UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

DESIGNAÇÃO DE TAREFAS EM APLICAÇÕES DE MULTIPROCESSADORES DE PROCESSAMENTO DIGITAL DE SINAL UTILIZANDO ALGORITMOS GENÉTICOS

FABIANA SIMÕES E SILVA

DISSERTAÇÃO DE MESTRADO

UNIVERSIDADE FEDERAL DE SÃO CARLOS
CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE PRODUÇÃO

DESIGNAÇÃO DE TAREFAS EM APLICAÇÕES DE MULTIPROCESSADORES DE PROCESSAMENTO DIGITAL DE SINAL UTILIZANDO ALGORITMOS GENÉTICOS

Fabiana Simões e Silva

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Produção da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Engenharia de Produção.

Orientadora: Profa. Dra. Vitória Pureza

SÃO CARLOS
2003

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

S586dt

Silva, Fabiana Simões e.
Designação de tarefas em aplicações de multiprocessadores de processamento digital de sinal utilizando algoritmos genéticos / Fabiana Simões e Silva. -- São Carlos : UFSCar, 2003.
84 p.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2003.

1. Logística empresarial. 2. Designação de tarefas. 3. Algoritmos genéticos. 4. Processamento de sinais – técnicas digitais. 5. Processamento paralelo. 6. Heurística I. Título.

CDD: 658.5 (20ª)

DEDICATÓRIA

"De tudo, ficaram três coisas:

a certeza de que estamos
sempre começando...
a certeza de que
é preciso continuar...
a certeza de que
seremos interrompidos
antes de terminar...

Portanto, devemos:

fazer da interrupção
um caminho novo...
da queda
um passo de dança...
do medo,
uma escada...
do sonho,
uma ponte...
da procura...
um encontro."

Fernando Pessoa

Aos meus pais Leize e Chicão, com muito amor. A melhor forma de retribuir todo carinho, esforço e apoio que vocês me deram, é mostrar que não foi em vão. Mais do que tudo valeu toda a dedicação de vocês por mim.

AGRADECIMENTOS

A minha orientadora Vitória Pureza, meus agradecimentos, minha admiração e minha confiança. Hoje sei a sorte que tive ao tê-la como minha orientadora e amiga ao longo desses anos.

Aos meus amigos Renato, Andreza, e Fábio, que foram, em muitos momentos da minha vida, a força que eu precisava para continuar.

Aos meus irmãos Luiz Francisco e Camila pela paciência e compreensão, pois mesmo não entendendo nada de algoritmos, eram as primeiras pessoas que estavam presentes quando conseguia bons resultados e saía pulando pela casa. Agradeço também a minha irmã gêmea Juliana, que sempre estava a minha disposição para tirar algumas dúvidas toda vez que eu a interrompia durante seu projeto de mestrado, e também por sempre estar presente na minha vida, nos momentos bons e ruins, me dando força e conselhos.

Agradeço principalmente ao meu tio Claudinei e ao Isaías, que sempre estiveram ao meu lado para me ajudar e ensinar em todas as vezes que precisei.

A minha querida vó Ofélia, que mesmo longe, sei que está rezando e torcendo por mim.

E um agradecimento especial a uma pessoa que não está mais presente neste mundo, mas que foi importante na minha vida e me deu muito apoio para começar esse projeto. Wilson, onde quer que você esteja, sei que torce por mim e essa conquista eu dedico também à você.

Enfim, agradeço a todos que colaboraram direta ou indiretamente na realização deste trabalho.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

RESUMO

O objetivo deste projeto consiste no desenvolvimento de algoritmos genéticos para resolução do problema de designação de tarefas em multiprocessadores de processamento digital de sinal (PDS). Especificamente, busca-se minimizar o atraso total em uma arquitetura de multiprocessadores particular, bastante utilizada em sistemas reais. Neste trabalho são apresentadas implementações de algoritmos genéticos, e os resultados computacionais decorrentes de sua aplicação a um conjunto de 117 exemplos gerados aleatoriamente e extraídos de contextos reais. O desempenho dos algoritmos é analisado, comparando-se a qualidade das soluções e os tempos computacionais requridos com os obtidos por uma heurística competitiva da literatura e por um algoritmo de busca *multiple starts*. Os algoritmos genéticos obtiveram menores valores de atraso em mais de 68% dos exemplos, a um tempo computacional maior.

Palavras-chave: Processamento paralelo, processamento de sinal digital, designação de tarefas, heurísticas, algoritmos genéticos.

ABSTRACT

This work consists in the development of genetic algorithms for the Task-to-Processor Assignment Problem in multiprocessor applications. Specifically, the objective is to find the task-to-processor assignment that minimizes the total delay in a particular multiprocessor digital signal processing architecture. We present a description of our algorithm implementations and the results found with a set of 117 randomly generated and real-life instances. The algorithms performance is compared with the results provided by a competitive dynamic list heuristic and a multiple start search algorithm. The results indicate lower delays in more than 68% of the instances, at a higher computational cost.

Key-words*: Parallel processing, digital signal processing, task assignment, heuristics, genetic algorithms.*

1 SISTEMAS DE PROCESSAMENTO PARALELO

Um sistema paralelo é um ambiente (*software* e/ou *hardware*) que permite que algoritmos sejam projetados de forma que suas diferentes partes sejam processadas por entidades de execução distintas. Ou seja, ao invés do algoritmo ser processado em seqüência, uma instrução por vez, ele será executado por várias entidades simultaneamente. Por exemplo, se tivermos um computador com vários processadores, cada um deles executará uma parte do programa simultaneamente e, conseqüentemente, este poderá ser completado mais rapidamente do que se apenas um processador tivesse que executar o programa inteiro. Além de permitir um processamento mais rápido, sistemas paralelos podem simplificar o projeto dos algoritmos.

O processamento paralelo (inerente a sistemas paralelos) é utilizado para atender os requerimentos computacionais de um grande número de algoritmos (aplicações) que surgiram na última década (HWANG, 1993; KUMAR *et al.*, 1994; QUINN, 1994). Tais aplicações são geralmente caracterizadas por uma necessidade de processamento rápido (muitas vezes em tempo real) de um volume grande de dados, a qual é muitas vezes comprometida em um ambiente de processamento seqüencial.

Um exemplo trivial de aplicação é a situação em que se tem um programa que precisa ler dados de um disco e realizar alguns cálculos de grande porte com estes dados. Uma vez que o disco pode transferir os dados da memória sem intervenção da CPU, faz sentido dividir o programa em duas partes e executá-las paralelamente: uma das partes lê os dados do disco para a memória e a outra parte realiza os cálculos. Isso também poderia ser feito com uma única parte seqüencial que alterna a leitura de blocos de dados com a realização dos cálculos. Entretanto, essa abordagem é difícil de ser implementada além de resultar em um processo menos eficiente (como o programa saberá quando o último bloco de dados foi lido e quando é hora de ler outro bloco?).

Um exemplo interessante de aplicação de processamento paralelo diz respeito ao processo de classificação de imagens meteorológicas na previsão de precipitação de chuvas através de um algoritmo de redes neurais (SILVA, 1999). O ambiente de processamento paralelo utilizado trouxe grandes economias em tempo de processamento nos processos de treinamento, teste e utilização das redes. LAUDON &

LAUDON (1999) reporta o exemplo da refinadora de petróleo Petro-Canadá, cujos sistemas de informação eram executados em diversos computadores departamentais de grande porte, mas que não forneciam à empresa uma visão unificada de suas operações e dados. Assim, primeiramente a Petro-Canadá adotou um *software* que integrava uma ampla faixa de funções comerciais, de modo a permitir o compartilhamento e utilização de dados das áreas de vendas, estoques e produção. Apesar do novo sistema permitir maior facilidade de acesso unificado dos dados comerciais da empresa, isso também implicou que os computadores teriam que processar um enorme volume de dados. Por esta razão, voltou-se para o processamento paralelo para o armazenamento de informações e manipulação de operações.

Em contextos como esses, o paralelismo tem sido apontado como a principal solução para o aumento de desempenho dos computadores e, como resultado, diversas máquinas paralelas baseadas em diferentes arquiteturas têm surgido no mercado. No entanto, para sua efetiva utilização, persistem dificuldades maiores de programação e de gerenciamento do que as encontradas em máquinas seqüenciais. De fato, a utilização do paralelismo resulta em um número de problemas que não são encontrados em um processamento seqüencial tais como projetar um algoritmo paralelo, particioná-lo em tarefas, coordenar a comunicação e sincronização, e programar as tarefas nas máquinas. Não obstante, em virtude dos benefícios decorrentes, um grande esforço de pesquisa em endereçar estas questões tem sido reportado na literatura (AMDAHL, 1967; CHU *et al.*, 1984; GAJSKY & PEIR, 1985; HWANG, 1993; LEWIS & EL-REWINI, 1993; LO *et al.*, 1991; LORD *et al.*, 1993; WU & GAJSKI, 1990; YANG & GERASOULIS, 1992).

