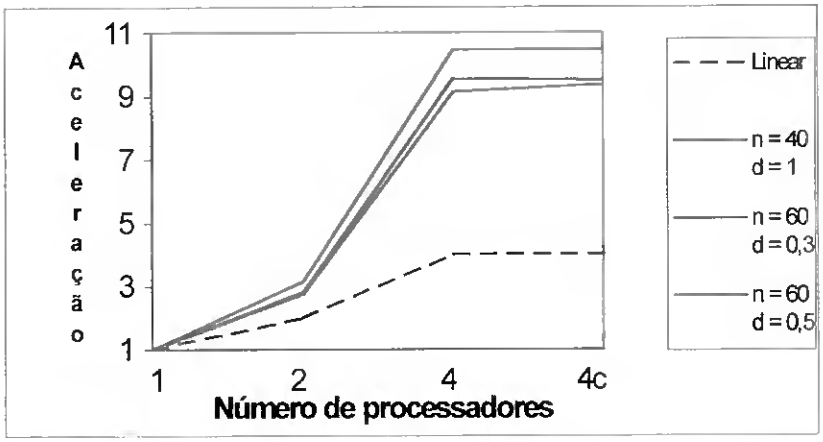


Gráfico 4.26 – Valores médios da aceleração para dimensão inferior ou igual a 60

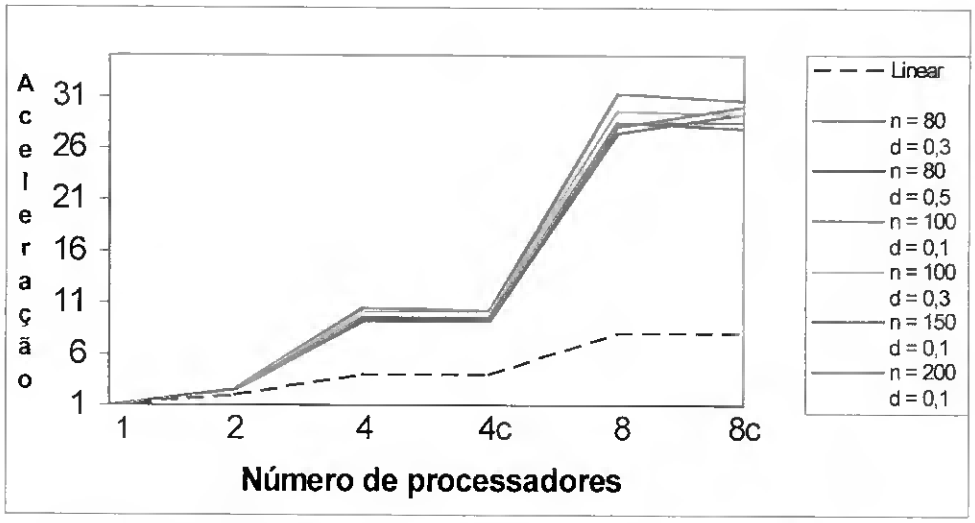


Gráfico 4.27 – Valores médios da aceleração para dimensão superior a 60

Neste gráfico verifica-se que para 4 processadores não houve alteração significativa das linhas da aceleração quando se introduziram mais comunicações. Para 8 processadores já se notam maiores diferenças tanto para melhor como para pior.

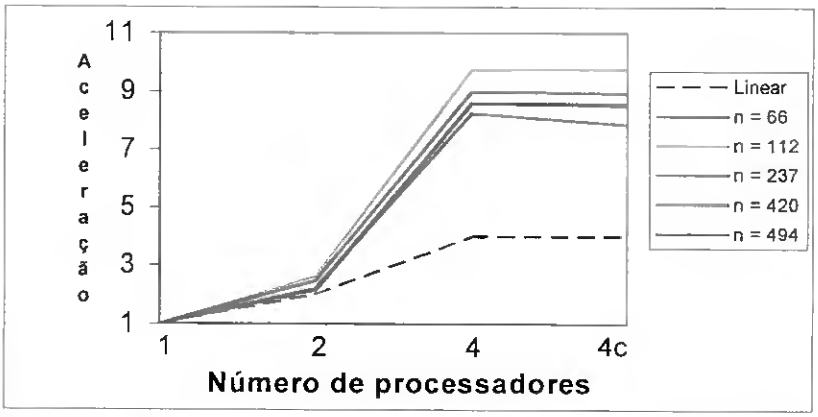


Gráfico 4.28 – Valores médios da aceleração para as matrizes da literatura

Estes três gráficos apresentam uma representação das acelerações médias muito semelhante, para todos os problemas. Para 2 processadores a aceleração é ligeiramente superior à linear. Para 4 e 8 processadores é sempre muito superior à linear.

Perante estes resultados pode-se concluir ser vantajosa a paralelização deste algoritmo para este tipo de problema, pois os tempos de execução são drasticamente reduzidos (para 4 e 8 processadores) e as deteriorações sofridas por algumas soluções foram relativamente insignificantes. Numa grande parte das instâncias a qualidade da solução manteve-se ou melhorou.

4.6.4.2 AUMENTAR A DIVERSIDADE DA POPULAÇÃO (ADP)

Para este estudo os processadores trataram todos o mesmo número de gerações e trabalharam com a mesma população que o algoritmo sequencial. Nas tabelas 4.27 e 4.28 apresentam-se os resultados obtidos em relação à *dp*, respectivamente para os dados gerados aleatoriamente e para as matrizes da literatura.

Como era de esperar, esta forma de paralelismo aumenta a diversidade da população e consequentemente melhora os resultados obtidos sequencialmente. De facto, passaram-se a ter várias subpopulações todas com a mesma dimensão da população do algoritmo sequencial num total de indivíduos igual ao número de processadores vezes a dimensão da população sequencial. Assim, na maioria dos problemas testados, tanto para os gerados como para os da literatura, os valores da *dp* melhoraram. Mantiveram-se quando as soluções já eram as óptimas (ou quase óptimas). Foram muito poucos aqueles em que existiu deterioração da solução, mas mesmo nestes casos foi insignificante. De uma forma geral, também os resultados melhoraram com o aumento do número de processadores. No entanto, o aumento de comunicações, (para 4 e 8 processadores) não obteve resultados homogéneos, tendo nuns casos melhorado e noutros piorado.

Estas constatações podem ser facilmente confirmadas nos gráficos 4.29 a 4.32 que representam os valores médios obtidos.

Diferença percentual										
n	*d*	Pr.		Processadores						
				1	2	4	8	4c	8c
20	1	10	Melhor	0	0	0	0	0	0
			Média	**1,3**	**0,92**	**1,62**	**1,21**	**0,55**	**0,89**
			Pior	7,79	8,04	7,79	7,79	4,33	7,79
40	1	10	Melhor	0	0	0	0	0	0
			Média	**0,06**	**0**	**0,06**	**0,06**	**0,06**	**0**
			Pior	0,56	0	0,56	0,56	0,56	0
60	0,3	10	Melhor	0	0	0	0	0	0
			Média	**0,2**	**0,07**	**0,04**	**0**	**0,01**	**0,01**
			Pior	1,38	0,28	0,25	0	0,14	0,14
	0,5	10	Melhor	0	0	0	0	0	0
			Média	**0,09**	**0,1**	**0**	**0**	**0**	**0,04**
			Pior	0,51	0,51	0	0	0	0,43
80	0,3	10	Melhor	0	0	0	0	0	0
			Média	**0,11**	**0,12**	**0**	**0**	**0,12**	**0**
			Pior	1,03	1,11	0,04	0,04	1,11	0,04
	0,5	10	Melhor	0	0	0	0	0	0
			Média	**0**	**0**	**0**	**0**	**0**	**0**
			Pior	0	0	0	0	0	0
100	0,1	10	Melhor	0	0	0	0	0	0
			Média	**0,75**	**0,36**	**0,31**	**0,13**	**0,39**	**0,09**
			Pior	3,75	1,82	1,67	0,67	1,82	0,67
	0,3	9	Melhor	0	0	0	0	0	0
			Média	**0**	**0,05**	**0**	**0**	**0,01**	**0**
			Pior	0	0,42	0	0	0,12	0
150	0,1	9	Melhor	0	0	0	0	0	0
			Média	**0,06**	**0,01**	**0,01**	**0,01**	**0,01**	**0,02**
			Pior	0,28	0,09	0,09	0,09	0,09	0,1
200	0,1	8	Melhor	0	0	0	0	0	0
			Média	**0,08**	**0,05**	**0,05**	**0,02**	**0**	**0**
			Pior	0,51	0,4	0,4	0,16	0	0

Tabela 4.27 – Resultados da *dp* com o AG paralelo para os dados gerados

Diferença percentual									
n	*d*	Pr.		Processadores					
				1	2	4	8	4c	8c
66	1	5	Melhor	0	0	0	0	0	0
			Média	**0**	**0**	**0**	**0**	**0**	**0**
			Pior	0	0	0	0	0	0
112	0,05	5	Melhor	0	0	0	0	0	0
			Média	**0**	**0**	**0**	**0**	**0**	**0**
			Pior	0,01	0	0	0	0	0
237	0,02	5	Melhor	0	0	0	0	0	0
			Média	**0,03**	**0,7**	**0,01**	**0**	**0,01**	**0**
			Pior	0,09	3,49	0,02	0,02	0,02	0
420	0,04	5	Melhor	1,38	0	0,67	0	0	0
			Média	**2,1**	**0,98**	**1,23**	**0,27**	**0,67**	**0,68**
			Pior	2,62	1,47	2,63	0,7	1,3	2,02
494	0,01	5	Melhor	1,11	0,29	0,29	0,29	0,2	0
			Média	**1,46**	**0,82**	**0,66**	**0,6**	**0,57**	**0,37**
			Pior	2,13	1,07	0,82	1,06	1,01	0,9

Tabela 4.28 – Resultados da *dp* com o AG paralelo para os dados da literatura

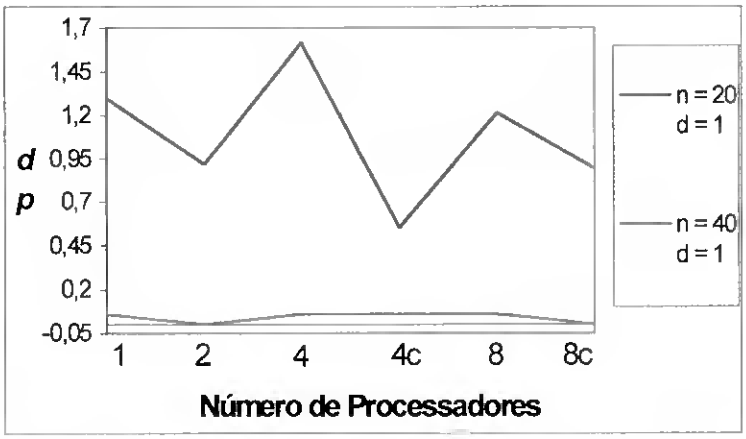


Gráfico 4.29 – Valores médios da *dp* para dimensão menor ou igual a 40

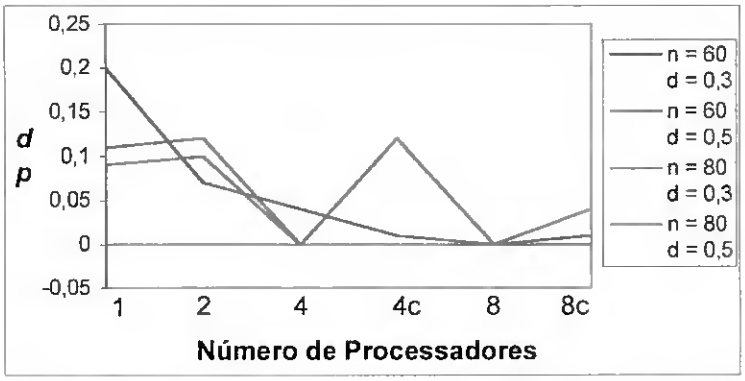


Gráfico 4.30 – Valores médios da *dp* para dimensão igual a 60 e a 80

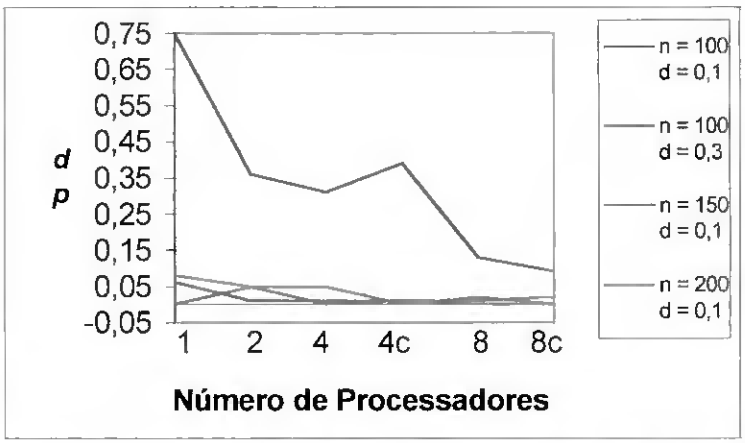


Gráfico 4.31 – Valores médios da *dp* para dimensão igual a 100, 150 e 200

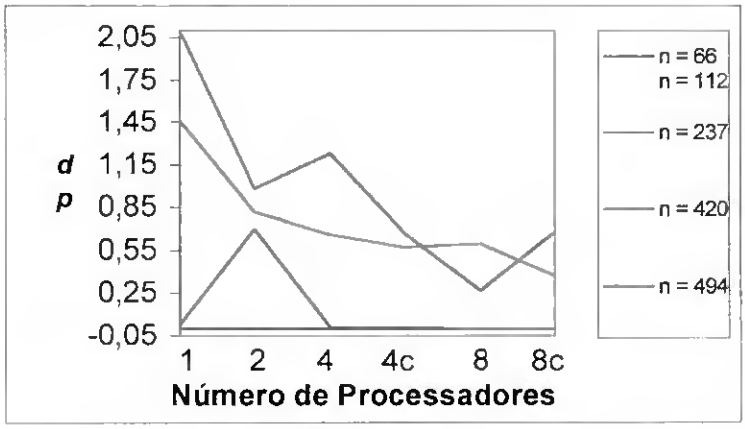


Gráfico 4.32 – Valores médios da *dp* para os dados da literatura

Nas tabelas 4.29 e 4.30 apresentam-se os resultados obtidos em relação à aceleração. De facto, ao manter cada processador com o mesmo trabalho do sequencial está-se a aumentar o trabalho total, consequentemente não se obtêm acelerações mas deteriorações do tempo de execução sequencial.

Aceleração								
n	*d*	Pr.		Processadores				
				2	4	8	4c	8c
20	1	10	Melhor	0,87	0,85	0,83	0,84	0,88
			Média	**0,82**	**0,8**	**0,77**	**0,79**	**0,82**
			Pior	0,73	0,75	0,72	0,74	0,73
40	1	10	Melhor	0,93	0,9	0,91	0,93	0,88
			Média	**0,8**	**0,79**	**0,78**	**0,79**	**0,76**
			Pior	0,73	0,67	0,71	0,72	0,64
60	0,3	10	Melhor	0,78	0,77	0,77	0,77	0,76
			Média	**0,75**	**0,74**	**0,73**	**0,73**	**0,73**
			Pior	0,69	0,69	0,69	0,69	0,68
	0,5	10	Melhor	0,8	0,81	0,77	0,79	0,8
			Média	**0,74**	**0,72**	**0,71**	**0,71**	**0,71**
			Pior	0,67	0,66	0,66	0,64	0,63
80	0,3	10	Melhor	0,71	0,71	0,71	0,71	0,7
			Média	**0,67**	**0,66**	**0,66**	**0,67**	**0,66**
			Pior	0,59	0,57	0,56	0,59	0,56
	0,5	10	Melhor	0,74	0,7	0,7	0,73	0,71
			Média	**0,67**	**0,66**	**0,66**	**0,66**	**0,66**
			Pior	0,6	0,59	0,59	0,59	0,59
100	0,1	10	Melhor	0,74	0,77	0,72	0,74	0,7
			Média	**0,69**	**0,69**	**0,67**	**0,68**	**0,64**
			Pior	0,62	0,62	0,58	0,61	0,56
	0,3	10	Melhor	0,75	0,78	0,73	0,78	0,7
			Média	**0,68**	**0,68**	**0,67**	**0,68**	**0,66**
			Pior	0,62	0,62	0,62	0,62	0,6
150	0,1	10	Melhor	0,68	0,68	0,67	0,68	0,65
			Média	**0,65**	**0,65**	**0,64**	**0,65**	**0,63**
			Pior	0,6	0,59	0,59	0,59	0,57
200	0,1	10	Melhor	0,7	0,69	0,68	0,7	0,67
			Média	**0,67**	**0,67**	**0,66**	**0,67**	**0,65**
			Pior	0,63	0,63	0,62	0,62	0,61

Tabela 4.29 – Resultados da aceleração para os dados gerados aleatoriamente

Aceleração								
n	*d*	Pr.		Processadores				
				2	4	8	4c	8c
66	1	5	Melhor	0,64	0,63	0,62	0,63	0,63
			Média	**0,61**	**0,61**	**0,59**	**0,59**	**0,59**
			Pior	0,57	0,57	0,53	0,54	0,55
112	0,05	5	Melhor	0,76	0,74	0,72	0,74	0,71
			Média	**0,7**	**0,7**	**0,67**	**0,68**	**0,64**
			Pior	0,66	0,66	0,64	0,64	0,59
237	0,02	5	Melhor	0,74	0,73	0,71	0,73	0,71
			Média	**0,69**	**0,67**	**0,67**	**0,68**	**0,66**
			Pior	0,63	0,63	0,62	0,62	0,6
420	0,04	5	Melhor	0,63	0,64	0,63	0,61	0,62
			Média	**0,6**	**0,61**	**0,59**	**0,59**	**0,57**
			Pior	0,55	0,58	0,55	0,56	0,54
494	0,01	5	Melhor	0,64	0,61	0,6	0,61	0,57
			Média	**0,6**	**0,59**	**0,56**	**0,58**	**0,56**
			Pior	0,58	0,56	0,52	0,55	0,55

Tabela 4.30 – Resultados da aceleração para as matrizes da literatura

Verifica-se que os tempos de execução vão aumentando com o número de processadores e com a dimensão do problema. No entanto o comportamento do algoritmo paralelo é bastante homogéneo. Os tempos de execução vão de 1.2 a 2 vezes o tempo de execução do algoritmo sequencial o que leva a desacelerações de 0,9 a 0,5. Considerando que o trabalho efectuado em paralelo é 2, 4 ou 8 vezes o efectuado pelo sequencial, os tempos de execução obtidos são bastante razoáveis (no máximo 2 vezes o tempo do sequencial).

Nos gráficos 4.33 e 4.34 representam-se os valores médios obtidos para a aceleração.

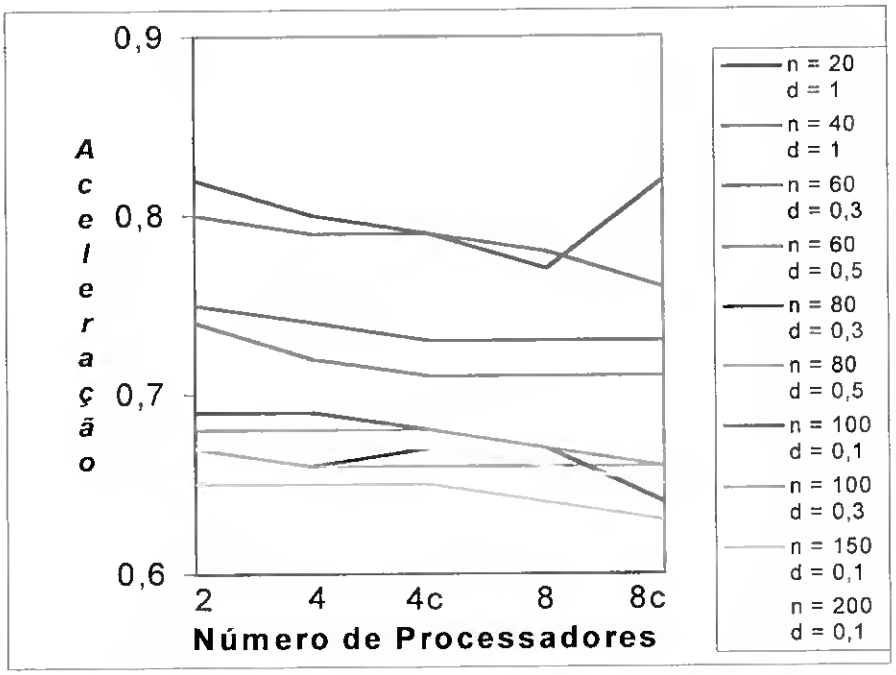


Gráfico 4.33 – Valores médios da aceleração para os dados gerados

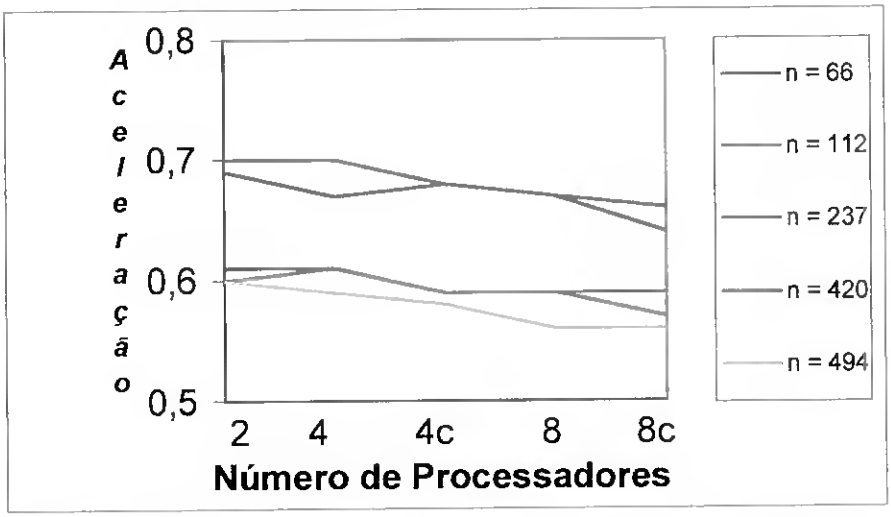


Gráfico 4.34 – Valores médios da aceleração para as matrizes da literatura

Constata-se que as acelerações oscilam muito pouco (menos de duas décimas para cada um dos tipos de dados). Tendencialmente pioram com o aumento do número de processadores e para o mesmo número de processadores pioram com o aumento das comunicações.

4.6.4.3 ANÁLISE COMPARATIVA DAS PARALELIZAÇÕES

Dos resultados obtidos com estas duas formas de paralelização, pode-se concluir que as duas trazem melhoramentos ao algoritmo sequencial.

A primeira traduz-se num melhoramento muito significativo dos tempos de execução (principalmente com 8 e 4 processadores) ficando a execução do algoritmo paralelo, em média, para 2, 4 e 8 processadores cerca de, respectivamente, 2.5, 10 e 30 vezes mais rápida do que a do algoritmo sequencial, com a contrapartida de uma eventual pequena deterioração da qualidade da solução. Realmente, com o algoritmo paralelo, o valor médio da *dp*, foi sempre menor que 1,7 para os dados gerados e que 2,4 para os da literatura, enquanto que com o sequencial foi sempre, respectivamente, menor que 1,3 e 2,1 .

A segunda paralelização melhora os valores médios da solução entre 21% a 60% e entre 30% a 75%, respectivamente, nos dados gerados e da literatura. Em contrapartida os tempos de execução aumentam de 1,2 a 2 vezes.

Nas tabelas 4.31 a 4.34 e nos gráficos 4.35 a 4.38 comparam-se os valores médios para os dois tipos de dados e de paralelizações.

Valores médios da diferença percentual								
	Problemas gerados	Parale-lização	Processadores					
			1	2	4	4c	8	8c
(A)	11 - 40	DTE	0,12	0,12	0,08	0,09		
(B)		ADP	0,12	0,06	0,03	0,02		
(C)	41 - 100	DTE	0,17	0,14	0,17	0,15	0,32	0,21
(D)		ADP	0,17	0,1	0,06	0,09	0,03	0,02

Tabela 4.31 – Valores médios da *dp* segundo o tipo de paralelismo para os dados gerados aleatoriamente

Valores médios da diferença percentual						
	Problemas da literatura	Parale-lização	Processadores			
			1	2	4	4c
(E)	1 - 25	DTE	0,72	0,75	0,11	0,74
(F)		ADP	0,72	0,5	0,38	0,25

Tabela 4.32 – Valores médios da *dp* segundo o tipo de paralelismo para as matrizes da literatura

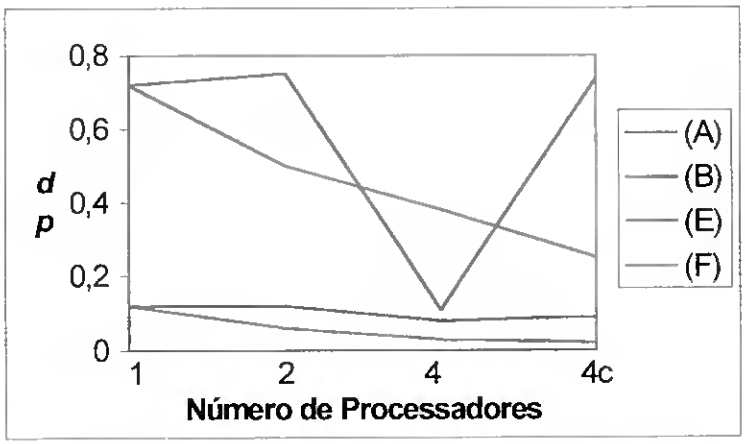


Gráfico 4.35 – Valores médios da *dp* com até 4 processadores

A paralelização ADP (linhas rosa e vermelha) levou nos dois tipos de dados a uma melhoria nos valores da *dp*. Com o aumento dos processadores a *dp*, em média, melhorou. Note-se que, os valores da diferença percentual obtidos com o algoritmo sequencial eram relativamente menores para os dados gerados aleatoriamente do que para os dados da literatura (tabelas 4.27 e 4.28). É pois, possível que as melhorias na *dp* com a paralelização ADP se façam sentir mais nos dados da literatura (linha vermelha), como se vê no gráfico 4.35.

Em relação à paralelização DTE verifica-se uma ligeira melhoria nos valores médios da *dp* para os dados gerados (linha azul) enquanto que para os dados da literatura (linha verde) houve uma acentuada melhoria com a utilização de 4 processadores e uma única troca de indivíduos entre eles.

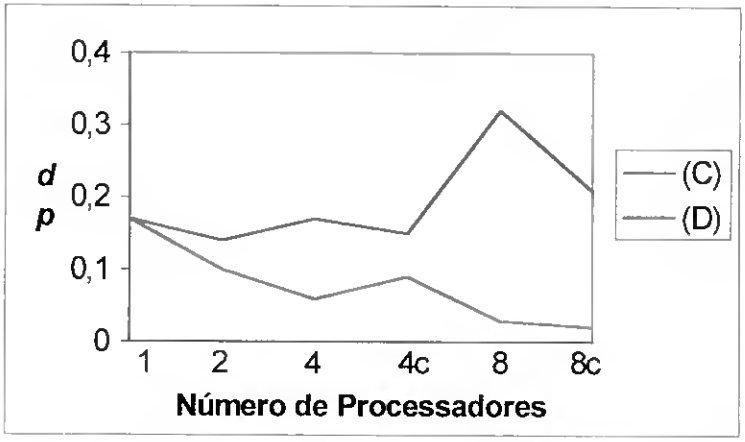


Gráfico 4.36 – Valores médios da *dp* com até 8 processadores

O comportamento do algoritmo para os problemas de maior dimensão gerados aleatoriamente é quase simétrico em relação a cada tipo de paralelização. Isto compreende-se, reparando que em relação à paralelização ADP (linha rosa) houve sempre melhoria, em média dos valores da *dp*, em relação à paralelização DTE (linha azul), como é esperado visto cada processador trabalhar com uma população menor, a *dp* pouco melhora para 4 processadores e piora para 8. A inclusão de mais trocas de indivíduos entre processadores aumenta a diversificação e melhora os resultados obtidos com uma só troca.

Valores médios da aceleração							
	Problemas gerados	Parale-lização	Processadores				
			2	4	4c	8	8c
(A)	11 - 40	DTE	2,91	9,72	9,77		
(B)		ADP	0,76	0,75	0,74		
(C)	41 - 100	DTE	2,59	9,76	9,66	28,89	29,29
(D)		ADP	0,67	0,67	0,67	0,66	0,65

Tabela 4.33 – Valores médios da aceleração segundo o tipo de paralelismo para os dados gerados aleatoriamente

Valores médios da aceleração					
	Problemas da literatura	Parale-lização	Processadores		
			2	4	4c
(E)	1 - 25	DTE	2,32	8,82	8,72
(F)		ADP	0,64	0,64	0,62

Tabela 4.34 – Valores médios da aceleração segundo o tipo de paralelismo para as matrizes da literatura

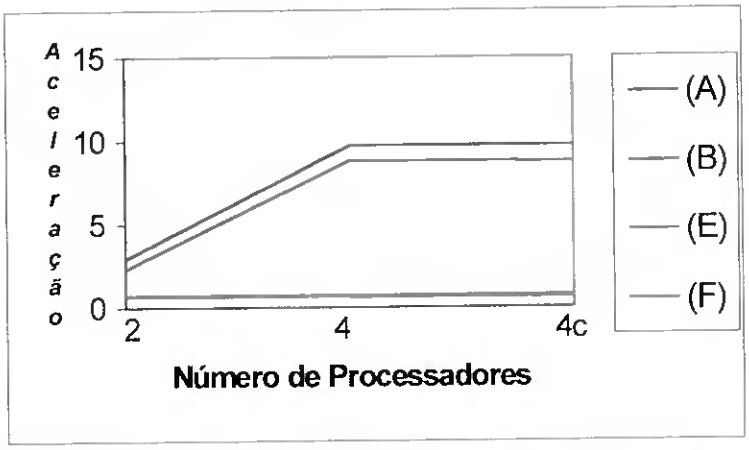


Gráfico 4.36 – Valores médios da aceleração com até 4 processadores

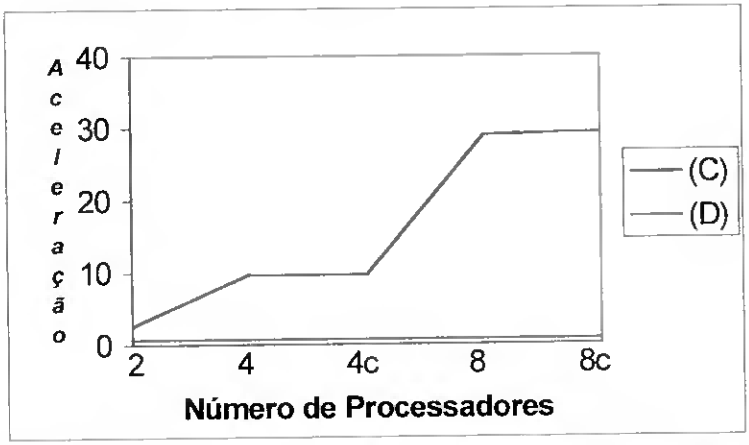


Gráfico 4.37 – Valores médios da aceleração com até 8 processadores

Em relação à aceleração, nota-se, a partir destas tabelas e destes gráficos, que, qualquer que seja o tipo de dados, existe uma aceleração acentuada com a paralelização DTE e uma desaceleração pouco acentuada com a ADP. Globalmente o aumento das comunicações não se reflecte de forma significativa na aceleração.

4.6.5 TESTES COMPLEMENTARES

Depois de realizado o estudo computacional descrito nas secções anteriores, envolvendo algoritmos exactos de Pesquisa em Árvore, Algoritmos Genéticos e respectivas paralelizações, verificou-se, muito recentemente, que havia um conjunto de problemas-teste, para o PQ 0-1, disponíveis na Internet no endereço http://mscmga.ms.ic.ac.uk/jeb/orlib. Para alguns desses problemas, conheciam-se os resultados obtidos com uma meta-heurística muito eficiente baseado em Pesquisa Tabu [Glover et al (1998)] já mencionada anteriormente. Tendo em conta que, em geral, os resultados obtidos com este tipo de meta-heurísticas são sempre muito bons, considerou-se que este estudo ficaria mais completo se o AG5 descrito em 4.5.3 (algoritmo 4.5) fosse testado nesses problemas.

Glover et al (1998) dividem estes dados em 6 conjuntos. Os três primeiros grupos, "a", "b" e "c", contêm dados já testados, também, por Pardalos e Rodgers (1990). Os outros conjuntos, "d", "e" e "f", foram gerados por Glover et al (1998), com a intenção de serem problemas muito difíceis. O conjunto "f" contém problemas com 500 variáveis e matrizes com densidades que variam entre 0,1 e 1. Glover et al (1998) não testaram este conjunto de dados num PC, devido às suas dimensões. No PC utilizado neste trabalho também não se conseguiu testá-los por falta de memória. Descrevem-se, na tabela 4.35, as características dos dados testados. Note-se que os problemas considerados são para um PQ 0-1 em que se pretende maximizar a f.o., ou seja na forma:

$$\begin{aligned} &max\ f(x) = x^{\mathrm{T}} Ax \\ &x \in \{0,1\}^n \end{aligned}$$

O facto de ser um problema de maximização não levanta qualquer dificuldade pois, como se sabe, para qualquer função f tem-se que:

$$\max_{x \in S} f(x) = -\min_{x \in S} f(-x).$$

Probl.	geração da matriz		nº	densi-	elementos (1)		diagonal (2)		nº (3)	var. (4)
nº	não diag	diagonal	var.	dade	> 0	< 0	> 0	< 0	total	fixas
1a	[-100, 100]	[-100, 100]	50	0,1	58	48	25	25	156	16
2a			60	0,1	79	84	31	28	222	18
3a			70	0,1	113	110	35	34	292	9
4a			80	0,1	145	159	46	33	383	5
5a			50	0,2	119	112	22	28	281	3
6a			30	0,4	91	83	11	19	204	0
7a			30	0,5	107	104	13	17	241	0
8a			100	0,625	149	155	53	46	403	52
1b	[0, 100]	[-63, 0]	20	1	187	0	0	20	207	0
2b			30	1	429	0	0	30	459	0
3b			40	1	771	0	0	40	811	1
4b			50	1	1208	0	0	50	1258	1
5b			60	1	1751	0	0	60	1811	0
6b			70	1	2388	0	0	70	2458	0
7b			80	1	3125	0	0	80	3205	0
8b			90	1	3962	0	0	90	4052	1
9b			100	1	4902	0	0	100	5002	1
10b			125	1	7663	0	0	125	7788	3
1c	[-50, 50]	[-100, 100]	40	0,8	309	316	16	24	665	0
2c			50	0,6	358	405	26	24	813	0
3c			60	0,4	334	367	34	26	761	0
4c			70	0,3	349	371	23	46	789	0
5c			80	0,2	284	357	45	35	721	0
6c			90	0,1	190	210	44	45	489	8
7c			100	0,1	247	248	49	51	595	20
1d	[-50, 50]	[-75, 75]	100	0,1	235	258	49	51	593	4
2d				0,2	516	500	56	44	1116	0
3d				0,3	727	698	48	52	1525	0
4d				0,4	1022	978	43	57	2100	0
5d				0,5	1202	1212	58	41	2513	0
6d				0,6	1478	1470	50	50	3048	0
7d				0,7	1697	1737	43	55	3532	0
8d				0,8	1956	1951	50	50	4007	0
9d				0,9	2203	2143	46	53	4445	0
10d				1	2377	2520	44	55	4996	0
1e	[-50, 50]	[-100, 100]	200	0,1	967	957	88	112	2124	0
2e				0,2	1943	1984	97	103	4127	0
3e				0,3	2959	2891	101	97	6048	0
4e				0,4	3926	3991	85	115	8117	0
5e				0,5	4883	4974	104	94	10055	0

(1) = número de elementos da matriz triangular (sem a diagonal) diferentes de zero
(2) = número de elementos da diagonal diferentes de zero
(3) = número de elementos da matriz triangular diferentes de zero
(4) = número de variáveis fixas definitivamente no vértice inicial

Tabela 4.35 – Características dos dados testados em Glover et al (1998)

Neste estudo, optou-se por testar o AG5, utilizando três valores diferentes para o número de indivíduos da população e para o número máximo de gerações a tratar. Estabeleceram-se esses valores (*Pop/Maxger*) em 40/35, 60/40 e 80/60. Na tabela 4.36 apresentam-se os resultados obtidos, incluindo, para comparação, os resultados obtidos com a heurística descrita em 4.4.2 (algoritmo 4.4). Nesta tabela, as soluções dos problemas "a", "b" e "c" são as óptimas e estão em itálico. Para os problemas "d" e "e" as melhores soluções conhecidas são as de Glover et al (1998) (lembre-se que neste caso está-se a maximizar). De facto, também foi utilizado o algoritmo de pesquisa em árvore descrito em 4.5.1 (APA) para resolver estes problemas, mas só foi encontrada a solução óptima para os problemas "a", "b" e "c" pois para os outros o algoritmo não chegou ao fim da pesquisa em árvore em menos de 15 horas (tempo limite que se tinha imposto).

Verifica-se que os resultados obtidos com o AG5 são bastante bons tanto em termos de qualidade de solução como em termos de tempos de execução computacional, qualquer que seja a dimensão da população. Os resultados obtidos com os problemas do tipo "a", "b" e "c" são melhores do que os obtidos com os outros. Isto era esperado porque os primeiros, são os problemas considerados mais fáceis. Somente, a execução exacta dos problemas do conjunto "c" é difícil e muito demorada [Glover et al (1998)].

Em relação à solução obtida, para os três primeiros conjuntos de problemas obteve-se, quase sempre, a solução óptima (em 18 dos 25 problemas). Mesmo para os problemas em que não se chega ao óptimo a solução obtida com o AG5 está muito próxima do óptimo. No pior caso (um em 25) a diferença percentual é igual a 1,27.

Para os problemas dos conjuntos "d" e "e" os resultados obtidos são ligeiramente piores. Mas, mesmo nestes caso a solução ou é igual à melhor conhecida ou está muito próxima. Exceptua-se um problema, em cada um destes conjuntos de dados, em que a solução obtida foi pior, sendo a diferença percentual de cerca de 5%.

		algoritmo 4.4			AG (40 / 35)				AG (60 /40)				AG (80 / 60)			
Nº	sol*	sol	*dp*	(2)	sol	*dp*	(1)	(2)	sol	*dp*	(1)	(2)	sol	*dp*	(1)	(2)
1a	*3414*	3381	0,97	0,06	3414	0,00	3	1,76	3414	0,00	3	2,97	3404	0,29	1	5,77
2a	*6063*	5790	4,50	0,11	5986	1,27	11	2,86	5989	1,22	4	4,95	5989	1,22	5	9,51
3a	*6037*	6035	0,03	0,11	6035	0,03	8	3,85	6035	0,03	18	6,76	6037	0,00	13	12,31
4a	*8598*	7017	18,39	0,11	8598	0,00	12	5,11	8530	0,79	8	8,96	8598	0,00	20	14,01
5a	*5737*	5712	0,44	0,11	5712	0,44	6	2,47	5712	0,44	4	4,01	5712	0,44	4	7,69
6a	*3980*	3980	0,00	0,11	3980	0,00	5	1,15	3980	0,00	3	1,98	3980	0,00	5	3,79
7a	*4541*	4541	0,00	0,06	4541	0,00	1	1,15	4541	0,00	2	2,03	4541	0,00	2	3,96
8a	*11109*	11109	0,00	0,11	11109	0,00	24	7,25	11109	0,00	10	12,20	11109	0,00	6	23,57
média			**3,04**	**0,10**		**0,22**	**9**	**3**		**0,31**	**7**	**5**		**0,24**	**7**	**10**
1b	*133*	85	36,09	0,00	133	0,00	1	0,33	133	0,00	1	0,55	133	0,00	1	0,95
2b	*121*	91	24,79	0,00	121	0,00	2	0,60	121	0,00	1	0,93	121	0,00	1	1,78
3b	*118*	102	13,56	0,06	118	0,00	1	0,93	118	0,00	1	1,65	118	0,00	1	3,08
4b	*129*	85	34,11	0,06	129	0,00	1	1,43	129	0,00	1	2,36	129	0,00	1	4,40
5b	*150*	150	0,00	0,00	150	0,00	3	2,03	150	0,00	1	3,35	150	0,00	1	6,32
6b	*146*	88	39,73	0,06	146	0,00	30	2,58	146	0,00	26	5,45	146	0,00	24	8,86
7b	*160*	160	0,00	0,06	160	0,00	1	3,57	160	0,00	1	5,82	160	0,00	1	10,60
8b	*145*	101	30,34	0,06	145	0,00	35	4,29	145	0,00	15	6,82	145	0,00	13	12,80
9b	*137*	75	45,26	0,06	137	0,00	3	4,89	137	0,00	3	8,13	137	0,00	3	14,45
10b	*154*	90	41,56	0,11	154	0,00	2	8,41	154	0,00	1	13,96	154	0,00	3	24,84
média			**24,88**	**0,05**		**0,00**	**8**	**3**		**0,00**	**5**	**5**		**0,00**	**5**	**9**
1c	*5058*	5058	0,00	0,00	5058	0,00	2	2,25	5058	0,00	4	3,96	5058	0,00	3	7,15
2c	*6213*	6213	0,00	0,00	6213	0,00	6	3,90	6213	0,00	10	6,32	6213	0,00	7	11,85
3c	*6665*	6649	0,24	0,06	6649	0,24	16	4,84	6649	0,24	9	7,91	6649	0,24	11	14,10
4c	*7398*	7398	0,00	0,00	7398	0,00	2	6,04	7398	0,00	2	10,44	7398	0,00	2	18,33
5c	*7362*	7336	0,35	0,06	7336	0,35	13	0,33	7342	0,27	35	12,80	7342	0,27	41	22,05
6c	*5824*	5805	0,33	0,06	5805	0,33	6	6,48	5805	0,33	11	11,21	5805	0,33	12	18,59
7c	*7225*	7147	1,08	0,00	7211	0,19	6	8,19	7211	0,19	10	14,40	7211	0,19	10	24,89
média			**0,29**	**0,03**		**0,16**	**7**	**5**		**0,15**	**12**	**10**		**0,15**	**12**	**17**
1d	6333	6077	4,04	0,00	6241	1,45	10	9,95	6248	1,34	27	15,82	6248	1,34	23	29,18
2d	6579	6240	5,15	0,06	6286	4,45	24	10,55	6240	5,15	11	15,22	6281	4,53	9	28,13
3d	9261	9158	1,11	0,06	9247	0,15	24	13,46	9247	0,15	17	22,91	9247	0,15	20	44,62
4d	10727	10684	0,40	0,06	10684	0,40	18	15,28	10684	0,40	13	25,17	10684	0,40	16	47,42
5d	11626	11162	3,99	0,06	11162	3,99	6	14,67	11390	2,03	18	26,48	11511	0,99	18	48,85
6d	14207	14157	0,35	0,06	14157	0,35	16	19,73	14157	0,35	23	35,22	14157	0,35	24	65,00
7d	14476	13969	3,50	0,06	13969	3,50	16	20,06	14188	1,99	10	35,60	13969	3,50	11	65,55
8d	16352	16308	0,27	0,06	16352	0,00	25	20,82	16352	0,00	30	37,75	16352	0,00	46	65,55
9d	15656	15557	0,63	0,06	15557	0,63	9	18,26	15557	0,63	9	31,21	15557	0,63	12	56,37
10d	19102	18391	3,72	0,11	18821	1,47	35	25,11	18772	1,73	30	40,06	18391	3,72	26	72,97
média			**2,32**	**0,06**		**1,64**	**18**	**17**		**1,38**	**19**	**29**		**1,56**	**21**	**52**
1e	16464	16438	0,16	0,06	16464	0,00	29	46	16464	0,00	31	75	16464	0,00	42	144
2e	23395	22880	2,20	0,11	23127	1,15	28	69	23020	1,60	33	117	23127	1,15	51	209
3e	25243	23930	5,20	0,11	23986	4,98	32	73	23939	5,17	39	132	23930	5,20	31	241
4e	35594	35594	0,00	0,22	35594	0,00	20	111	35594	0,00	20	192	35594	0,00	49	429
5e	35154	34761	1,12	0,22	34816	0,96	29	102	34816	0,96	31	206	34816	0,96	26	347
média			**1,74**	**0,14**		**1,42**	**28**	**80**		**1,55**	**31**	**144**		**1,46**	**40**	**274**
média			**8,09**	**0,07**		**0,66**	**13**	**16,37**		**0,63**	**13**	**29,16**		**0,65**	**15**	**54,51**

(1) - geração em que foi encontrada a melhor solução

(2) - Tempo de execução em segundos num Pentium MMX a 200 Mhz

Tabela 4.36 – Resultados obtidos com o AG5

Em relação aos tempos de execução computacional, verifica-se que este AG é bastante rápido, inclusive para uma população de 80 indivíduos. O pior tempo de execução, cerca de 7 minutos, ocorreu no problema 4e, com 200 variáveis e cuja matriz triangular tinha 8117 elementos não nulos. Excepto para estes problemas de 200 variáveis e para os problemas "d" no caso da população de 80 indivíduos, os tempos de execução computacional foram sempre inferiores a 40 segundos. Desta forma é possível, em aplicações reais, executar em tempo útil o AG5 variando a dimensão da população ou com uma população ainda maior.

Ressalta também, das colunas (1), que a melhor solução é encontrada, em geral, em poucas gerações. Isto, por um lado, significa que na maioria dos problemas testados a solução heurística não influenciou o resultado, uma vez que ela só é introduzida na população após completados dois terços do total de gerações. Por outro, leva a supor que, em termos de qualidade, os resultados seriam semelhantes, se tivesse sido imposto um limite menor para o número máximo de gerações a efectuar, e nesse caso, o tempo de execução computacional seria ainda menor.

Na tabela 4.37 resumem-se os resultados obtidos com o AG5 e representam-se, nos gráficos 4.38 e 4.39, os valores médios das diferenças percentuais e dos tempos de execução computacional, para cada uma das populações.

Constata-se que, em média, os resultados obtidos com a menor população são os melhores, excepto para os problemas do tipo "d". Para estes a população de dimensão igual a 60 conduz a melhores resultados. Em média, a população de dimensão igual a 80 não melhorou nenhum resultado. Assim, para estes dados, as populações de dimensão 40 e 60 foram as mais adequadas.

Probl.		AG (40/35)		AG (60/40)		AG (80/60)	
		dp	tempo	*dp*	tempo	*dp*	tempo
"a"	Melhor	0	1	0	2	0	4
	Média	**0,22**	**3**	**0,31**	**5**	**0,24**	**10**
	Pior	1,27	7	1,22	12	1,22	24
"b"	Melhor	0	0	0	1	0	1
	Média	**0**	**3**	**0**	**5**	**0**	**9**
	Pior	0	8	0	14	0	25
"c"	Melhor	0	0	0	4	0	7
	Média	**0,16**	**5**	**0,15**	**10**	**0,15**	**17**
	Pior	0,35	8	0,33	14	0,33	25
"d"	Melhor	0	10	0	15	0	28
	Média	**1,64**	**15**	**1,38**	**29**	**1,56**	**52**
	Pior	4,45	25	5,15	40	4,53	73
"e"	Melhor	0	46	0	75	0	144
	Média	**1,42**	**80**	**1,55**	**144**	**1,46**	**274**
	Pior	4,98	111	5,17	206	5,2	429

Tabela 4.37 – Resultados para cada conjunto de dados

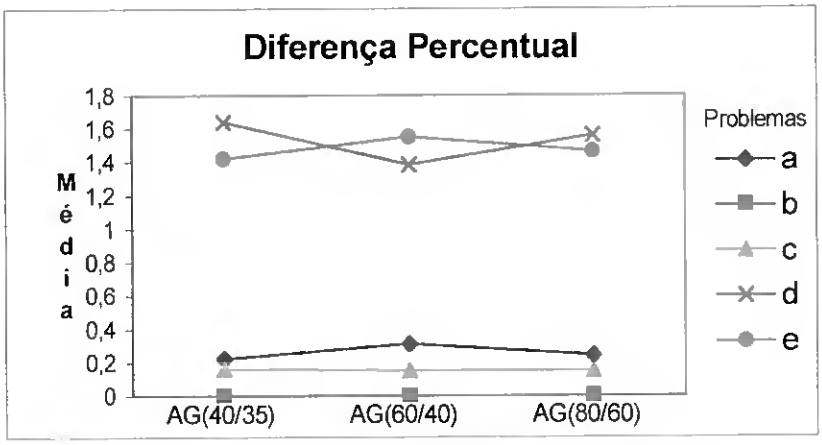


Gráfico 4.38 – Valores médios da diferença percentual

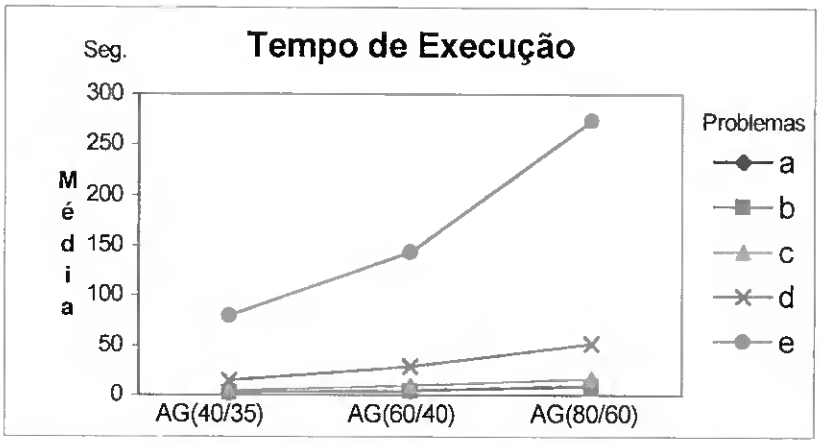


Gráfico 4.39 – Valores médios do tempo de execução (em segundos)

Verifica-se que os tempos médios de execução computacional são muito bons, inferiores a 1 minuto, para os problemas dos conjuntos "a", "b", "c" e "d", mesmo quando se utilizou uma população de 80 indivíduos. Note-se que nestes conjuntos alguns problemas já são de grande dimensão, 125 e 100 variáveis com matrizes densas.

Relativamente a estes resultados os tempos de execução para resolução dos problemas maiores, de 200 variáveis, pioraram significativamente, mas em termos absolutos também os tempos de execução computacional para resolver estes problemas foram bons, inferiores a 5 minutos mesmo com uma população de 80 indivíduos, e exequíveis em tempo útil.

Face a estes resultados não pareceu interessante repetir o estudo da paralelização, já feito com alguma exaustão na secção anterior. Embora seja de prever uma melhoria nos tempo de execução com a população distribuída pelos processadores e uma eventual melhoria dos resultados mantendo a mesma população em cada processador. As eventuais melhorias que se poderiam obter seriam insignificantes dada a boa qualidade dos resultados já obtidos.

4.7 CONCLUSÕES

Os resultados obtidos mostram que a opção de resolver exactamente Problemas de Optimização Combinatória é, do ponto de vista de tempo de execução computacional

dos algoritmos, em geral, impraticável para problemas de média e grande dimensão, mesmo dispondo de um bom majorante do valor óptimo e de algoritmos eficientes. Não existe também à priori nenhuma forma de prever, para um certo problema, qual o comportamento que o algoritmo de pesquisa em árvore vai ter. Mesmo sabendo que quanto menor for o número de variáveis livres mais fácil deverá ser a pesquisa, não é suficiente para calcular o tamanho da árvore e consequentemente o tempo de execução do algoritmo exacto.

No que diz respeito à tentativa de reduzir o tempo de execução computacional deste tipo de algoritmos, os resultados obtidos, tanto para os problemas gerados como para os problemas da literatura, mostram que, em geral, a utilização de processamento paralelo é vantajosa, levando a importantes reduções, podendo inclusivé atingir eficiências superiores a 1. No entanto, é necessário ter em conta que, nalguns casos, também é possível não obter nenhum ganho (eficiências praticamente nulas).

A topologia utilizada mostrou-se bem ajustada a este tipo de algoritmos. Ao executar uma pesquisa da árvore em profundidade e simultaneamente em largura originou árvores diferentes das obtidas pelo algoritmo sequencial, mas na generalidade dos problemas, com um número de vértices semelhante ao do algoritmo sequencial. Permitiu uma distribuição equilibrada do trabalho pelos processadores sem que isso acarretasse um dispêndio de tempo com as comunicações muito elevado, visto que, obteve boas acelerações para os problemas (a maioria) em que o número de vértices pesquisado sequencialmente ou paralelamente era semelhante (ou mesmo ligeiramente superior no algoritmo paralelo).

No que respeita aos AGs desenvolvidos ficou claro que a sua utilização é adequada a este problema. Em particular, a incorporação de técnicas de optimização num AG mostrou-se bastante interessante. De facto, o AG5 obteve soluções óptimas ou muito próximas das óptimas para todos os problemas testados (gerados aleatoriamente, matrizes da literatura e um conjunto de problemas-teste). Glover et al (1998) referem que os resultados com o seu algoritmo de PT superaram todos os outros métodos heurísticos (ou meta-heurísticos) conhecidos. Então, a partir da experiência computacional descrita em 4.6.5, onde o AG5 se mostrou competitivo com a PT de Glover et al (1998), pode-se afirmar que este método é também dos mais eficientes.

Os testes revelaram ainda uma boa consistência do AG5 em relação à dimensão e densidade das matrizes dos problemas, permitindo a obtenção de uma solução próxima da óptima (eventualmente a óptima) em tempo razoável mesmo para grandes dimensões.

A paralelização do AG5 também foi vantajosa. Se o objectivo for a diminuição do tempo de execução consegue-se uma enorme redução do tempo de execução computacional(principalmente com 4 e 8 processadores) sem um prejuízo significativo para a qualidade da solução. Se o objectivo for melhorar ainda mais a qualidade da solução também é atingido sem um aumento considerável no tempo de execução computacional.

CAPÍTULO 5

PROBLEMA DE OPTIMIZAÇÃO DE ROTAS DE VEÍCULOS

5.1 INTRODUÇÃO

A versão básica do Problema de Optimização de Rotas de Veículos (PORV), genericamente conhecido na literatura de língua inglesa por "Vehicle Routing Problem", consiste em determinar um conjunto de rotas a efectuar por uma frota de veículos, a partir de um depósito central, para servir um conjunto de clientes. As capacidades dos veículos, as localizações e procuras dos clientes, bem como a localização do depósito são conhecidas. Pretende-se, então, determinar quais os clientes que serão servidos por cada veículo e qual a configuração das rotas a efectuar pelos veículos de forma a minimizar a distância total percorrida.

Podem-se representar os dados deste problema através de um grafo $G = (V, E)$ onde $V = \{v_0, v_1, ..., v_n\}$ é o conjunto dos vértices e $E = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ é o conjunto dos arcos. Considere-se que o vértice v_0 corresponde ao depósito e que os outros vértices representam os clientes. No depósito está sediada uma frota de m veículos idênticos de capacidade Q. A cada cliente $v_i \in V - \{v_0\}$ está associada uma procura q_i, $q_i \geq 0$. Uma matriz, não negativa, de distâncias $C = (c_{ij})$ está associada aos arcos de E. Dependendo dos casos, c_{ij} pode ser interpretado como a distância entre os clientes v_i e v_j ou como o tempo gasto para ir de v_i a v_j (pode representar um custo espacial ou um custo temporal). Na maioria dos casos tem-se $c_{ij} = c_{ji}$ e trabalha-se portanto com uma matriz simétrica. O PORV básico consiste, então, em determinar um conjunto de m rotas de custo total mínimo e tais que:

(i) Cada rota comece e termine no depósito.

(ii) Cada cliente seja visitado uma e só uma vez por um único veículo.

(iii) A procura total em cada rota não exceda Q.

Muitas extensões e variantes deste problema são descritas na literatura e têm sido alvo de várias formalizações matemáticas e de diferentes algoritmos. Boas revisões bibliográficas e análises acerca dos estudos desenvolvidos sobre este problema encontram-se, entre outros, em Christofides (1985), Laporte e Norbert (1987), Almeida (1988), Bodin (1990), Laporte (1992) e Desrosiers et al (1995).

Neste capítulo, apresenta-se, na secção 2, uma formalização matemática para o PORV, referenciam-se alguns trabalhos importantes desenvolvidos para o resolver e resumem-se as principais vantagens e desvantagens das várias abordagens. Na secção 3, descrevem-se os Algoritmos Genéticos desenvolvidos e implementados neste trabalho. Expõe-se, também, a paralelização efectuada. Apresenta-se, na secção 4, o estudo computacional realizado, os resultados obtidos e respectiva análise. Finaliza-se este capítulo com algumas conclusões e comentários, na secção 5.

5.2. FORMALIZAÇÃO E MÉTODOS DE RESOLUÇÃO

Algumas das formalizações matemáticas mais utilizadas, na resolução do PORV básico, estão descritas em Almeida (1988) e Schütz (1989). Entre elas, apresenta-se uma formalização do PORV (formalização 5.1) como um problema de Programação Não Linear com variáveis binárias [Fisher e Jaikumar (1978)], por ser a que está subjacente aos AGs desenvolvidos neste trabalho e descritos na secção 3 deste capítulo.

Considere-se as constantes:

n – o número de clientes

o depósito corresponde ao cliente 0

m – o número de veículos

c_{ij} – a distância entre i e j, $i, j = 0,...,n$

q_i – a procura do cliente i, $i = 1,...,n$

Q – a capacidade de cada veículo

e as variáveis:

$$x_{ijk} = \begin{cases} 1, \text{ se o veículo } k \text{ vai de } i \text{ para } j \text{ directamente} \\ 0, \text{caso contrário} \end{cases}$$

$$y_{ik} = \begin{cases} 1, \text{ se o cliente } i \text{ é servido pelo veículo } k \\ 0, \text{ caso contrário} \end{cases}$$

Seja $y_k = (y_{0k}, y_{1k}, ..., y_{nk})$

então tem-se:

$$min \sum_k f(y_k) \tag{5.1}$$

$$\text{s.a} \sum_k y_{ik} = \begin{cases} 1 & , i = 1,...,n \\ m & , i = 0 \end{cases} \tag{5.2}$$

$$\sum_i q_i y_{ik} \leq Q \quad , k = 1,...,m \tag{5.3}$$

$$y_{ik} = 0 \text{ ou } 1 \quad , i = 0,...,n \; ; \; k = 1,...,m \tag{5.4}$$

em que para cada k, $k = 1,...,m$

$$f(y_k) = min \sum_{i,j} c_{ij} x_{ijk} \tag{5.5}$$

$$\text{s.a} \sum_j x_{ijk} = y_{ik} \quad , i = 0,...,n \tag{5.6}$$

$$\sum_i x_{ijk} = y_{jk} \quad , j = 0,...,n \tag{5.7}$$

$$\sum_{i,j \in S \times S} x_{ijk} \leq |S| - 1 \quad , S \subseteq \{1,...,n\}, \; 2 \leq |S| \leq n-1 \tag{5.8}$$

$$x_{ijk} = 0 \text{ ou } 1 \quad , i, j = 0,...,n \tag{5.9}$$

Formalização 5.1 – Formalização do Problema básico de Optimização de Rotas de Veículos

Em (5.1) o objectivo é minimizar uma soma de funções não lineares $f(y_k)$. O valor de $f(y_k)$, para cada k, $k = 1,...,m$, é obtido resolvendo um subproblema cuja função objectivo (5.5) minimiza a distância percorrida por cada veículo, para servir um determinado subconjunto de clientes. Neste subproblema as restrições (5.6) e (5.7) obrigam a que um veículo que chega a um cliente também parta desse cliente e as restrições (5.8) garantem que não existem subcircuitos que não contenham o depósito. As condições (5.9) definem as variáveis deste subproblema como binárias.

As restrições (5.2) garantem que os clientes estão afectos a um veículo e que o depósito é servido por todos os veículos. As condições (5.3) não permitem que as

capacidades dos veículos sejam excedidas. As restrições (5.4) definem as variáveis como binárias.

Esta formalização, põe em evidência os dois Problemas de Optimização Combinatória que estão englobados num PORV. Por um lado, as restrições (5.2), (5.3) e (5.4) têm a estrutura de um Problema de Afectação Generalizada, onde é determinada a afectação dos clientes aos veículos. Por outro lado, para as variáveis y_{ik} fixas satisfazendo (5.2), (5.3) e (5.4) e para cada veículo k, cada parcela $f(y_k)$ da função objectivo (5.1), dada por (5.5) a (5.9), define um Problema do Caixeiro Viajante (PCV), onde é estabelecida a configuração de cada rota. Assim, nesta formalização o PORV é definido como um Problema de Afectação Generalizada Não Linear.

O PORV é um Problema de Optimização Combinatória pertencente à classe dos problemas *NP*–difíceis [Lenstra e Rinnooy Kan (1981)]. A grande dificuldade deste problema e as inúmeras aplicações reais justificam os muitos trabalhos que lhe têm sido dedicados. Uma bibliografia extensa dos artigos publicados sobre este problema encontra-se em Laporte e Osman (1995).

Os algoritmos exactos raramente são aplicáveis de uma forma eficiente a problemas com mais de 50 clientes, mesmo para problemas menores estes métodos são lentos [Desrochers et al (1992), Fisher (1994), Hadjiconstantinou et al (1995)]. Os poucos casos em que problemas com 100 clientes foram resolvidos exactamente, referem que ou não foi necessário recorrer à Pesquisa em Árvore, ou o número de vértices da árvore que foi necessário pesquisar era muito pequeno [Fisher (1994)].

Por isso, a maior parte da investigação, sobre o PORV, tem-se dirigido a algoritmos heurísticos. São bem conhecidas, entre outras, a heurística construtiva de rota de Clarke e Wright (1964), as heurísticas de duas fases de Gillet et Miller (1974), Christofides et al (1979) e Fisher e Jaikumar (1981) e as heurísticas de melhoramento de rotas de Christofides e Eilon (1969) e de Russel (1977).

Mais recentemente várias meta-heurísticas e algoritmos híbridos têm sido desenvolvidos para o PORV. Podem-se citar, entre outros, o algoritmo híbrido de Osman (1993) que combina Arrefecimento Simulado e Pesquisa Tabu, o algoritmo baseado num Sistema de Colónia de Formigas de Bulhnmeier et al (1999) e os trabalhos

baseados em algoritmos de Pesquisa Tabu de Taillard (1993), Gendreau et al (1994), Rego (1994), Rochat e Taillard (1995), Rego (1998) e Kelly e Xu (1999). A maioria destes algoritmos de PT faz a pesquisa da vizinhança da solução actual movendo clientes de uma rota para outra. Se a PT recorrer somente a funções de memória de curto termo [Taillard (1993), Gendreau et al (1994)] os clientes mais próximos do depósito sofrem mais trocas do que os outros. Uma forma de diversificar a pesquisa é incluir uma penalidade no custo de um movimento, proporcional à frequência com que este é efectuado [Taillard (1993), Gendreau et al (1994)]. Em Gendreau et al (1997) é feito um estudo comparativo de sete versões de heurísticas de PT para o PORV num conjunto de 14 problemas-teste descritos em Christofides et al (1979). A maioria das melhores soluções conhecidas foram obtidas com a PT de Taillard (1993), destacando-se também a actuação dos algoritmos de Rochat e Taillard (1995) e de Gendreau et al (1994).

Com base nos trabalhos divulgados na literatura, propondo diversas abordagens para resolução do PORV e respectivos resultados, podem-se tecer alguns comentários:

- Os métodos exactos, salvo em casos particulares ou de pequena dimensão, não se têm mostrado aplicáveis em tempo útil.

- As abordagens heurísticas, são em geral muito rápidas e produzem boas soluções, mas frequentemente ficam presas num óptimo local.

- A inclusão de procedimentos pós-optimais, nos algoritmos heurísticos, leva em geral a melhoramentos na qualidade da solução. No entanto, os procedimentos utilizados nas heurísticas obtêm frequentemente um óptimo local porque ao pesquisarem a vizinhança de uma solução, não aceitam soluções piores e se não existir nenhuma solução melhor terminam.

- Entre as heurísticas, o método de duas fases de Fisher e Jaikumar (1981) é um dos que se tem apresentado como mais robusto e eficiente, apesar de computacionalmente mais demorado [Schütz (1990)].

- As meta-heurísticas utilizam uma grande quantidade de parâmetros (principalmente a PT) para guiar a pesquisa, intensificando-a nalgumas regiões e diversificando-a a todas as regiões do espaço de soluções.

- Nas meta-heurísticas a sintonização (determinação do melhor valor) de cada um desses parâmetros pode ser difícil e depende totalmente da especificidade do problema. No entanto, muito bons resultados têm sido alcançados, para o PORV, principalmente com algoritmos de PT.

- Os resultados obtidos com as meta-heurísticas têm vindo a melhorar consideravelmente, devido à introdução de técnicas mais sofisticadas na condução da pesquisa e à combinação de alguns princípios subjacentes a cada um dos métodos, resultando em bons algoritmos híbridos.

- Comparando, entre si, as meta-heurísticas verifica-se que as implementações de PT obtêm, em geral, melhores e excelentes resultados. No entanto, salienta-se que foram desenvolvidos muitos algoritmos de PT para o PORV, enquanto poucas tentativas foram feitas utilizando outras meta-heurísticas, nomeadamente AGs e SCFs.

- Na generalidade dos casos, as meta-heurísticas têm-se mostrado mais eficientes que as heurísticas no que respeita à qualidade das soluções do PORV. Em contrapartida são computacionalmente mais exigentes e mais demoradas.

- As meta-heurísticas requerem, em geral, um esforço computacional elevado.

Ultimamente, a opção mais óbvia para abreviar o tempo de execução de um algoritmo é a utilização de máquinas paralelas. Entre as meta-heurísticas, as que aparecem, nitidamente, como boas candidatas a serem paralelizadas são o AG e o SCF. No entanto, considerando que a meta-heurística SCF pode ser vista como uma variante do AG, optou-se neste trabalho por desenvolver um AG para o PORV e estudar a sua paralelização.

5.3 UMA IMPLEMENTAÇÃO DE ALGORITMOS GENÉTICOS

Como já se expôs, no capítulo 3, os AGs foram inicialmente desenvolvidos para problemas na área de Inteligência Artificial [Holland (1975)]. Baseiam-se fortemente em processos aleatórios e em probabilidades. Com probabilidades de selecção e de mutação adequadas [Fonseca (1998)], os indivíduos da população tendem a reproduzir-se de forma a que pelo menos uma cópia dos melhores indivíduos seja seleccionada para se reproduzir originando indivíduos mais aptos. Os AGs têm, também, sido aplicados com sucesso a problemas onde os outros métodos existentes, para os resolver, não conseguem sequer encontrar uma solução admissível [Flemming (1997)], bem como a problemas multi-objectivo [Fonseca (1995)]. Tendo presentes estes aspectos, relativos aos AGs, e também o facto do PORV ser um POC muito diferente dos problemas referidos, optou-se por estudar a aplicação dos princípios subjacentes aos AGs, ao PORV básico. Adaptaram-se os operadores genéticos ao PORV básico implementando um AG adaptado. Fizeram-se, também, algumas experiências variando os parâmetros envolvidos. Implementou-se, também, um AG básico e um AG híbrido. Descrevem-se seguidamente as várias abordagens realizadas. No fim indica-se a forma utilizada para paralelizar o AG.

5.3.1 CODIFICAÇÃO

Apesar de, como foi referido no capítulo 3, os AGs terem surgido baseados numa codificação binária dos indivíduos [Holland (1975), Goldberg (1989)], tem-se mostrado necessária a utilização de outro tipo de codificação, principalmente em alguns Problemas de Optimização Combinatória [Potvin (1996), Huntley e Brown (1996)].

Para o PORV a codificação binária não parece adequada. Uma alternativa natural seria usar abordagens análogas às que tem sido utilizadas para o PCV [Potvin (1996)]. No entanto, no PCV tem-se apenas uma rota e no PORV tratam-se m rotas, com número de clientes por rota e subconjunto de clientes em cada rota diferentes de uma solução para outra. A codificação por permutações de n, que traduz no PCV o

caminho e no PORV a configuração das várias rotas, torna particularmente difícil, através do cruzamento de dois indivíduos, a obtenção de um novo indivíduo que represente uma solução admissível e que as partes que herdou dos pais tenham um significado idêntico ao que tinham nos pais. Por outro lado, tanto na representação por permutações de n como por adjacências seria necessário acrescentar informação sobre onde começa e termina cada rota. Assim, considerou-se que as representações já utilizadas para o PCV, também não seriam muito adequadas para o PORV.

Então, optou-se por codificar cada indivíduo x^i, $i = 1,...,Pop$, por um cromossoma com n genes x_j^i, $j = 1,...,n$, onde n é o número de clientes do problema, podendo cada gene x_j^i tomar um valor de 1 a m, onde m é o número de veículos, previamente definido.

Neste caso, em vez de um indivíduo representar a configuração das rotas, representa as afectações dos clientes aos veículos. De acordo com a formalização 5.1 tem-se:

$$y_{0k} = 1 \text{ e } y_{jk} = \begin{cases} 1 & \text{se } x_j^i = k \\ 0 & \text{caso contrário} \end{cases} \quad j = 1,...,n;\ k = 1,...,m \tag{5.10}$$

Por exemplo, o indivíduo

1	1	3	2	2	2	2	3	1	2

Indica que:

os clientes 1, 2, 9 são servidos pelo veículo 1,

os clientes 4, 5, 6, 7, 10 são servidos pelo veículo 2,

os clientes 3, 8 são servidos pelo veículo 3.

e de acordo com a formalização 5.1, tem-se:

$$y_{11} = 1,\ y_{21} = 1,\ y_{91} = 1,$$
$$y_{42} = 1,\ y_{52} = 1,\ y_{62} = 1,\ y_{72} = 1,\ y_{102} = 1,$$
$$y_{33} = 1,\ y_{83} = 1,$$

sendo as restantes variáveis y iguais a zero, que para poder ser uma afectação admissível dos clientes aos veículos tem de verificar as restrições (5.2), (5.3) e (5.4).

O valor da função objectivo (5.1), para cada indivíduo x^i, vai-se representar por $f(x^i)$ e é igual à distância total percorrida pelos m veículos para servirem os clientes de acordo com a afectação definida por esse indivíduo e verificando as restrições (5.6) a (5.9). Para cada veículo $k, k = 1,...,m$, e respectiva afectação $y_k = (y_{0k},...,y_{nk})$ obtida segundo (5.10), a rota a efectuar (valores de x_{ijk}) e a distância a percorrer (valor de $f(y_k)$) são determinadas através da resolução de um PCV. Assim vem:

$$f(x^i) = \sum_{k=1}^{m} f(y_k) \ , \ i = 1,...,Pop \tag{5.11}$$

Nesta implementação, os PCV foram resolvidos heuristicamente, devido ao facto do PCV ser também um problema pertencente à classe dos problemas *NP*–difíceis. Utilizou-se o Método da Inserção Mais Afastada, por ser computacionalmente muito rápido e se encontrarem bons resultados referidos na literatura [Syslo et al (1983), Golden et al (1980)].