Em várias aplicações, sistemas paralelos são projetados para o processamento digital de sinais digitais, ou de forma mais curta, o processamento digital de sinais (PDS). Como o termo sugere, PDS é o processamento de sinais por meios digitais. Historicamente, as origens de processamento de sinal estão na engenharia elétrica; neste contexto, um sinal significa um sinal elétrico levado por uma linha telefônica ou por uma onda de rádio. De forma geral, entretanto, um sinal é uma corrente de informação representando qualquer coisa desde preços de ações da bolsa até dados enviados por um satélite.

Em muitos casos, o sinal está inicialmente na forma de uma voltagem ou corrente elétrica analógica produzida, por exemplo, por um microfone. Em algumas situações, os dados já estão na forma digital – tais como a saída de um CD *player*. Um sinal analógico precisa ser convertido para a forma digital (ou seja, forma numérica) antes que técnicas PDS sejam aplicadas. Um sinal de voltagem analógico, por exemplo, pode ser digitalizado usando um circuito integrado que gera uma saída digital na forma de um número binário, cujo valor representa a voltagem elétrica de entrada.

Sinais geralmente precisam ser processados de várias formas. Por exemplo, o sinal de saída de um equipamento pode estar contaminado com "ruídos" elétricos indesejados. Processar o sinal usando um circuito especializado de forma que se possa remover ou pelo menos reduzir a parte indesejada pode melhorar a qualidade do sinal. Em outros casos, o processamento permite extrair informações importantes do sinal que são usadas em diferentes aplicações de grande importância prática (SMITH, 1997):

- **Telecomunicação**: PDS revolucionou as indústrias de telecomunicações através da geração e detecção de tons de sinal, do deslocamento das bandas de freqüência, da filtragem dos sinais, entre outros.
- **Processamento de áudio**: A representação digital de dados é importante para prevenir a degradação freqüentemente associada à manipulação e armazenamento analógico. Isto é perceptível comparando-se CDs às fitas cassete. Uma vez que tais dados estejam na forma digital é possível a utilização de técnicas de PDS para a filtragem, adição, subtração e edição destes sinais.
- **Localização de eco**: Um método comum para se obter informação sobre um objeto remoto é emitir e receber uma onda do objeto. Por exemplo, um radar opera através da emissão de ondas de rádio e o exame das ondas de eco recebidas. PDS produziu um grande avanço nas áreas que trabalham com as ondas resultantes de objetos ao se utilizar, por exemplo, os sonares. Sonares são divididos em duas categorias: os ativos e os passivos. Em sonares ativos, sinais são emitidos e as ondas de resposta analisadas; em sonares passivos não são emitidas ondas, mas apenas realizada a análise das ondas que chegam até os sensores. PDS possibilitou o uso de geração e compressão de pulso, além da filtragem de sinais detectados.

- **Processamento de imagens**: Imagens são sinais com características especiais. Em primeiro lugar, tais sinais são medidos com parâmetros espaciais, enquanto a maioria dos sinais é medida em função do tempo. Segundo, eles contêm uma grande quantidade de dados que exigem armazenamento físico. Terceiro, o julgamento da qualidade é mais subjetivo que objetivo. Tais características tornaram o processamento de imagens um subgrupo distinto dentro de PDS. O uso das técnicas de PDS em processamento de imagens pode ser observado em:
    - **Área médica** – O uso de PDS é caracterizado por técnicas como a tomografia computadorizada que se utiliza raios X, formando imagens e digitalizando as mesmas que são então armazenadas em sistemas computacionais. Estas informações armazenadas são utilizadas para o cálculo das imagens que correspondem a fatias do corpo humano. Além do uso de PDS em tomografia computadorizada, existem outras aplicações, tais como a ressonância magnética que se utiliza ondas de rádio e campo magnético para a detecção de imagens do corpo humano que respondam a diferentes ondas. A ressonância magnética não pode ser aplicada sem o uso das técnicas de PDS.

    - **Área comercial** – Sistemas comerciais necessitam ser sobretudo baratos, e PDS traz vantagens quanto à compressão de dados que apresentam informações redundantes. A compressão pode ser aplicada a sistemas de televisão ou filmes onde existe redundância entre os quadros apresentados. Exemplos práticos do uso de PDS são os videofones, programas de computador que apresentam filmes, televisão digital, entre outros.

Como resultado de sua grande aplicabilidade, desde 1988, o mercado de processadores digitais de sinal tem crescido em mais de 40% por ano. Fontes da empresa Texas Instruments estimam que o crescimento do mercado de processamento digital de sinal juntamente com o processamento de sinal misto e equipamentos analógicos relacionados chegará a 50 bilhões de dólares nos próximos 4 anos (FIGURA 1.1).

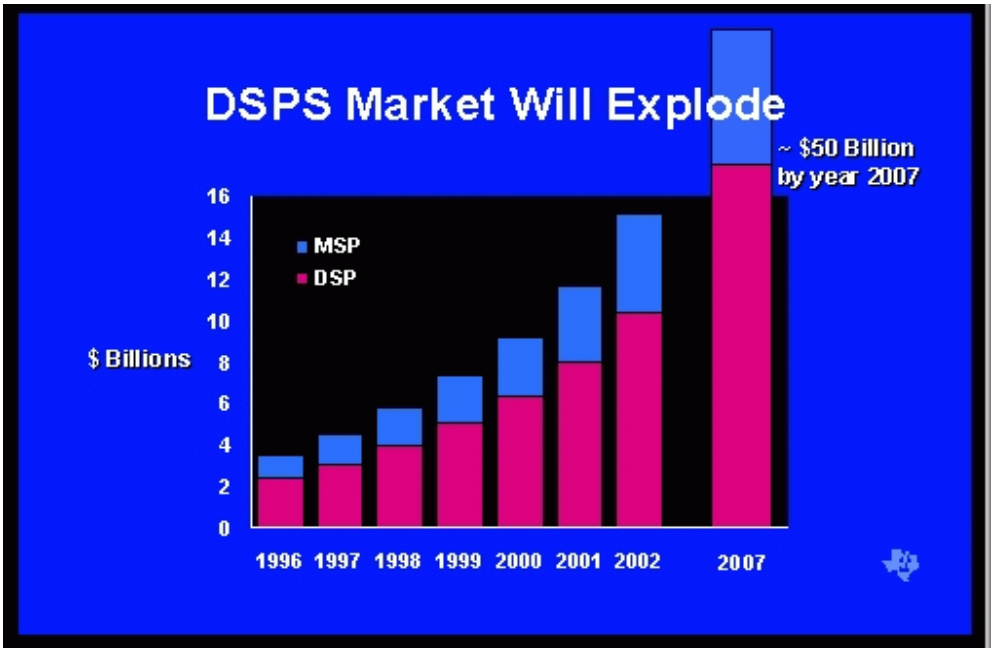


FIGURA 1.1 - Mercado de processamento digital de sinais.
(fonte: http://www.ti.com/corp/docs/investor/speeches/steph97/stephens97.shtml)

Este trabalho aborda o problema de designação de tarefas em um sistema de multiprocessadores PDS. Conforme visto anteriormente, a designação de tarefas a processadores é uma questão que surgiu com o processamento paralelo. Sua importância é crucial para o desempenho do sistema, uma vez que se inapropriada, pode eliminar os ganhos decorrentes da paralelização e impedir a plena exploração do potencial do sistema. O problema de designação de tarefas a multiprocessadores é combinatório, e assim como a grande maioria dos problemas de designação de tarefas, pertence à classe NP-completo. Desta forma, a resolução ótima de sistemas a partir de um certo número de tarefas torna-se inviável, o que exige a utilização de métodos heurísticos. Métodos heurísticos não garantem a otimalidade da solução, porém permitem a obtenção de soluções sub-ótimas de alta qualidade. Em particular, este trabalho aborda a utilização de algoritmos genéticos. Algoritmos genéticos são uma ferramenta heurística poderosa caracterizada por diversas técnicas de busca baseadas nos princípios darwinianos de evolução. Eles vêm sendo aplicados com sucesso em

uma série de problemas combinatórios tais como projetos de circuitos e o problema do caixeiro viajante.

O texto da dissertação é estruturado como se segue. No capítulo 2 é apresentada uma descrição do problema de designação de tarefas na arquitetura de multiprocessadores PDS selecionada. No capítulo 3 são apresentados os fundamentos básicos dos algoritmos genéticos. No capítulo 4 são descritas as implementações em algoritmos genéticos e os resultados computacionais decorrentes. No capítulo 5 são apresentadas as conclusões e as perspectivas do trabalho realizado.

2 O PROBLEMA DE DESIGNAÇÃO DE TAREFAS EM APLICAÇÕES DE MULTIPROCESSADORES DIGITAIS DE SINAL

Em ambientes paralelos, o algoritmo é fragmentado em tarefas, as quais são designadas para execução em processadores ou unidades de CPU distintas. A "melhor" designação das tarefas depende da aplicação. Em particular, este trabalho visa aplicações PDS em que se almeja o menor atraso total. O atraso total é definido como o tempo decorrido entre o surgimento do sinal na porta de entrada e a produção do sinal processado na porta de saída. Sua minimização é particularmente importante em sistemas PDS que precisam responder rapidamente aos sinais que chegam.

O cálculo do atraso total requer a escolha de uma arquitetura de multiprocessadores caracterizada, dentre vários aspectos, pela conectividade entre os processadores. Um exemplo de uma arquitetura com três processadores (CORRÊA *et al.*, 1999) é mostrada na FIGURA 2.1.