5.3.2 POPULAÇÃO INICIAL

Em geral, nos AGs a população inicial é gerada aleatoriamente. Neste trabalho utilizaram-se duas abordagens. Numa a população inicial é gerada aleatoriamente (versão básica) e na outra é gerada heuristicamente (versão adaptada e versão híbrida).

Na primeira, para cada indivíduo geram-se n números aleatórios igualmente distribuídos no intervalo [1, m]. Desta forma os indivíduos podem não corresponder a soluções admissíveis do PORV. Para estes o valor da função objectivo passou a ser dado pelo simétrico da quantidade total de capacidade excedida, isto é,

$$f(x^i) = \sum_{k=1}^{m} min\left\{0, \left(Q - \sum_{i=1}^{n} q_i \, y_{ik}\right)\right\} \tag{5.12}$$

Na outra, utilizou-se, para gerar a população inicial, um método heurístico computacionalmente rápido. Implementou-se um algoritmo do tipo construtivo de rota, que se descreve em seguida.

HEURÍSTICA CONSTRUTIVA DE ROTA

No PORV básico formalizado em 5.1. o número de veículos é constante e conhecido. Assim, nesta heurística, as m rotas são formadas paralelamente e os clientes são inseridos nas rotas segundo uma abordagem ávida. O algoritmo é inicializado escolhendo m clientes e formando m rotas do tipo depósito – cliente – depósito. Seguidamente, em cada iteração, é inserido em cada rota o cliente que não viola as restrições e minimiza o aumento da distância percorrida pelo veículo para o incluir entre os clientes que já estão nessa rota. Depois são aplicados procedimentos pós-optimais que consistem na troca de um e dois clientes entre rotas. Finalmente é resolvido para cada rota um PCV.

Pode-se resumir o algoritmo pelos seguintes Passos (algoritmo 5.1).

Considere-se que:

0 representa o depósito

$R = \{r_1,...,r_m\}$ representa o conjunto das m rotas

$C_L = \{1,...,n\}$ representa o conjunto dos clientes livres

$C_k = \varnothing, k = 1,...,m$, representa o conjunto dos clientes afectos à rota r_k

$Ocp(k) = 0, k = 1,...,m$, representa a ocupação do veículo k

$d(k), k = 1,...,m$, representa a distância percorrida pelo veículo k

Então,

Passo 1:

Escolher m clientes iniciais, $s_1,...,s_m$

Inicializar as m rotas fazendo

$r_k = (0, s_k, 0)$

$C_L = C_L - \{s_1,...,s_m\}$

$C_k = C_k + \{s_k\}, k = 1,...,m$

$Ocp(k) = Ocp(k) + q_{s_k}, k = 1,...,m$

$d(k) = c_{0s_k} + c_{s_k 0}, k = 1,...,m$

Passo 2:

Para cada rota $r_k \in R$ determinar qual o cliente $i \in C_L$ com menor custo de inserção e que satisfaz a restrição de capacidade dessa rota, isto é, o cliente $i \in C_L$ que verifica:

$$\begin{cases} Ocp(k) + q_i \leq Q \\ c_{ins}(i,k) = \min\limits_{l \in C_L} c_{ins}(l,k), \\ \qquad \text{onde } c_{ins}(l,k) = \min\limits_{j \in C_k} \{c_{jl} + c_{lj} - c_{j\,j+1}\} \end{cases} \tag{5.13}$$

Inserir p clientes em p rotas ($p \leq m$), segundo a ordenação crescente dos custos de inserção, associados às rotas $r_k \in R$ e clientes $i \in C_L$ que verificam (5.10).

Actualizar as rotas em cada inserção, isto é, fazer:

$r_k = (0,...,j,i,j+1,...,0)$

$C_L = C_L - \{i\}$ $\qquad$ $C_k = C_k + \{i\}$

$Ocp(k) = Ocp(k) + q_i$ $\qquad$ $d(k) = d(k) + c_{ins}(i,k)$

Passo 3:

Repetir o Passo 2 enquanto $C_L \neq \varnothing$ e os veículos tiverem capacidade disponível para servir os clientes livres.

Passo 4:

Se $C_L = \varnothing$ melhorar a solução aplicando procedimentos pós-optimais baseados na troca de clientes entre rotas.

Caso contrário terminar. (Não encontrou uma solução admissível).

Passo 5:

Terminar obtendo as rotas $r_k \in R$, as distâncias $d(k), k = 1,...,m$, e a distância total percorrida $D = \sum_{k=1}^{m} d(k)$.

Algoritmo 5.1 – Algoritmo Construtivo de Rota

Em geral, ao mudar os clientes iniciais de cada rota, as afectações subsequentes são diferentes, levando a outra solução. Desta forma, variando a escolha dos m primeiros clientes a serem inseridos, pode-se gerar os indivíduos para formar a população inicial (de dimensão *Pop*). Neste trabalho, para três dos indivíduos, escolheram-se os clientes iniciais com base na localização e/ou procura dos clientes. Para os restantes *Pop*-3 indivíduos, geraram-se aleatoriamente os clientes iniciais.

Os três critérios diferentes utilizados, no Passo 1 do algoritmo 5.1, para escolher os clientes iniciais, $s_1,...,s_m$, foram:

Critério 1 – Começa por escolher o cliente mais distante do depósito. Depois, escolhe os clientes, um de cada vez, mais afastados dos que já foram escolhidos.

Então:

$$\begin{aligned} s_1 &= k \in C_L : c_{0k} = \max_{i \in C_L} \{c_{0i}\} \\ s_p &= k \in C_L : \delta_k = \max_{i \in C_L} \{\delta_i\}, \; p = 2,...,m, \\ &\text{onde } \delta_i = \sum_{j \in C_L^c} c_{ij} \end{aligned} \tag{5.14}$$

Critério 2 - Quando alguns clientes têm uma procura muito superior aos outros e ocupam uma parte considerável da capacidade dos veículos, é interessante que os clientes de maior procura fiquem, desde o início, distribuídos pelas várias rotas. Quando as procuras dos clientes são semelhantes, esta escolha, baseada na maior procura, pode levar a que dois clientes, que estejam próximos e que numa boa solução seriam incluídos na mesma rota, fiquem em rotas diferentes conduzindo a uma solução pior. Por isso, este critério escolhe os clientes com maior procura desde que razoavelmente distantes dos que já foram escolhidos. Assim:

$$s_1 = k \in C_L : q_k = \max_{i \in C_L} \{q_i\}$$
$$s_p = k \in C_L : q_k = \max_{i \in C_L \cap Dist} \{q_i\}, p = 2,...,m \qquad (5.15)$$
$$\text{onde } Dist = \left\{ i : c_{ij} > \frac{\sum_{l=1}^{n} c_{0l}}{2n}, \ j \in C_L^c \right\}$$

Critério 3 - Este critério combina os dois anteriores, começa pelo cliente de maior procura e os outros são os que estiverem mais distantes dos que já foram escolhidos.

$$s_1 = k \in C_L : q_k = \max_{i \in C_L} \{q_i\}$$
$$s_p = k \in C_L : \delta_k = \max_{i \in C_L} \{\delta_i\}, \ p = 2,...,m \qquad (5.16)$$
$$\text{onde } \delta_i = \sum_{j \in C_L^c} c_{ij}$$

No Passo 4 do algoritmo 5.1, utilizaram-se sequencialmente os seguintes procedimentos pós-optimais:

(i) Tenta distribuir equilibradamente a procura pelas rotas. Para isso retira clientes de veículos que tenham a sua capacidade quase ou completamente preenchida e insere-os noutros que estejam menos cheios, de forma a que as distâncias percorridas sejam simultaneamente reduzidas (figura 5.1). Assim, para todas as rotas

$$r_k \in R \text{ tais que } Ocp(k) > 0{,}95Q$$

determina

$$r_l \in R, l \neq k, \text{ com } Ocp(l) < 0{,}9Q, i \in r_k, j \in r_l$$

tais que:

$$\begin{cases} Ocp(l) + q_i \leq Q \\ \Delta = \min_{i \in C_k} \min_{r_l \in R} \left\{ \min_{j \in C_l} \left\{ c_{i-1\,i+1} - c_{i-1\,i} - c_{i\,i+1} + c_{j\,i} + c_{i\,j+1} - c_{j\,j+1} \right\} \right\} \end{cases}$$

Se $\Delta < 0$ então faz:

$$Ocp(k) = Ocp(k) - q_i \text{ e } Ocp(l) = Ocp(l) + q_i$$
$$r_k = (0,...,i-1,i+1,...,0) \text{ e } r_l = (0,...,j-1,i,j+1,...,0)$$

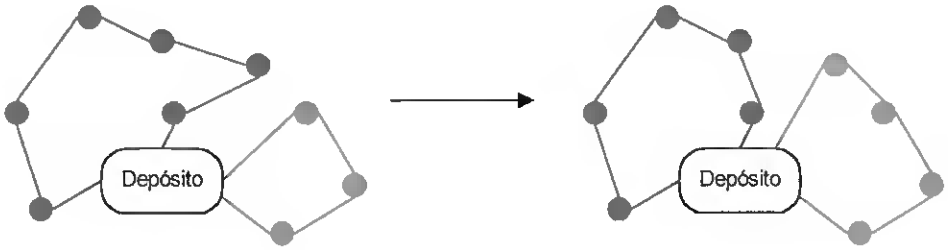


Figura 5.1 – Passagem de um cliente de uma rota para outra

(ii) Troca clientes entre duas rotas, desde que as distâncias diminuam. Em cada troca um cliente de uma rota e um cliente de outra rota trocam as suas posições, desde que melhorem a solução (figura 5.2). Isto é, para todas as rotas

$$r_k, r_l \in R,\ k \neq l, \text{ e para todos } i,\ i-1,\ i+1 \in r_k,\ j,\ j-1,\ j+1 \in r_l$$

$$\text{Se } \begin{cases} Ocp(k) + q_j - q_i < Q \\ Ocp(l) + q_i - q_j < Q \\ (c_{j-1\,i} + c_{i\,j+1}) + (c_{i-1\,j} + c_{j\,i+1}) < (c_{j-1\,j} + c_{j\,j+1}) + (c_{i-1\,i} + c_{i\,i+1}) \end{cases}$$

Faz:

$$Ocp(k) = Ocp(k) + q_j - q_i \text{ e } Ocp(l) = Ocp(l) + q_i - q_j$$
$$r_k = (0,...,i-1,j,i+1,...,0) \text{ e } r_l = (0,...,j-1,i,j+1,...,0)$$

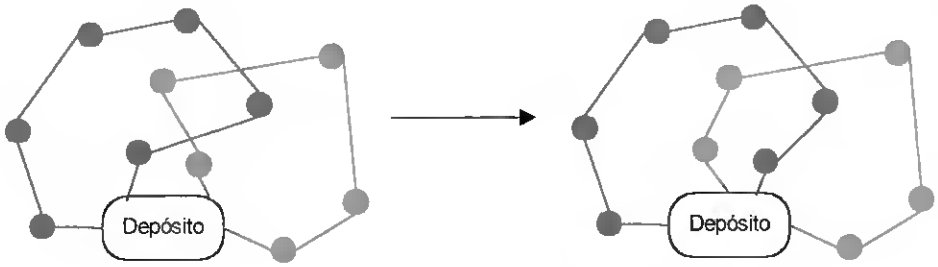


Figura 5.2 – Troca de um cliente entre duas rotas

(iii) Como no procedimento anterior, troca clientes entre duas rotas. Mas, neste caso, em cada troca, dois clientes consecutivos de uma rota ocupam a posição de outros dois clientes consecutivos da outra rota, desde que melhorem a solução (figura 5.3). Isto é, para todas as rotas

$$r_k, r_l \in R,\ k \neq l, \text{ e para todos } i, i-1, i+1, i+2 \in r_k,\ j, j-1, j+1, j+2 \in r_l$$

$$\text{Se } \begin{cases} Ocp(k) + q_j + q_{j+1} - q_i - q_{i+1} < Q \\ Ocp(l) + q_i + q_{i+1} - q_j - q_{j+1} < Q \\ \left(c_{j-1\,i} + c_{i\,i+1} + c_{i+1\,j+2}\right) + \left(c_{i-1\,j} + c_{j\,j+1} + c_{j+1\,i+2}\right) < \\ < \left(c_{j-1\,j} + c_{j\,j+1} + c_{j+1\,j+2}\right) + \left(c_{i-1\,i} + c_{i\,i+1} + c_{i+1\,i+2}\right) \end{cases}$$

Faz:

$$Ocp(k) = Ocp(k) + q_j + q_{j+1} - q_i - q_{i+1}$$
$$Ocp(l) = Ocp(l) + q_i + q_{i+1} - q_j - q_{j+1}$$
$$r_k = (0,\ldots,i-1, j, j+1, i+2,\ldots,0)$$
$$r_l = (0,\ldots,j-1, i, i+1, j+2,\ldots,0)$$

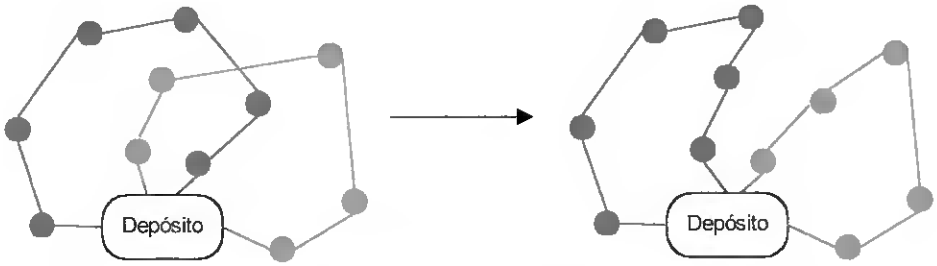


Figura 5.3 – Troca de dois clientes consecutivos entre duas rotas

5.3.2 FUNÇÃO DE APTIDÃO E SELECÇÃO

As formas de seleccionar a população, de acordo com o exposto em 3.3.3, baseiam-se na aptidão dos indivíduos (valor da função objectivo, classificação dos indivíduos, etc.). Os métodos de selecção mais utilizados são a amostragem com reposição (SAR), ou sem reposição (SUS). Na versão adaptada testaram-se estas duas

formas de selecção, nas outras versões utilizou-se apenas a selecção SUS. Implementaram-se, estas duas formas de selecção, da seguinte forma:

SELECÇÃO SAR

Como o PORV é um problema de minimização, é necessário definir a função de aptidão $Apt\left(x^i\right)$ de forma a que os indivíduos mais aptos sejam os que têm menor valor da função objectivo, $f\left(x^i\right)$ (de (5.11)), mantendo o valor da aptidão positivo. Considerou-se, então, que a aptidão era dada por:

$$Apt\left(x^i\right) = \frac{\sum_{j=1}^{Pop} f\left(x^j\right)}{f\left(x^i\right)} \ , \ i = 1,...,Pop \tag{5.17}$$

Em cada geração uma subpopulação de indivíduos (de dimensão *subpop*) da população actual é seleccionada para se reproduzir, através dos operadores genéticos, dando origem a uma nova população. A cada indivíduo corresponde uma parte da *roleta* de acordo com a sua aptidão de forma a que os mais aptos (melhores soluções) tenham uma maior probabilidade de serem seleccionados. A *roleta* foi construída da seguinte forma:

$$\begin{aligned} &roleta\ (0) = 0\ , \\ &roleta\ (i) = roleta\ (i-1) + Apt\left(x^i\right)\ ,\ \ i = 1,...,Pop\ . \end{aligned} \tag{5.18}$$

Ao indivíduo i corresponde o intervalo compreendido entre $roleta(i-1)$ e $roleta(i)$. O indivíduo a seleccionar é aquele cujo intervalo na roleta contiver o número aleatório gerado uniformemente no intervalo $[0, roleta(Pop)]$.

Nesta subpopulação pode-se encontrar o mesmo indivíduo mais do que uma vez.

Por exemplo, considere-se a população de 10 indivíduos da tabela 5.1

Indiv.	1	2	3	4	5	6	7	8	9	10
f	6	4	2	5	4	5	8	12	7	10
Apt	10,5	15,75	31,5	12,6	15,75	12,6	7,875	5,25	9	6,3
roleta	10,5	26,25	57,75	70,35	86,1	98,7	106,575	111,825	120,825	127,125

Tabela 5.1 – Exemplo da aptidão e roleta para uma população de 10 indivíduos (selecção SAR)

Seja a dimensão da subpopulação igual a 8 e suponha-se que foram obtidos os números aleatórios da tabela 5.2.

nºs aleat.	24	51	74	30	112	66	125	88

Tabela 5.2 – Números aleatórios gerados em [0, 127.125]

Seriam seleccionados os indivíduos da tabela 5.3 (ver figura 5.4):

selec.	2	3	5	3	9	4	10	6

Tabela 5.3 – Indivíduos seleccionados

SELECÇÃO SUS

Ordenaram-se os indivíduos pelo valor da função objectivo. Atribuiu-se uma aptidão 0 ao pior indivíduo e uma aptidão 2 ao melhor. Aos outros indivíduos foram atribuídas aptidões entre 0 e 2 de acordo com a sua classificação e igualmente espaçadas, isto é, fez-se $s = 2$ na expressão da função de avaliação dada em (3.2) .

Na versão básica a classificação dos indivíduos é feita considerando dois subconjuntos da população. Um primeiro contendo os indivíduos admissíveis ordenados por ordem crescente do valor da função objectivo (dada em (5.11), sendo o melhor o que percorre menor distância) e um segundo contendo os não admissíveis ordenados por ordem decrescente do valor da função objectivo, isto é, o pior indivíduo será o que mais excede a capacidade dos veículos de acordo com (5.12). Nas versões adaptada e híbrida os indivíduos são ordenados por ordem crescente do valor da função objectivo dada em (5.11).

Sejam:

$perm(i)$ o vector que indica a classificação (de 0 a *Pop*-1) do indivíduo *i* na população;

$const = 2/Pop - 1$.

Então a função de aptidão é dada por:

$$Apt(x^i) = 2 - perm(i) \times const\ ,\ i = 1,...,Pop \tag{5.19}$$

Construiu-se a roleta como em (5.18). Gerou-se um número aleatório uniformemente distribuído no intervalo $[0,1]$ e formou-se um ponteiro (com a dimensão da população) adicionando sucessivamente uma unidade. Para o exemplo anterior seleccionavam-se os indivíduos da tabela 5.4 (ver figura 5.5):

Indiv.	1	2	3	4	5	6	7	8	9	10
f	6	4	2	5	4	5	8	12	7	10
Apt	0,89	1,78	2,00	1,33	1,56	1,11	0,44	0,00	0,67	0,22
roleta	0,89	2,67	4,67	6,00	7,56	8,67	9,11	9,11	9,78	10,00

pont.	0,7	1,7	2,7	3,7	4,7	5,7	6,7	7,7	8,7	9,7

selec.	1	2	3	3	4	4	5	6	7	9

Tabela 5.4 – Aptidão, roleta, ponteiro e indivíduos seleccionados para uma população de 10 indivíduos (selecção SUS)

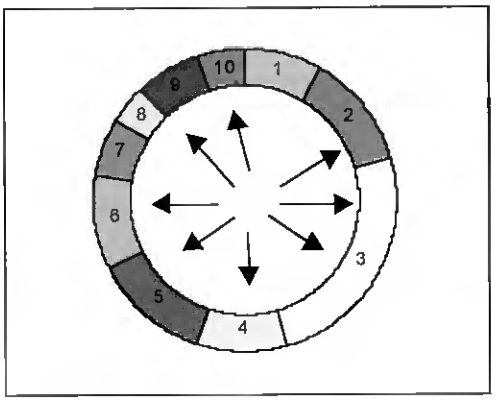


Figura 5.4 – Selecção SAR

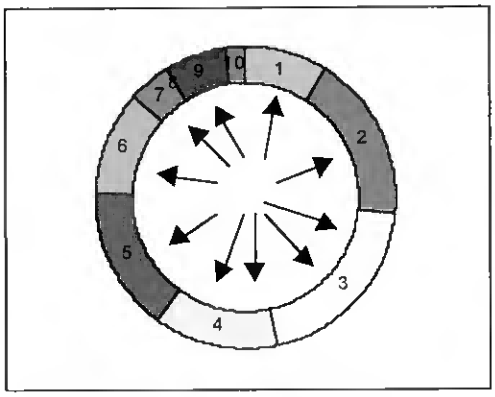


Figura 5.5 – Selecção SUS

5.3.4 CRUZAMENTO

O cruzamento de dois indivíduos foi repetido em cada geração um número de vezes igual a metade da dimensão da subpopulação. Para isso agruparam-se, aleatoriamente, os indivíduos seleccionados em pares (pais), de modo que não haja pares com elementos iguais. Obtiveram-se dois novos indivíduos (filhos) trocando, no máximo, uma certa percentagem dos alelos dos genes que ocupam o mesmo locus nos pais. Os locus, dos alelos a serem trocados, correspondem aos números aleatórios gerados uniformemente no intervalo $[1,n]$.

Nesta forma de cruzamento os filhos têm sempre os seus genes herdados (locus e alelo) de pelo menos um dos pais.

Nas versões adaptada e híbrida do AG só se efectuou a troca dos alelos que mantinham a solução admissível. Por exemplo, sejam:

Pai 1 [1 3 1 4 3 1 2 4 4 2 4 2]

Pai 2 [2 2 3 3 2 3 3 4 4 1 4 1]

e o número máximo de alelos a trocar igual a 4. Então, geram-se 4 números aleatórios diferentes no intervalo $[1,12]$, (por exemplo 5, 7, 1, 8). Se a capacidade dos veículos permitir as duas primeiras trocas e não permitir a terceira, obtêm-se:

Filho 1 [1 3 1 4 2 1 3 4 4 2 4 2]

Filho 2 [2 2 3 3 3 3 2 4 4 1 4 1]

Supondo que os dois indivíduos pais eram representados, graficamente, por:

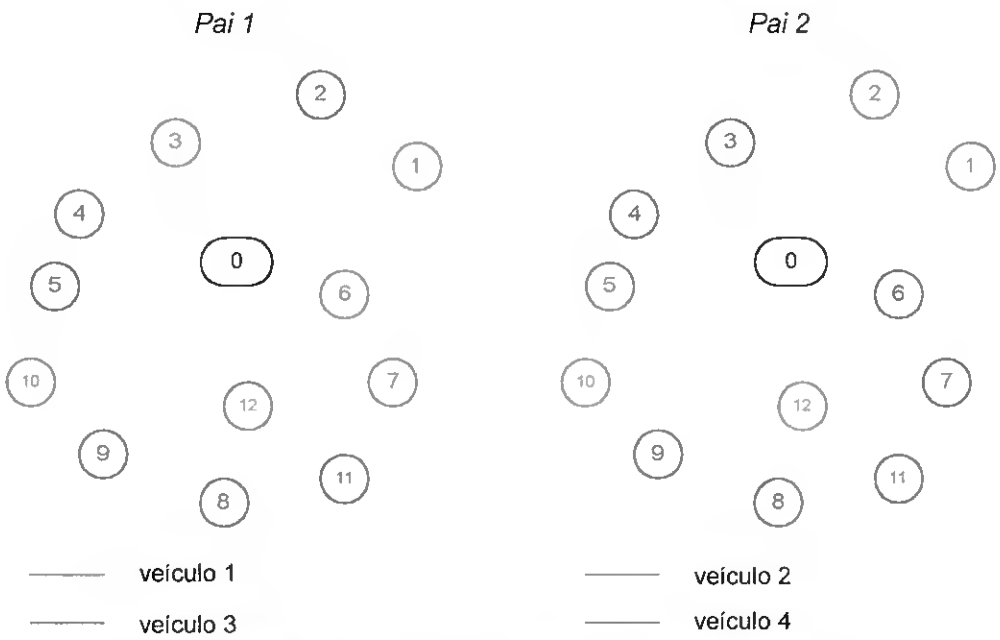


Figura 5.6 – Uma representação gráfica dos indivíduos que vão ser cruzados

Os filhos, de acordo com o exemplo, seriam:

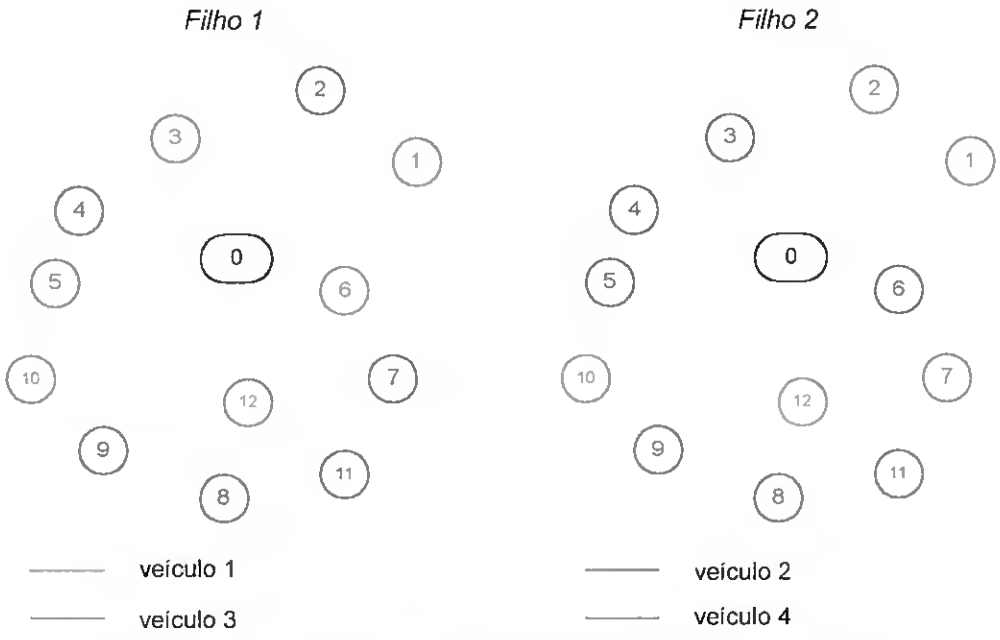


Figura 5.7 – Representação gráfica dos indivíduos resultantes do cruzamento

Com este tipo de actuação do operador cruzamento obtêm-se afectações, em geral, com agrupamentos de clientes pouco diferentes dos existentes nas afectações que lhes deram origem. No entanto, esta troca aleatória de clientes entre agrupamentos possibilita uma pesquisa diversificada pelo espaço das afectações admissíveis. Como, durante o algoritmo, em cada iteração são executados vários cruzamentos entre afectações distintas é esperado que alguns deles possam conduzir às melhores afectações dos clientes aos veículos.

Após esta operação, dos quatro indivíduos envolvidos (pais e filhos), testaram-se as seguintes formas para escolher os que fariam parte da nova população (mencionadas em 3.3.6):

a) Passam a fazer parte da nova população os dois mais aptos (que são os que têm menor valor da função objectivo). Por exemplo, supondo que se tinha os seguintes valores da função objectivo, calculados de acordo com (5.11):

$$f(Pai1) = 483, \qquad f(Filho1) = 487,$$
$$f(Pai2) = 472, \qquad f(Filho2) = 480,$$

o *Pai 2 e o Filho 2* seriam os mais aptos e, por isso, fariam parte da nova população;

b) Os pais são substituídos pelos filhos independentemente dos respectivos valores da aptidão. Assim, a nova população é formada pelos filhos e a solução final é a melhor encontrada durante todas as gerações, podendo não pertencer à última geração;

c) Como em b) mas, paralelamente forma-se uma lista com os cinco melhores indivíduos que vão sendo introduzidos um a um na população nas gerações seguintes;

d) Os filhos substituem os piores indivíduos da população.

5.3.5 MUTAÇÃO

Para cada indivíduo (pai) mudou-se, no máximo, o alelo de um número previamente fixo dos seus genes, obtendo-se um novo indivíduo (filho). Os locus destes genes também foram escolhidos aleatoriamente com distribuição uniforme no intervalo $[1,n]$. Lembrando que o alelo (valor) k de um gene j, num indivíduo i, significa que o cliente j vai ser servido pelo veículo k ($y_{ik} = 1$), a mutação consistiu em afectar, quando possível, esse cliente ao veículo que tivesse maior capacidade livre. No caso da versão adaptada a mutação só foi efectivada se mantivesse a admissibilidade. Esta operação foi executada um certo número de vezes em cada geração.

Considerando, por exemplo, que se aplicava a mutação ao *Pai 1* do exemplo anterior (figura 5.6), e supondo que o veículo 3 tinha maior capacidade livre (já que só tem dois clientes para servir) provavelmente ser-lhe-ia afectado mais um cliente, bastando para isso que pelo menos um dos números aleatórios gerados fosse diferente de 2 e 5 e, na versão adaptada, que a sua procura não excedesse a capacidade livre do veículo 3.

Assim, este operador terá um efeito muito reduzido em problemas muito "apertados", isto é, nos casos em que a procura total dos clientes é quase igual à capacidade total dos veículos, mas nos outros poderá contribuir para equilibrar o peso das rotas.

Após esta operação foram testadas duas alternativas em relação aos indivíduos que fariam parte da nova população (ver 3.7):

a) o filho substitui sempre o pai;

b) o filho substitui o pai quando é melhor.

5.3.5.1 DOMINÂNCIA

Partindo da hipótese que nas melhores soluções de cada problema certos clientes são afectos da mesma forma optou-se, na versão adaptada, por utilizar uma nova forma de mutação que, continuando com a analogia genética, transmitisse à nova população certas características *dominantes* "*boas*" eliminando outras *dominantes* "*más*" e as *recessivas*.

Considerou-se como características dominantes "boas" ("más") as afectações comuns dos três indivíduos mais aptos (menos aptos), de toda a população. As afectações diferentes destas eram recessivas.

Sendo D^+ e D^- os conjuntos dos genes dominantes "bons" e "maus" respectivamente e considerando os indivíduos ordenados de uma forma estritamente decrescente das aptidões, isto é, x^1, x^2, x^3, são os mais aptos e x^{Pop}, $x^{Pop\text{-}1}$, $x^{Pop\text{-}2}$, os menos aptos (com valores de aptidão diferentes), tem-se:

$$D^+(j)=\begin{cases} x_j^1 & \text{se} \quad x_j^1 = x_j^2 = x_j^3 \\ 0 & \text{caso contrário} \end{cases}, \quad j = 1,...,n \tag{5.20}$$

$$D^-(j)=\begin{cases} x_j^{Pop} & \text{se} \quad x_j^{Pop} = x_j^{Pop-1} = x_j^{Pop-2} \neq D^+(j) \\ 0 & \text{caso contrário} \end{cases}, \quad j = 1,...,n \tag{5.21}$$

Escolheu-se aleatoriamente um certo número de indivíduos e mudaram-se, quando a admissibilidade se mantinha, os alelos dos genes dominantes "maus" para os alelos dos genes dominantes "bons", se existissem, senão para os alelos dos genes do melhor indivíduo. Transmitiram-se, também, os valores dos genes dominantes "bons" aos genes recessivos. Assim, desde que se mantenha a admissibilidade, os genes dominantes "maus" são sempre alterados, os recessivos só são alterados se existir um gene dominante "bom".

Então, para um indivíduo x^i, escolhido aleatoriamente no intervalo $[2, Pop]$, procedeu-se da seguinte forma:

Para $j = 1$ até n

Se $x_j^i = D^-(j)$

Então $x_j^i = x_j^1$ (desde que seja admissível)

Senão

Se $x_j^i \neq D^+(j) \wedge D^+(j) \neq 0$

Então $x_j^i = D^+(j)$

Por exemplo, suponha-se que o melhor indivíduo de uma população, os genes dominantes "bons" e "maus" são, respectivamente:

x^1 [1 2 2 2 2 1 1 3 3 3]

D^+ [0 2 0 2 0 1 1 0 0 0]

D^- [0 0 0 0 0 0 3 2 0 0]

Seja o indivíduo x^i dado por:

x^i [1 1 3 2 2 3 3 2 3 1]

Então supondo que as capacidades dos veículos permitem as alterações no indivíduo x^i, obtém-se:

x^i [1 2 3 2 2 1 1 3 3 1]

Representando, graficamente, o indivíduo x^i por:

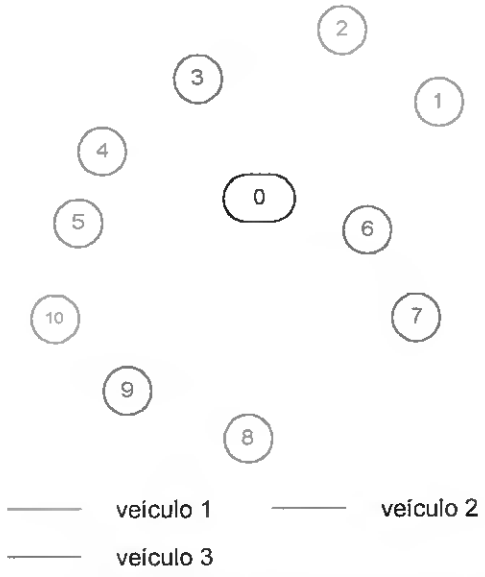


Figura 5.8 – Uma representação gráfica do indivíduo seleccionado para ser sujeito ao operador dominância

Obtinha-se de acordo com o exemplo e após a actuação deste operador:

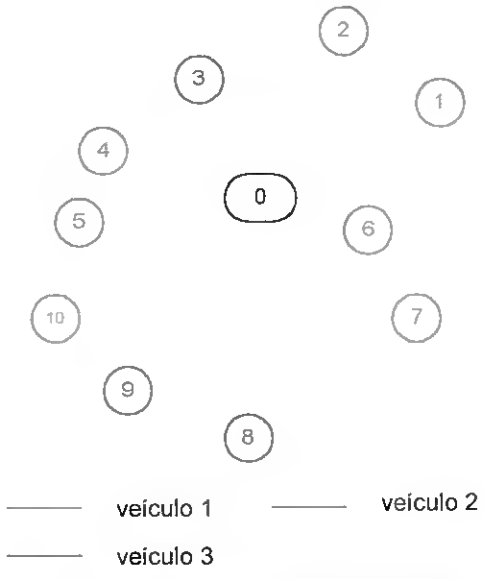


Figura 5.9 – Representação gráfica do indivíduo resultante da aplicação do operador dominância

Notando que o melhor indivíduo tinha, neste exemplo, a seguinte representação:

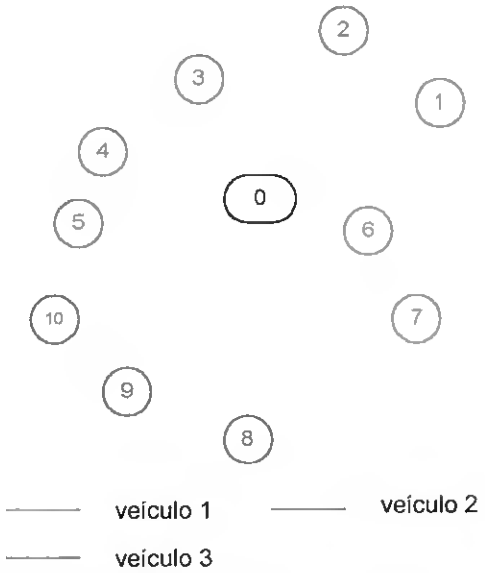


Figura 5.10 – Representação gráfica do melhor indivíduo

Verifica-se, pelo menos em alguns casos, que o indivíduo que sofre a aplicação deste operador fica com agrupamentos de clientes mais semelhantes aos do melhor indivíduo. Este operador pode, portanto, levar a uma intensificação da pesquisa em torno das melhores afectações, o que será vantajoso, mas simultaneamente pode levar a uma convergência muito rápida para soluções idênticas atingindo, consequentemente, apenas um óptimo local.