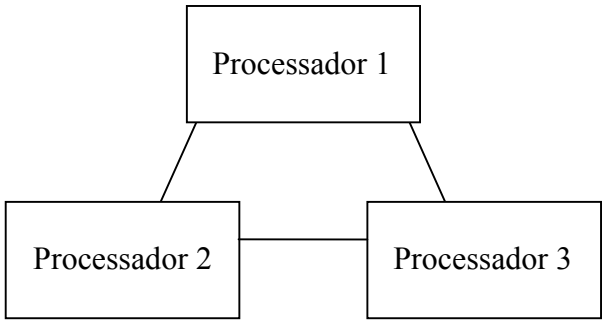


FIGURA 2.1 – Arquitetura com três processadores.
(fonte: CORREA *et al*., 1999)

No sistema da FIGURA 2.1, os processadores estão conectados por uma rede de comunicação completa, isto é, todos os processadores estão conectados entre si. Neste trabalho é abordada uma arquitetura de propósito geral, utilizada em muitas aplicações PDS (GOUBRAN *et al.*, 1991), cujas características são apresentadas a seguir.

2.1 Características da arquitetura de multiprocessadores

A arquitetura, ilustrada na FIGURA 2.2, consiste de um processador mestre (processador 0) e N processadores em linha cujo número é definido de acordo com a necessidade (a princípio, o número de processadores é ilimitado). As setas indicam o fluxo de dados entre os processadores.

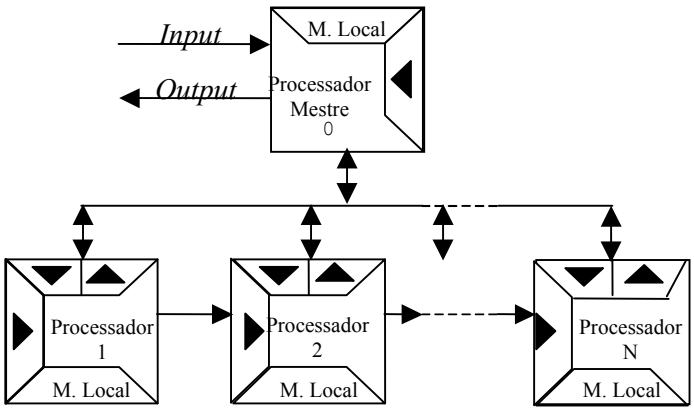


FIGURA 2.2 – Arquitetura utilizada.
(fonte: Adaptado de CHINNECK *et al.,* 2003)

A memória compartilhada é o veículo de comunicação entre os processadores. Os processadores em linha se comunicam escrevendo na memória do próximo processador em linha. Nesse caso, a memória compartilhada é "*write-only*" para o processador em linha que precede um processador *i* e "*read-only*" para o processador em linha que sucede o processador *i*. De maneira similar, o processador mestre usa a memória nos processadores em linha; entretanto, a transferência de informação ocorre em ambas as direções. Ou seja, o processador mestre pode enviar/receber dados de/para todos os processadores em linha.

Nesta arquitetura, o processador mestre é a porta de entrada e saída do sinal e apenas ele está conectado diretamente aos demais processadores, enquanto os processadores em linha estão conectados diretamente com os processadores adjacentes. Exceto pelos aspectos relativos à conectividade, todos os processadores na arquitetura são idênticos. Processadores típicos são os encontrados nas famílias TMS-320 (Texas Instruments), ADSP-21000 (Analog Devices) ou DSP-56000 (Motorola).

A memória limitada em cada processador é configurada em 4 seções: uma para a memória local, duas para a memória dividida com o mestre (uma para dados

enviados para o mestre e uma para os dados lidos do mestre) e uma para memória dividida com seu predecessor. Muitos processadores PDS estão baseados na arquitetura Harvard onde a memória local pode também ser dividida em memória de programa e memória dos dados.

Há um limite na memória total do programa e um limite separado no total de memória dos dados. A soma da memória de programa de todas as tarefas designadas para um dado processador não deve exceder a memória total de programa disponível para o processador.

Os relógios internos dos processadores são coordenados por um relógio de sincronização que possui uma velocidade pré-definida, geralmente correspondente a maior taxa de amostragem para os sinais que chegam. A comunicação interprocessador é também coordenada pelo relógio de sincronização: cada ciclo de relógio é dividido em um subciclo de leitura e um subciclo de escrita. Durante o curto subciclo de leitura, cada processador copia sua memória de entrada compartilhada na memória local, mas não tem permissão para escrever para sua memória de saída compartilhada. Durante o longo subciclo de escrita (quando a maior parte do processamento acontece), os processadores podem modificar sua memória de saída compartilhada. Isso significa que todo o processamento precisa ser realizado em um ciclo de relógio.

2.2 Diagrama de tarefas

A fragmentação de um algoritmo em um sistema de processamento paralelo resulta no chamado diagrama de tarefas (FIGURA 2.3). Um diagrama de tarefas é um grafo direcionado acíclico onde cada tarefa é um nó e os arcos denotam o fluxo de dados entre as tarefas.

Os arcos do grafo representam implicitamente as relações de precedência entre as tarefas do diagrama. Se uma tarefa *x* é predecessora de uma tarefa *y*, então *y* só pode ser executada após a execução de *x*. No exemplo da FIGURA 2.3, a tarefa c1 é predecessora direta de d1 e sucessora direta de b1. A tarefa a tem como sucessoras diretas as tarefas b1 e b2, mas não possui, por sua vez, tarefa predecessora. Para efeitos de resolução do problema de designação, considera-se a tarefa hipotética *input* como tarefa predecessora de a. De maneira similar, a tarefa h da FIGURA 2.3 tem como predecessora a tarefa g e como sucessora a tarefa hipotética *output*.

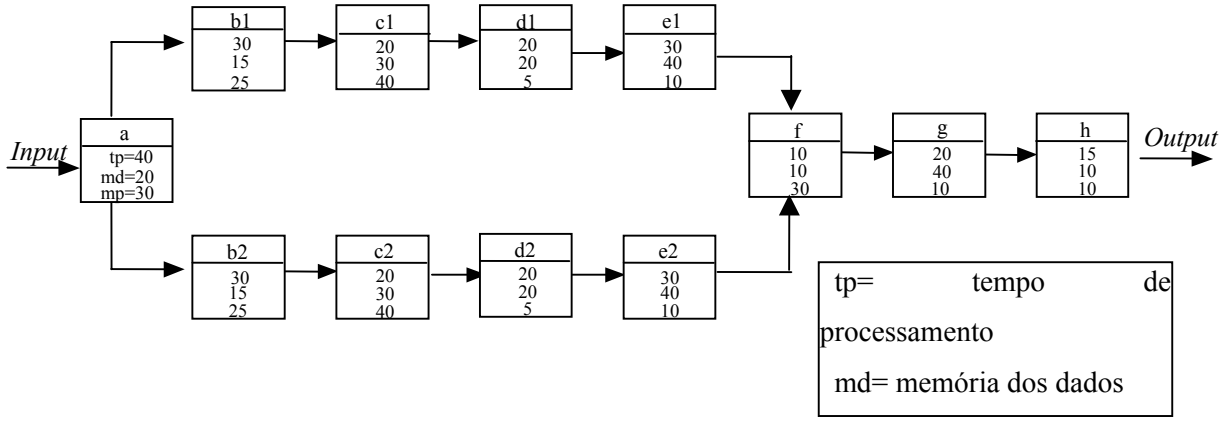


FIGURA 2.3 - Diagrama de tarefas.

Três parâmetros estão associados a cada tarefa: tempo de processamento, memória de dados e memória de programa. Como os processadores da arquitetura são idênticos, estes três parâmetros são mostrados nesta ordem como uma porcentagem do total disponível em cada processador dentro dos blocos de tarefas na FIGURA 2.3. A identificação das tarefas é dada no topo de cada bloco. Por exemplo, a tarefa f tem como parâmetros 10, 10, 30, o que significa que a tarefa f utiliza 10% de tempo de processamento, 10% de memória dos dados e 30% de memória de programa do total disponível do processador para o qual esta tarefa foi designada. A designação das tarefas de um dado diagrama deve ser tal que o tempo de processamento, memória de dados e memória de programa de cada processador não sejam excedidos.

Considere a designação de tarefas do diagrama da FIGURA 2.3. Note na TABELA 2.1 que a somatória dos tempos de processamento, memória dos dados e memória de programa das tarefas designadas ao processador 0 (a, f, g e h) totalizam respectivamente, 85%, 80% e 80% da capacidade deste processador. Como tais porcentagens individuais não excedem a capacidade total do processador (100% para cada um dos parâmetros), tal designação é factível. O mesmo se verifica para os demais processadores.

TABELA 2.1 – Exemplo de designação factível de tarefas.

Processador	0	1	2	3
Tarefas e requerimentos associados	a = 40 ,20, 30 f = 10, 10, 30 g = 20, 40, 10 h = 15, 10, 10	b1 = 30 ,15 ,25 b2 = 30 ,15, 25 c1 = 20, 30, 40	c2 = 20, 30, 40 d2 = 20 ,20, 5 e2 = 30, 40, 10	d1 = 20 ,20, 5 e1= 30, 40, 10
Utilização (Σ)	85, 80, 80	80, 60, 90	70, 90, 55	50, 60, 15

2.3 Cálculo do atraso

O atraso total depende do tempo requerido para cada tarefa, da relação de precedência das tarefas, do relógio do sistema, e do tempo de comunicação entre os processadores. Enquanto os dois primeiros fatores são especificados no diagrama de tarefas e a taxa do relógio é usualmente relacionada com a entrada de sinal esperado, os tempos de comunicação interprocessadores são mais difíceis de prever. Eles dependem da arquitetura, do método de comunicação entre processadores, e de uma designação factível de tarefas para os processadores.

Uma vantagem particular da arquitetura empregada é que os tempos de comunicação interprocessadores são limitados. Seja *i* uma tarefa do diagrama e *j* sua sucessora direta. Então o tempo de comunicação entre *i* e *j* é igual a:

- 0 ciclo : se processador[*j*]=processador[*i*].
- 1 ciclo: se processador[*j*]=processador[*i*] + 1 (há comunicação direta entre os processadores) ou processador[*j*]=0 e processador[*i*]≠0 ou processador[*j*]≠0 e processador[*i*]=0 (uma das tarefas está no processador mestre e a outra em um processador em linha).
- 2 ciclos: as demais situações, onde não há comunicação direta entre os processadores.

Como exemplo de cálculo de tempos de comunicação, considere na FIGURA 2.4, uma designação parcial do diagrama de tarefas da FIGURA 2.3. A tarefa a foi designada ao processador 0, as tarefas b1, b2 e c2 ao processador 1 e a tarefa d2 ao processador 3 (os números no topo das caixas indicam o processador da tarefa). A tarefa

a não tem tarefas predecessoras e tem as tarefas b1 e b2 como sucessoras. Como processador[a]=0 e processador[b1] $\neq$ 0, então o tempo de comunicação entre a tarefa a e a tarefa b1 é 1 ciclo. De igual forma, o tempo de comunicação entre a tarefa a e a tarefa b2 é 1 ciclo. A tarefa sucessora de b2 é a tarefa c2, e como ambas foram designadas ao mesmo processador (processador 1), o tempo de comunicação entre elas é de 0 ciclos. A tarefa c2 tem como sucessora a tarefa d2. Como não há comunicação direta entre os processadores a que c2 e d2 foram designadas (processador[c2]=1 e processador[d2]=3), o tempo de comunicação entre elas é de 2 ciclos.

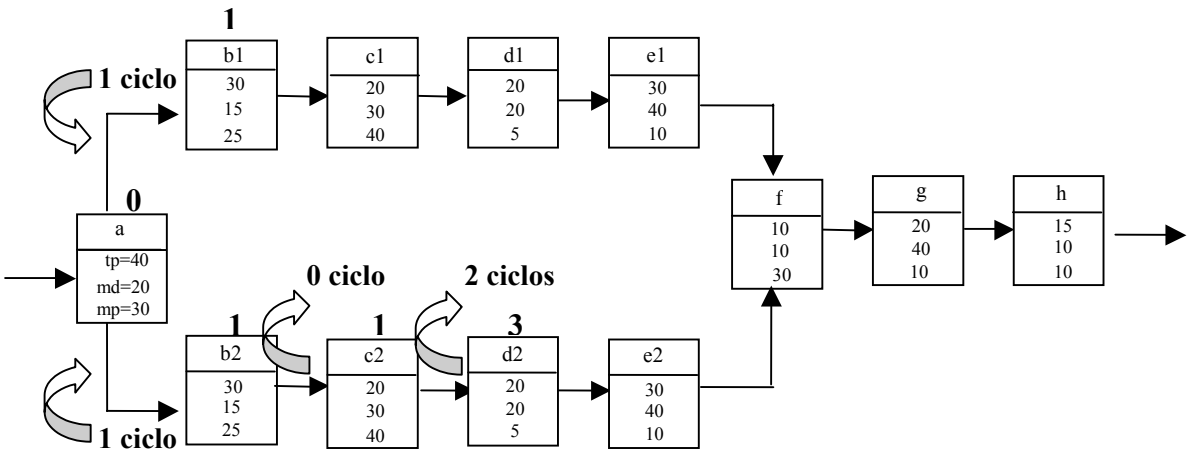


FIGURA 2.4 – Designação parcial para o exemplo de cálculo de tempos de comunicação.

Sem atrasos de relógio ou de comunicação interprocessador, encontrar o atraso total mínimo reduz-se a obter o comprimento do **caminho crítico** do diagrama de tarefas. Quando o relógio é adicionado, atrasos surgem quando os processadores devem estar sincronizados no fim do ciclo de relógio antes que a comunicação interprocessador possa ocorrer. Por causa do relógio, o tempo de comunicação interprocessadores depende da designação das tarefas nos processadores. Por esta razão, somente quando é definida a designação de tarefas aos processadores tal que os limites dos 3 parâmetros sejam respeitados, é que se torna possível calcular o atraso total.

No diagrama da FIGURA 2.3, a comunicação entre processadores ocorre entre os seguintes pares de processadores: 0 a 1 (a tarefa a envia para as tarefas b1 e b2); 1 a 2 (a tarefa b1 envia para a tarefa c1); 1 a 3 (a tarefa c2 envia para a tarefa d2); 2 a 0 (a tarefa e1 envia para a tarefa f); 3 a 0 (a tarefa e2 envia para a tarefa f). As tarefas designadas ao processador 0 requerem um total de 85 unidades de tempo, assim o atraso de comunicação entre a tarefa a e a tarefa b1 é de 15 unidades de tempo. Esta é a espera até o fim do ciclo de relógio uma vez que existe uma ligação direta entre o processador 0 (onde a tarefa a reside) e o processador 1 (onde a tarefa b1 reside). O atraso de comunicação entre a tarefa c2 (processador 1) e a tarefa d2 (processador 3) é ainda maior: primeiro, há a espera de 20 unidades de tempo até o fim do ciclo no processador 1; então são necessárias mais 100 unidades de tempo (um ciclo de relógio) para a transmissão de dados para o processador 3 através do processador 0. A FIGURA 2.5 ilustra o cálculo do atraso total (5 ciclos) para esta designação. Os números no topo das caixas indicam o processador da tarefa e os números em negrito sobre as setas indicam o atraso acumulado em número de ciclos.

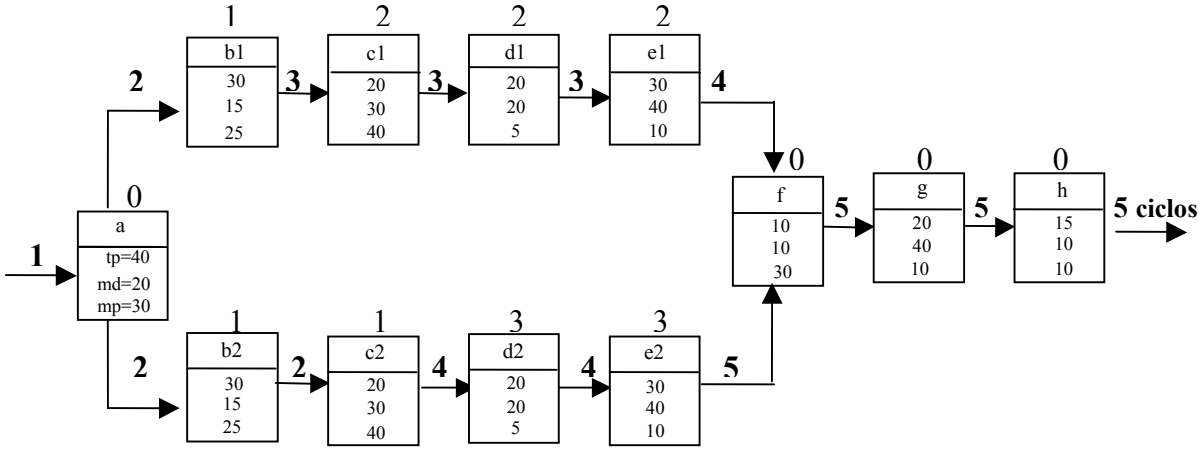


FIGURA 2.5 – Cálculo do atraso total.

2.4 Trabalhos relacionados

De acordo com KWOK & AHMAD (1999), problemas de programação (*scheduling*) podem se apresentar em duas formas: estática e dinâmica. Na programação estática, características do programa como tempo de processamento, comunicação, entre outros, são conhecidas antes da execução do programa. Na programação dinâmica, por

outro lado, poucas hipóteses podem ser feitas antes da execução. Em particular, este trabalho trata de um problema de programação estática. KWOK & AHMAD (1999) sugerem uma taxonomia parcial do problema de programação estática em ambientes de multiprocessadores (FIGURA 2.6).