Com esta abordagem pretendeu-se, também, transpor para a codificação não binária a ideia do "esquema" (descrita em 3.3), subjacente aos AGs (com codificação binária) desenvolvidos por Holland (1975).

Neste operador, o filho só substituiu o pai quando melhorava o valor da função objectivo.

5.3.5.2 PÓS-OPTIMIZAÇÃO

De acordo com a forma como se definiram os operadores cruzamento e mutação, já descritos, é esperado que a sua actuação produza uma pesquisa diversificada no espaço das afectações admissíveis dos clientes aos veículos e que a intensifique em

torno das melhores afectações. Mas, se por um lado a aleatoriedade da pesquisa deverá, em termos probabilisticos, encontrar boas afectações, por outro lado o facto de não ter em conta a configuração das rotas, pode trazer algumas desvantagens.

Assim, na versão híbrida, manteve-se o operador cruzamento na forma descrita, mas substituíram-se os dois operadores de mutação anteriores por outro, que tem em conta a configuração das rotas e a distância nelas percorrida, baseado em procedimentos de *k*-trocas de clientes entre rotas. Introduziram-se, para isso, no AG híbrido, os procedimentos de pós-optimização (i), (ii) e (iii), incluídos na heurística construtiva de rota, utilizada para gerar a população inicial, descritos em 5.3.2.1. Aplicaram-se estes procedimentos aos indivíduos resultantes de cruzamentos admissíveis, e também no final, à melhor solução obtida durante as várias gerações.

Para o exemplo dado a propósito do operador cruzamento e considerando que para as duas afectações resultantes se obtinham as seguintes rotas:

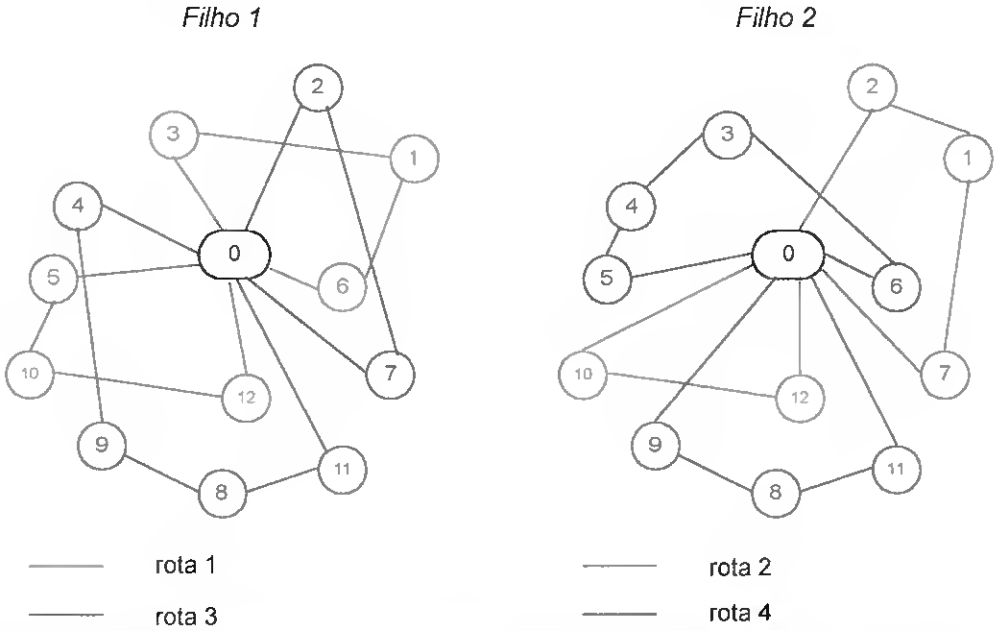


Figura 5.11 – Exemplo gráfico das rotas para os indivíduos resultantes do cruzamento

Podiam-se obter, após a actuação dos procedimentos de pós-optimização, aplicados a cada um dos indivíduos, as rotas da figura 5.12, supondo que para o *Filho 1* a aplicação do procedimento (i) permitia a inserção do cliente 4 na rota 1 entre o cliente 3 e o depósito e a do procedimento (ii) produzia a troca dos clientes 1 (rota 1) e 2 (rota

3) e respectivas ligações. Para o *Filho 2* apenas o cliente 6 mudava de rota devido à actuação do procedimento (i).

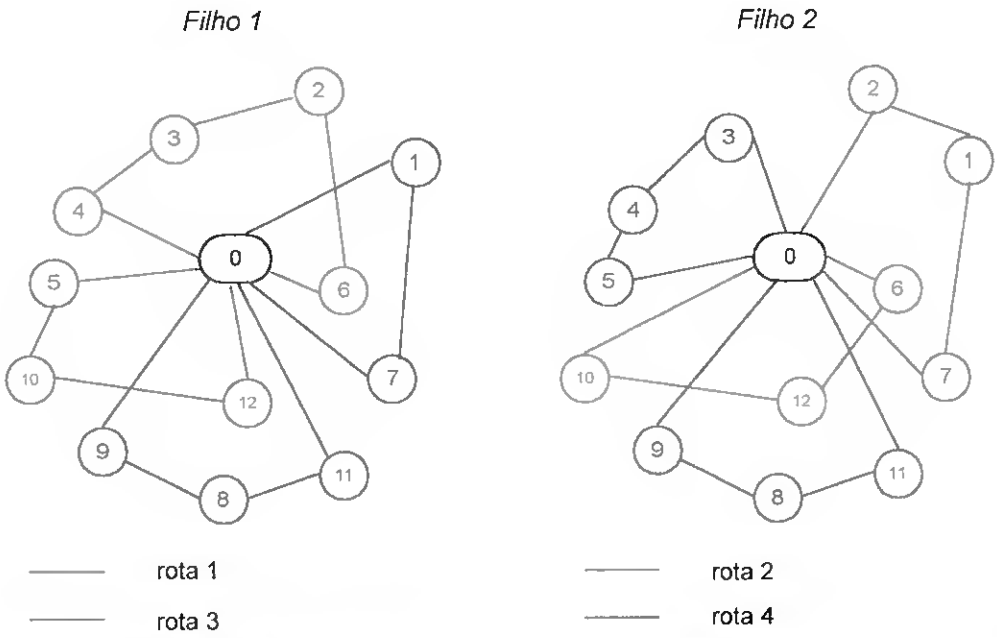


Figura 5.12 – Exemplo gráfico dos indivíduos resultantes da aplicação dos procedimentos de pós-optimização

Subjacente ao desenvolvimento destes AGs para o PORV básico está a suposição de que a maioria das soluções obtidas através das heurísticas usuais (construtivas, de melhoramento e de duas fases), contém boas distribuições dos clientes pelos veículos e boas rotas. Assim, um AG ao combinar várias destas soluções deverá ter uma forte probabilidade de, após um certo número de iterações, encontrar uma solução próxima da óptima.

Os operadores, no AG adaptado, actuam especialmente a nível das afectações de clientes aos veículos provocando, em cada iteração, pequenas alterações nas soluções. No AG híbrido o operador de pós-optimização também provoca algumas alterações nas afectações mas actua, principalmente, a nível das configurações das rotas a partir das trocas efectuadas.

Partindo de pressupostos idênticos, Kelly e Xu (1999), propõem um algoritmo em que inicialmente geram um grande conjunto de rotas distintas, tiradas de várias soluções obtidas com heurísticas simples. Posteriormente, identificam as "boas" rotas e juntam-nas numa solução resolvendo um Problema de Partição de Conjunto através

duma meta-heurística de Pesquisa Tabu. Em cada iteração trocam uma rota da solução corrente por uma (semelhante) do conjunto de rotas geradas, e eventualmente corrigem, utilizando procedimentos ávidos, as situações de clientes com múltiplas afectações e de clientes não servidos.

Diferentemente dos AGs que operam com um conjunto de soluções modificando algumas, em cada iteração, através da aplicação dos operadores genéticos, este algoritmo constrói e modifica uma solução a partir de um conjunto de rotas. A escolha da rota a substituir e dos procedimentos de obtenção da nova solução é conduzida através de funções de memória de curto termo, de memória de longo termo e de oscilação estratégica, envolvendo um grande número de parâmetros.

Por exemplo, considerando a figura 5.12 e supondo que numa iteração a solução corrente deste algoritmo era o *Filho* 1 e que entre o conjunto das rotas geradas se encontrava a rota 2 do *Filho* 2, esta poderia substituir a rota 3 (figura 5.13). Mas como o cliente 2 passava a estar afecto a dois veículos eram utilizados procedimentos ávidos para tornar a solução admissível (figura 5.13).

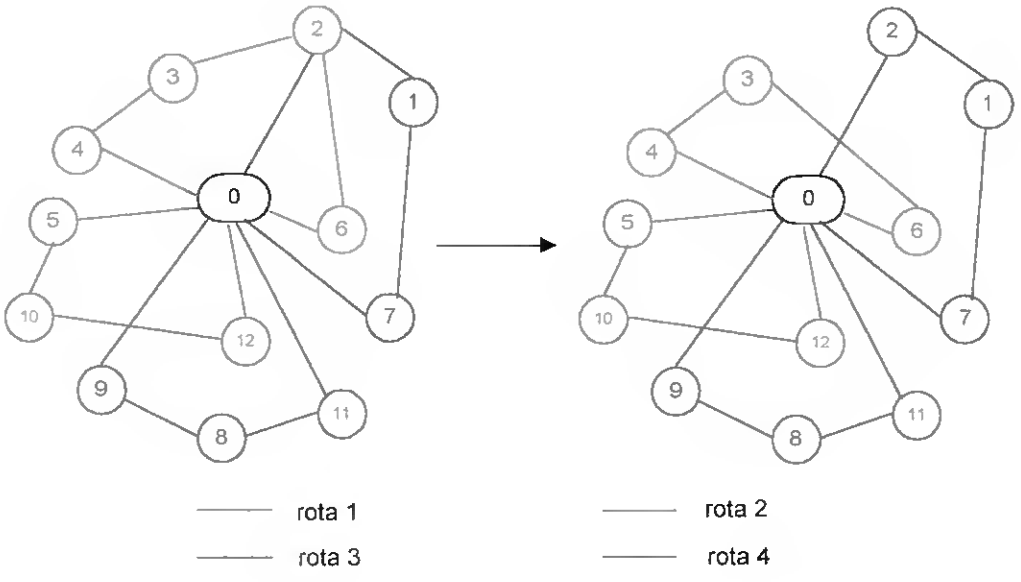


Figura 5.13 – Exemplo de uma substiutição de rota e obtenção de uma solução admissível [Kelly e Xu (1999)]

Após um número pré-determinado de iterações o algoritmo termina. À solução obtida são aplicadas duas heurísticas para o PCV (um método 3-optimal e uma PT) para melhorar a configuração das rotas [Kelly e Xu (1999)].

Rego (1998) propõe um algoritmo de PT, para o PORV, cujos movimentos para passar de uma solução a outra se baseiam numa cadeia de ejecção de arestas, partindo também da suposição que em cada iteração apenas algumas das arestas da solução actual não pertencem à solução óptima.

Em cada iteração identifica uma *estrutura referencial em flor* que consiste num subgrafo com vários ciclos e um caminho. O caminho (*caule*) liga o centro da flor (que corresponde ao depósito) a um vértice (*raiz*). Cada ciclo (*pétala*) corresponde a uma rota. A selecção das arestas a serem apagadas e inseridas para modificar a estrutura referencial em flor é definida por um processo de PT (a *baixo nível*) . Um algoritmo de PT (a *alto nível*) conduz, também, a pesquisa incorporando o processo da cadeia de ejecção.

Por exemplo, considerando de novo a figura 5.12 e supondo que a solução actual era o *Filho* 1, tem-se na figura 5.14 uma estrutura referencial em flor

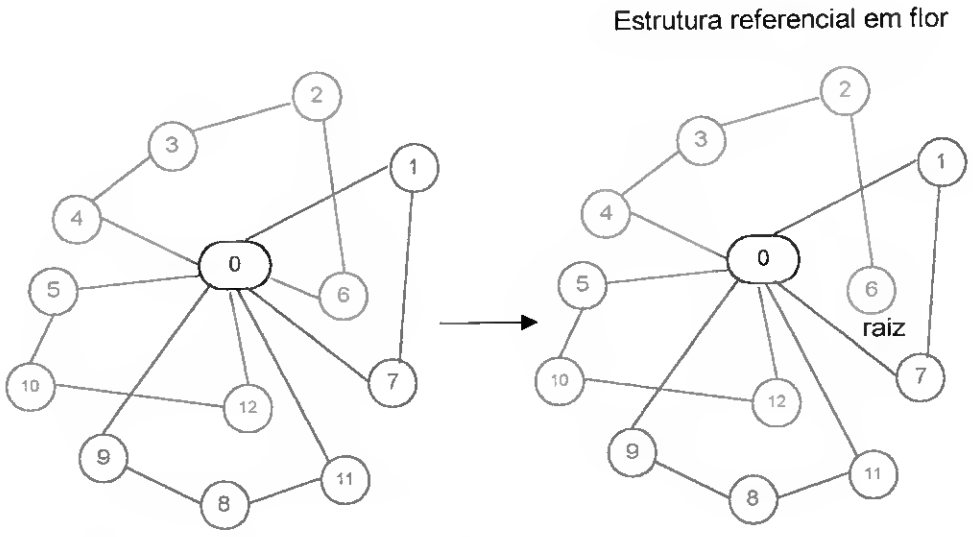


Figura 5.14 – Exemplo para a construção de uma estrutura referencial em flor

Aplicando as regras de ejecção [Rego (1998)] poder-se-ia obter as estruturas em flor da figura 5.15:

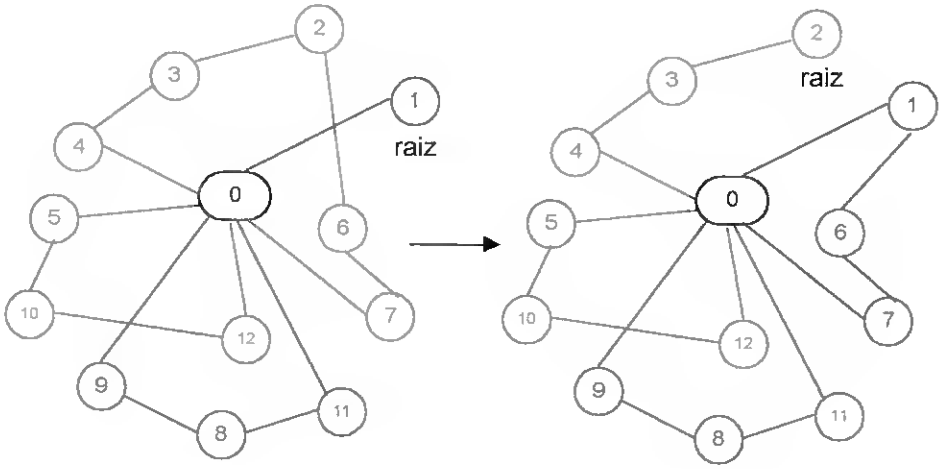


Figura 5.15 – Exemplo da aplicação das regras de ejecção

Seguindo as regras de passagem da estrutura referencial em flor a uma solução [Rego (1998)] poder-se-ia obter a solução da figura 5.16.

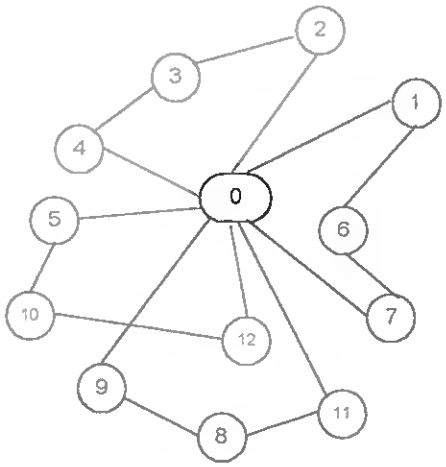


Figura 5.16 – Exemplo de uma nova solução

A utilização das regras de ejecção e a passagem da estrutura referencial em flor a uma solução pode produzir soluções intermédias não admissíveis e reduzir ou aumentar o número de veículos, que neste algoritmo não pode estar previamente fixo. No final, as rotas da solução obtida são reoptimizadas utilizando um algoritmo de PT para o PCV.

Estes dois algoritmos de PT provocam alterações apenas numa solução enquanto os AGs propostos alteram várias soluções em cada iteração. A utilização de PT envolve, em geral, um grande número de parâmetros. Valores diferentes para esses parâmetros ocasionam soluções bastante diferentes. Por exemplo, Rego (1998) melhora as soluções obtidas alterando apenas o intervalo de variação de um parâmetro. A determinação dos melhores valores para os parâmetros, principalmente no caso do algoritmo de Kelly e Xu (1999) que opera com um número considerável de parâmetros, é uma tarefa algo exigente.

De acordo com a descrição da forma de aplicar os operadores genéticos resumem-se, nos algoritmos 5.2, 5.3 e 5.4, as três versões do AG aqui desenvolvidas. Nestes algoritmos sejam:

x – melhor solução

sol – valor da função objectivo para x

α – o número máximo de genes a trocar em cada cruzamento

β – o número máximo de genes a serem modificados em cada mutação

Selec – vector que contém os índices dos indivíduos seleccionados para se reproduzirem

Maxger – o número máximo de gerações a executar

pm – probabilidade de mutação

Adm – conjunto dos indivíduos admissíveis

Subpop – número de indivíduos da subpopulação que se vai reproduzir

Cruza – o número de cruzamentos a executar em cada geração

Muta – o número de indivíduos a sofrerem mutações em cada geração

Dom – o número de indivíduos aos quais se aplica o operador dominância em cada geração

Passo 1:

Gerar $n \times Pop$ números aleatórios uniformemente distribuídos em $[1,...,m]$ obtendo Pop indivíduos, $x^1,...,x^{Pop}$

Calcular $f(x^i)$ de acordo com (5.11) para $x^i \in Adm$

e de acordo com (5.12) para $x^i \notin Adm$

Fazer $x = x^i : f(x^i) = \min_{x^j \in Adm} f(x^j)$ e $sol = f(x)$

$Geração = 0$

Passo 2:

Calcular $Apt(x^i)$, $i = 1,...,Pop$, conforme (5.19)

Actualizar x e sol. $Geração = Geração + 1$

Construir a *roleta* como em (5.18).

Para $j = 1$ até Pop

Nal = nº aleatório gerado uniformemente em $[0,1]$

$Selec(j) = i$ talque $roleta(i-1) < Nal + j - 1 \leq roleta(i)$

Passo 3:

Para $j = 0$ até $Pop - 2$

$Nal1$ = nº aleatório gerado uniformemente em $[1, Pop - j]$

$Pai1 = Selec(Nal1)$

$Pai2 = Pai1$

Enquanto $Pai2 = Pai1$

$Nal2$ = nº aleatório gerado uniformemente em $[1, Pop - j]$

$Pai2 = Selec(Nal2)$

$j = j + 2$

Gerar α números aleatórios diferentes uniformemente distribuídos em $[1, n]$

Cruzar $Pai1$ e $Pai2$ obtendo $Filho1$ e $Filho2$ trocando sempre os alelos correspondentes aos locus dos α números aleatórios

Substituir indivíduos de acordo com 5.3.4.

Passo 4:

Para $j = 1$ até Pop

Para $i = 1$ até n

Nal = nº aleatório gerado uniformemente em $[0,1]$

Se $Nal < pm$

Mutar o indivíduo j alterando o alelo i de acordo com 5.3.5

Substituir indivíduo de acordo com 5.3.5.

Passo 5:

Actualizar x e sol

Se $Geração \leq Maxger$

voltar ao Passo2 (com a nova população)

Caso contrário

terminar obtendo x, sol e respectivas rotas

Algoritmo 5.2 – Algoritmo Genético versão básica

Passo 1:

Executar a heurística construtiva de rota, variando os clientes iniciais de acordo com o exposto no Passo 1 (do algoritmo 5.1) e com (5.14), (5.15) e (5.16) até obter *Pop* indivíduos, $x^1,...,x^{Pop}$

Calcular $f(x^i)$, $i = 1,...,Pop$, de acordo com (5.11)

Fazer $x = x^i : f(x^i) = \min_{j=1,...,Pop} f(x^j)$ e $sol = f(x)$

$Geração = 0$

Passo 2:

$Geração = Geração + 1$

Calcular $Apt(x^i)$, $i = 1,...,Pop$, conforme (5.17) ou (5.19)

Construir a *roleta* como em (5.18)

Seleccionar subpopulação a cruzar aleatoriamente através do método SAR ou SUS (descritos em 5.3.3)

Passo 3:

Para $j = 1$ até *Cruza*

$Nal1$ = nº aleatório gerado uniformemente em $[1, subpop]$

$Pai1 = x^i$ tal que $i - 1 < Nal1 \leq i$

$Pai2 = Pai1$

Enquanto $Pai2 = Pai1$

$Nal2$ = nº aleatório gerado uniformemente em $[1, subpop]$

$Pai2 = x^k$ tal que $k - 1 < Nal2 \leq k$

Gerar α números diferentes uniformemente distribuídos em $[1, n]$

Cruzar *Pai*1 e *Pai*2 obtendo *Filho*1 e *Filho*2 trocando sempre os alelos correspondentes aos locus dos α números aleatórios

Substituir indivíduos de acordo com 5.3.4.

Passo 4:

Para $j = 1$ até *Muta*

Gerar β números diferentes uniformemente distribuídos em $[1, n]$

Mutar *Pai* (indivíduo *j*) obtendo *Filho*, alterando os alelos correspondentes aos locus dos β números aleatórios, como em 5.3.5

Substituir indivíduo conforme descrito em 5.3.5

Passo 5:

Para $j = 1$ até *Dom*

Nal = nº aleatório gerado uniformemente em $[2, Pop]$

$Pai = x^i$ tal que $i - 1 < Nal \leq i$

Aplicar a *Pai* o operador dominância obtendo *Filho*

Se $A(Filho) > A(Pai)$ então substituir *Pai* por *Filho*

Passo 6:

Actualizar *x* e *sol*

Se $Geração \leq Maxger$

voltar ao Passo2 (com a nova população)

Caso contrário terminar obtendo *x*, *sol* e respectivas rotas

Algoritmo 5.3 – Algoritmo Genético versão adaptada

Na versão híbrida considerou-se que o algoritmo convergia quando durante cinco gerações consecutivas o valor global da aptidão da população não aumentava.

Passo 1:

Executar a heurística construtiva de rota, descrita anteriormente (algoritmo 5.1), variando os clientes iniciais de acordo com o exposto no Passo 1 (dessa heurística) e com (5.14), (5.15) e (5.16) até obter *Pop* indivíduos, $x^1,...,x^{Pop}$

Calcular $f(x^i)$, $i = 1,...,Pop$, de acordo com (5.11)

Fazer $x = x^i : f(x^i) = \min_{j=1,...,Pop} f(x^j)$ e $sol = f(x)$

$Geração = 0$

Passo 2:

$Geração = Geração + 1$

Calcular $Apt(x^i)$, $i = 1,...,Pop$, conforme (5.19)

Construir a *roleta* como em (5.18)

Para $j = 1$ até Pop

Nal = nº aleatório gerado uniformemente em $[0,1]$

$Selec(j) = i$ talque $roleta(i-1) < Nal + j - 1 \leq roleta(i)$

Passo 3:

Para $j = 0$ até $Pop - 2$

$Nal1$ = nº aleatório gerado uniformemente em $[1, Pop - j]$

$Pai1 = Selec(Nal1)$

$Pai2 = Pai1$

Enquanto $Pai2 = Pai1$

$Nal2$ = nº aleatório gerado uniformemente em $[1, Pop - j]$

$Pai2 = Selec(Nal2)$

$j = j + 2$

Gerar α números aleatórios diferentes uniformemente distribuídos em $[1, n]$

Cruzar *Pai*1 e *Pai*2 obtendo *Filho*1 e *Filho*2 trocando sempre os alelos correspondentes aos locus dos α números aleatórios

Se *Filho*1 (*Filho*2) é admissível aplicar-lhe os procedimentos pós-optimais (i), (ii) e (iii) descritos em 5.3.2.1.

Substituir *Pai1* e *Pai2* conforme descrito em (5.3.4 a))

Passo 4:

Actualizar x e sol

Se $Geração \leq Maxger \wedge$ o algoritmo não convergiu

voltar ao Passo2 (com a nova população)

Caso contrário

terminar obtendo x, sol e respectivas rotas

Algoritmo 5.4 – Algoritmo Genético versão híbrida

5.3.6 PARALELIZAÇÃO

Paralelizou-se a melhor destas versões do AG utilizando 2, 4 e 8 transputers. O modelo utilizado, a topologia e as comunicações efectuadas foram exactamente as mesmas adoptadas e descritas no capítulo 4, para a paralelização do AG desenvolvido para o problema PQ 0-1.

Assim, também, tanto para 4 como para 8 transputers utilizaram-se duas topologias (top4 e top4c, top8 e top8c). Numa só são trocados indivíduos entre dois processadores uma vez (top4 e top 8). Na outra são trocados alguns indivíduos várias vezes entre diferentes processadores (top4c e top8c). Pretendia-se, com estas topologias, verificar se uma maior troca de indivíduos entre populações distintas melhorava a solução.

O encaminhamento dos dados para os processadores bem como as trocas de indivíduos entre processadores foram feitas como descrito em 4.5.4.

O último Passo do AG (para qualquer uma das versões) passa a ser, no algoritmo paralelo:

Último Passo:

Actualizar x e *sol*
Se *Geração* $\leq$ *Maxger*
Se *Geração* $= {}^{a}/_{b}$ *Maxger* , com $a = 1$; $b = 3$ (top4),
$a = 1, 2$; $b = 3$ (top4c)
$a = 1$; $b = 4$ (top8),
$a = 1, 2, 3$; $b = 4$ (top8c)
Trocar indivíduos
Voltar ao Passo2 (com a nova população)
Caso contrário
terminar obtendo x, *sol* e respectivas rotas

5.4 EXPERIÊNCIA COMPUTACIONAL

Testaram-se as três versões do AG descritas na secção anterior num conjunto de problema conhecidos da literatura (tabela 5.5), e disponíveis na Internet, no endereço http://www–apache.imag.fr/~paugerat/VRP/instances/, de onde foram obtidos:

Nº dos problemas	Fonte dos problemas	Tipo dos dados
1 a 5, 8, 11	Eilon et al (1971)	Gerados uniformemente.
6, 7, 9		Iguais ao 8, com outro número e capacidade dos veículos.
10,12		Iguais ao 11, com outro número e capacidade dos veículos.
13 a 15	Fisher (1994)	Obtidos a partir de aplicações a casos reais.
16 e 17	Christofides et al (1979)	Estruturados - os clientes formam grupos (clusters).
18		Junção dos clientes dos problemas 5 e 11.
19		Junção dos clientes do problema 18 com os primeiros 49 clientes do problema 8.

Tabela 5.5 – Problemas-teste da literatura obtidos na Internet

As suas características estão resumidas na tabela 5.6. Nesta tabela os valores do número de veículos e da melhor solução (ou da solução óptima) são os que constavam dos ficheiros com os dados obtidos da Internet. Os valores sublinhados correspondem à solução óptima.

Todos estes ficheiros de dados contêm as coordenadas cartesianas do depósito e dos clientes. Como não referem o tipo de distâncias que foram utilizadas (inteiros ou reais), mas como o melhor valor da solução, neles indicado, é inteiro, calcularam-se as distâncias usando reais e arredondou-se o resultado à unidade.

Nº do problema	Nº de Clientes	Nº de Veículos	(1)	Melhor solução
1	21	4	0.938	375
2	22	3	0.755	569
3	29	3	0.944	534
4	32	4	0.918	835
5	50	5	0.971	521
6	75	7	0.886	683
7	75	8	0.947	735
8	75	10	0.974	832
9	75	14	0.974	1032
10	100	4	0.911	681
11	100	8	0.911	817
12	100	14	0.930	1077
13	44	4	0.898	728
14	71	4	0.957	238
15	134	7	0.945	1165
16	100	10	0.905	820
17	120	7	0.982	1065
18	150	12	0.931	1053
19	199	17	0.937	1351

$$(1) = \frac{\text{Total da procura dos clientes}}{\text{Capacidade total dos veículos}}$$

Tabela 5.6 – Características dos problemas-teste

Inicialmente fez-se um estudo comparativo das três versões do AG implementadas, incluindo variação de parâmetros e as várias abordagens aos operadores genéticos descritas em 5.3. Depois compararam-se os resultados obtidos através da

melhor versão dos AGs com os resultados de outras heurísticas. Por fim procedeu-se à paralelização, da versão do AG que obteve melhores resultados.

As diferentes versões dos AG foram implementadas em Turbo C e executadas num PC com um processador Pentium MMX 200 MHz e com 16 MB de RAM.

5.4.1 ALGORITMO GENÉTICO BÁSICO

Começou-se por testar o AG básico (algoritmo 5.2). Como era de esperar, a população inicial não continha sequer uma solução admissível. Em geral, ao fim de algumas gerações (entre 15 e 30) o AG encontrava uma solução admissível mas de muito fraca qualidade e no fim do algoritmo (com *maxger* igual a 40 e também a 80) essa solução pouco ou nada tinha melhorado. Este ("mau") comportamento foi semelhante em todos os testes, que incluíram vários valores para α e β e escolha dos indivíduos a fazerem parte da nova população de acordo com as diferentes estratégias mencionadas. Assim, este tipo de algoritmo "cego" em relação ao problema (complexo e não binário) mostrou-se, naturalmente, muito ineficiente.

5.4.2 ALGORITMO GENÉTICO ADAPTADO

Em relação ao AG adaptado (algoritmo 5.3) analisou-se, primeiro, a contribuição dos diferentes operadores para a sua eficiência. Para isso, executou-se o AG, várias vezes, variando os parâmetros respeitantes aos operadores e mantendo constante a dimensão da população e o número de gerações. Estabeleceu-se a dimensão da população em 40 ($Pop = 40$), com o intuito de garantir alguma diversidade e simultaneamente obter tempos de execução reduzidos. Note-se que a geração heurística da população é computacionalmente muito pesada. Fixou-se em 35 o número máximo de gerações a executar ($Maxger = 35$). Mantendo estes valores foram realizados os seguintes testes:

5.4.2.1 SELECÇÃO E CRUZAMENTO

Começou-se por estudar qual dos métodos de selecção descritos produzia melhores resultados. Para isso, testou-se o AG só com os operadores selecção e cruzamento. Para a selecção aleatória com reposição (SAR) considerou-se adequado que cerca de 75% da população fosse seleccionada para se reproduzir (*subpop* = 30). Como cada cruzamento envolve dois indivíduos executaram-se 15 cruzamentos (*Cruza* = 15) em cada geração. Para a selecção SUS, como já foi explicado anteriormente, o número de cruzamentos é igual a metade da população (*subpop* = *Pop*), por isso, efectuaram-se 20 cruzamentos em cada geração.

Simultaneamente estudou-se qual o melhor valor para o número máximo de genes a serem trocados (α) em cada cruzamento.

Testaram-se as diferentes formas, descritas em 5.3.4 de escolher os indivíduos que vão fazer parte da nova população e concluiu-se que a melhor era a de a), isto é, os filhos só substituem os pais quando têm um menor valor da função objectivo. Nos outros casos, o algoritmo convergia mais rapidamente mas nunca melhorou a qualidade da solução, tendo geralmente piorado. Os resultados, apenas para este caso a), encontram-se nas tabelas 5.7 e 5.8. Estas tabelas contêm as diferenças percentuais (*dp*) obtidas, em que,

$$dp = \frac{sol - sol^*}{sol^*} \times 100 \quad , \quad \text{onde } sol \text{ é a solução obtida}$$

e sol^* é a melhor solução conhecida

Em cada tabela o melhor valor médio das diferenças percentuais está em negrito.

Os valores médios das diferenças percentuais estão representados nos gráficos 5.1 e 5.2.