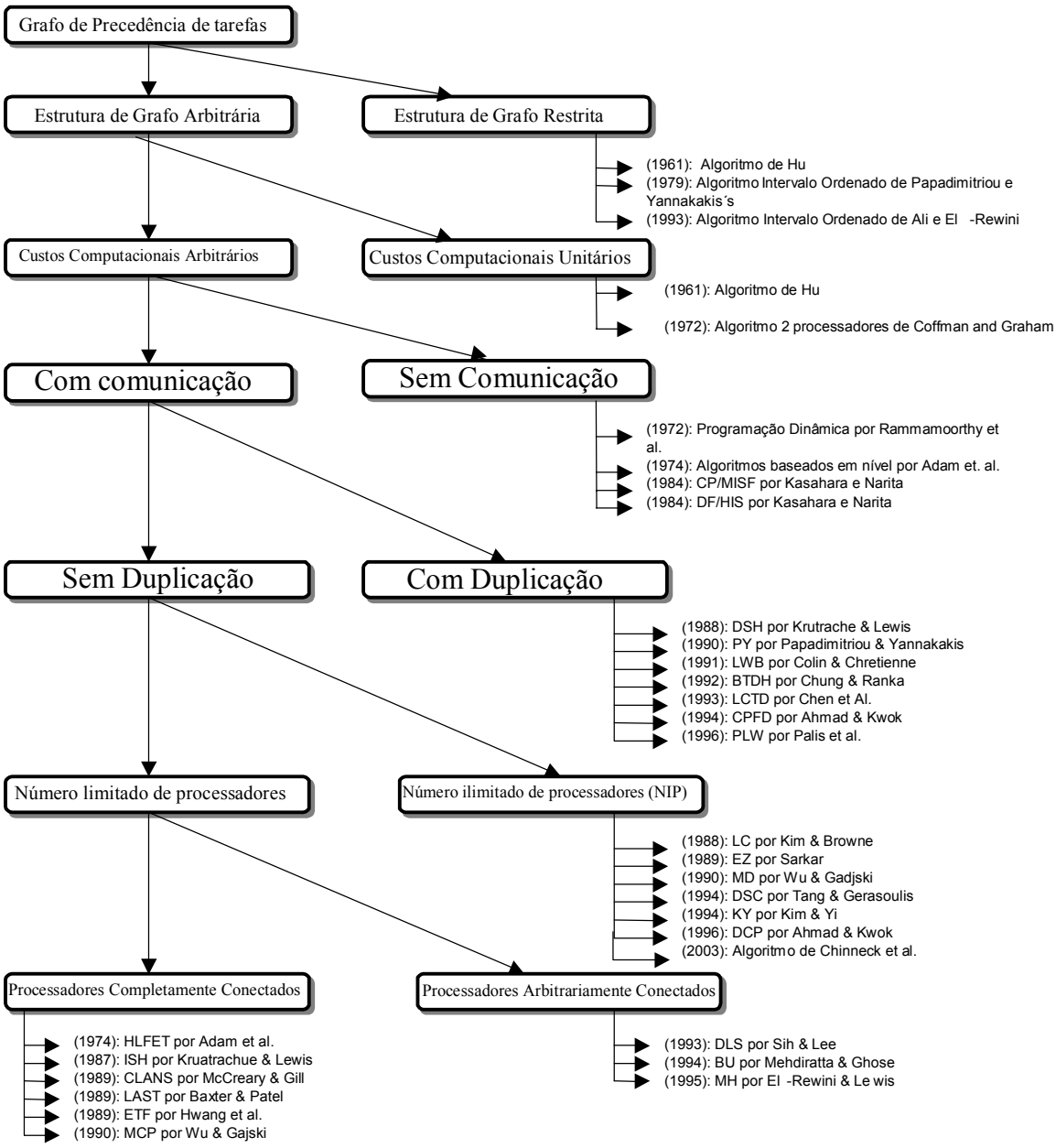


FIGURA 2.6 – Taxonomia parcial do problema de programação estática em sistemas paralelos.

(fonte: Adaptado de KWOK & AHMAD , 1999)

O problema do mínimo atraso com tempos de comunicação interprocessador descrito neste trabalho se enquadra quase perfeitamente na classe NIP (número ilimitado de processadores) desta classificação, pois possui as seguintes características: estrutura de grafo arbitrária, custos computacionais arbitrários, com custos computacionais, não duplicação e um número ilimitado de processadores. Entretanto, a relevância da rota de comunicação, uma de suas mais importantes características, não é considerada na classificação de KWOK & AHMAD (1999).

A solução para o problema do mínimo atraso depende dos tempos de comunicação intertarefas, por sua vez, resultado das designações das tarefas aos processadores. Minimizar o atraso total difere do problema de designação de tarefas em processadores discutidas por LEE & MESSERSCHMITT (1987) e LEE *et al.*(1989). Enquanto a abordagem de fluxo de dados sincronizado é a mesma, LEE & MESSERSCHMITT (1987) ignoram o tempo de comunicação interprocessadores, de maneira que seu problema é idêntico o problema de balanceamento de linha de montagem, por sua vez NP-completo.

A literatura reporta várias proposições de algoritmos para resolução de problemas similares ao problema de designação de tarefas com vistas à minimização do atraso total. NORMAN & THANISCH (1993) descrevem um modelo geral e apresentam uma compilação de vários algoritmos e resultados associados; em todos os casos os algoritmos têm complexidade exponencial ou requerem restrições adicionais que simplificam o domínio do problema a fim de obter complexidade polinomial. Por exemplo, CHOUDHARY *et al.*(1994) descrevem um algoritmo de tempo polinomial para diagramas de tarefas cuja topologia é estritamente serial-paralela para um número fixo de processadores. BILLIONNET (1994) propõem um algoritmo ótimo de tempo polinomial para grafos com topologias em árvore e custos de comunicação uniformes.

Alguns pesquisadores utilizam abordagens de corte mínimo que incluem tempos de comunicação entre processadores, mas que não podem representar relações de precedência. Programação inteira 0-1 tem sido utilizada, mas também apresenta problemas com a representação de relações de precedência. A literatura reporta a proposição de heurísticas de agrupamento, porém não há garantia que a solução ótima

seja descartada. HU (1961) sugere uma boa heurística para grafos do tipo árvore, mas não considera tempo de comunicação interprocessadores.

Heurísticas que constróem listas de programação (*list scheduling heuristics*) fornecem a ordem com a qual as tarefas devem ser designadas a processadores. Novamente, tempos de comunicação interprocessador não são considerados. KASAHARA & NARITA (1984) descrevem uma interessante combinação de lista de programação e *branch & bound* com percorrimento em profundidade, cuja aplicação está limitada a problemas de pequeno porte. JAIN & RAJARAMAN (1994) descrevem um método para calcular os limitantes inferior e mínimo superior de programações ótimas de multiprocessadores usando um modelo no qual o número de processadores é conhecido e não existem atrasos de comunicação.

Em contextos de gestão de operações, em particular de programação da produção, existem problemas com algumas características do problema tratado neste trabalho. Conforme mencionado anteriormente nesta seção, o problema de minimização do atraso sem tempos de comunicação entre processadores é idêntico ao problema de balanceamento de linha de montagem. O problema de balanceamento de linha tem como objetivo definir o número mínimo de estações de trabalho atribuídas a elementos de trabalho com alguma relação de precedência. Um outro possível objetivo é o de minimizar o tempo de ciclo, isto é, quantidade máxima de tempo de processamento em cada estação, para um dado conjunto de estações de trabalho (BELLMAN *et al.,* 1982).

Existem também problemas de programação de operações que envolvem a decisão de seqüenciamento de tarefas. O *Job Shop* tradicional, por exemplo, é caracterizado por diferentes fluxos das ordens de produção entre as máquinas e diferentes números de operações por ordem, as quais são processadas apenas uma vez em cada máquina. Em tais ambientes, uma grande variedade de produtos (cada qual com sua seqüência de processamento) torna o roteamento das operações bastante complexo.

Problemas de planejamento, programação e controle de projetos têm sido objeto de interesse desde o advento das técnicas PERT (*Program Evaluation and Review Technique* – Programa de Avaliação e Técnica de Revisão) e CPM (*Critical Path Method* – Método do Caminho Crítico). A questão básica desta área é encontrar uma programação viável tal que algum objetivo da administração seja otimizado. O

problema surge quando as quantidades de recursos solicitados pelas atividades são limitadas. Por esta razão, a realização das atividades resulta em um incremento da duração do projeto além daquela determinada sem a consideração da limitação de recursos. O objetivo é tomar as decisões de seqüenciamento, sujeitas a restrições de recursos e precedência, de modo a minimizar o incremento da duração. (BELL & HAN, 1991).

A técnica CPM é utilizada para calcular o caminho crítico de uma dada rede de projeto. O caminho crítico é o caminho de atividades (arcos) através de eventos (nós) cuja somatória dos tempos condiciona a duração do projeto. Por meio do caminho crítico obtém-se a duração total do projeto e a folga das tarefas que não controlam o término do projeto. As tarefas pertencentes ao caminho crítico merecem, portanto uma maior atenção por parte dos gestores.

Como exemplo de cálculo do caminho crítico, suponha que um torno apresenta defeitos na bomba de lubrificação que precisam ser reparados. O projeto, portanto, consiste na realização das atividades de reparação do equipamento. Primeiramente, listam-se as atividades, as dependências entre as atividades e os tempos de realização das atividades em uma seqüência lógica (TABELA 2.2):

TABELA 2.2 – Exemplo de projeto.