	Cruzamento									
Pr. Nº	α = 0	α = 10	α = 20	α = 30	α = 40	α = 50	α = 60	α = 70	α = 80	α = 90
1	1,07	1,07	1,07	0,00	1,07	1,07	1,07	1,07	1,07	1,07
2	0,18	0,18	0,18	0,18	0,18	0,18	0,00	0,18	0,00	0,18
3	1,12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
4	5,63	4,91	4,91	5,03	5,03	4,91	4,91	5,03	5,03	4,43
5	8,83	8,25	8,25	7,87	8,64	7,87	8,25	8,64	7,87	8,25
6	8,35	7,61	7,61	6,73	7,61	7,61	7,61	7,61	7,61	7,61
7	9,93	9,80	9,80	9,80	9,80	9,80	9,80	9,66	9,80	9,80
8	12,02	11,78	11,78	11,78	11,78	11,78	11,78	11,78	11,78	11,78
9	15,50	12,98	12,98	12,98	12,98	12,98	12,98	12,98	12,98	12,98
10	5,29	5,14	5,14	5,14	5,14	5,14	5,14	5,14	5,14	5,14
11	8,94	6,85	6,85	7,47	6,85	7,47	7,47	7,47	7,47	7,47
12	13,18	11,98	10,49	10,49	10,49	10,49	10,49	10,49	10,49	10,49
13	4,67	2,47	1,92	3,30	2,34	3,30	3,02	3,16	3,30	2,75
14	1,68	1,68	1,68	1,68	1,68	1,68	1,68	1,68	1,68	1,68
15	14,68	13,22	12,88	13,91	13,91	13,56	13,91	13,91	13,91	13,91
16	17,20	14,88	14,88	14,88	14,88	14,88	14,88	14,88	14,88	14,88
17	21,78	21,78	21,78	21,78	21,78	21,78	21,78	21,78	21,78	21,78
18	9,97	7,79	7,79	7,79	7,79	7,79	7,79	7,79	7,79	7,79
19	9,99	6,88	6,88	6,88	6,88	6,14	6,88	6,14	6,14	6,88
média	8,95	7,86	**7,73**	7,77	7,83	7,81	7,87	7,86	7,83	7,83

Tabela 5.7 – Valores da diferença percentual para a selecção SAR

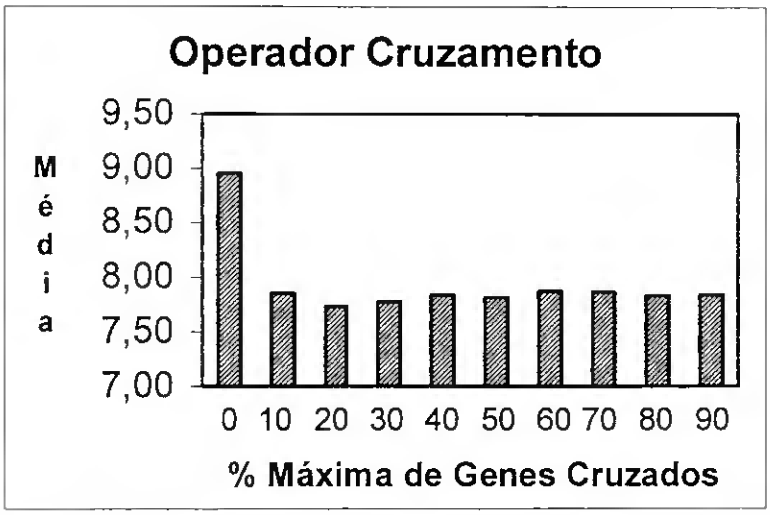


Gráfico 5.1- Valores médios da diferença percentual para a selecção SAR

	Cruzamento									
Pr. Nº	α = 0	α = 10	α = 20	α = 30	α = 40	α = 50	α = 60	α = 70	α = 80	α = 90
1	1,07	1,07	1,07	1,07	0,00	1,07	1,07	1,07	1,07	0,00
2	0,18	0,18	0,18	0,18	0,00	0,00	0,00	0,18	0,00	0,00
3	1,12	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
4	5,63	4,43	4,43	5,03	4,43	4,43	4,43	4,91	4,43	4,43
5	8,83	8,25	8,25	8,06	7,87	7,87	8,06	8,45	7,87	7,87
6	8,35	7,03	6,88	6,73	6,88	7,61	7,61	7,61	7,61	7,61
7	9,93	9,80	9,80	9,80	9,66	9,80	9,80	9,80	9,80	9,80
8	12,02	11,78	11,78	11,78	11,78	11,78	11,78	11,78	11,78	11,78
9	15,50	12,98	12,98	12,98	12,98	12,98	12,98	12,98	12,98	12,98
10	5,29	4,85	4,85	5,14	5,14	5,14	5,14	5,14	5,14	5,14
11	8,94	7,59	6,85	7,47	7,47	7,47	7,47	7,47	7,47	7,47
12	13,18	11,98	11,98	10,49	10,49	10,49	11,98	10,03	10,49	10,49
13	4,67	3,30	2,61	3,16	3,30	3,30	1,79	3,30	3,16	1,65
14	1,68	1,68	1,68	1,68	1,68	1,68	1,68	1,68	1,68	1,68
15	14,68	13,91	13,82	13,91	13,99	13,91	13,91	13,82	13,91	13,91
16	17,20	14,88	14,88	14,88	14,88	14,88	14,88	14,88	14,88	14,88
17	21,78	21,78	21,78	21,78	21,78	21,78	21,78	21,78	21,78	21,78
18	9,97	7,60	7,60	7,79	7,79	7,79	7,79	7,79	7,79	7,79
19	9,99	6,88	6,88	6,88	6,51	6,14	6,88	6,14	6,14	6,14
média	8,95	7,89	7,81	7,83	7,72	7,80	7,84	7,83	7,79	**7,65**

Tabela 5.8 – Valores da diferença percentual para a selecção SUS

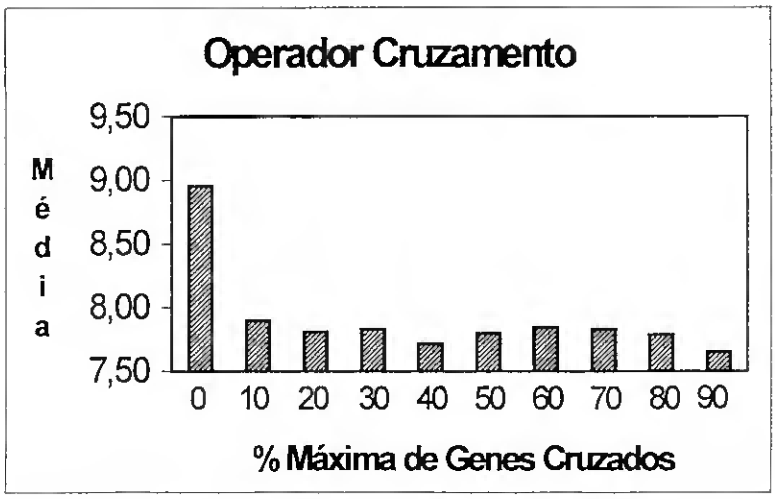


Gráfico 5.2 – Valores médios da diferença percentual para a selecção SUS

A coluna $\alpha = 0$ corresponde à melhor solução obtida na população inicial, sem ter ainda havido actuação dos operadores. Verifica-se que nenhum valor de α conduz sempre, para todos os problemas, à melhor solução. Mas, a melhor média é obtida com $\alpha = 20\%$ e $\alpha = 90\%$ para a selecção SAR e SUS, respectivamente. Note-se, no entanto, que a variação dos valores médios das diferenças percentuais, para os vários valores de α, é reduzida. O melhor valor da população inicial é, em geral, melhorado. Em média, a diferença percentual decresce cerca de 1.3%, com a aplicação do operador cruzamento durante as várias gerações.

Verifica-se que a diferença entre os dois tipos de selecção é pouco significativa. No entanto, optou-se por utilizar apenas a selecção SUS, nos testes seguintes, visto ter obtido resultados ligeiramente melhores. Utilizou-se $\alpha = 90\%$ por ser o valor que em média conduziu a maior melhoramento.

5.4.2.2 MUTAÇÃO

A mutação foi aplicada em cada geração a toda a população. De acordo com o exposto em 5.3.5 foram alterados até β por cento dos genes. Mais uma vez os melhores resultados foram obtidos quando se procedeu à substituição dos pais pelos filhos somente no caso de haver melhoramento, como descrito em 5.3.5 b). Os resultados obtidos para a *dp* e a representação gráfica dos valores médios encontram-se na tabela 5.9 e no gráfico 5.3.

Nota-se, a partir da tabela 5.9 e do gráfico 5.3, que este operador não melhorou o melhor valor médio obtido somente com o operador cruzamento. No entanto, para $\beta = 50\%$ o valor médio é praticamente igual ao melhor valor médio obtido só com o cruzamento ($\alpha = 90\%$).

	Mutação								
Pr. Nº	β = 10	β = 20	β = 30	β = 40	β = 50	β = 60	β = 70	β = 80	β = 90
1	0,00	1,07	0,00	0,00	0,00	1,07	1,07	0,00	0,00
2	0,18	0,18	0,00	0,00	0,00	0,18	0,18	0,00	0,00
3	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00	0,00
4	4,43	4,43	4,43	4,43	4,43	4,43	5,03	4,43	4,43
5	7,87	7,87	7,87	7,87	7,87	7,87	8,45	7,87	7,87
6	7,47	7,61	7,61	7,61	7,61	7,61	7,61	7,61	7,61
7	9,80	9,66	9,80	9,80	9,80	9,66	9,39	9,80	9,80
8	11,78	11,78	11,78	11,78	11,78	11,78	11,78	11,78	11,78
9	12,98	12,98	12,98	12,98	12,98	12,98	12,98	12,98	12,50
10	5,14	5,14	5,14	5,14	5,14	5,14	5,14	5,14	5,14
11	7,47	7,47	7,47	6,85	7,47	7,47	7,47	6,85	7,47
12	10,49	10,49	10,49	10,49	10,49	10,03	10,49	10,49	10,49
13	1,79	3,57	3,30	3,30	1,79	3,30	2,20	3,16	3,30
14	1,68	1,68	1,68	1,68	1,68	1,68	1,68	1,68	1,68
15	13,91	13,91	13,91	13,91	13,91	13,91	13,91	13,91	13,91
16	15,73	14,88	14,88	14,88	14,88	14,88	14,88	14,88	14,88
17	21,78	21,78	21,78	21,78	21,78	21,78	21,78	21,78	21,78
18	7,79	7,79	7,79	7,79	7,79	7,79	7,79	7,79	7,79
19	6,14	6,14	6,14	6,14	6,14	6,14	6,14	6,14	6,14
média	7,71	7,81	7,74	7,71	**7,66**	7,77	7,79	7,70	7,71

Tabela 5.9 – Valores da diferença percentual com selecção SUS, cruzamento e mutação

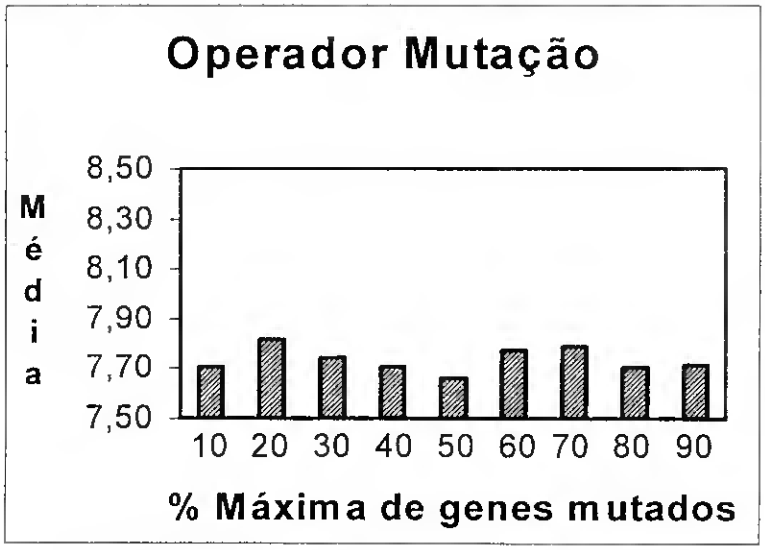


Gráfico 5.3 – Valores médios da diferença percentual

5.4.2.3 DOMINÂNCIA

Neste teste retirou-se o operador mutação e aplicaram-se conjuntamente os operadores cruzamento e dominância. Para testar a influência deste último nas soluções deram-se os valores 1 , 5 e 10 ao parâmetro *Dom* (número de indivíduos seleccionados para aplicação do operador). Os resultados das *dp* obtidos e a representação gráfica dos valores médios encontram-se na tabela 5.10 e no gráfico 5.4.

	Dominância		
	α = 90%		
Pr. Nº	*Dom* = 1	*Dom* = 5	*Dom* = 10
1	1,07	0,00	0,00
2	0,00	0,00	0,00
3	0,00	0,00	0,00
4	4,43	4,43	4,43
5	8,64	8,06	7,87
6	7,61	7,61	7,61
7	9,80	9,80	9,66
8	11,78	11,78	11,78
9	12,98	12,98	12,98
10	5,14	5,14	5,14
11	6,85	6,85	7,47
12	10,49	10,49	10,49
13	4,26	3,98	3,57
14	1,68	1,68	1,68
15	13,91	13,91	13,91
16	14,88	14,88	14,51
17	21,78	21,78	21,78
18	7,79	7,79	7,79
19	6,14	6,14	6,88
média	7,85	**7,75**	7,77

Tabela 5.10 – Valores da diferença percentual

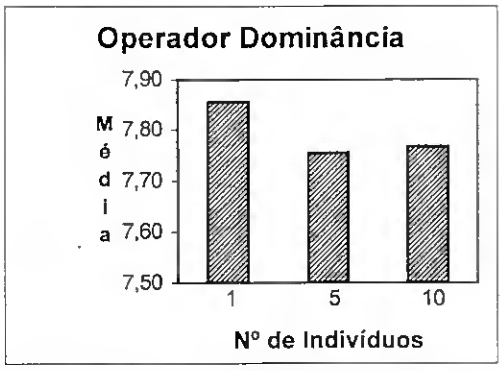


Gráfico 5.4 – Valores médios da diferença percentual

Mais uma vez os melhores valores médios da *dp* obtidos apenas com a aplicação do cruzamento não foram melhorados com a introdução do operador dominância.

5.4.2.4 MUTAÇÃO E DOMINÂNCIA

Testou-se, ainda, a actuação dos operadores mutação e dominância conjuntamente. Neste caso deram-se os valores $\alpha = 90\%$, $\beta = 50\%$ e *Dom* = 5. Obteve-se um valor médio da *dp* igual a 7.78, não havendo, portanto, melhoria dos resultados obtido com a actuação isolada do cruzamento.

Após estes testes, foi possível concluir que o operador cruzamento contribui na pesquisa de melhores soluções conduzindo a população inicial a outras mais aptas. A inclusão dos operadores mutação e dominância, agindo separadamente ou conjuntamente, não melhora os valores médios obtidos apenas com o operador cruzamento, embora possa melhorar ligeiramente uma ou outra solução.

Em todos os testes não existe uma predominância absoluta de um valor para os diferentes operadores, mas as oscilações médias verificadas também não são significativas.

No gráfico 5.5 representam-se os melhores valores médios das *dp* obtidas com as diferentes variações do AG adaptado.

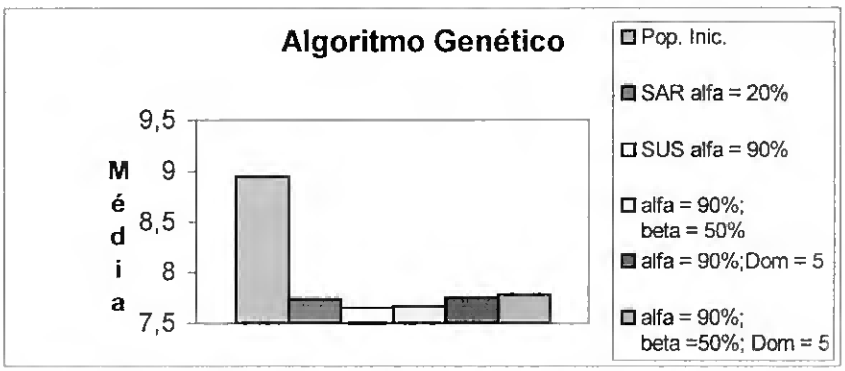


Gráfico 5.5 - Diferenças percentuais para variações dos parâmetros do AG adaptado

O tempo de execução deste algoritmo ($\alpha = 90\%$ e $\beta = 50\%$) foi sempre bastante bom, variou entre 1,6 segundos para o menor problema e 147 segundos para o problema 19 (o maior). Em média o tempo de execução foi igual a 31 segundos.

5.4.3 ALGORITMO GENÉTICO HÍBRIDO

De acordo com a experiência anterior com o AG adaptado, neste caso não se julgou necessário repetir os mesmos testes (valores de α e formas de substituição). Optou-se, então, por fixar $\alpha = 90\%$ e substituir os pais de acordo com 5.3.4 a). Manteve-se a dimensão da população em 40 e o número máximo de gerações em 35. O valor médio obtido para a *dp* foi igual a 5,03, bastante melhor do que o obtido com o AG adaptado.

Em contrapartida, os tempos de execução foram sempre superiores aos do AG adaptado, chegando, num problema de maior dimensão, a demorar 7 vezes o tempo que o AG adaptado tinha demorado. Para o AG híbrido os tempos de execução variaram entre 2 segundos e 639 segundos (para o maior problema). Em média o tempo de execução foi igual a 103,47 segundos.

É esperado que aumentando a dimensão da população, principalmente no caso dos problemas de maior dimensão, seja possível obter melhores resultados com qualquer uma destas versões do AG, com a contrapartida de o tempo de execução aumentar consideravelmente, principalmente o tempo da geração da população inicial, que por ser feita heuristicamente é mais demorada. No caso do AG híbrido o tempo gasto na aplicação dos procedimentos pós-optimais é, também, muito grande, como se pode ver comparando os tempos do AG adaptado e do híbrido.

Para verificar se uma população maior conduzia realmente a melhores resultados, executou-se de novo a versão híbrida, para os problemas com um número de clientes superior a 50, estabelecendo nestes a dimensão da população em 80 e o número máximo de gerações em 60. Obteve-se uma *dp* média igual a 3,17 e os tempos de execução variaram entre 44 e 1525,77 segundos.

A tabela 5.11 mostra os tempos de execução do AG híbrido, os tempos gastos para gerar a população inicial, bem como a percentagem do tempo total que essa parte do algoritmo representa.

Probl. Nº	*Pop* =40 e 80			*Pop* =40		
	(a)	(b)	(c)	(a)	(b)	(c)
1	2,03	0,11	5,42	2,03	0,11	5,42
2	3,96	0,39	9,85	3,96	0,39	9,85
3	5,39	0,77	14,29	5,39	0,77	14,29
4	5,93	0,55	9,27	5,93	0,55	9,27
5	7,80	1,76	22,56	7,80	1,76	22,56
6	213,46	13,68	6,41	33,79	6,76	20,01
7	149,29	10,88	7,29	23,24	5,55	23,88
8	44,40	15,28	34,41	13,35	6,87	51,46
9	86,81	71,76	82,66	20,06	16,7	83,25
10	177,58	52,42	29,52	90,99	26,87	29,53
11	473,79	32,14	6,78	48,9	16,26	33,25
12	440,60	18,52	4,20	69,45	9,29	13,38
Média (1)	**134,25**	**18,19**	**19,39**	**27,07**	**7,66**	**26,35**
13	19,67	1,59	8,08	19,67	1,59	8,08
14	225,22	16,98	7,54	74,78	7,58	10,14
15	1525,77	103,02	6,75	481,87	53,08	11,02
Média (2)	**590,22**	**40,53**	**7,46**	**192,11**	**20,75**	**9,75**
16	390,60	21,70	5,56	185,82	10,99	5,91
17	299,95	55,88	18,63	98,79	28,24	28,59
18	455,22	100,50	22,08	140,99	50,22	35,62
19	1459,67	200,60	13,74	639,07	103,46	16,19
Média (3)	**651,36**	**94,67**	**15,00**	**266,17**	**48,23**	**21,58**
Média (4)	**315,11**	**37,82**	**16,58**	**103,47**	**18,27**	**22,72**

(a) = Tempo total de execução em segundos

(b) = Tempo para gerar a população inicial em segundos

(c) = Percentagem do tempo total de execução gasto com a geração da população inicial

Tabela 5.11 - Tempos de execução da geração da população e total do AG híbrido

Como era esperado, verifica-se pela média (4) que, quer para uma população quer para a outra, a percentagem do tempo total do algoritmo gasta com a geração da população é elevada, variando de uma forma acentuada de problema para problema, chegando a atingir cerca de 83% do tempo total de execução. No entanto, representa menos quando a população é maior, o que mostra que o operador que executa a pós-optimização é, realmente, a parte mais pesada do algoritmo. Enquanto, em média, com o aumento da população o tempo gasto para a gerar quase duplicou, o tempo total de execução triplicou.

Os valores das médias (1), (2) e (3) revelam que, para os problemas de Fisher (1994), a percentagem do tempo total consumida com a geração da população é consideravelmente menor do que para os outros problemas.

Os gráficos 5.6 e 5.7 apresentam os valores médios obtidos, para as *dp* e para os tempos de execução, com o AG adaptado ($\alpha = 90\%$ e $\beta = 50\%$), com o AG híbrido com população igual a 40, e com o AG híbrido com população igual a 40 e a 80, respectivamente, para os problemas de dimensão menor ou igual a 50 e dimensão superior a 50.

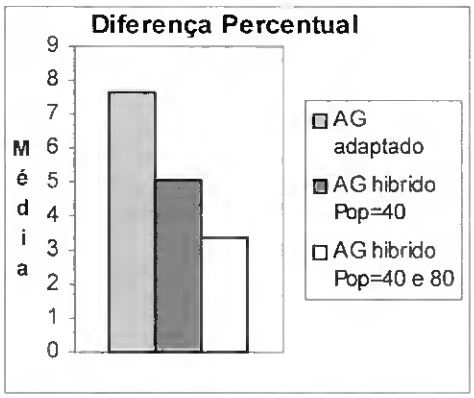


Gráfico 5.6 - Valores médios da diferença percentual

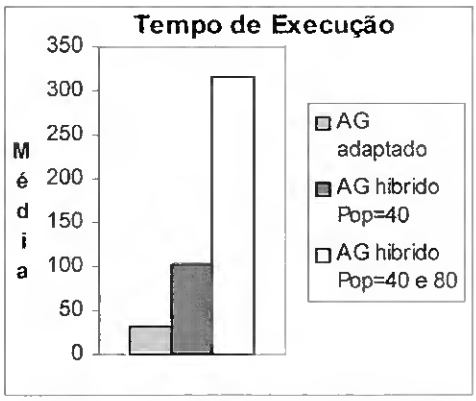


Gráfico 5.7 - Valores médios dos tempos de execução (em segundos)

Nestes gráficos ressalta que, em média, a introdução de procedimentos de pós-optimização melhora consideravelmente (cerca de 34%) os resultados mas quase triplica os tempos de execução. O aumento da população ao implicar um aumento da aplicação de procedimentos pós-optimais tem o mesmo efeito: melhora (33%) os resultados e triplica o tempo de execução.

5.4.4 TESTES COMPARATIVOS

Além de se terem comparado as soluções obtidas, através dos AGs, com as melhores conhecidas, utilizaram-se também, para comparar principalmente os tempos de execução computacional, outros dois algoritmos. Por ser uma heurística de duas fases computacionalmente muito rápida e com referências de bons resultados [Schütz (1990)] adaptou-se a implementação, em Pascal, descrita em Schütz (1990) para Turbo C. Na primeira fase os clientes são afectos aos veículos de uma forma semelhante à do algoritmo de duas fases descrito em Christofides et al (1979). Na segunda fase utiliza-se o Método da Inserção Mais Afastada para resolver heuristicamente os PCV envolvidos. No fim à solução obtida são aplicados, também, os mesmos procedimentos pós-optimais ((i), (ii) e (iii)) da heurística construtiva de rota descrita em 5.3.2.1. Com esta implementação obtêm-se duas soluções. Numa os clientes são afectos aos veículos sequencialmente e na outra paralelamente. Para tentar aferir melhor a eficiência do AG, executou-se, também, um algoritmo com a heurística construtiva de rota, que serviu para gerar a população inicial, mas recorrendo só aos três critérios de escolha do cliente inicial já descritos em 5.3.2.1 ((5.14), (5.15) e (5.16)). Assim, com este algoritmo, obtiveram-se três soluções. Comparou-se a melhor das soluções obtidas com cada um destes métodos com a obtida pelo AG híbrido. Repare-se que, qualquer um destes métodos pode, devido ao número de veículos estar previamente definido, não obter solução admissível, por ficarem clientes por servir.

Estas duas heurísticas também foram implementados em Turbo C e executadas no mesmo PC com um processador Pentium MMX 200 MHz e com 16 MB de RAM.

Na tabela 5.12 apresenta-se um quadro comparativo entre o AG híbrido, as heurísticas já mencionadas e a melhor solução conhecida. Os resultados referem-se ao AG híbrido, com uma população de 80 e 40 indivíduos e um máximo de 60 e 35 gerações, respectivamente para os problemas de dimensão maior do que 50 e menor ou igual a 50.

Probl. Nº	Melhor solução	Heurística de 2 fases			Heurística Construtiva			AG híbrido		
		solução	*dp*	tempo	solução	*dp*	tempo	solução	*dp*	tempo
1	375	390	4,00	0,000	381	1,60	0,000	375	0,00	2,088
2	569	669	17,57	0,000	570	0,18	0,000	569	0,00	3,956
3	534	*		0,000	559	4,68	0,055	534	0,00	5,385
4	835	921	10,30	0,055	882	5,63	0,055	837	0,24	5,989
5	521	601	15,36	0,055	569	9,21	0,110	537	3,07	7,857
6	683	752	10,10	0,165	754	10,40	0,440	725	6,15	213,571
7	735	820	11,56	0,110	840	14,29	0,440	784	6,67	149,341
8	832	987	18,63	0,110	1009	21,27	0,220	878	5,53	44,396
9	1032	1282	24,22	0,055	*		0,164	1077	4,36	86,813
10	681	777	14,10	0,440	717	5,29	2,033	705	3,52	177,418
11	817	984	20,44	0,275	902	10,40	1,209	878	7,47	473,462
12	1077	1261	17,08	0,165	1232	14,39	0,769	1153	7,06	440,550
Média (1)			**14,85**	**0,119**		**8,85**	**0,458**		**3,67**	**134,236**
13	728	879	20,74	0,055	778	6,87	0,110	727	-0,14	19,725
14	238	260	9,24	0,110	260	9,24	0,495	242	1,68	225,385
15	1165	1483	27,30	0,495	1385	18,88	3,626	1175	0,86	1477,967
Média (2)			**19,09**	**0,220**		**11,67**	**1,410**		**0,80**	**574,359**
16	820	1037	26,46	0,220	1028	25,37	0,769	831	1,34	390,550
17	1065	1334	25,26	0,330	1325	24,41	1,813	1049	-1,50	300,165
18	1053	1278	21,37	0,714	1174	11,49	4,066	1135	7,79	455,824
19	1351	1645	21,76	1,923	1538	13,84	4,835	1434	6,14	1502,582
Média (3)			**23,71**	**0,797**		**18,78**	**2,871**		**3,44**	**662,28**
Média (4)			**17,53**	**0,28**		**11,52**	**1,12**		**3,17**	**314,90**

* - Não obteve solução admissível

tempo - tempo de execução em segundos

Tabela 5.12 - Comparação entre a melhor solução conhecida, e as soluções obtidas com o algoritmo de duas fases, o algoritmo construtivo de rota e o AG híbrido

Verifica-se que a solução obtida com o AG híbrido atinge em 3 problemas o óptimo, em 2 (problemas 13 e 17) obtém um valor melhor do que a melhor solução conhecida, em 4 está a menos de 2% da melhor solução conhecida. No total, em 12 dos

19 problemas a solução obtida com o AG está próxima da melhor solução conhecida (menos de 5%), o que já não ocorre com as outras duas heurísticas. Enquanto as outras duas heurísticas obtêm as melhores soluções para os dados de Eilon et al (1971), o AG obtém, também, muito boas soluções para os problemas de menor dimensão deste conjunto, mas já não obtém bons resultados para os de dimensão 75 e 100.

Nos problemas para os quais este algoritmo obteve as melhores soluções, a procura é, em geral, inferior a 95% da capacidade total dos veículos, no entanto, este facto não explica o bom comportamento do algoritmo, visto noutros problemas igualmente pouco "apertados" não se obterem bons resultados, e num muito "apertado" (problema 17) obter uma solução muito boa. Parece pois, não ser muito significativa a relação entre o comportamento do algoritmo e a percentagem de ocupação dos veículos.

Mas, em problemas muito "apertados" este AG não deve ser muito eficiente, visto não serem admissíveis a maioria das trocas de valores entre os indivíduos pais. Assim, o AG pode convergir rapidamente sem grandes alterações à população inicial. Isto ocorreu nos problemas 8 e 9.

É de salientar que os melhores resultados obtidos com o AG correspondem aos problemas de Fisher (1994), extraídos de aplicações reais, aos problemas 16 e 17, que são os únicos que têm os clientes em grupos [Christofides et al (1979)], que é, em geral, o tipo de estrutura dos dados das aplicações reais, e aos problemas de pequena dimensão.

Em relação aos tempos de execução computacional nota-se que os despendidos com o AG são bastante superiores aos dos outros dois algoritmos (ambos muito rápidos, principalmente o de duas fases). No entanto, o tempo de execução do AG foi em geral inferior a 5 minutos, mesmo trabalhando com uma população de 80 indivíduos, o que é ainda razoável. Em contrapartida os resultados obtidos com o AG foram sempre melhores (na média a dp foi de 3% contra 12% e 18% dos outros algoritmos).

Constata-se, assim, que este AG quando comparado com algumas heurísticas (construtiva e de duas fases) é mais eficiente. Em relação às melhores soluções conhecidas e para problemas de maior dimensão já não é tão eficiente.

5.4.5 OBSERVAÇÕES

É considerado [Rego (1994)] que os resultados obtidos para o PORV com algoritmos que utilizam distâncias inteiras "subestimam" o valor real da solução. De facto, não é correcto comparar os resultados obtidos com distâncias reais com os obtidos com distâncias inteiras. Como já se referiu, neste trabalho utilizaram-se distâncias inteiras por serem inteiros os valores das melhores soluções conhecidas (disponíveis nos ficheiros da Internet), por questões de tempo de execução computacional e porque na aplicação real que se pretendia estudar usavam-se distâncias inteiras (visto trabalhar-se em metros, reduzindo-os na solução final a quilómetros, como descrito no capítulo 6). Para o estudo comparativo entre as versões dos AGs, também não era importante utilizar distâncias reais. Por outro lado, só se obtém uma comparação totalmente fiável determinando a distância percorrida com as rotas obtidas através da mesma matriz de distâncias. Foi isto que se fez no caso dos problemas 13 a 15, que estão também descritos em Fisher (1994) com as respectivas soluções (rotas a efectuar), aplicando-se-lhes a matriz de distâncias obtida neste trabalho para melhor poder comparar os resultados. Então, na tabela 5.13 apresentam-se os valores óptimos (para os problemas 13 e 14) e melhor solução conhecida (para o problema 15) obtidas com as matrizes de distâncias aqui utilizadas.

		AG híbrido	
Pr. Nº	sol*	sol.	*dp*
13	*717*	727	1,39
14	*238*	242	1,68
15	1114	1175	5,48

Tabela 5.13 - Problemas de Fisher (1994)

Verifica-se que as soluções obtidas pelo AG são piores do que as obtidas pelo algoritmo de Fisher (1994). No entanto para os problemas 13 e 14 a solução encontrada por este AG está próxima da óptima. Para o problema 15, de maior dimensão (134 clientes), o resultado do AG não é bom.

Para os problemas: 5, 8, 11, 16, 17, 18 e 19, são conhecidos da literatura os resultados com distâncias reais, mas não referem qual o número de veículos utilizado nessas soluções, embora indiquem que os problemas foram executados sem limite no número de veículos [Gendreau et al (1997), Rochat et Taillard (1995), Rego (1994)]. Ao contrário, nos testes realizados neste trabalho impôs-se o número de veículos a utilizar. Apresenta-se na tabela 5.14 os resultados obtidos com a versão híbrida do AG, para esses problemas, sendo o número de veículos o indicado na tabela 5.6 e utilizando uma matriz de distâncias reais.

	Matriz real				Matriz inteira			
		AG híbrido				AG híbrido		
Pr. Nº	sol*	sol.	*dp*	tempo	sol*	sol.	*dp*	tempo
5	*524,61*	536,62	2,29	106,15	*521*	537	3,07	7,857
8	835,26	897,92	7,50	131,70	832	878	5,53	44,396
11	*826,14*	873,46	5,73	1714,23	817	878	7,47	473,462
16	819,56	829,73	1,24	1674,73	820	831	1,34	390,550
17	1042,11	1199,01	15,06	817,58	1065	1049	-1,50	300,165
18	1028,42	1155,69	12,38	587,20	1053	1135	7,79	455,824
19	1291,45	1479,03	14,52	3118,63	1351	1434	6,14	1502,582
média			8,39	1164,32			4,26	453,55

Tabela 5.14 - Comparação das soluções obtidas com uma matriz real e uma matriz inteira

As melhores soluções reais, sol* (apresentadas nesta tabela) foram todas obtidas com algoritmos de Pesquisa Tabu. O algoritmo de Taillard (1993) obteve todas estas soluções excepto a do problema 19 obtida por Rochat et Taillard (1995). Em Gendreau et al (1997) estão referenciados vários outros algoritmos que também obtiveram algumas destas soluções. Posteriormente, também, Rego (1998) e Kelly e Xu (1999) as obtiveram. Em relação ao problema 17 apresenta-se a solução obtida por Taillard (1993) e referenciada como a melhor em Rego (1998), (e também em Gendreau et al (1997), Kelly e Xu (1999)), embora Rego (1994) tenha obtido uma solução melhor igual a 1039.

Em relação aos tempos de execução o AG híbrido é sempre mais demorado quando as matrizes de distâncias são reais.

Comparando os valores das *dp* obtidos para as duas matrizes (real e inteira) verifica-se que para o problema 5, o único em que é conhecida a solução óptima (real e inteira) o AG obteve melhor resultado com a matriz real. Para o problema 8 também obteve melhor resultado com a matriz real e para o problema 16 o resultado é semelhante nos dois casos. Para os restantes problemas, os melhores valores da *dp* foram obtidos com a matriz inteira. Assim, é de supor que o comportamento deste AG piore com matrizes reais. A qualidade das soluções obtidas com este AG foi, em geral, inferior à obtida com os algoritmos de PT, o que é notório nos problemas 17, 18 e 19.

Não é possível comparar os tempos de execução do AG híbrido com o das outras meta-heurísticas de PT, porque os tempos referenciados na literatura foram obtidos em computadores diferentes.

5.4.6 PARALELIZAÇÃO

Paralelizou-se o AG híbrido de acordo com o modelo e topologias adoptados (descritos em 5.3.6). Analogamente ao exposto no capítulo 4, visou-se com esta paralelização dois objectivos:

1 - Diminuir o tempo de execução;

2 - Aumentar a diversidade da população.

Da mesma forma que anteriormente, para o objectivo 1, cada processador gerou aleatoriamente Pop/p indivíduos e executou $Maxger/p$ gerações, com p = 2, 4 ou 8. Para o objectivo 2, cada processador executou o mesmo número máximo de gerações e trabalhou com uma população da mesma dimensão do algoritmo sequencial. Nestes testes, para os problemas com dimensão menor ou igual a 50, trabalhou-se com uma população de 40 indivíduos e um número máximo de 35 gerações a executar. Para os outros a população tem 80 indivíduos e no máximo executam-se 60 gerações.

Para o objectivo 2, estabeleceu-se em cinco o número de indivíduos a trocar entre processadores. Para o objectivo 1, estabeleceu-se em cinco e em três indivíduos consoante fossem utilizados 2 ou 4 e 8 processadores. Estes indivíduos são escolhidos aleatoriamente entre as populações de cada processador.

5.4.6.1 DIMINUIR O TEMPO DE EXECUÇÃO

Fixou-se em 10 indivíduos a dimensão mínima da população e em 10 gerações o menor valor de *Maxger*, por processador. Desta forma, os problemas com um número de clientes menor ou igual a 50 só foram executados em 1, 2 e 4 processadores. Na tabela 5.15 apresentam-se os resultados obtidos em relação à *dp*.

Constata-se, a partir desta tabela, que no geral e em média (média (4)) os valores da *dp* pioraram com este tipo de paralelização. Somente o conjunto de problemas de Fisher (1994) melhorou com 2 e 4 processadores (média (2)). Pontualmente alguns dos outros problemas também sofreram uma ligeira melhoria para algum número de processadores. Uma vez que o AG híbrido, executado sequencialmente, já não obtinha resultados bons para alguns problemas, esta divisão da população pelos processadores ainda piora esses resultados. De facto produz uma menor diversidade e consequentemente provoca a convergência, em cada processador, para uma solução de fraca qualidade. Isto explica que tenha havido, quase sempre, melhoria com a introdução de mais trocas de indivíduos entre os processadores. Estas trocas podem trazer alguma diversidade.

Diferença percentual							
Probl. Nº	Nº clientes	Processadores					
		1	2	4	8	4c	8c
1	21	0,00	0,00	0,00		0,00	
2	22	0,00	0,00	0,00		0,00	
3	29	0,00	1,12	1,12		1,12	
4	32	0,24	0,72	0,72		2,16	
5	50	3,07	1,73	5,37		3,65	
6	75	6,15	6,30	7,32	10,25	6,30	10,25
7	75	6,67	8,03	8,03	9,25	8,30	8,16
8	75	5,53	8,53	13,22	7,81	13,34	7,09
9	75	4,36	17,15	7,66	9,30	9,30	9,30
10	100	3,52	3,38	3,96	3,23	3,67	3,38
11	100	7,47	8,94	8,94	7,47	8,94	7,71
12	100	7,06	9,47	10,21	10,68	9,47	9,75
média(1)		**3,67**	**5,45**	**5,55**	**8,28**	**5,52**	**7,95**
13	44	-0,14	-0,14	-0,14		-0,14	
14	71	1,68	1,68	1,68	4,62	1,68	2,10
15	134	0,86	-1,80	0,60	3,52	-1,72	3,35
média(2)		**0,80**	**-0,09**	**0,71**	**4,07**	**-0,06**	**2,73**
16	100	1,34	1,46	1,34	3,17	4,63	1,34
17	120	-1,50	10,14	8,92	9,11	8,45	6,67
18	150	7,79	11,40	9,40	8,07	11,30	7,60
19	199	6,14	10,07	8,44	10,36	10,07	11,03
média(3)		**3,44**	**8,27**	**7,03**	**7,68**	**8,61**	**6,66**
média(4)		**3,17**	**5,17**	**5,09**	**7,45**	**5,29**	**6,75**

Tabela 5.15 - Valores da *dp* obtidos com a paralelização visando o objectivo 1

Os gráficos 5.8, 5.9, 5.10 e 5.11 representam os valores da *dp* obtidos segundo a dimensão dos problemas. Os valores médios da *dp*, segundo o tipo de dados, obtidos para os problemas que foram executados em 2, 4 e 8 processadores, estão representados no gráfico 5.12.

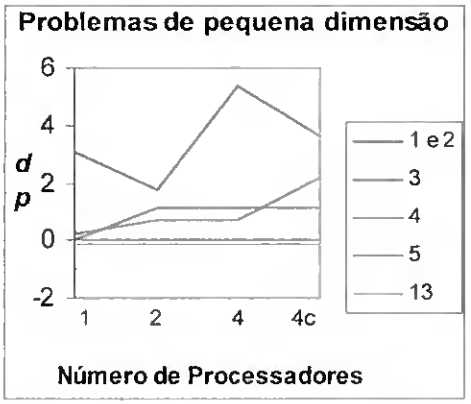


Gráfico 5.8 - Problemas de dimensão menor ou igual a 50

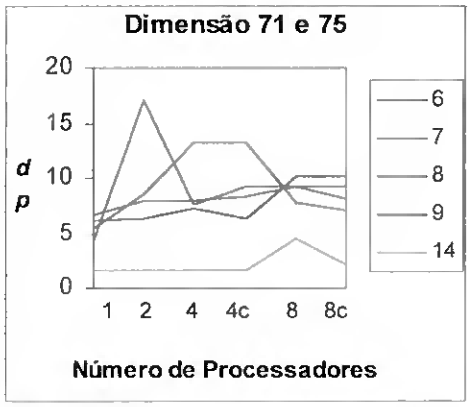


Gráfico 5.9 - Problemas de dimensão superior a 50 e inferior a 100

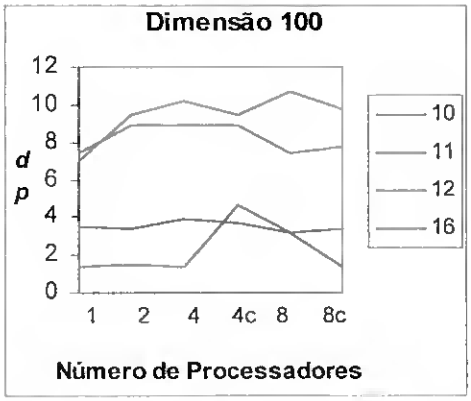


Gráfico 5.10 - Problemas de dimensão 100

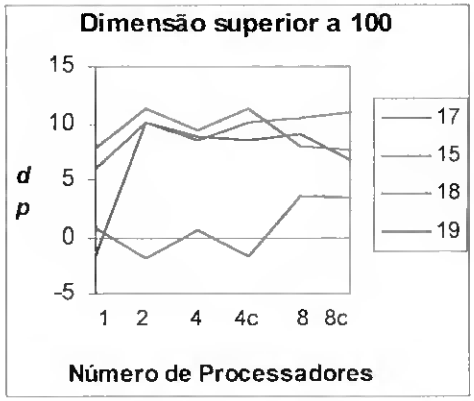


Gráfico 5.11 - Problemas de dimensão superior a 100

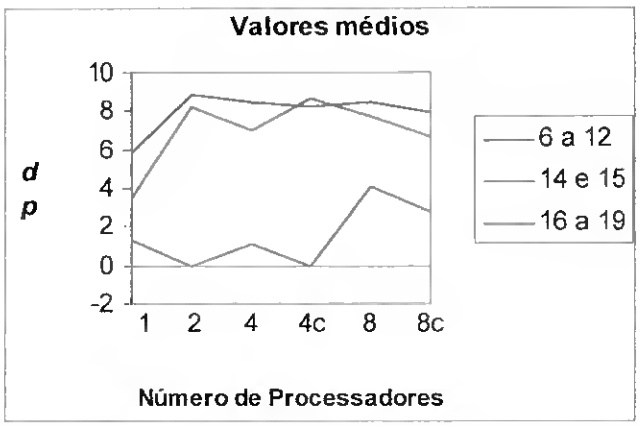


Gráfico 5.12 - Valores médios da *dp* para os três conjuntos de problemas

Em relação aos problemas de pequena dimensão, que só foram paralelizados em 2 e 4 processadores, não houve melhoramentos com o aumento dos processadores ou das comunicações, excepto para o problema 5 cujo comportamento oscilou, melhorando com 2 processadores piorando muito com 4 e menos com 4c.

Esta paralelização também não trouxe melhoramentos ao valor da *dp*, em relação ao algoritmo sequencial, para os problemas de dimensão 71 e 75. A *dp* dos problemas de dimensão 75 oscilou muito com o número de processadores. Para o problema de dimensão 71 apenas piorou com 8 processadores, principalmente com uma única troca de indivíduos entre processadores.

Para os problemas de dimensão 100 também não se verificou um comportamento homogéneo, notando-se que, para os três pertencentes ao mesmo conjunto de dados, os resultados da *dp* pioram com o aumento do número de veículos.

A *dp*, para os problemas de dimensão superior a 100, piora sempre os resultados obtidos sequencialmente, excepto para o problema 15 que melhora para 2 e 8 processadores. Para o problema 17 piora significativamente. Em todos, a *dp* oscila o seu valor consoante o número de processadores.

Em relação aos valores médios obtidos, para cada tipo de dados, verifica-se que a paralelização só melhorou os resultados da *dp* para os dados de Fisher (1994).

Na tabela 5.16 têm-se os valores da aceleração. Era esperado que as acelerações fossem muito superiores à linear, como aconteceu no capítulo 4. No entanto, as médias para 2, 4 e 8 processadores situam-se em torno de 2, 5 e 12, respectivamente, e ainda pioram com o aumento da troca de indivíduos entre processadores, passando a aceleração a ser 4 e 8 (para 4c e 8c, respectivamente). Tem-se então que, para este problema utilizando o AG híbrido que inclui geração heurística da população inicial e procedimentos pós-optimais relativamente morosos, mesmo reduzindo o trabalho em cada processador, as acelerações médias são lineares ou pouco superiores à linear.

Probl. Nº	Nº clientes	Processadores				
		2	4	8	4c	8c
1	21	3,10	11,03		6,13	
2	22	2,12	6,11		4,72	
3	29	1,17	4,11		2,95	
4	32	1,88	6,94		4,56	
5	50	1,46	4,39		3,35	
6	75	2,50	5,34	8,98	3,79	5,55
7	75	4,52	5,91	12,16	3,85	6,96
8	75	2,52	5,86	11,34	5,59	7,54
9	75	1,04	2,15	8,08	1,96	5,52
10	100	2,01	5,25	12,64	4,89	7,49
11	100	0,96	3,37	10,33	2,27	6,96
12	100	1,98	6,35	18,77	4,41	11,86
média(1)		2,11	5,57	11,76	4,04	7,41
13	44	1,96	6,00		5,49	
14	71	2,22	7,10	19,16	5,15	12,01
15	134	1,78	5,51	15,64	3,87	8,49
média(2)		1,99	6,20	17,40	4,84	10,25
16	100	2,48	5,64	15,74	4,74	9,95
17	120	1,96	3,61	9,57	2,55	6,49
18	150	1,16	2,37	5,05	2,61	3,01
19	199	2,05	4,93	7,52	1,99	5,04
média(3)		1,91	4,14	9,47	2,97	6,12
média(4)		2,05	5,37	11,92	3,94	7,45

Tabela 5.16 - Valores da aceleração para a paralelização visando o objectivo 1

Nos gráficos 5.13 e 5.14 representam-se, respectivamente, os valores médios da aceleração para os problemas paralelizados em 2 e 4 processadores e para os paralelizados em 2, 4 e 8 processadores.

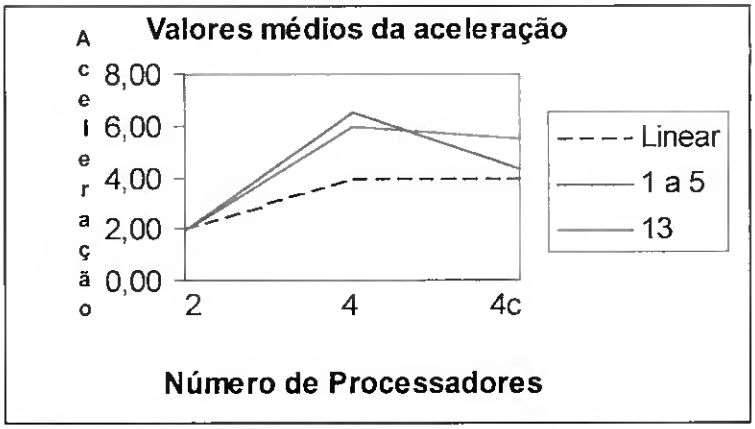


Gráfico 5.13 - Valores médios da aceleração para os problemas paralelizados em 2 e 4 processadores

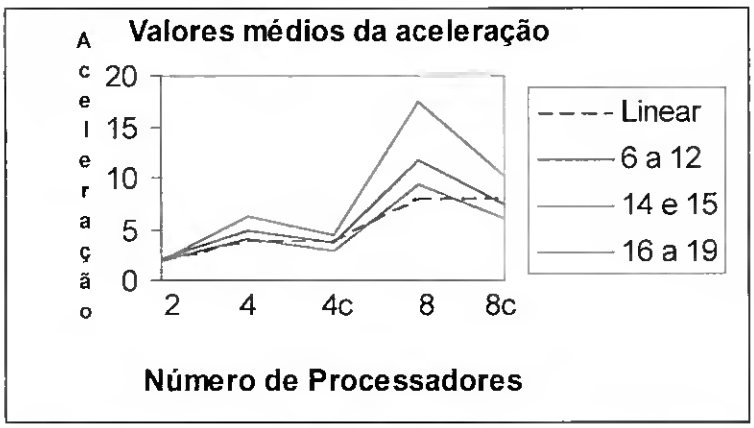


Gráfico 5.14 - valores médios da aceleração para os problemas paralelizados em 2, 4 e 8 processadores

O gráfico 5.13 mostra que para os problemas de menor dimensão a aceleração obtida com 4 processadores é superior à linear piorando com a introdução de mais comunicações entre os processadores.

A aceleração obtida com 2 processadores, nestes dois gráficos é igual à linear.

No gráfico 5.14 verifica-se que, somente para os dados de Fisher (1994), a aceleração é sempre superior à linear, para 4 e 8 processadores. Para os dados de Eilon et al (1971) é ligeiramente superior à linear, excepto quando se introduzem mais trocas de indivíduos entre processadores (4c e 8c). Para os dados de Christofides et al (1979) só é ligeiramente superior à linear para 8 processadores, com uma só troca de indivíduos entre eles.

Relativamente a esta paralelização e aos resultados obtidos não houve, em termos gerais, vantagem em paralelizar o AG híbrido. Por um lado a qualidade da solução piorou, nalguns casos consideravelmente, por outro a aceleração obtida não foi muito interessante, principalmente tendo em conta a deterioração da solução. No que respeita aos algoritmos paralelos uma aceleração linear é considerada óptima, pois normalmente não é atingida, mas sabendo que, neste caso, cada processador realiza muito menos trabalho que o algoritmo sequencial, num só processador, não se pode considerar que estas acelerações sejam boas.

5.4.6.2 AUMENTAR A DIVERSIDADE DA POPULAÇÃO

Para esta paralelização os processadores executam cada um o mesmo trabalho do que o executado por um só processador, sendo, por isso, de esperar que haja desacelerações com a introdução de mais processadores (como já aconteceu no capítulo 4). Então, e perante os resultados obtidos com a paralelização anterior, é de supor que este tipo de paralelização só poderá ser vantajosa para os problemas que não atinjam bons resultados com o algoritmo sequencial e cuja dimensão não seja excessivamente grande. Assim, optou-se por testar esta paralelização apenas nos problemas que tinham obtido, com o algoritmo sequencial, valores da *dp* superiores a 5% e que tinham dimensão não superior a 100, nomeadamente, nos problemas 6, 7, 8, 11 e 12.

Na tabela 5.17 apresentam-se os resultados obtidos para a *dp*.

Probl. Nº	Nº clientes	Processadores					
		1	2	4	8	4c	8c
6	75	6,15	7,47	6,30	6,44	6,59	5,71
7	75	6,67	4,22	4,35	4,08	4,08	2,86
8	75	5,53	6,97	4,69	5,65	5,65	5,29
11	100	7,47	8,94	6,98	6,98	6,98	5,39
12	100	7,06	7,99	7,61	6,87	6,87	5,85
média		6,58	7,12	5,99	6,00	6,03	5,02

Tabela 5.17 - Valores da diferença percentual visando o objectivo 2

Verifica-se que em média o valor da *dp* piora com 2 processadores mas melhora com 4 e 8. A introdução de mais trocas de indivíduos entre processadores piorou o valor médio da *dp*, no caso de 4 processadores, mas melhorou no caso de 8 processadores. Foi, aliás neste último caso (8 processadores e trocas de indivíduos entre processadores por três vezes) que se obteve o melhor valor médio da *dp*.

No gráfico 5.15 representam-se os valores da *dp* para estes problemas.

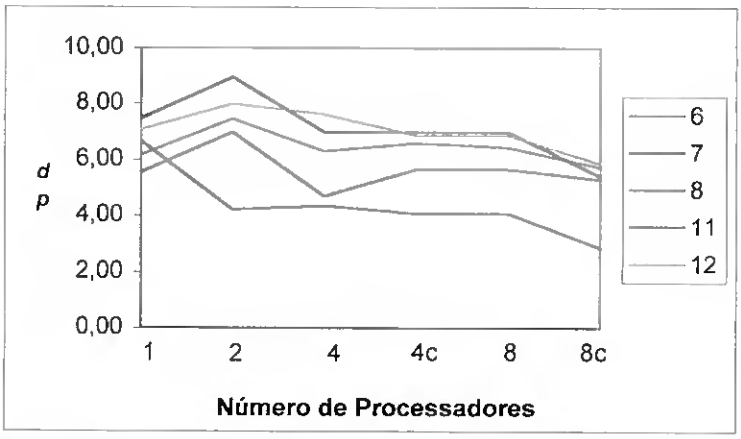


Gráfico 5.15 - Valores da diferença percentual visando o objectivo 2

Nota-se que, em geral, os valores da *dp*, para esta paralelização, têm um comportamento pouco homogéneo. A *dp* melhorou com 8c processadores para todos os problemas e piorou com 2 processadores para todos os problemas, excepto para o problema 7. De facto, para o problema 7, a paralelização teve um comportamento mais homogéneo, pois não piorou o valor da *dp*, com a introdução de mais processadores ou comunicações, e melhorou consideravelmente a *dp*, obtida sequencialmente, com 2 e 8c processadores.

Globalmente pode-se dizer que este tipo de paralelização só teve efeitos mais significativos quando se utilizaram os 8 processadores e mais trocas de indivíduos entre processadores.

Para o estudo da aceleração apresentam-se a tabela 5.18 e o gráfico 5.16.

Probl. Nº	Nº clientes	Processadores				
		2	4	8	4c	8c
6	75	0,38	0,25	0,25	0,17	0,13
7	75	0,38	0,38	0,36	0,28	0,19
8	75	0,79	0,67	0,52	0,71	0,33
11	100	0,64	0,49	0,30	0,27	0,14
12	100	0,55	0,51	0,43	0,37	0,27
média		0,55	0,46	0,37	0,36	0,21

Tabela 5.18 - Valores da aceleração

À medida que se utilizam mais processadores o tempo de comunicações aumenta e como cada processador efectua todo o trabalho que era feito num só sequencialmente, o tempo total de execução piora o que acarreta deteriorações da aceleração.

Verifica-se que, para estes problemas, a execução paralela gastou, em média, de 2 a 6 vezes o tempo gasto pela sequencial, tendo nalguns casos gasto nove vezes mais. Estes resultados podem ser constatados em termos de aceleração na tabela 5.18. Da análise desta tabela nota-se que o tempo de execução computacional utilizado 8 ou 4c processadores é muito semelhante, mas para alguns problemas é bastante maior com 4c processadores. Como os resultados, obtidos para a *dp*, são semelhantes, nestes dois casos, é de supor que será melhor utilizar mais processadores do que fazer mais trocas de indivíduos entre eles.

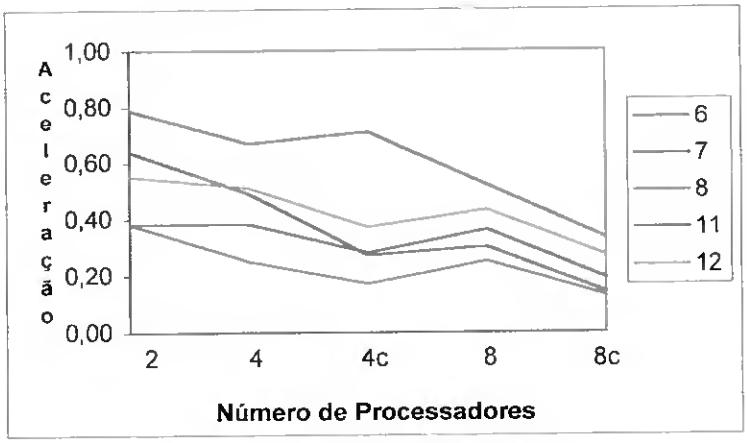


Gráfico 5.16 - Valores da aceleração

Globalmente, tendo em consideração os resultados obtidos, tanto para a *dp* como para a aceleração, este tipo de paralelização não é vantajosa para o AG híbrido. Realmente, embora com mais processadores se possa conseguir uma maior diversidade da população e com isso conseguir pesquisar outras regiões do espaço das soluções, o custo em termos de tempo de execução computacional é muito elevado, não havendo, em contrapartida, uma garantia de que efectivamente a qualidade da solução vá melhorar, com efeito umas vezes melhorou outras não.

5.5 CONCLUSÕES

Com base na experiência computacional descrita, na secção anterior, pode-se concluir que para o PORV básico:

- A utilização de um AG básico não é apropriada.

- A utilização de um AG, com adaptações na codificação e nos operadores, apresenta dificuldades de implementação. No caso da codificação e operadores utilizados, neste trabalho, não se obtiveram resultados satisfatórios.

- A introdução de procedimentos de intensificação da pesquisa em torno de boas soluções, ou de técnicas para diversificar a pesquisa a todo o espaço de soluções admissíveis, num AG, resulta num melhoramento significativo da qualidade da solução e num aumento bastante grande de tempo de execução computacional. No caso da versão híbrida, implementada neste trabalho, a introdução de procedimentos pós-optimais, que consistiram na troca de clientes entre rotas, foi bastante proveitosa.

- No que respeita à qualidade da solução, o AG híbrido mostrou-se bastante melhor do que algumas heurísticas, nomeadamente do que a heurística construtiva de rota e a heurística de duas fases, também implementadas neste trabalho. Revelou ser menos eficiente do que algoritmos de Pesquisa Tabu, cujos resultados estão descritos na literatura.

- Em relação aos tempos de execução computacional, o AG híbrido é muito mais lento que as heurísticas testadas. No entanto, os tempos de execução são, em geral, razoáveis. A sua execução permite resolver os problemas em tempo útil.

- Os resultados obtidos com o AG híbrido, para problemas reais ou com os dados estruturados, foram bons.

- A paralelização do AG híbrido só trouxe melhoramentos à qualidade da solução, em alguns casos, nos quais houve um aumento muito acentuado do tempo de execução computacional. Assim, deverá ser mais eficiente aumentar a população do algoritmo e executá-lo sequencialmente.

CAPÍTULO 6

UMA APLICAÇÃO DA OPTIMIZAÇÃO DE ROTAS DE VEÍCULOS: ESTUDO DAS ROTAS DE RECOLHA DE RESÍDUOS SÓLIDOS NA REGIÃO RURAL DE FARO

6.1 INTRODUÇÃO

O PORV desempenha um dos principais papéis na gestão da recolha e/ou distribuição de produtos, para a generalidade das empresas e instituições.

Quando se tenta resolver problemas reais surgem sempre várias aspectos que não se enquadram na versão básica do PORV, descrita no capítulo anterior. Assim, em geral, é necessário impor mais algumas restrições. As mais frequentes são:

(i) *Restrição na duração das rotas – Limite temporal*: neste caso c_{ij} representa um custo temporal e cada cliente $v_i \in V - \{v_0\}$ tem associado um tempo de serviço t_i, $t_i \geq 0$. A duração total de cada rota não pode exceder um valor L previamente estabelecido [Desrosiers et al (1995)].

De forma análoga, pode ser estabelecido um *Limite na distância* a percorrer por cada rota [Kelly e Xu (1999), Rego (1998), Renaud et al (1996), etc].

(ii) *Janelas de tempo para servir os clientes*: cada cliente $v_i \in V - \{v_0\}$, deve ser visitado num certo intervalo de tempo $[a_i, b_i]$. Em geral os veículos podem chegar antes do inicio da janela de tempo e ficar estacionados à espera [Gambardella et al (1999), Taillard et al (1997), Potvin e Bengio (1996), etc].

(iii) *Frota heterogénea*: a frota é composta por veículos com diferentes capacidades, condições de transporte e custos operacionais. Neste caso, é necessário estabelecer as rotas de acordo com os custos e as características específicas de cada veículo [Ochi et al (1998), Taillard (1996), etc.].

Em geral, pretende-se minimizar os custos totais das rotas, que podem ser definidos em termos das distâncias percorridas ou dos tempos gastos (incluindo neste caso os tempos de espera nos clientes). No entanto, outro objectivo, muito frequente, é minimizar o número de veículos a utilizar. Em várias aplicações estes dois objectivos surgem combinados ou são optimizados de uma forma hierárquica.

Devido à variedade de situações reais é praticamente impossível estabelecer um modelo global que abranja todas os casos. Muitas formalizações e algoritmos com características específicas adequadas ao problema concreto em estudo encontram-se descritos na literatura.

Algumas das aplicações mais comuns a problemas reais são:

- Entrega de mercadorias [Evans e Norback (1985), Golden e Wasil (1987), Osman (1993), Semet e Taillard (1993), Rochat e Semet (1994)].

- Distribuição de jornais [Nygard et al (1988)].

- Transporte de pessoas [Desrosiers et al (1986), Bodin e Berman (1979), Graham e Nuttle (1986), Bodin (1990), Raghavendra et al (1992)].

- Recolha de notas e títulos nas agências de um Banco [Lambert et al (1993)].

- Recolha de dinheiro de máquinas de venda [Bohoris e Thomas (1995)].

- Recolha de resíduos sólidos [Beltrami e Bodin (1972), Sculli et al (1987), Bodin (1990)].

Constata-se que em todos estes trabalhos os custos com a recolha/distribuição diminuíram consideravelmente (de 5 a 30%). É claro que uma redução de custo nas rotas, mesmo que percentualmente pequena, vai-se reflectir significativamente nos custos totais, para sectores de actividade que as fazem diariamente, ou com grande frequência.

Neste capítulo, apresenta-se uma aplicação dos algoritmos, desenvolvidos, descritos e testados anteriormente (no capítulo 5) para o PORV, com as devidas adaptações, ao problema real da recolha de resíduos sólidos na região rural de Faro

Esta tarefa é executada pelos Serviços Municipalizados da Câmara Municipal de Faro e vinha sendo feita em moldes que não serviam nem a população abrangida, nem os funcionários da Câmara. Esta situação devia-se, como foi explicado pelo responsável pelos Serviços, ao facto de as rotas, em vigor na altura, terem sido construídas de uma forma empírica e de acordo com as disponibilidades que iam tendo e as necessidades que supunham existir, sem ter havido um estudo mais rigoroso sobre os vários factores envolvidos. Os Serviços não dispunham de dados concretos sobre o número de contentores a recolher, os locais exactos onde se encontravam (pontos de recolha), as quantidades recolhidas, os dias em que os locais eram recolhidos e as distâncias

percorridas. O director dos Serviços Municipalizados, por um lado pretendia melhorar os vários factores envolvidos: tempo de recolha, distâncias percorridas, frequência de recolha, etc., e por outro desejava que a recolha fosse feita de uma forma sistematizada, de modo a que se pudesse saber, facilmente e em qualquer momento, o que estava a ser recolhido, como, quando e por quem. Isto permitiria um maior controlo de toda a recolha, possibilitando uma melhor resposta a eventuais solicitações e/ou reclamações da população e uma adequação rápida a eventuais alterações ou flutuações nos resíduos a recolher. Havia ainda que atender à alteração das leis de trabalho, que diminuíam o número de horas semanais de trabalho dos funcionários. Então, iria ser necessária a contratação de mais pessoal e a aquisição de mais viaturas se as rotas continuassem a ser feitas daquele modo.

Assim, os Serviços Municipalizados pretendiam melhorar a operação de recolha, tanto a nível de qualidade como de custos. Dispuseram-se, então, a fazer um levantamento das rotas que praticavam, dos lugares onde havia contentores, da quantidade de contentores recolhidos, da quantidade de resíduos recolhida e das distâncias percorridas.

Com a intenção de obter uma boa solução, que correspondesse aos objectivos dos Serviços Municipalizados, fez-se um estudo sobre as várias abordagens possíveis. Optou-se, então, por formalizar este caso concreto como um Problema de Optimização de Rotas de Veículos com limites diferentes na duração das rotas.

Numa primeira fase, com o intuito de dar rapidamente uma resposta satisfatória à solicitação dos Serviços Camarários e, também, de apreender todas as características especificas deste problema particular, adaptaram-se as implementações do algoritmo de duas fases e do algoritmo do tipo construtivo de rota, que já foram utilizados e descritos no capítulo anterior. As melhores soluções obtidas foram encaminhadas para os Serviços, que as puseram em prática com bons resultados. Com base em toda a experiência já adquirida, tentou-se obter uma solução melhor, fazendo as alterações necessárias para este caso à versão adaptada do AG desenvolvido para o PORV básico e descrito em 5.3.

Apresenta-se, na secção 6.2, o problema e tecem-se alguns comentários sobre os dados utilizados. Depois, na secção 6.3, descreve-se o modelo adoptado para abordar

este caso concreto. Expõe-se, na secção 6.4, alguns aspectos relativos à implementação computacional. Apresentam-se, na secção 6.5, os resultados computacionais obtidos e respectiva análise. Por fim esboçam-se algumas conclusões na secção 6.6.

6.2 DESCRIÇÃO DO PROBLEMA

A recolha de resíduos sólidos na região rural de Faro, era feita ao longo da semana, incluindo o Sábado de manhã, por 2 veículos, que operavam cada um, num total de 38 horas, em 11 períodos (manhã e tarde) com durações diferentes (de duas a quatro horas). A recolha era feita em 201 pontos, que se encontravam distribuídos pela região, havendo em muitos deles, mais do que um contentor para recolher. A frequência de recolha destes pontos, durante a semana, variava entre uma e quatro vezes.

As rotas de recolha nestes períodos, embora tivessem partes comuns, eram sempre diferentes. Isto levava a que, por um lado, alguns pontos fossem recolhidos em dias seguidos e ficassem depois 3 ou 4 dias sem serem recolhidos. Por outro, cada ponto não era servido sempre pelo mesmo veículo, o que dificultava a atribuição de responsabilidades e a adequação a flutuações pontuais de volume de resíduos a recolher numa localidade (devido a feiras, festas, etc.).

Os veículos partiam do depósito central, situado num extremo da cidade de Faro, percorriam o seu itinerário de recolha, que terminava no aterro, situado já no Concelho de Loulé, aproximadamente a 10 Km do depósito, onde descarregavam. Tinham depois que regressar ao depósito no fim do período, para recomeçarem vazios, a partir do depósito, o período seguinte. O tempo necessário para este regresso (sem recolha), incluindo o destinado à higiene dos funcionários, era aproximadamente de 30 minutos.

Os Serviços Camarários disponibilizaram 12 mapas da região onde estavam assinalados os pontos de recolha e os itinerários efectuados em cada dia da semana por cada veículo. Continham, também, o número total de contentores recolhidos e respectivo peso, e a distância percorrida. Estas informações foram organizadas nas tabelas 6.1 e 6.2. Nestas tabelas, as linhas "rota de recolha" e "tempo de recolha"

	Segunda		Terça		Quarta		Quinta		Sexta		Sábado	Total
Horário de trabalho	9-12.	13-16,5	8-12.	13-16,5	8-12.	13-16,5	8-12.	13-16,5	8-12.	13-15.	8-11.	
Período (minutos)	180	210	240	210	240	210	240	210	240	120	180	2280
Depósito - Aterro (min.)	145	175	205	175	210	175	190	175	205	90	155	1900
Pontos recolhidos	17	29	24	19	20	14	21	19	27	12	24	226
Contentores 1100 litros	13	59	30	20	18	28	45	20	27	11	22	293
Contentores 800 litros	87	27	68	67	86	41	30	67	86	28	79	666
Total de contentores	100	86	98	87	104	69	75	87	113	39	101	959
Peso recolhido (kg)	5100	5300	7300	7300	5900	4900	4200	3800	5800	1500	2700	53800
Rota Dep. - At. (Km)	37	51	46	30	36	37	62	30	36	29	30	424
Rota de recolha (Km)	15,5	35,4	22,3	17,6	19,1	13,1	29,2	17,6	23,7	9,3	21,7	224,5
Tempo de recolha (min.)	102	143,8	157,6	150,2	176,2	127,2	124,4	150,2	180,4	50,6	138,4	1501

Tabela 6.1 - Características das rotas anteriormente efectuadas (veículo 1)

	Segunda		Terça		Quarta		Quinta		Sexta		Sábado	Total
Horário de trabalho	9-12.	13-16,5	8-12.	13-16,5	8-12.	13-16,5	8-12.	13-16,5	8-12.	13-15.	8-11.	
Período (minutos)	180	210	240	210	240	210	240	210	240	120	180	2280
Depósito - Aterro (min.)	145	180	210	175	205	180	205	175	205	90	145	1915
Pontos recolhidos	20	26	26	9	22	26	26	9	24	13	19	220
Contentores 1100 litros	34	31	59	14	38	31	59	14	30	29	21	360
Contentores 800 litros	50	57	27	56	62	57	27	56	68	18	52	530
Total de contentores	84	88	86	70	100	88	86	70	98	47	73	890
Peso recolhido (kg)	6600	7400	7100	4600	5300	5800	3700	3600	5100	3800	5300	58300
Rota Dep. - At. (Km)	29	23	44	29	29	23	44	29	46	18	18	332
Rota de recolha (Km)	16	12,5	26,5	14	15	12,5	26,5	14	22,3	6,5	8,7	174,5
Tempo de recolha (min.)	119	159	175	145	177	159	170	145	157,6	67	126,4	1600

Tabela 6.2 - Características das rotas anteriormente efectuadas (veículo 2)

referem-se respectivamente à distância percorrida e ao tempo gasto entre o primeiro e o último pontos recolhidos. Este tempo foi calculado considerando que a velocidade dos veículos, quando não há recolha, é de 30 Km / hora. Na tabela 6.3 encontra-se o resumo desses dados.

Resumo	Horas trabalho	Horas Dep.-At.	Horas recolha	Pontos recolha	Conten - tores	Peso (Ton.)	Dist. rec. (Km)	Dist. Tot. (Km)
Veículo 1	38	31,67	25,02	226	959	53,800	224,5	534
Veículo 2	38	31,92	26,67	220	890	58,300	174,5	442
Total	76	63,59	51,68	446	1849	112,1	399	976

Tabela 6.3 – Resumo das tabelas 6.1 e 6.2

Nota-se, a partir destas tabelas, que embora em relação aos valores totais semanais (tabela 6.3) as diferenças entre um veículo e o outro não sejam muito significativas, já em relação aos vários períodos de recolha (tabelas 6.1 e 6.2) é grande a variação entre o número de contentores recolhidos, respectivo peso e distâncias percorridas, inclusive para períodos com o mesmo número de horas. Isto deve-se precisamente ao facto de, contrariamente ao que é desejado pelos Serviços Camarários, as rotas serem sempre diferentes e não haver uma boa distribuição, na semana, dos pontos que são recolhidos duas ou três vezes. Assim, por exemplo, pode acontecer que na primeira recolha da semana os contentores estejam muito cheios e na segunda praticamente vazios.

Resumindo, os objectivos dos Serviços Municipalizados da Câmara Municipal de Faro, em relação à recolha de resíduos sólidos eram:

1 Diminuir o número de horas semanais de recolha, (principalmente porque os funcionários teriam que passar a 37 horas de trabalho semanal em 1998 e 36 horas em 1999).

2 Tentar eliminar as recolhas ao sábado.

3 Distribuir pela semana as recolhas ao mesmo ponto, de acordo com a respectiva frequência.

4 Tanto quanto possível, construir rotas de recolha que se repitam nalguns períodos.

5 Passar a recolher 2 vezes por semana os pontos que só são recolhidos uma vez (14 pontos com 69 contentores).

6 Diminuir as distâncias totais percorridas.

Além das restrições de capacidade dos veículos é necessário ainda que:

1 As rotas comecem e terminem no depósito. Imediatamente antes de voltarem ao depósito passem no aterro para descarregar.

2 Cada período de recolha não exceda quatro horas e cada veículo não faça mais de 36 horas semanais.

3 O tempo (sem recolha) entre o aterro e o fim do período seja de cerca de 30 minutos.

DADOS UTILIZADOS

A partir das tabelas 6.1 e 6.2 verifica-se que não é possível estabelecer uma mesma relação entre os vários factores envolvidos (número de contentores por rota, peso recolhido, distâncias percorridas, duração das rotas, etc.) para todas as rotas. A variação destes valores de período para período é muito grande (ver tabela 6.4). Optou-se, então, por trabalhar, para cada factor, com valores próximos dos médios.

	Média	Mínimo	Máximo
Quilos por contentor	60,63	26,7	84,1
Limite de contentores por veículo	148,45	107,02	337,08
Tempo de colecta por contentor (min.)	1,65	1	2,1
Tempo de colecta por ponto de recolha(min)	6,95	4,2	16,11
Velocidade de colecta (metros / min.)	128,67	68,8	246,2

Tabela 6.4 – Estatísticas referentes às tabelas 6.1 e 6.2

No modelo teórico do PORV básico supõe-se que as procuras de cada cliente são previamente conhecidas com exactidão e não têm flutuações. Neste caso, a quantidade de resíduos em cada contentor oscila muito de dia para dia, não sendo possível estabelecer com exactidão a quantidade que vai ser recolhida. Por isso, apesar da capacidade dos veículos ser de 10 toneladas, considerou-se que era de apenas 9 toneladas, esperando, também, que os veículos nunca ficassem completamente cheios e que permitissem algumas flutuações na quantidade de resíduos. Desta forma, obtiveram-se os limites de contentores por veículo, em cada período (tabela 6.4).