ATIVIDADES	DESCRIÇÃO	ATIVIDADES PREDECESSORAS	TEMPO PARA REALIZAÇÃO
A	Retirar placa e proteções.	-	1 hora
B	Esgotar óleo e fazer banho químico	A	3 horas
C	Lavar cabeçote.	A	2 horas
D	Trocar rolamentos.	B	3 horas
E	Trocar reparo da bomba de lubrificação.	B e C	2 horas
F	Montar, abastecer e testar o conjunto.	D e E	4 horas

Com esta tabela constrói-se a rede da FIGURA 2.7 (as setas expressam as relações de precedência entre as atividades).

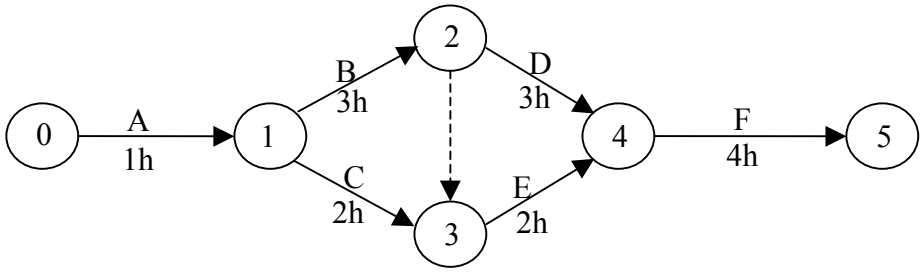


FIGURA 2.7 – Exemplo de rede de projeto.

Na rede da FIGURA 2.7 há três caminhos de atividades entre o evento 0 e o evento 5. Especificamente:

- A-B-D-F, com duração de 11 horas.
- A-C-E-F, com duração de 9 horas.
- A-B-atividade fantasma-E-F, com duração de 10 horas.

Há, pois, um caminho com duração superior ao demais (A-B-D-F) que condiciona a duração do projeto (11 horas). Obtém-se com isso, o caminho crítico e tempo mínimo de finalização do projeto. Note, entretanto, que a consideração de uma única estimativa para a duração das atividades é muitas vezes irrealista. Neste caso, adota-se a abordagem probabilística PERT. A programação e o controle são feitos da mesma maneira que em CPM. Entretanto, ao invés de obter uma data de término do projeto, é especificada uma data esperada de finalização do projeto ou a probabilidade de que o projeto seja completado até uma certa data.

Como pode ser observado, o problema de programação de projetos guarda similaridades com o problema de designação de tarefas tratado neste trabalho. A principal diferença é que para este último, os custos dos arcos não são conhecidos previamente, pois dependem da designação das tarefas aos processadores, o que indica sua maior complexidade e dificuldade de resolução.

2.5 A heurística de CHINNECK *et al* (2003)

A versão do problema de atraso mínimo na arquitetura da FIGURA 2.2 foi inicialmente tratada por CHINNECK *et al*. (2003). Os autores propõem uma heurística que constrói uma lista de programação dinâmica de tarefas (KWOK & AHMAD, 1999). A idéia geral é a de ordenar a lista de tarefas não designadas de forma que as tarefas no caminho crítico sejam designadas primeiro, e em ordem, o que tanto explora as vantagens dos processadores em linha da arquitetura como as vantagens das conexões do processador mestre. A heurística baseia-se em três princípios centrais:

1. Como as designações afetam os atrasos de comunicação entre tarefas, o caminho crítico não pode ser previsto com exatidão. Portanto, quando o grafo de tarefas é examinado, o conjunto de caminhos quase críticos (Sncp) precisa ser considerado juntamente com o caminho crítico corrente estimado (CP) pois um caminho quase-crítico pode se tornar crítico conforme as designações são feitas. Um caminho quase crítico tem um comprimento menor que CP por um valor igual ou menor que um parâmetro de tolerância (CPtol). Por definição, o conjunto de caminhos críticos (Scp) é a união do conjunto dos caminhos quase críticos e CP. Ou seja, Scp={CP} $\cup$ Sncp.

2. Como o caminho crítico real não é conhecido ao longo da efetuação das designações, a escolha do caminho do conjunto Scp precisa ser continuamente atualizada de forma que as tarefas do caminho crítico mais provável sejam designadas primeiro.

3. O processador mestre é, geralmente, a melhor escolha para: (1) tarefas de Scp que estão próximas de *input* e *output* e; (2) tarefas que ocorrem no maior número de elementos de Scp, ou seja, as tarefas no maior número de caminhos críticos ou quase críticos.

A heurística é composta de duas fases:

Fase 1: Tarefas são designadas ao processador mestre usando o princípio 3, de forma que as tarefas mais próximas a *input* e *output* são selecionadas, com prioridade para aquelas que se encontram no maior número de caminhos críticos ou quase críticos (*input* e *output* são consideradas tarefas do grafo com requerimentos iguais a 0, 0, 0 e pertencem a Scp).

Fase 2: Começando de *input*, tarefas ao longo do caminho crítico estimado são designadas aos processadores da linha. A ordenação da lista é adaptativa e usa os princípios 1 e 2 para reavaliar a ordem da lista depois de cada designação. Quando *output* é alcançado, o algoritmo é repetido começando de *input* e considerando os caminhos que se seguem da próxima tarefa que permanece não designada. Desta maneira, as tarefas no "próximo caminho mais crítico" e no "próximo próximo caminho mais crítico", etc., são designadas.

Um exemplo numérico da aplicação deste algoritmo pode ser encontrado em CHINNECK *et al* (2003).

Experimentos computacionais reportados pelos autores contemplam um conjunto de 34 diagramas (5 a 96 tarefas) gerados aleatoriamente e extraídos de contextos reais. Os autores comparam as soluções da heurística com as obtidas por um algoritmo *branch&bound* (problemas pequenos) e com um testador de existência de uma designação factível com atraso menor que algum limitante especificado (problemas maiores). Os resultados indicam que a heurística provê soluções de grande qualidade. Por ser bastante rápida, é particularmente adequada para aplicações que exigem soluções em tempo real.

Apesar de utilizar um grande volume de conhecimento do problema, a heurística de CHINNECK *et al.* (2003) é uma heurística que constrói deterministicamente uma única solução. Portanto, é interessante investigar a qualidade das soluções decorrentes da aplicação de métodos iterativos como buscas locais e meta-heurísticas, uma vez que estes permitem a melhoria das soluções geradas por heurísticas construtivas. Apesar dos métodos iterativos geralmente requererem um tempo computacional superior ao das heurísticas construtivas, reduções adicionais de atraso poderiam ser obtidas em aplicações PDS onde o *download* do algoritmo no sistema de multiprocessadores é realizado uma única vez. Nestes casos, a obtenção da designação não precisa, necessariamente, ser realizada em tempo real.

Com este objetivo, este trabalho visa investigar a utilização de algoritmos genéticos para resolução do problema do atraso mínimo sob as mesmas condições e ambiente de multiprocessadores utilizado por CHINNECK *et al* (2003). No que diz

respeito a esta classe de algoritmos, a literatura aponta vários trabalhos que tratam o problema de minimização do tempo total de execução, sem atrasos de comunicação (veja, por exemplo, CORRÊA *et al.*(1999); ZOMAYA *et al.*(1999); HOU *et al.*(1994); AHMAD & DHODHI(1996)). A maioria dos algoritmos reportados envolve a aplicação de operações de cruzamento e mutação em uma heurística de lista de programação. Como se trata do problema de balanceamento de linha, as listas de programação precisam satisfazer as relações de precedência entre tarefas e os operadores genéticos são projetados de forma a manter esta propriedade. Dependendo da estratégia de elaboração da lista de programação, pode haver uma considerável redução do espaço de soluções possíveis de serem geradas e, neste caso, os algoritmos não têm a garantia de produzir uma solução ótima com probabilidade maior que 0 (CORRÊA *et. al*, 1999). A seguir são apresentados os fundamentos básicos dos algoritmos genéticos.

3 ALGORITMOS GENÉTICOS

Como o próprio nome diz, o funcionamento dos Algoritmos Genéticos tem extrema vinculação com os princípios de evolução e hereditariedade. Nesta área da biologia, pesquisadores como Charles Darwin evidenciaram o fato de que as espécies efetivamente se modificam e que nem todos os organismos que nascem estão destinados à sobrevivência ou reprodução. Os indivíduos com mais chance de sobreviver seriam os que possuem características mais apropriadas para enfrentar as condições ambientais. Um dos principais pontos dos estudos de Darwin foi o aspecto das variações apresentadas entre indivíduos da mesma espécie. Segundo ele, todas as novas espécies são produzidas por meio de uma seleção natural destes indivíduos; ao longo das gerações, as espécies podem se diversificar, mantendo atributos selecionados que as tornam mais adaptadas ao ambiente em que vivem.

De acordo com MICHALEWICZ (1996), durante os últimos 30 anos têm crescido o interesse no tratamento de certas classes de problemas através de sistemas baseados nos princípios de evolução. Tais sistemas compõem a chamada Computação Evolutiva (CE), um importante ramo da ciência da computação. A Computação Evolutiva surgiu no final dos anos 60, quando John Holland começou a estudar a possibilidade de incorporar os mecanismos naturais de seleção e sobrevivência na resolução de problemas de Inteligência Artificial (IA).