Os mapas tinham apenas as distâncias entre os sucessivos locais de recolha. Assim, para se obter a matriz de distâncias entre todos os locais de recolha (201) existentes na zona rural de Faro, foi necessário calculá-las. Calcularam-se ainda, as distâncias entre os locais de recolha e o depósito e também entre estes e o aterro. É óbvio que a matriz de distâncias obtidas, dados os meios de que se dispunha, não corresponderá exactamente às distâncias reais; as diferenças existentes, no entanto, não deverão ser muito significativas.

6.3 MODELO UTILIZADO

Em Beltrami e Bodin (1972) é descrito um estudo executado para determinar rotas eficientes de recolha de resíduos para o Departamento Sanitário da cidade.de Nova York. Este trabalho refere, de uma forma detalhada, os vários problemas que surgem na abordagem, formalização e resolução desta situação real.

Em geral, para qualquer aplicação real de optimização de rotas, dispõe-se de um mapa da região que pode ser transformado num grafo com n vértices e um conjunto E de arcos. Uma das primeiras questões que se levanta, [Beltrami e Bodin (1972)], para se poder enquadrar o problema e mesmo para se poder passar do mapa ao grafo, é saber se a recolha é feita nos vértices ou nos arcos (em alguns casos poder-se-á ter um problema misto, [Pandit e Muralidharan (1995)], parte da recolha nos vértices e parte nos arcos). Se a recolha for feita nos vértices, estes correspondem aos pontos de recolha e os arcos

fazem a ligação entre vértices adjacentes. Se, ao contrário a recolha for feita nos arcos, os vértices representam as intersecções ou ligações entre os arcos.

No caso da recolha de resíduos sólidos na região rural de Faro, constatou-se, através dos mapas, que era mais apropriado utilizar um grafo com recolha nos vértices, visto as localizações dos contentores estarem muito dispersas pela região. Como foi referido, pelos Serviços, que todas as ruas têm dois sentidos de circulação, considerou-se que o grafo era não orientado.

Neste caso concreto, embora só sejam utilizados dois veículos, como a recolha tem de ser feita, ao longo da semana, em períodos de duração limitada, foi possível considerar que cada período corresponde a um veículo, que para além do seu limite de capacidade tem também um limite temporal de 4 horas e ainda que o produto do número de períodos (veículos) pela sua duração não pode exceder 72 horas (2 veículos × 36 horas semanais). Neste caso, teria que se adicionar ao PORV básico já descrito estas duas restrições. Assim, considerando os seguintes parâmetros:

t_{ij} – duração do percurso entre i e j, $i, j = 0,...,n$

t_i – tempo para servir o cliente i, $i = 1,...,n$

T_k – limite temporal da rota k, $k = 1,...,n$

à formalização 5.1, dada no capítulo anterior, devem ser acrescentadas as seguintes restrições:

$$T_k \leq 4\,\text{horas},\ k = 1,...,m \wedge \sum_k T_k \leq 72\,\text{horas} \qquad (6.1)$$

$$\sum_i t_i y_{ik} + \sum_i \sum_j t_{ij} x_{ijk} \leq T_k \qquad k = 1,...,m \qquad (6.2)$$

Em (6.1) estabelecem-se os limites de duração das rotas e as desigualdades (6.2) garantem que a duração das rotas não é excedida.

Inicialmente, pensou-se em formalizar o problema como um PORV com k veículos não necessariamente idênticos, k menor ou igual a 20 (2 períodos por dia × 5

dias × 2 veículos), em que o número de clientes era igual ao produto do número de pontos de recolha pela frequência de cada ponto (201 × frequência), incluindo mais uma restrição que não permitia que um ponto de recolha fosse recolhido mais de uma vez pelo mesmo veículo. No entanto, esta formalização, logo mostrou ter a grande desvantagem de não cumprir os objectivos 3 e 4 e de tornar praticamente impossível a afectação posterior das rotas aos dias da semana, [Beltrami e Bodin (1972)]. Para além disso o problema teria grandes dimensões (20 veículos e 460 clientes).

Então, decidiu-se fazer primeiro a afectação das rotas aos dias da semana e determiná-las posteriormente. Foi possível fazer a afectação das rotas, sem saber previamente que pontos elas recolhiam, porque, embora fosse um objectivo distribuir, pela semana, as recolhas ao mesmo ponto, era indiferente os dias em que eram feitas. Por exemplo, um ponto que fosse recolhido duas vezes por semana tanto podia ser à 2ª e 5ª- feira, como à 3ª e 6ª- feira ou à 2ª e 6ª- feira. A única excepção era o ponto de recolha do mercado abastecedor de Estói que devia ser recolhido à 2ª e 5ª- feira de manhã. Assim a afectação das rotas aos dias da semana só dependia da frequência dos pontos de recolha.

Subdividiu-se, então, o problema em dois PORV de menores dimensões, separando os pontos de recolha de acordo com a sua frequência. Como só há quatro pontos de recolha com frequência 4, não foram tratados isoladamente e optou-se por considerá-los como pontos duplos com frequência 2 (construiu-se uma réplica de cada um, obtendo a partir de um ponto i, com $i = 1,...,4$, de frequência 4, dois pontos i_1 e i_2 com frequência 2). Impôs-se a condição:

$$y_{i_1 k} + y_{i_2 k} \leq 1 \qquad i = 1,...,4; k = 1,...,m \tag{6.3}$$

impedindo que os pontos i_1 e i_2 com $i = 1,...,4$, sejam recolhidos pelo mesmo veículo. Assim, e de acordo com a afectação definida, conseguiu-se que esses pontos fossem recolhidos em dias diferentes.

Não foram utilizados algoritmos de afectação para fazer a distribuição das rotas pelos dias da semana, porque as restrições do problema e os objectivos 2, 3 e 4, não permitem muitas hipóteses. Além disso, uma análise dos dados (tabela 6.1 e 6.2),

confirmada pelos Serviços, mostra claramente que rotas de pequena duração (por exemplo as de 6ª - feira de 2 horas) não são produtivas, pois é maior o tempo gasto a chegar ao primeiro contentor a recolher e a ir do último ao aterro e deste ao depósito, do que o tempo gasto, efectivamente, com a recolha. Devido aos limites de horas de trabalho dos funcionários dos Serviços, diários (máximo de 4 horas por período) e semanais (de acordo com (6.1)), concluiu-se que era desejável que as rotas tivessem entre 3 e 4 horas. No entanto, optou-se por fazer duas distribuições alternativas: uma (Alternativa 1) com o principal objectivo de minimizar o tempo total de recolha e as distâncias; e a outra (Alternativa 2), tendo principalmente em conta a distribuição das rotas pela semana.

Na Alternativa 1 fazem-se por veículo e por semana 9 períodos, 4 com 4 horas de limite de duração e os outros 5 com 3 horas e meia, num total de 33 horas e meia, ficando ainda a manhã de quarta-feira livre. Na Alternativa 2, tem-se, por veículo, 10 períodos semanais, 8 com limite de duração de 3 horas e meia e os outros 2 com 3 horas, num total de 34 horas e meia semanais. Neste caso é necessário que uma das rotas (rota 12) seja feita pelos dois veículos (um em cada dia), o que não chega a ser um inconveniente, pois os Serviços sabem sempre qual o veículo que está afecto a cada um dos dias.

Apresentam-se estas duas afectações nas tabelas 6.5 e 6.6.

VEÍCULO 1			VEÍCULO 2		
Rota nº	Duração (horas)	Períodos	Rota nº	Duração (horas)	Períodos
1	4	2ª f. e 5ª f. de manhã	4	4	2ª f. e 5ª f. de manhã
2	4	3ª f. e 6ª f. de manhã	5	4	3ª f. e 6ª f. de manhã
3	3,5	3ª f. e 5ª f. de tarde	6	3,5	3ª f. e 5ª f. de tarde
7	3,5	2ª f., 4ªf. e 6ª f. de tarde	8	3,5	2ª f., 4ªf. e 6ª f. de tarde

Tabela 6.5 – Afectação das rotas aos períodos de recolha (Alternativa 1)

VEÍCULO 1			VEÍCULO 2		
Rota nº	Duração (horas)	Períodos	Rota nº	Duração (horas)	Períodos
9	3,5	2ª f. e 5ª f. de manhã	13	3,5	2ª f. e 4ª f. de manhã
10	3,5	3ª f. e 5ª f. de tarde	14	3,5	3ª f. e 6ª f. de manhã
11	3,5	4ª f. e 6ª f. de manhã	15	3,5	3ª f. e 5ª f. de tarde
12	3	3ª manhã	12	3	5ª manhã
7	3,5	2ª f., 4ªf. e 6ª f. de tarde	8	3,5	2ª f., 4ª f. e 6ª f. de tarde

Tabela 6.6 – Afectação das rotas aos períodos de recolha (Alternativa 2)

Então, obtiveram-se:

- Para os pontos com frequência 2 e 4, um PORV com 151 clientes, 6 ou 7 veículos, consoante se trate da Alternativa 1 ou 2, e limites de duração das rotas de 3,5 e 4 horas (PORV fr. 2 e 4);

- Para os pontos com frequência 3, um PORV com 50 clientes, 2 veículos e limites de duração das rotas de 3,5 horas em ambas as alternativas (PORV fr. 3).

Para estes PORV pode-se adaptar a formalização 5.1 (dada na secção 5.2) obtendo a formalização 6.1, em que se consideram as constantes:

n – número de clientes (pontos de recolha de contentores),

O depósito corresponde ao cliente 0 e o aterro ao cliente n+1

m – número de veículos

c_{ij} – a distância entre i e j, $i, j = 0,...,n$

t_{ij} – duração do percurso entre i e j, $i, j = 0,...,n+1$

q_i – a procura do cliente i, $i = 1,...,n$

t_i – tempo para servir o cliente i, $i = 1,...,n$

Q – capacidade do veículo k, $k = 1,...,m$

T_k – limite temporal da rota k, $k = 1,...,m$

e as variáveis:

$$x_{ijk} = \begin{cases} 1 & \text{se o veículo k vai de i para j directamente} \\ 0 & \text{caso contrário} \end{cases}$$

$$y_{ik} = \begin{cases} 1 & \text{se o cliente i é servido pelo veículo k} \\ 0 & \text{caso contrário} \end{cases}$$

Seja $y_k = (y_{0k}, y_{1k}, ..., y_{nk}, y_{n+1k})$

Então tem-se:

$$min \quad \sum_{k=1}^{m} f(y_k) \tag{6.4}$$

$$\text{s.a} \quad \sum_{k=1}^{m} y_{ik} = \begin{cases} 1 & , i = 1,...,n \\ m & , i = 0,\ n+1 \end{cases} \tag{6.5}$$

$$\sum_{i=1}^{n} q_i\, y_{ik} \le Q \qquad , k = 1,...,m \tag{6.6}$$

$$y_{ik} \in \{0,1\} \qquad , i = 0,...,n+1; k = 1,...,m \tag{6.7}$$

em que para cada $k,\ k = 1,...,m$

$$f(y_k) = min \sum_{i=0}^{n+1} \sum_{j=0}^{n+1} c_{ij}\, x_{ijk} \tag{6.8}$$

$$\text{s.a} \quad \sum_{i=1}^{n} x_{ijk} = y_{jk} \qquad , j = 1,...,n+1 \tag{6.9}$$

$$\sum_{j=1}^{n} x_{ijk} = y_{ik} \qquad , i = 0,...,n \tag{6.10}$$

$$\sum_{i=1}^{n} t_i y_{ik} + \sum_{i=0}^{n} \sum_{j=1}^{n+1} t_{ij} x_{ijk} \le T_k \tag{6.11}$$

$$\sum_{i,j \in S \times S} x_{ijk} \le |S| - 1 \qquad , S \subseteq \{1,...,n\},\ 2 \le |S| \le n-1 \tag{6.12}$$

$$x_{0\,n+1\,k} = 0 \tag{6.13}$$

$$x_{n+1\,0\,k} = 1 \tag{6.14}$$

$$x_{ijk} \in \{0,1\} \qquad , i, j = 0,...,n+1 \tag{6.15}$$

Formalização 6.1 - Formalização matemática para o PORV fr.2 e 4 e para o PORV fr.3

A função objectivo (6.4) minimiza a soma de funções não lineares $f(y_k)$. As restrições (6.5) garantem que os clientes estão afectos a um veículo e que o depósito e o aterro são servidos por todos os veículos. As condições (6.6) não permitem que a capacidade dos veículos seja excedida. Em (6.7) as variáveis são definidas como binárias. Para cada k, $k = 1,...,m$, a função $f(y_k)$ define um subproblema cuja função objectivo (6.8) minimiza a distância percorrida por cada veículo para recolher um determinado subconjunto de pontos de recolha. Neste subproblema as equações (6.9) e (6.10) garantem que um veículo que chega a um ponto de recolha também parte desse ponto de recolha, todos os veículos partem do depósito e chegam ao aterro. As condições (6.11) não permitem que o limite da duração das rotas seja excedido. As restrições (6.12) eliminam a formação de subcircuitos entre o depósito e o aterro. As equações (6.13) e (6.14) impõem que os veículos não vão directamente do depósito para o aterro e que do aterro sigam imediatamente para o depósito. As condições (6.15) definem as variáveis como binárias.

Para o PORV fr.2 e 4 acrescentam-se à formalização 6.1 as desigualdades (6.3) que não permitem que os clientes duplos (os de frequência 4) pertençam à mesma rota.

De acordo com a frequência dos veículos e a alternativa utilizada têm-se os valores de n iguais a 151 e 50, os de m iguais a 6 (ou 7) e 2 e Q igual a 9 toneladas. Os valores de q_i, t_i e t_{ij} foram definidos de acordo com a tabela 4. Os valores de T_k variaram entre duas horas e meia e três horas e meia, visto que se retirou à duração das rotas (ver tabelas 6.5 e 6.6) meia hora destinada ao descarregamento dos veículos no aterro, ao percurso do aterro ao depósito e à higiene dos funcionários.

O limite de duração das rotas foi tratado como uma restrição adicional de uma forma análoga às restrições de capacidade dos veículos. Assim, os algoritmos utilizados não permitem que estes limites sejam violados.

O facto de estes limites serem diferentes (dois em cada alternativa) poderia levar a uma formalização do problema como um PORV com uma frota de veículos heterogénea. Este tipo de problema aparece em muitas situações reais, geralmente associado a muitas outras condicionantes. Este modelo é importante, principalmente nos casos em que certos clientes só podem ser servidos por determinado tipo de veículo,

quer devido à capacidade do veículo, quer devido à localização geográfica do cliente que não permite o acesso de alguns veículos [Semet e Taillard (1993), Rochat e Semet (1994)], noutros casos os custos de cada ligação entre clientes depende do tipo de veículo que a faz, sendo por isso necessário decidir qual o melhor veículo para fazer cada percurso[Taillard (1996)].

No caso deste trabalho, o importante é determinar um bom calendário para visitas ao mesmo ponto e boas rotas para recolher os contentores, não sendo relevante, para qualquer ponto, se este é recolhido numa rota de maior ou menor duração. Como foi exposto, nas secções 5.3 e 5.4, nos métodos utilizados as rotas são inicializadas com um cliente–"semente", depois os outros clientes (pontos de recolha) vão sendo introduzidos nas rotas enquanto a capacidade dos veículos e a duração da rota permitirem. A afectação destes às rotas tem em conta o custo de inserção de cada cliente, ainda livre, na rota. Este custo é calculado em relação ao clientes–"semente", no caso da heurística de duas fases, ou em relação a cada um dos clientes que já pertencem à rota (depósito e aterro incluídos) no caso da heurística construtiva de rota utilizada também para gerar a população inicial do AG. Assim, optou-se por considerar o facto de as rotas não terem todas a mesma duração apenas na afectação do cliente–"semente", do ponto de recolha do mercado abastecedor de Estói e dos quatro pontos de frequência 4. Como os critérios de escolha dos clientes–"semente" se baseiam na maior distância (ao depósito e ao aterro simultaneamente) e na maior procura, começou-se a afectação dos clientes-"semente" pelas rotas com maior duração.

6.4 IMPLEMENTAÇÃO COMPUTACIONAL

Como, a partir dos dados disponíveis (tabelas 6.1 e 6.2), não se conseguiu estabelecer uma relação exacta entre o tempo gasto para recolher cada contentor e o tempo gasto no percurso entre os pontos de recolha, nem saber o valor médio do tempo gasto exclusivamente com a recolha dos contentores ou exclusivamente com o percurso entre contentores, considerou-se como parâmetro principal na duração das rotas o tempo gasto na recolha dos contentores, isto porque é o factor de menor variação de período para período (ver tabela 6.4). Assim, sendo:

$$t_i = \sigma\, q_i \ , \quad i = 1,...,n \quad e$$
$$t_{ij} = \lambda\, c_{ij} \ , \quad i = 0,...,n;\ j = 1,...,n+1 \tag{6.16}$$

resolveu-se considerar, após alguns testes computacionais,

$$\sigma = 1{,}6 \ e \ \lambda = 0{,}2 . \tag{6.17}$$

Em qualquer um destes algoritmos as rotas formadas inicialmente são do tipo: depósito – cliente-"semente" – aterro – depósito.

No caso da heurística de duas fases o custo de inserção de cada cliente ainda livre numa rota é calculado apenas em relação à rota depósito – cliente-"semente" – aterro, independentemente dos outros clientes que já estejam afectos a essa rota, uma vez que na primeira fase desta heurística somente se obtém uma afectação admissível dos clientes aos veículos. A ordem pela qual eles serão servidos é determinada na segunda fase. Assim, nesta heurística, tratou-se a restrição do limite temporal como a do limite de capacidade, isto é, a restrição (6.11) passou a ser dada por:

$$\sum_{i=1}^{n} t_i y_{ik} \le T_k \qquad k = 1,...,m \tag{6.18}$$

sendo, neste caso,

$$t_i = \sigma\, q_i \ , \quad i = 1,...,n \text{ com } \sigma = 1{,}7 .$$

Então, um cliente só pode ser afecto a um veículo se verificar as restrições (6.6) e (6.18).

Na heurística construtiva de rota e no AG, os custos de inserção de cada cliente numa rota são calculados em relação a todos os clientes que já fazem parte dessa rota, como descrito em 5.3, considerando que a rota começa no depósito (cliente 0) e termina no aterro (cliente n+1). O cliente é inserido na melhor posição, conhecendo-se sempre a rota já formada. Assim, nestes algoritmos, utilizaram-se os valores de (6.16) e (6.17) na restrição (6.18), da formalização 6.1, obtendo:

$$\sum_{i=1}^{n} 1.6\, q_i y_{ik} + \sum_{i=0}^{n} \sum_{j=1}^{n+1} 0.2\, c_{ij}\, x_{ijk} \le T_k \qquad k = 1,...,m \tag{6.19}$$

Para uma melhor comparação entre os algoritmos, o tempo da duração das rotas da melhor solução, obtida pela heurística de duas fases foi, no fim, calculado de acordo com (6.18). Na solução final, para os três algoritmos, foi necessário adicionar meia hora ao tempo de duração de cada rota, correspondente ao trajecto aterro – depósito e higiene dos funcionários, para se obter o tempo total de cada rota.

Para se poderem afectar as rotas obtidas aos períodos de recolha cumprindo os requisitos já expostos, impôs-se que o ponto de recolha do mercado abastecedor de Estói e os quatro pontos duplos fossem afectos a rotas de 4 horas de duração, na Alternativa 1 e de 3 horas e meia na Alternativa 2. Desta forma, a rota que incluísse o ponto de recolha do mercado abastecedor de Estói era considerada, de acordo com as tabelas 6.5 e 6.6, a nº 1 ou nº 4, na Alternativa 1, e a nº 9 na Alternativa 2. As rotas que incluíssem os pontos com frequência 4 eram, na Alternativa 1, a nº 1 (ou nº 4) e a nº 2 (ou nº 5). Na Alternativa 2 existem várias formas de as afectar às rotas números 9, 10, 11, 12, 13, 14 e 15.

Em relação às distâncias trabalhou-se em metros, reduzindo depois o resultado final a quilómetros.

Em relação ao AG gerou-se, para resolver os dois PORV uma população de 50 indivíduos. Os operadores cruzamento e mutação foram aplicados de forma a nunca produzirem indivíduos não admissíveis e só quando são mais aptos (melhores soluções) substituem o(s) pai(s). Em cada iteração são executados 20 cruzamentos e uma mutação. Estabeleceu-se em 50 o limite de gerações a executar e considerou-se que o algoritmo converge se em 5 gerações consecutivas o valor global da aptidão não diminuir. Com base na experiência anterior com o AG, supôs-se que 50 gerações é um valor suficientemente grande para que uma população de 50 indivíduos convirja. Por outro lado, como só é permitido que os filhos substituam os pais quando são melhores, a aptidão global da população nunca piora. Então, se em 5 gerações consecutivas não há melhoramento, globalmente, provavelmente também não se obteria melhoramentos continuando a execução do algoritmo.

Como foi referido anteriormente, para, com brevidade, apresentar uma boa solução aos Serviços Camarários e para se ter uma maior sensibilização aos aspectos envolvidos, aplicaram-se inicialmente ao problema, a heurística de duas fases e a

heurística construtiva de rota. A melhor solução obtida (conseguida com a heurística construtiva de rota) foi entregue aos Serviços Camarários que a puseram em prática com resultados muito satisfatórios e próximos dos estabelecidos computacionalmente. A única observação negativa que foi feita, referia-se a que em dois períodos um veículo tinha ficado completamente cheio, não tendo sido possível recolher os últimos dois contentores.

Se a informação contida nas tabelas 6.1 e 6.2 fosse consistente esta situação não deveria ter ocorrido. Com efeito, verifica-se que as rotas que recolhiam mais contentores não ultrapassavam os 113 e nunca excediam 75% da capacidade de carga dos veículos. Ora, até perfazer os 90% da sua capacidade ainda seria possível recolher mais 35 contentores, ou seja um máximo de 148. Acontece que nas rotas que foram inicialmente propostas à Câmara nunca eram recolhidos mais de 125 contentores. Como não era possível obter dados mais fiáveis optou-se, para contornar o problema, por estabelecer em 115 o limite máximo de contentores a recolher em cada rota. Em termos de duração das rotas, a implementação no terreno correspondeu aproximadamente ao estabelecido computacionalmente.

Os programas foram codificados em TURBO C e executados num PC com um processador Pentium MMX 200 MHz e com 16 MB de RAM.

6.5 RESULTADOS COMPUTACIONAIS

Apresentam-se, na tabela 6.7, os melhores resultados obtidos com cada algoritmo e, na tabela 6.8, a diferença percentual (*dp*) entre as distâncias totais percorridas, dada por:

$$dp = \frac{\text{distância obtida} - 976}{976} \text{ x } 100 \text{ .}$$

Algoritmo	PORV fr. 2 e 4		PORV fr. 3	PORV			
	ALT. 1	ALT. 2		ALT. 1		ALT. 2	
	Distância (Km)	Distância (Km)	Distância (Km)	Distância Total(Km)	Tempo de Exec.(seg.)	Distância Total(Km)	Tempo de Exec.(seg.)
2 fases	762,30	784,60	246,15	1008,45	27,03	1030,75	25,66
Construtivo	642,80	717,30	207,9	850,70	28,96	925,20	30,77
Genético	633,90	666,60	184,35	818,25	204,45	850,95	165,00

Tabela 6.7 – Resumo dos resultados

Algoritmo	*dp*	
	ALT. 1	ALT. 2
2 fases	3,32	5,61
Construtivo	-12,84	-5,20
Genético	-16,16	-12,81

Tabela 6.8 – Diferença percentual

O comportamento dos algoritmos para esta aplicação real foi semelhante ao observado com os dados da literatura (ver 5.4). As melhores soluções, para as duas alternativas, foram obtidas com o AG e as piores com o algoritmo de 2 fases. Em relação aos tempos de execução, também neste caso, o algoritmo de 2 fases é o mais rápido e o AG é o mais lento, sendo consideravelmente mais demorado que os outros dois.

Salienta-se ainda que mesmo o algoritmo de duas fases traz melhoramentos importantes para os percursos de recolha, porque apesar das rotas obtidas serem mais longas que as praticadas pelos Serviços Municipalizados, são recolhidos mais 14 pontos com 69 contentores (objectivo 5), o tempo total de recolha é reduzido (objectivo 1) em 7 ou 9 horas (consoante seja a Alternativa 2 ou 1) e os objectivos 2, 3 e 4 são, também, atingidos. Os outros dois métodos, principalmente o AG, conseguem ainda, diminuir a distância total a percorrer (objectivo 6), satisfazendo todos os objectivos e obtendo, por isso, melhores soluções em todos os aspectos.

Obviamente obtêm-se menores distâncias com a Alternativa 1, visto conter menos duas rotas. No entanto, apesar de percorrer uma maior distância, a Alternativa 2

pode ser interessante porque os veículos ficam sempre mais vazios. Esta alternativa pode adaptar-se melhor às flutuações populacionais que ocorrem no Algarve e mesmo aos períodos de festas em que o volume de resíduos aumenta.

Em relação aos tempos de execução, as duas heurísticas são muito rápidos. O AG foi mais lento, como era esperado, pois sabe-se que, em geral, este tipo de métodos baseados em pesquisa local são mais demorados. Mas, o tempo obtido (menos de 205 segundos), é completamente aceitável, permitindo inclusive a variação dos parâmetros para obter outras soluções. De qualquer forma, a melhoria obtida com o AG em relação à qualidade da solução justificaria a sua utilização mesmo que fosse mais demorado.

Estes métodos permitem a inclusão de mais pontos de recolha ou mais contentores, alterando apenas o ficheiro de entrada que contém os dados.

As tabelas 6.9, 6.10, 6.11 e 6.12 resumem os resultados obtidos com o AG (visto ser a melhor solução) e podem ser facilmente comparadas com as tabelas 6.1, 6.2 e 6.3.

Nota-se que a diferença entre a distância total percorrida e a distância percorrida para recolher os contentores (entre o primeiro e o último contentor de cada rota) é muito maior nas rotas praticadas anteriormente pelos Serviços Camarários do que nas obtidas pelo AG. É também de salientar que, apesar de se recolherem mais pontos e contentores, o tempo gasto na recolha dos resíduos, com as soluções do AG, é, ainda assim, inferior ao praticado pelos Serviços.

Comparando as duas alternativas, verifica-se que os acréscimos nas distâncias e nos tempos gastos a recolher, da Alternativa 2, devem-se sobretudo à existência de mais dois períodos. Constata-se que as rotas da Alternativa 2 estão muito folgadas, permitindo reduzir, em média, 10 minutos na duração de cada período correspondente às rotas 9, 10, 11, 13, 14 e 15, isto levaria a que para esta alternativa se fizessem também 33,5 horas de trabalho semanal por veículo.

Qualquer uma destas alternativas permite, devido ao tempo livre ainda existente em relação às 36 horas semanais, que alguns pontos sejam recolhidos com maior frequência ou a inclusão de outros pontos de recolha.

VEÍCULO 1										
	Segunda		Terça		Quarta		Quinta		Sexta	
Horário	8-12	13-16,5	8-12	13-16,5		13-16,5	8-12	13-16,5	8-12	13-16,5
Período (min.)	240	210	240	210		210	240	210	240	210
Dep.-At.(min)	199	155	190	180	livre	155	199	180	190	155
Pontos recolh.	34	19	24	27		19	34	27	24	19
Contentores	113	95	115	109		95	113	109	115	95
Peso (kg)	7458	6270	7590	7194		6270	7458	7194	7590	6270
Rota Dep-At(Km)	84,45	13,95	31,25	26,35		13,95	84,45	26,35	31,25	13,95
Rota rec. (Km)	64,85	10,75	21,4	19,15		10,75	64,85	19,15	21,4	10,75
Recolha (min.)	159,8	148,6	170,3	165,6		148,6	159,8	165,6	170,3	148,6
VEÍCULO 2										
	Segunda		Terça		Quarta		Quinta		Sexta	
Horário	8-12	13-16,5	8-12	13-16,5		13-16,5	8-12	13-16,5	8-12	13-16,5
Período (min.)	240	210	240	210		210	240	210	240	210
Dep.-At.(min)	189	158	192	180	livre	158	189	180	192	158
Pontos recolh.	22	31	26	22		31	22	22	26	31
Contentores	114	95	115	108		95	114	108	115	95
Peso (kg)	7524	6270	7590	7128		6270	7524	7128	7590	6270
Rota Dep-At(Km)	32,80	27,50	40,50	41,60		27,50	32,80	41,60	40,50	27,50
Rota rec. (Km)	22,70	22,05	30,30	31,50		22,05	22,70	31,50	30,30	22,05
Recolha (min.)	169	147	172	160		147	169	160	172	147

Tabela 6.9 – Características das rotas propostas (Alternativa 1)

Resumo	Horas trabalho	Horas Dep-At	Horas recolha	Pontos recolha	Conten-tores	Peso (Ton.)	Dist.rec (Km)	Dist.Tot (Km)
Veículo 1	33,5	26,72	23,95	227	959	63,294	243,05	415,95
Veículo 2	33,5	26,60	24,03	233	959	63,294	235,15	402,3
Total	67	53,32	47,98	460	1918	126,59	478,2	818,25

Tabela 6.10 - Resumo da tabela 6.9

VEÍCULO 1										
	Segunda		Terça		Quarta		Quinta		Sexta	
Horário	8,5-12	13-16,5	9-12	13-16,5	8,5-12	13-16,5	8,5-12	13-16,5	8,5-12	13-16,5
Período (min.)	210	210	180	210	210	210	210	210	210	210
Dep.-At.(min)	160	155	143	166	173	155	160	166	173	155
Pontos recolh.	16	19	23	33	23	19	16	33	23	19
Contentores	97	95	86	99	100	95	97	99	100	95
Peso (kg)	6402	6270	5676	6534	6600	6270	6402	6534	6600	6270
Rota Dep-At(Km)	25,6	13,95	28,2	36,35	65,95	13,95	25,6	36,35	65,95	13,95
Rota rec. (Km)	10,5	10,75	24,5	26,2	52,35	10,75	10,5	26,2	52,35	10,75
Recolha (min.)	130	148,6	136	145,7	146	148,6	130	145,7	146	148,6
VEÍCULO 2										
	Segunda		Terça		Quarta		Quinta		Sexta	
Horário	8,5-12	13-16,5	8,5-12	13-16,5	8,5-12	13-16,5	9-12	13-16,5	8,5-12	13-16,5
Período (min.)	210	210	210	210	210	210	180	210	210	210
Dep.-At.(min)	159	158	166	165	159	158	143	165	166	158
Pontos recolh.	21	31	20	19	21	31	23	19	20	31
Contentores	94	95	98	100	94	95	86	100	98	95
Peso (kg)	6204	6270	6468	6600	6204	6270	5676	6600	6468	6270
Rota Dep-At(Km)	42	28	43	22	42	28	28,2	22	43	28
Rota rec. (Km)	35	22	35	18	35	22	24,5	18	35	22
Recolha (min.)	146	147	150	156	146	147	136	156	150	147

Tabela 6.11 – Características das rotas propostas (Alternativa 2)

Resumo	Horas trabalho	Horas Dep-At	Horas recolha	Pontos recolha	Conten-tores	Peso (Ton.)	Dist. rec.	Dist. Tot.
Veículo 1	34,5	26,77	23,73	224	963	63,558	234,85	425,85
Veículo 2	34,5	26,62	24,68	236	955	63,030	266,85	425,1
Total	69	53,38	48,41	460	1918	126,59	501,7	850,95

Tabela 6.12 – Resumo da tabela 6.11

Como já se referiu, as rotas inicialmente obtidas com o algoritmo construtivo foram postas em prática pelos Serviços Camarários, tendo sido feito apenas o reparo em relação a dois períodos em que os veículos ficaram muito cheios. No entanto, os

Serviços Municipalizados passaram a efectuar a recolha dos resíduos sólidos segundo essas rotas (da Alternativa 2).

As rotas obtidas com o AG, foram também entregues aos Serviços Municipais, mas não chegaram a ser implementadas na prática porque, por um lado, estavam satisfeitos com as outras e não valia a pena estar de novo a alterar tudo (visto que "leva sempre algumas semanas até aos funcionários se adaptarem a essa mudança"). Por outro lado, como está a ser construído o novo aterro sanitário e se prevê a aquisição de novos veículos e contentores, incluindo alguns para recolha selectiva, foi sugerido, pelos Serviços Municipalizados, que se repetisse o estudo quando toda esta fase de remodelação estiver concluída, alterando então de novo as rotas.

Posteriormente, apenas para efeitos de estudo, testou-se também a versão híbrida do AG, descrita em 5.3. Definiu-se a dimensão da população em 80 e 40 indivíduos consoante fosse o PORV fr.2 e 4 ou o PORV fr.3. e estabeleceu-se, respectivamente, em 60 e 35 o número máximo de gerações a executar. O melhor resultado obtido com o AG híbrido, para as duas alternativas, foi o mesmo já obtido com a versão adaptada do AG. Assim, neste caso, a utilização do AG híbrido não foi vantajosa. Além de não ter obtido melhores soluções demorou mais do dobro do tempo de execução, 496 segundos e 453 segundos respectivamente para a Alternativa 1 e para a Alternativa 2.

6.6 CONCLUSÕES

Os algoritmos utilizados neste trabalho conseguem resolver o problema da recolha de resíduos sólidos com uma diminuição de pelo menos 10% no número de horas e ainda recolhendo mais 69 contentores. Para além disso, fazem uma boa distribuição pela semana dos locais a recolher. No que respeita às distâncias totais o AG é muito eficiente, atingindo reduções de 13 ou 16 % (respectivamente com a Alternativa 2 e a Alternativa 1).

Os resultados obtidos mostram que o recurso a métodos heurísticos, para resolução de problemas reais, se traduz em melhoramentos significativos quer a nível de custos quer a nível de qualidade, obtendo-se boas soluções em tempo útil.

Os bons resultados obtidos com o AG permitem confirmar que este tipo de métodos globais, baseados na pesquisa local e que trabalham com um conjunto de soluções, são eficientes. Em geral, estas técnicas de resolução são lentas, sendo esse o principal defeito que normalmente lhes é apontado. No entanto, neste caso, os tempos de execução são bastante reduzidos, permitindo alterações de acordo com a característica sazonal do volume de resíduos na região de Faro. Possibilita, também, a execução do algoritmo repetidas vezes com variação dos valores dados em (6.17), obtendo-se rotas diferentes e eventualmente mais adaptadas a semanas específicas.

CAPÍTULO 7

CONCLUSÕES FINAIS E TRABALHOS FUTUROS

Nos capítulos anteriores já foram apontadas as principais conclusões retiradas das várias experiências computacionais realizadas. Assim, esboçam-se neste capítulo as conclusões gerais e expõem-se alguns dos trabalhos que se gostaria de vir a fazer, sugeridos no desenrolar deste.

Neste trabalho foram desenvolvidos algoritmos para o Programa Quadrático 0-1 e para o Problema de Optimização de Rotas de Veículos. Implementaram-se algumas versões, para serem executadas em máquinas sequenciais, de um algoritmo exacto de Pesquisa em Árvore e de um Algoritmo Genético para o Programa Quadrático 0-1. Estudou-se também a aplicação de um Algoritmo Genético ao Problema de Optimização de Rotas de Veículos. A melhor versão de cada algoritmo foi, depois, implementada para ser executada em máquinas paralelas com memória distribuída.

O Programa Quadrático 0-1 é um problema de variáveis booleanas sem restrições e de formalização muito simples. O Problema de Optimização de Rotas de

Veículos é um problema de formalização matemática mais complexa, para a qual têm sido utilizadas diferentes abordagens, que incluem sempre um conjunto de várias restrições. Assim, em termos de formalização matemática estes dois problemas são totalmente distintos, o que constituiu um motivo para o estudo comparativo da implementação de um método genérico (uma meta-heurística) para os resolver. Optou-se pelo Algoritmo Genético por, entre outros motivos, ser um método directamente aplicável ao Programa Quadrático 0-1 mas de difícil aplicação ao Problema de Optimização de Rotas de Veículos. Assim, supõe-se, à priori, que se possam obter bons resultados num (no Programa Quadrático 0-1) e não tão bons no outro (no Problema de Optimização de Rotas de Veículos), sendo, por isso, interessante o estudo comparativo da utilização de Algoritmos Genéticos na resolução dos dois problemas.

ALGORITMOS SEQUENCIAIS

Em relação ao desempenho dos algoritmos sequenciais destacam-se o algoritmo exacto (APA) e o Algoritmo Genético para o Programa Quadrático 0-1.

Os resultados obtidos indicam que o APA é consideravelmente mais eficiente que o algoritmo exacto de Pardalos e Rodgers (1990), pois os mesmos problemas são resolvidos em cerca de um terço do tempo de execução e as árvores de pesquisa construídas têm cerca de metade dos vértices. Os resultados teóricos, apresentados no capítulo 4, demonstram que a forma proposta nesta tese para determinação e actualização dos limites inferiores da função objectivo é mais eficiente. A experiência computacional não só confirmou os resultados teóricos como também mostrou que os limites propostos tornam o algoritmo de Pesquisa em Árvore muito mais eficiente.

Como era esperado, o Algoritmo Genético, desenvolvido para o Programa Quadrático 0-1, obteve, inclusive nas versões menos elaboradas, um desempenho igual ou superior a uma versão de uma heurística, com bons resultados referidos na literatura [Faustino (1992)], embora apresente tempos de execução maiores. A melhor versão do Algoritmo Genético mostrou-se competitiva com uma meta-heurística de PT, que obteve os melhores resultados para um conjunto de problemas-teste [Glover et al (1998)]. Em termos de tempos de execução do Algoritmo Genético não é possível

comparar os resultados obtidos com os mencionados no trabalho de Glover et al (1998), por terem sido utilizados computadores diferentes. Contudo, pode-se verificar que este Algoritmo Genético tem tempos de execução muito mais reduzidos do que os esperados para uma meta-heurística.

As várias versões do Algoritmo Genético para o Problema de Optimização de Rotas de Veículos não obtiveram resultados tão interessantes. Em geral, os resultados foram de qualidade inferior aos obtidos por outras meta-heurísticas existentes para o Problema de Optimização de Rotas de Veículos baseadas em PT. No entanto, os Algoritmos Genéticos obtiveram resultados consideravelmente melhores que heurísticas não muito elaboradas. Também, a melhor versão dos Algoritmos Genéticos desenvolvidos neste trabalho, AG híbrido, obteve muito bons resultados para problemas-teste, conhecidos da literatura, extraídos de aplicações reais ou com dados estruturados. Estes bons resultados foram confirmados na aplicação destes Algoritmos Genéticos a um caso real. Em relação aos tempos de execução computacional, os Algoritmos Genéticos mostraram-se bastante mais lentos que as outras duas heurísticas utilizadas, sendo, no entanto, os seus tempos de execução bastante aceitáveis.

Confirmou-se a suposição inicial de que um Algoritmo Genético seria eficiente na resolução do Programa Quadrático 0-1 e que seria difícil a sua implementação eficiente para o Problema de Optimização de Rotas de Veículos. Neste trabalho, a implementação do Algoritmo Genético para o Problema de Optimização de Rotas de Veículos revelou-se eficiente apenas para um tipo específico de instâncias, as que tinham dados estruturados ou eram provenientes de aplicações reais.

Adaptar um Algoritmo Genético ao Problema de Optimização de Rotas de Veículos não foi tarefa fácil. As maiores deficiências da abordagem realizada devem-se a:

(i) Não se ter permitido que na geração da população inicial fossem produzidos indivíduos (soluções) que violassem as restrições, o que obrigou a utilizar uma heurística para gerar a população inicial;

(ii) Na aplicação dos operadores genéticos também não foram permitidas violações das restrições;

(iii) Cada indivíduo representar uma afectação dos clientes aos veículos e não uma solução (as rotas) do Problema de Optimização de Rotas de Veículos.

Em relação aos aspectos (i) e (ii) também se fizeram experiências gerando aleatoriamente a população inicial e permitindo violações das restrições, mas obtiveram-se fracos resultados. Nestas experiências penalizaram-se os indivíduos não admissíveis atribuindo-lhes uma aptidão menor. Assim, não eram escolhidos para se reproduzirem tantas vezes como os indivíduos admissíveis. Talvez seja possível obter melhores resultados penalizando a não admissibilidade de uma forma que permita aos indivíduos não admissíveis serem seleccionados mais vezes para reprodução, supondo que a sua combinação com indivíduos admissíveis possa diversificar a pesquisa.

O aspecto (iii) pode levar a que, eventualmente, se encontre a melhor afectação possível dos clientes aos veículos, e nesse sentido o Algoritmo Genético seja muito eficiente, mas as rotas dos veículos, para servirem os clientes que lhes estão afectos, obtidas através da resolução heurística dos Problemas do Caixeiro Viajante não sejam as melhores. Assim, a solução do Problema de Optimização de Rotas de Veículos pode não ser a óptima mesmo tendo sido obtida a afectação óptima dos clientes aos veículos. A determinação das rotas óptimas, para uma afectação obtida pelo Algoritmo Genético, envolvia a resolução exacta de um Problema do Caixeiro Viajante para cada veículo, o que é proibitivo em termos de tempo de execução computacional.

ALGORITMOS PARALELOS

A utilização de algoritmos paralelos foi, em geral, vantajosa. Em particular, nos casos em que os algoritmos sequenciais já eram eficientes, a sua paralelização trouxe uma redução do tempo de execução ou uma melhoria na qualidade da solução. Os bons resultados obtidos com a paralelização levam a concluir que as topologias utilizadas foram adequadas.

De facto, para a maioria das instâncias do Programa Quadrático 0-1 testadas, foram obtidas reduções significativas do tempo de execução do APA. Assim, foi

possível, no mesmo limite de tempo de execução do algoritmo sequencial, encontrar a solução óptima para instâncias de maior dimensão.

Também o Algoritmo Genético paralelo para o Programa Quadrático 0-1 mostrou poder melhorar a qualidade da solução (com um pequeno aumento do tempo de execução computacional), ou reduzir o tempo de execução computacional sem deteriorar significativamente a qualidade da solução. Como as soluções obtidas pelo algoritmo sequencial já são as óptimas ou muito próximas da óptima, pode-se utilizar o algoritmo paralelo para diminuir o tempo de execução computacional. Alternativamente pode-se trabalhar em cada processador com uma população apenas um pouco menor que a população utilizada sequencialmente, conseguindo uma diminuição (ou manutenção) do tempo de execução sequencial e simultaneamente melhorar, eventualmente, a qualidade da solução.

A paralelização do Algoritmo Genético híbrido para o Problema de Optimização de Rotas de Veículos não foi vantajosa. A diminuição do tempo de execução reflectiu-se numa perda de qualidade das soluções e o aumento da diversidade da população originou um aumento excessivo dos tempos de execução sem uma melhoria equivalente da qualidade das soluções. A paralelização visando a diminuição do tempo de execução diminui a dimensão da população em cada processador o que levou a que não existisse suficiente diversidade nos indivíduos conduzindo o Algoritmo Genético híbrido a uma solução subóptima. A geração heurística da população inicial e a aplicação dos procedimentos de pós-optimização, com tempos elevados de execução computacional, são os factores responsáveis pela ineficiência da paralelização do Algoritmo Genético híbrido visando o aumento da diversidade populacional.

Esta tese evidencia as vantagens da utilização de algoritmos paralelos de Pesquisa em Árvore e de Algoritmos Genéticos paralelos bem como aponta para as situações em que a sua utilização pode não ser proveitosa. Os algoritmos paralelos e topologias utilizados neste trabalho, para uma rede de transputers, podem ser utilizados, com pequenas adaptações, noutros tipos de máquinas paralelas com memória distribuída. Para máquinas paralelas com uma relação potência de cálculo/velocidade de comunicação semelhante à dos transputers é esperado que o desempenho dos algoritmos, aqui desenvolvidos e testados, se mantenha.

Resolução de um Caso Real

Durante o tratamento da aplicação real efectuado neste trabalho sobressaíram, de forma clara, as muitas dificuldades que se levantam quando se pretende aplicar a teoria aos casos práticos. Mas, de uma forma incontestável, os resultados obtidos mostram que, apesar dessas dificuldades, a utilização de métodos heurísticos, particularmente de meta-heurísticas, na resolução de um problema real são de grande valia. De facto, os algoritmos utilizados neste trabalho conseguiram resolver o problema real em estudo, de acordo com os objectivos dos Serviços Camarários e satisfazendo todas as restrições por eles impostas, obtendo soluções qualitativamente muito melhores do que as anteriormente praticadas. Também, em termos de custos (que no caso concreto são as distâncias percorridas) atingiram-se reduções de até 16 %.

Trabalhos Futuros

A forma utilizada (topologia e comunicações) para paralelizar o algoritmo de Pesquisa em Árvore pode vir a ser testada em algoritmos de Pesquisa em Árvore para outros Problemas de Optimização Combinatória. É de supor que se voltem a obter bons resultados.

Ressalta, da experiência computacional realizada, que a introdução de procedimentos baseados em pós-optimização nos Algoritmos Genéticos, substituindo a mutação aleatória, foi sempre proveitosa. Então, talvez se possam obter resultados ainda melhores se forem também introduzidos procedimentos, que provoquem mutações nos indivíduos, baseados em funções de memória que permitam intensificar a pesquisa em torno de bons indivíduos e diversificar a pesquisa a outras regiões do espaço das soluções a partir dos indivíduos menos aptos. Seria interessante poder vir a testar computacionalmente esta suposição.

Face aos bons resultados obtidos com o Algoritmo Genético (AG5) para o Programa Quadrático 0-1 seria também interessante adaptá-lo e testá-lo em problemas quadráticos 0-1 com restrições, ou mesmo noutro tipo de problemas quadráticos.

O Algoritmo Genético híbrido, desenvolvido para o Problema de Optimização de Rotas de Veículos, poderá eventualmente ficar mais eficiente mediante algumas alterações que tenham em conta os aspectos (i), (ii) e (iii) focados atrás. As alterações, que podem ser testadas, separadamente ou em conjunto em trabalhos futuros, decorrem da observação dos resultados da experiência computacional apresentada e são:

- Utilizar heurísticas mais eficientes para a resolução dos Problemas do Caixeiro Viajante envolvidos;

- Permitir afectações não admissíveis recorrendo a parâmetros penalizadores incluídos no cálculo da função objectivo;

- Alterar a actuação dos operadores genéticos tendo em conta que as soluções podem ser avaliadas segundo dois critérios. Um relativo à violação das restrições de capacidade e o outro às distâncias percorridas;

- Gerar a maior parte da população inicial aleatoriamente e somente alguns (poucos) indivíduos heurísticamente.

Se alguma alteração conduzir a um Algoritmo Genético eficiente para o Problema de Optimização de Rotas de Veículos então, tendo em conta os bons resultados obtidos com o Algoritmo Genético paralelo para o Programa Quadrático 0-1, é de considerar também a sua paralelização.

Os excelentes resultados que se obtiveram no estudo e resolução de um caso concreto reforçam a intenção de, no futuro, continuar a desenvolver métodos e técnicas de resolução de Problemas de Optimização Combinatória motivados por solicitações da vida real.

Na sequência deste trabalho surge, de uma forma natural, a intenção de, futuramente, adaptar e testar os algoritmos paralelos desenvolvidos em outras máquinas paralelas, utilizando também outras topologias e um número maior de processadores.

REFERÊNCIAS

AARTS, E.; J. KORST; P. LAARHOVEN (1997), *Simulated annealing*, in Local Search in Combinatorial Optimization, editado por E. Aarts e J. Lenstra, John Wiley & Sons.

ALMEIDA, M. T. (1988), *Problema básico de distribuição e suas extensões – uma revisão bibliográfica*, Doc. Trabalho nº 46, CEMAPRE.

ALVES, M. L. e M. T. ALMEIDA (1992), *Simulated annealing for the simple plant location problem: a computational study*, Investigação Operacional, vol. 12, nº 2, pp. 145-157.

AMDAHL, G. M. (1967), *Validity of the single processor approach to achieving large scale computing capabilities*, AFIPS Conference Proceedings 30, pp. 483-485.

ANDROULAKIS, I. P. e C. A. FLOUDAS (1998), *Distributed branch and bound algorithms for global optimization*, in Parallel Processing of Discrete Problems, editado por P. Pardalos, Springer Verlag.

BADEAU, P.; M. GENDREAU; F. GUERTIN; J.-Y. POTVIN; E. TAILLARD (1997), *A parallel tabu search heuristic for the vehicle routing problem with time windows*, Transportation Research-C5, pp. 109-122.

BAKER, J. (1987), *Reducing bias and inefficiency in the selection algorithm*, Conference Proceedings ICGA 2, pp. 14-21.

BARAHONA, F. (1986), *A solvable case of quadratic 0-1 programming*, Disc. Appl. Math., 13, pp. 23-26.

BARAHONA, F.; M. JÜNGER; G. REINELT (1989), *Experiments in quadratic 0-1 programming*, Math. Prog., 44, pp.127-137.

BASS, J. (1996), *Multiple instructions multiple data systems I*, Parallel Processing Module, Msc Control Systems, The University of Sheffield.

BELTRAMI, E. e L. BODIN (1972), *Networks and vehicle routing for municipal waste collection*, Networks, 4, pp. 65-94.

BERNER, S. (1996), *Parallel methods for verified global optimization practice and theory*, J. Global Optimization, 9, pp. 1-22.

BERTSEKAS, D. P. e J. N. TSITSIKLIS (1989), *Parallel and Distributed Computation, Numerical Methods*, Prentice-Hall, Englewood Cliffs, N. J.

BODIN, L. (1990), *Twenty years of routing and scheduling*, Ops. Res., 38, pp. 571-579.

BODIN, L. e L. BERMAN (1979), *Routing and scheduling of school buses by computer*, Transp. Science, vol. 13, pp. 113-129.

BOHORIS, G. e J. THOMAS (1995), *A heuristic for vehicle routeing and depot staffing*, J.O.R.S., vol. 46, pp. 1184-1191.

BOROS, E.; Y. CRAMA; P. HAMMER (1990), *Upper bounds for quadratic 0-1 maximization*, Ops. Res. Letters, 9, pp. 73-79.

BOROS, E. e P. HAMMER (1991), *The max-cut problem and quadratic 0-1 optimization; polyhedral aspects, relaxations and bounds*, Ann. of Ops. Res., vol. 33, pp. 151-180.

BULLNHEIMER, B.; R. F. HARTL; C. STRAUSS (1999), *Applying the ant system to the vehicle routing problem*, in Metaheuristics: Advances and Trends in Local Search for Optimization, editado por S. Voss, S. Martello, I. Osman, C. Roucairol, Kluwer Academic Publishers.

CARTER, M. W. (1984), *The indefinite zero-one quadratic problem*, Disc. Appl. Math., 7, pp. 23-44.

CHARDAIRE, P. e A. SUTTER (1995), *A decomposition method for quadratic zero-one programming*, Manag. Sci.,. 41, pp. 704-712.

CHIPPERFIELD, A. e P. FLEMING (1996), *Parallel genetic algorithms*, in Parallel and Distributed Computing Handbook, ed. A. Zomaya, McGraw-Hill .

CHRISTOFIDES, N. (1985), *Vehicle routing* in The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization, editado por E. Lawler, J. Lenstra, A . Rinnooy Kan, D. Shmoys, John Wiley, New York.

CHRISTOFIDES, N. e S. EILON (1969), *An algorithm for the vehicle dispatching problem*, Op. Res. Q., 20, nº 3, pp. 309-318.

CHRISTOFIDES, N.; A. MINGOZZI; P. TOTH (1979), *The vehicle routing problem*, in Combinatorial Optimization, editado por N. Christofides, A. Mingozzi, P. Toth e C. Sandi, John Wiley.

CLARKE, G. e J. WRIGHT (1964), *Scheduling of vehicles from a central depot to a number of delivery points*, Ops. Res., 12, pp. 568-581.

CORRÊA R.; A. FERREIRA; S. PORTO (1998), *Selected algorithmic techniques for parallel optimization*, in Handbook of Combinatorial Optimization, Vol. 3, editado por D.-Z. Du e P. Pardalos, Kluwer Academic Publishers.

CROES, G. A. (1958), A *method for solving traveling salesman problems*, Ops. Res., 6, pp. 791-812.

DAMBERG, O.; A. MIGDALAS; S. STORØY (1997), *Parallel algorithms for network problems*, in Parallel Computing in Optimization, editado por A. Migdalas, P. Pardalos e S. Storøy, Kluwer Academic Publishers.

DESROCHERS, M.; J. DESROSIERS; M. SOLOMON (1992), *A new optimization algorithm for the vehicle routing problem with time windows*, Ops. Res., 40, pp. 342-354.

DESROSIERS, J.; F. SOUMIS; M. DESROCHERS; M. SAUVÉ (1986), *Vehicle routing and scheduling with time windows*, Math. Prog. Study, 26, pp. 249-251.

DESROSIERS, J.; Y. DUMAS; M. SOLOMON; F. SOUMIS (1995), *Time constrained routing and scheduling* in Network Routing. Handbooks in Operations Research and Management Science, editado por M. Ball, T. Magnanti, C. Monma, G. Nemhauser. North-Holland.

DOWD, K. (1993), *High Performance Computing,* O' Reilly and Ass., Sebastopol, EEUU.

DUNCAN, R. (1990), *A survey of parallel computers,* Computer (IEEE Society), pp. 5-16.

EVANS, S. e J. NORBACK (1985), *The impact of a decision-support system for vehicle routing in a foodservice supply situation*, J.O.R.S., vol. 36, pp. 467-472.

FAUSTINO, A. M. (1992), *Complementaridade Linear e Aplicação em Optimização Global*, Tese de Doutoramento, Universidade de Coimbra.

FAUSTINO, A. M. (1998), *Comunicação Oral.*

FISHER, M. (1994), *Optimal solution of vehicle routing problems using minimum k-trees*, Ops. Res., 42, pp. 626-642.

FISHER, M. e R. JAIKUMAR (1978), *A decomposition algorithm for large-scale vehicle routing*, Work. Pap. 78-11-05, Dep. of Decision Sciences, Univ. Pennsylvania, Philadelphia.

FISHER, M. e R. JAIKUMAR (1981), *A generalized assignment heuristic for vehicle routing*, Networks, 11, pp. 109-124.

FLEMING, P. J. (1997), *Apontamentos de Processamento Paralelo*, Mestrado em Eng. Sistemas e Computação, UCEH, Universidade do Algarve.

FLYN, M. J. (1966), *Very high-speed computing systems,* Proceedings of the IEEE 54, pp. 1901-1909.

FOGEL, L.; A. OWENS; M. WASH (1966), *Artificial Intelligence Through Simulated Evolution*, Wiley.

FONSECA, C. (1995), *Multiobjective Genetic Algorithms with Application to Control Engineering Problems*, Ph.D. Thesis, Dept. of Automatic Control and Systems Engineering, The University of Sheffield.

FONSECA, C. (1998), *Comunicação Oral.*

FÜRER, M. (1997), *Parallel algorithms and complexity*, in Parallel Computing in Optimization, editado por A. Migdalas, P. Pardalos e S. Storøy, Kluwer Academic Publishers.

GALLO, G.; P. L. HAMMER; B. SIMEONE (1980), *Quadratic knapsack problems*, Math. Prog., 12, pp.132-149.

GAMBARDELLA, L.; E. TAILLARD; G. AGAZZI (1999), *MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows*, Technical Report IDSIA-06-99, IDSIA, Lugano, Suiça.

GAREY, M. R. e D. S. JOHNSON (1979), *Computers and Intractability: A Guide to the Theory of NP-Completness*, Freeman, S. Francisco.

GENDREAU, M.; A. HERTZ; G. LAPORTE (1994), *A tabu search heuristic for the vehicle routing problem*, Manag. Sci., 40, pp. 1276-1290.

GENDREAU, M.; G. LAPORTE; J.-Y. POTVIN (1997), *Vehicle routing: modern heuristics*, in Local Search in Combinatorial Optimization, editado por E. Aarts e J. Lenstra, John Wiley & Sons.

GENDRON, B. e T. G. CRAINIC (1994), *Parallel branch-and-bound algorithms: survey and synthesis*, Ops. Res., 42, pp.1042-1066.

GILLET, B. e L. MILLER (1974), *A heuristic algorithm for the vehicle dispatching problem*, Ops. Res., 22, pp. 340-349.

GLOVER, F. (1986), *Future paths for integer programming and links to artificial intelligence*, Comp. Ops. Res., vol. 5, pp. 533-549.

GLOVER, F. (1989), *Tabu search – Part I*, ORSA J. on Computing, 1, pp. 190-206.

GLOVER, F. (1990), *Tabu search – Part II*, ORSA J. on Computing, 2, pp. 4-32.

GLOVER, F. (1995), *Tabu search fundamentals and uses*, Graduate School of Business, University of Colorado.

GLOVER, F. (1996), *Tabu search and adaptative memory programming – advances, applications and challenges*, Graduate School of Business, University of Colorado.

GLOVER, F.; J. KELLY; M. LAGUNA (1995), *Genetic algorithms and tabu search: hibrids for optimization*, Comp. Ops. Res., vol. 22, pp. 111-134.

GLOVER, F.; G. KOCHENBERGER; B. ALIDAEE (1998), *Adaptive memory tabu search for binary quadratic programs*, Manag. Sci., 44, pp .336-345.

GLOVER, F. e M. LAGUNA (1997), Tabu Search, Kluwer Academic Publishers.

GOLDBERG, D. (1989), *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company.

GOLDEN, B.; L. BODIN; T. DOYLE; W. STEWART (1980), *Aproximate travelling salesman algorithms*, Ops. Res., 28, pp. 694-711.

GOLDEN, B.; G. LAPORTE; E. TAILLARD (1997), *An adaptative memory heuristic for a class of vehicle routing problems with minmax objective*, Comp. Ops. Res., 24, pp. 445-452.

GOLDEN, B. e E. WASIL (1987), *Computerized vehicle routing in the soft drink industry*, Ops. Res., 35, pp. 6-17.

GRAHAM, D. e H. NUTTLE (1986), *A comparison of heuristics for a school bus scheduling problem*, Transp. Res. B, vol. 20, pp.175-182.

GULATI, V. P.; S. K. GUPTA; A. K. MITTAL (1984), *Unconstrained quadratic bivalent programming problem*, E.J.O.R., vol. 15, pp. 121-125.

GUSTAFSON, J. L. (1988), *Reevaluating Amdahl's law*, Communications of the ACM 31, pp. 532-533.

HADJICONSTANTINOU E.; N. CHRISTOFIDES; A. MINGOZZI (1995), *A new exact algorithm for the vehicle routing problem based on q-path and k-shortest path relaxations*, Ann. Ops. Res., vol. 61, pp. 21-44.

HAMMER, P. L.; P. HANSEN; B. SIMEONE (1984), *Roof duality, complementation and persistency in quadratic 0-1 optimization*, Math. Prog., 28, pp. 121-155.

HANSEN, P. (1979), *Methods of nonlinear 0-1 programming*, Ann. of Disc. Math., 5, pp. 53-70.

HANSEN, P.; B. JAUMARD; M. RUIZ; J. XIONG (1993), *Global minimization of indefinite quadratic functions subject to box constraints*, Naval Res. Logistics, vol. 40, pp. 373-392.

HOARE, C. A. R. (1978), *Communicating sequential processes*, CACM, 21, pp. 666-667.

HOCKNEY, R. W. e C. R. JESSHOPE (1990), *Parallel Computers*, Adam Higher Ltd. Bristol EEUU.

HOLLAND, J. (1975), *Adaptation in natural and artificial systems*, University of Michigan Press, Ann Arbor.

HOLMQVIST, K.; A. MIGDALAS; P. PARDALOS (1997), *Parallelized heuristics for combinatorial search*, in Parallel Computing in Optimization, editado por A. Migdalas, P. Pardalos e S. Storøy, Kluwer Academic Publishers.

HUI, L. S. (1984), *An improved enumerative algorithm for solving quadratic zero-one programming*, E.J.O.R., vol. 15, pp. 110-120.

HUNTLEY, C. e D. BROWN (1996), *Parallel genetic algorithms with local search*, Comp. Ops. Res., vol. 23, pp. 559-571.

IBARAKI, T. (1987), *Enumerative approaches to combinatorial optimization*, Ann. of Ops. Res., vol. 10-11.

INMOS, Ltd. (1988), *Occam 2 Reference Manual*, editado por C. A. R. Hoare, Prentice Hall.

JANSEN, R. (1987), *A note on superlinear speedup*, Parallel Computing, 4, pp. 211-213.

KELLY, J. e J. XU (1999), *A set partitioning-based heuristic for the vehicle routing problem*, INFORMS J. Computing, vol. 11, pp. 161-172.

KINDERVATER, G. A. P. e H. W. J. M. TRIENEKENS (1988), *Experiments with parallel algorithms for combinatorial problems*, E.J.O.R., vol. 33, pp.65-81.

KIRKPATRICK, S.; C. GELATT, Jr.; M. VECCHI (1983), *Optimization by simulated annealing*, Science, 220, pp. 671-680.

KÖRNER, F. (1983), *An efficient branch and bound algorithm to solve the quadratic integer programming problem*, Computing, vol. 30, pp. 253-261.

KÖRNER, F. (1992), *Remarks on a difficult test problem for quadratic boolean programming*, Optimization, vol. 26, pp. 355-357.

KRARUP, J. e P. A. PRUZAN (1978), *Computer aided layout design*, Math. Prog. Study, 9, pp.75-94.

KUMAR, V. (1994), *Introduction to Parallel Computing*, The Benjamin/Cumming Publishing Company, Inc., Redwood City, EEUU.

LAI, T. H. e S. SAHNI (1984), *Anomalies in parallel branch-and-bound algorithms*, Communications of the ACM 27, pp. 594-602.

LAMBERT, V.; G. LAPORTE; LOUVEAUX (1993), *Designing collection routes through Bank branches*, Comp. Op. Res., vol. 20, pp. 783-791.

LAPORTE , G. (1992), *The vehicle routing problem: An overview of exact and approximate algorithms*, E. J. O. R., vol. 59.

LAPORTE , G. e Y. NOBERT (1987), *Exact algorithms for the vehicle routing problems*, Ann. Disc. Math., 31, pp. 147-184.

LAPORTE , G.; Y. NOBERT; M. DESROCHERS (1985), *Optimal routing under capacity and distance restrictions*, Ops. Res., 33, pp. 1050-1073.

LAPORTE, G. e I. OSMAN (1995), *Routing problems: A bibliography*, Ann. of Ops. Res., vol. 61, pp. 227-238.

LENSTRA, J. e A. RINNOOY KAN (1981), *Complexity of vehicle routing and scheduling problems*, Networks, 11, pp. 221-227.

LI, G. e B. W. WAH (1984), *Computational efficiency of parallel approximate branch-and-bound algorithms*, Report TR-EE 84-6, School of Electrical Engineering, Purdue University, West Lafayette, IN.

LOOTSMA, F. A. e K. M. RAGSDELL (1988), *State-of-the-art in parallel non linear optimization*, Parallel Comp., 6, pp.133-155.

LUZ, C. (1998), *A lower bound for the quadratic 0-1 minimization problem*, Doc. Trabalho, EST Instituto Politécnico de Setúbal.

McBRIDE, R. D. e J. S. YOMARK (1980), *An implicit enumeration algorithm for quadratic integer programming*, Manag. Sci., 26, pp.282-296.

MEGGIDO, N. (1983), *Applying parallel computation algorithms in the design of serial algorithms,* Journal of the ACM 30, pp.852-865.

MICHALEWICZ, Z. (1996), Genetic Algorithms + Data Structures = Evolution Programs. Springer.

MÜHLENBEIN, H. (1997), *Genetic algorithms*, in Local Search in Combinatorial Optimization, editado por E. Aarts e J. Lenstra, John Wiley & Sons.

NYGARD, K.; P. GREENBERG; W. BOLKAN; E. SWENSON (1988), *Generalized assignment methods for the deadline vehicle routing problem*, in Vehicle Routing: Methods and Studies, editado por B. Golden e E. Assad , North-Holland.

OCHI, L.; D. VIANNA; L. DRUMMOND; A. VICTOR (1998), *An evolutionary hybrid metaheuristic for solving the vehicle routing problem with heterogeneous fleet*, Lect. Notes in Comp. Sci., 1391, pp. 187-195.

ORTÍ, E. Q. (1996), *Algoritmos paralelos para resolver equaciones matriciales de Riccati en problemas de control*, Tese de Doutoramento, Universidad Politécnica de Valencia.

OSMAN, I. (1993), *Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem*, Ann. Ops. Res., vol. 41, pp. 421-451.

PANDIT, R. e B. MURALIDHARAN (1995), *A capacitated general routing problem on mixed networks,* Comp. Ops. Res., vol. 22, pp. 465-478.

PARDALOS P. M. (1989), *Performance of parallel branch-and-bound algorithms for quadratic 0-1 programming*, Tech. Rep., Comp. Science Dep., Pennsylvania State University.

PARDALOS P. M.; A. T. PHILLIPS; J. B. ROSEN (1992), *Topics in Parallel Computing in Mathematical Programming,* Science Press, New York.

PARDALOS P. M. e X. LI (1990), *Parallel Branch-and-bound algorithms for combinatorial optimization*, Supercomputer, 7, pp. 23-30.

PARDALOS, P. M. e G. P. RODGERS (1990), *Computational aspects of a branch-and-bound algorithm for quadratic zero-one programming*, Computing, vol. 45, pp. 131-144.

PARDALOS, P. M. e G. P. RODGERS (1992), *A branch-and-bound algorithm for the maximum clique problem*, Comp. Ops. Res., vol. 19, pp. 363-375.

PARKINSON, D. (1986), *Parallel efficiency can be greater than unity*, Parallel Computing, vol. 3, pp. 261-262.

PEKNY, J. F. e D. L. MILLER (1992), *A parallel branch-and-bound algorithm for solving large asymmetric traveling salesman problems*, Math. Prog., 55, pp. 17-33.

PICARD, J. C. e H. D. RATLIFF (1974), *Minimum cuts and related problems*, Networks, 5, pp. 357-370.

PIRES, F. M. (1994), *Problemas Lineares Complementares Monótonos*, Tese de Doutoramento, Universidade do Algarve.

POLI, R. e W. LANGDON (1998), *A review of theoretical and experimental results on schemata in genetic programming*, Lect. Notes in Comp. Sci., 1391.

POLJAK, S. e H. WOLKOWICZ (1993), *Convex relaxations of 0-1 quadratic programming*, DIMACS Tech. Rep. 93-18.

POTVIN, J.-Y. (1996), *Genetic algorithms for the traveling salesman problem*, Ann. Ops. Res., vol. 63, pp. 339-370.

POTVIN, J.-Y. e S. BENGIO (1996), *The vehicle routing problem with time-windows – Part II: genetic search*, INFORMS Journal on Computing, vol. 8, pp. 165-172.

RAGHAVENDRA, A; T. KRISHNAKUMAR; R. MURALIDHAR; D. SARVANAN (1992), *A practical heuristic for a large scale vehicle routing problem*, E.J.O.R., vol. 57, pp. 32-38.

REEVES, C. (1996), *Modern heuristic techniques* in Modern Heuristic Search Methods, editado por V. Rayward-Smith, I. Osman, C. Reeves e G. Smith, John Wiley, New York.

REGO, C. (1994), *Uma heurística tabu para a determinação de rotas de veículos*, Investigação Operacional, vol. 14, nº 2, pp. 207-232.

REGO, C. (1998), *A subpath ejection method for the vehicle routing problem*, Manag. Sci., 44, pp. 1447-1459.

RENAUD, J.; G. LAPORTE; F. BOCTOR (1996), *A tabu search heuristic for the multi-depot vehicle routing problem*, Comp. Ops. Res., vol.23, pp. 229-235.

ROCHAT, Y. e E. TAILLARD (1995), *Probabilistic diversification and intensification in local search for vehicle routing*, J. of Heuristics, vol.1, nº1, pp. 147-167.

ROCHAT, Y.; F. SEMET (1994), *A tabu search approach for delivering pet food and flour in Switzerland*, J.O.R.S., vol. 45, pp. 1233-1246.

RUANO, A. E. (1992), *Applications of Neural Networks to Control Systems*, Ph.D. Thesis, UCNW, UK.

RUSSEL, R. A. (1977), *An effective heuristic for the m-tour travelling salesman problem with some side conditions*, Ops. Res., 25, pp. 517-524.

SCHÜTZ, G. (1989), *Problema de Optimização de Rotas de Veículos – Um Estudo Computacional da Heurística de Fisher e Jaikumar*, Dissertação de Mestrado, Faculdade de Ciências da Universidade de Lisboa.

SCHÜTZ, G. (1990), *Problema de optimização de rotas de veículos – heurísticas de duas fases*, Doc. Trabalho nº 4 – 90 CEMAPRE, ISEG, Universidade Técnica de Lisboa.

SCULLI, D.; K. MOK; S. CHEUNG (1987), *Scheduling vehicles for refuse collection*, J.O.R.S., vol.38, pp.233-239.

SEMET, F. e E. TAILLARD (1993), *Solving real-life vehicle routing problems efficiently using tabu search*, Ann. Ops. Res., vol. 41, pp. 468-488.

SØREVIK, T. (1997), *A programmer's view of parallel computers*, in Parallel Computing in Optimization, editado por A. Migdalas, P. Pardalos e S. Storøy, Kluwer Academic Publishers.

SYSLO, M.; N. DEO; J. KOWALIK (1983), *Discrete Optimization Algorithms with PASCAL Programs*, Prentice-Hall.

TAILLARD, E. (1993), *Parallel iterative search methods for vehicle routing problems*, Networks, 23, pp. 661-673.

TAILLARD, E. (1996), *A heuristic column generation method for the heterogeneous fleet VRP*, Doc. Trabalho CRT-96-03, Universidade de Montreal, Canadá.

TAILLARD, E. (1999), *Ant Systems*, Technical Report IDSIA-05-99, IDSIA, Lugano, Suiça.

TAILLARD, E.; P. BADEAU; M. GENDREAU; F. GUERTIN; J.-Y. POTVIN (1997), *A tabu search heuristic for the vehicle routing problem with soft time windows*, Transp. Science, vol. 31, pp. 170-186.

TAILLARD, E.; L.GAMBARDELLA; M. GENDREAU; J.-Y. POTVIN (1997), *La programmation à mémoire adaptative ou l'évolution des algorithmes évolutifs*, Technical Report IDSIA-79-97, IDSIA, Lugano, Suiça.

TAILLARD, E.; L.GAMBARDELLA; M. GENDREAU; J.-Y. POTVIN (1998), *A unified view of meta-heuristics*, Technical Report IDSIA-19-98, IDSIA, Lugano, Suiça.

TRIENEKENS, H. (1989), *Parallel branch-and-bound and anomalies*, Technical Report EUR-CS-89-01, Erasmus University, Dep. of Computer Sciences, The Netherlands.

VAZ PATO, M. e L. LOURENÇO (1997), *A standard genetic algorithm for clustering with precedence constraints*, Investigação Operacional, vol. 12, nº 2, pp. 145-157.