A Computação Evolutiva engloba todas as técnicas de resolução de problemas complexos de busca e aprendizagem que se baseiam na simulação dos processos descritos nas teorias da evolução. De acordo com PALAZZO (1997), a Computação Evolutiva divide-se em 5 paradigmas, como pode ser visto na FIGURA 3.1.

A Computação Evolutiva baseia-se fundamentalmente no uso de Algoritmos Evolutivos (AE). AE são algoritmos probabilísticos que mantêm uma população de indivíduos P(t)= $\{x_1^t,....,x_n^t\}$ em cada iteração t. Cada indivíduo representa uma solução possível para o problema, implementada com alguma estrutura de dados S. Cada solução x_i^t é avaliada para dar uma medida de sua qualidade ou

aptidão. Uma nova população é então formada pela seleção dos indivíduos mais aptos. Alguns membros da nova população sofrem transformações através dos operadores genéticos para formar novas soluções. AE têm como finalidade a sobrevivência dos indivíduos mais aptos, a qual é obtida pela sua maior e inerente chance de sobrevivência e, conseqüentemente, maior possibilidade de gerar descendentes que herdarão suas características.

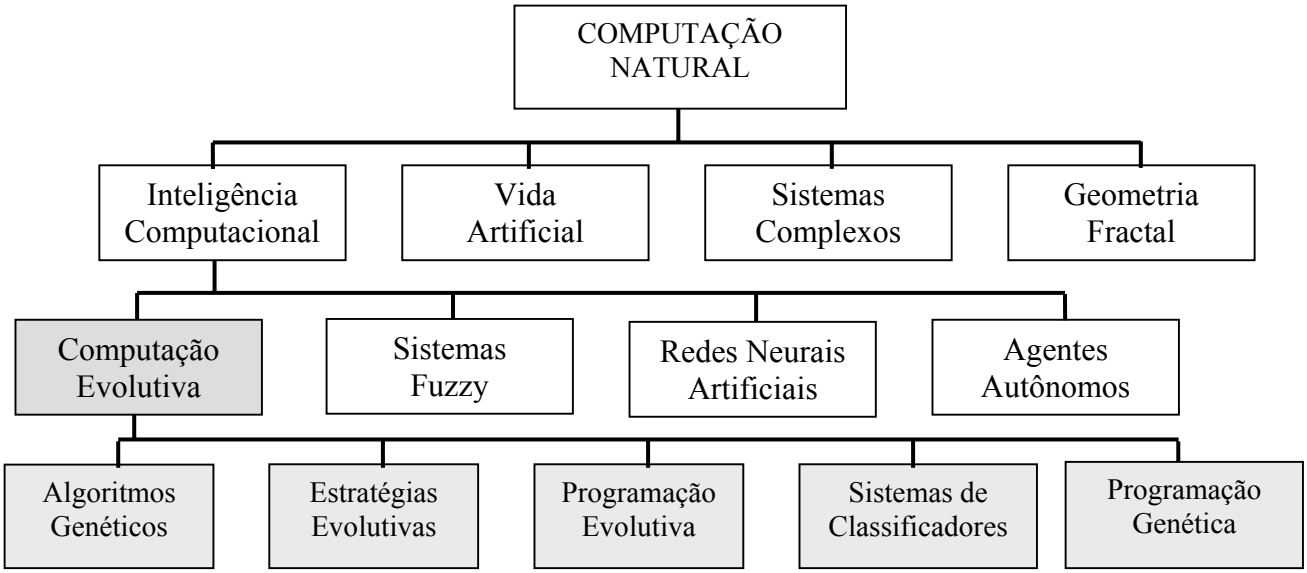


FIGURA 3.1 - Taxonomia dos Sistemas Naturais.
(fonte: PALAZZO, 1997)

Muitos AE podem ser formulados para um mesmo problema. Tais programas podem diferir de várias formas; usar diferentes estruturas de dados para codificar um indivíduo, diferentes operadores genéticos para transformar os indivíduos e diferentes valores de parâmetros (tamanho da população, probabilidades de aplicar diferentes operadores, entre outros). Todavia, eles têm um princípio comum: uma população de indivíduos sofre algumas transformações e durante este processo de evolução os indivíduos lutam por sua sobrevivência. Um algoritmo evolutivo é considerado eficiente quanto melhor for o seu desempenho na solução de um problema, independentemente de sua fidelidade aos conceitos biológicos.

Algoritmos evolutivos partem de um grupo ou **população** de soluções candidatas (**cromossomas** ou **indivíduos**). As variáveis do problema estão geralmente

associadas aos **genes** em um cromossoma e os valores que podem assumir correspondem aos **alelos** daquele gene. Através da seleção natural e operadores genéticos, cromossomas com melhor aptidão são obtidos. A seleção natural garante que os cromossomas mais aptos gerem descendentes nas populações futuras. Usando um operador de **cruzamento**, AE combina genes de dois cromossomas pais, previamente selecionados, para formar dois novos cromossomas filhos, os quais, espera-se, sejam mais aptos que os pais. Alguns cromossomas sofrem **mutação**, ou seja, uma pequena modificação estrutural com o propósito de conferir diversidade à população.

Algoritmos genéticos clássicos (AG) representam a forma mais simples de implementação de AE. Algoritmos genéticos utilizam vetores de *bits* (tiras binárias) ou caracteres para representar os cromossomas, e operações simples de manipulação dos *bits* para implementar cruzamento, mutação e outros operadores genéticos. Suponha, por exemplo, um problema de otimização em que se deseja maximizar a função $f(x)=5x_1 - 6x_2 + 15x_3 - 8x_4 + 12x_5 - 4x_6 - 5x_7$ ($x_i=\{0,1\}$, i=1..7). Na FIGURA 3.2, pode-se ver a simplicidade da representação de um possível cromossoma composto por sete genes (cada um associado a uma das variáveis), por meio de uma tira binária. Como será visto na seção 3.1.4, operações de cruzamento e mutação modificam os valores das variáveis, gerando novas soluções.

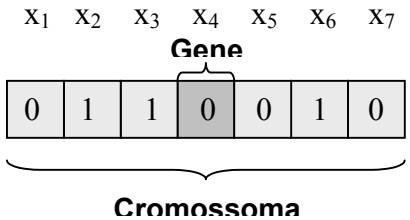


FIGURA 3.2 - Exemplo de representação de um cromossoma com 7 genes binários (f(x)=5).

Segundo Michalewicz (1996), a diferença básica entre algoritmos genéticos e evolutivos diz respeito ao uso de tiras binárias. Elas requerem uma modificação do problema original em uma forma apropriada; isso incluiu o mapeamento entre soluções potenciais e a representação binária, utilizando-se decodificadores. Algoritmos evolutivos não alteram o problema, mas adotam a representação de uma solução potencial usando estruturas de dados "naturais", e aplicam operadores

"genéticos" apropriados. Em outras palavras, para resolver um problema usando um programa evolutivo, pode-se tanto transformar o problema em uma forma apropriada para o algoritmo (AG) como transformar o algoritmo em uma forma apropriada para o problema (AE). Apesar dessa diferença conceitual, ao longo desse texto, será utilizado o termo algoritmos genéticos tanto para denotar algoritmos genéticos clássicos como evolutivos.

A estrutura básica de um algoritmo genético é mostrada na FIGURA 3.3.

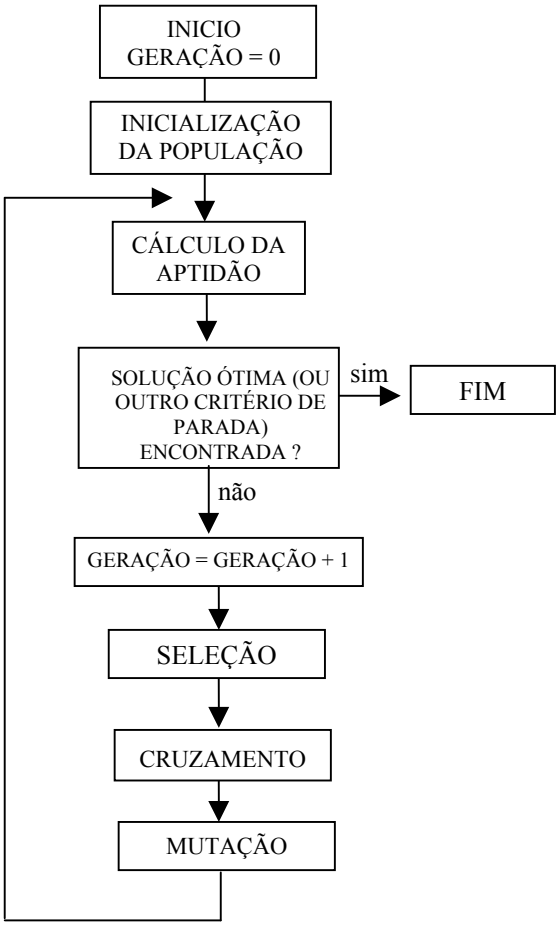


FIGURA 3.3 - Fluxograma básico de um algoritmo genético.

Algoritmos genéticos geralmente não incorporam conhecimento específico do problema a ser resolvido, tendo como guia apenas a função de aptidão. Entretanto, constitui uma vantagem o fato de operarem em uma população de soluções, ao contrário de outras meta-heurísticas (Busca Tabu (GLOVER & LAGUNA, 1997), *Simulated Annealing* (KIRKPATRICK *et al.*, 1983), GRASP (FEO & RESENDE, 1994), entre outras) que geram uma solução a cada iteração. Um algoritmo genético

representa uma busca multidirecional mantendo uma população de soluções possíveis e promove a troca de informação entre elas. A população sofre uma evolução simulada: em cada geração as "boas" soluções reproduzem, enquanto as soluções "piores" morrem.

Um algoritmo genético (como qualquer programa evolutivo) para um problema particular deve ter os cinco seguintes componentes:

- Uma representação das possíveis soluções do problema;
- Uma maneira para criar a população inicial de soluções;
- Uma função de avaliação que desempenhe o papel do meio ambiente, classificando as soluções em termos de sua "aptidão";
- Operadores genéticos que alteram a composição dos filhos;
- Valores para parâmetros genéticos (tamanho da população, probabilidades de aplicação dos operadores genéticos, entre outros.).

Estes componentes são discutidos mais detalhadamente a seguir.

3.1 Componentes de um AG

3.1.1 Representação das soluções (*encoding*)

Os algoritmos genéticos em sua forma original (AG clássicos) utilizam a representação de tiras binárias (veja FIGURA 3.2 da seção 3), para representar uma solução do problema abordado. Esta representação mostrou-se eficiente em vários problemas. Entretanto, à medida que as aplicações de AG foram crescendo, observou-se que esta representação poderia não ser a mais adequada.

Para o Problema do Caixeiro Viajante (TSP), por exemplo, a codificação binária não é tão natural. Considere um problema com 5 cidades e 20 arestas. Uma possibilidade para representar uma solução em que a rota usa as arestas 2, 4, 5, 11 e 15 é utilizar a tira binária com 20 posições < 01011000001000100000 >. Uma representação deste tipo não é adequada para o TSP já que somente um pequeno subconjunto dos cromossomas representa soluções factíveis do TSP. Mais especificamente, apenas um subconjunto dentre aqueles em que existem 5 genes com valor 1 são soluções factíveis, dado que muitos dos cromossomas que contêm cinco "1" e quinze "0" não formam um

circuito válido. A representação binária requereria neste caso, algoritmos que corrijam soluções, já que mesmo a mudança de um único *bit* pode resultar em uma rota inválida.

Surgiram, portanto novas alternativas. Para o Problema do Caixeiro Viajante, existe, por exemplo, a chamada "representação de caminho"**,** mais natural que a tira binária. Nesta representação, a rota 5-1-7-8-9-4-6-2-3 é representada simplesmente pelo vetor de números inteiros <5 1 7 8 9 4 6 2 3> onde a posição *i* armazena a *i*-ésima cidade a ser visitada.

No problema de designação de tarefas, cada cromossoma foi representado por um vetor de números inteiros. O vetor possui n posições correspondentes às tarefas do grafo; cada posição i do vetor armazena o número do processador onde a tarefa i foi designada na solução e a ordenação topológica das tarefas pode ou não ser feita previamente para definição deste vetor. Caso o seja, qualquer tarefa sucessora de uma dada tarefa *i* estará sempre em uma posição *j* > *i* do vetor (veja FIGURA 3.4).

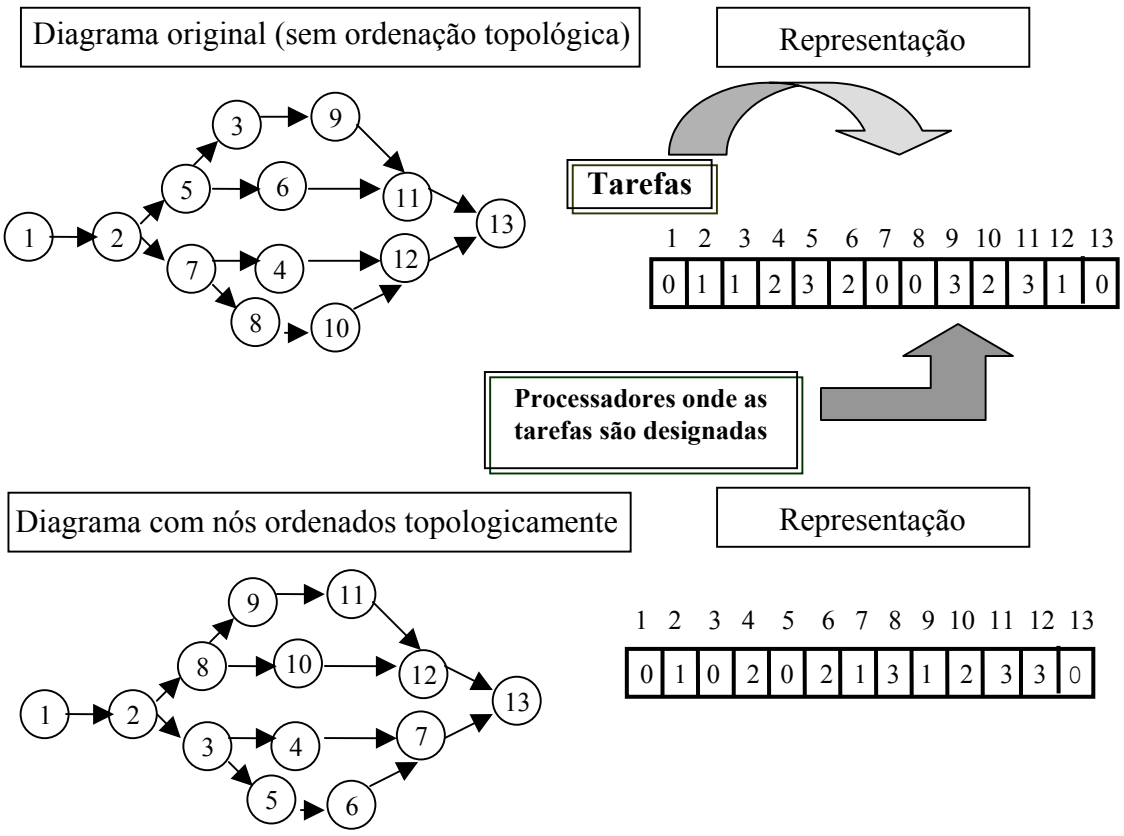


FIGURA 3.4 - Representação de um cromossoma antes e depois da ordenação topológica.

Independentemente do tipo de representação selecionado, deve-se sempre verificar se a representação está corretamente associada às soluções do problema analisado. Ou seja, toda solução deve ter um cromossoma associado e, reciprocamente, todo cromossoma gerado pelo AG deve estar associado a uma solução válida do problema analisado.

3.1.2 Geração da população inicial

A geração da população inicial de cromossomas em um AG muitas vezes não é uma tarefa das mais difíceis. De fato, em muitos problemas a população é gerada de forma aleatória. Além da facilidade inerente, ao se lidar com um problema novo pode-se aprender muito iniciando a população desta maneira. Fazer com que uma população gerada aleatoriamente evolua até se produzir uma solução com grande valor de adaptação, é um bom teste para saber como está funcionando a implementação do AG. Isso porque características essenciais e críticas da solução final devem ser conseqüência do processo evolutivo e não das características dos métodos usados para gerar a população inicial.

Em problemas onde a geração de uma população inicial não é tão simples, costuma-se usar perturbações de uma solução gerada por algum especialista no problema ou perturbações aleatórias da saída de algum algoritmo construtivo guloso. Uma técnica que combina perturbações aleatórias com uma regra de seleção gulosa é chamada de GRASP (*Greedy Randomized Adaptive Search Procedure*) (FEO & RESENDE, 1994). Algoritmos construtivos gulosos utilizam uma regra de seleção determinística para determinar iterativamente qual elemento deve ser incluído para compor a solução. Como resultado, geram uma única solução. A técnica GRASP, por sua vez, elege aleatoriamente um elemento a partir de uma lista dos k melhores candidatos ($k \geq 1$), permitindo a geração de diferentes soluções.

Como o problema de designação de tarefas em multiprocessadores pode ser considerado um problema novo dadas suas características em relação a tempos de comunicação interprocessador, foi adotada numa etapa inicial dos experimentos, a geração aleatória da população inicial. Numa etapa posterior, além de soluções geradas aleatoriamente, foram incluídas na população inicial a solução gerada pelo algoritmo de CHINNECK *et al.* (2003), e algumas soluções resultantes de pequenas perturbações do mesmo que alteram pouco a boa qualidade da solução original. Como resultado, obtém-se uma população inicial mista.

3.1.3 Função de aptidão das soluções

Avaliar um cromossoma corresponde a determinar seu valor de aptidão que se refletirá em uma maior ou menor probabilidade de sobrevivência. Em problemas de otimização numérica, o valor de aptidão pode ser equivalente ao valor da função objetivo da solução analisada. Em MICHALEWICZ (1996), tem-se exemplos de otimização de funções matemáticas simples (como $f(x) = x.\text{sen}(10\pi.x) + 1.0$ onde x $\in$ R), cujo o objetivo é encontrar os valores das variáveis que maximizam ou minimizam tais funções. Para tais situações, a função de aptidão utilizada é o valor normalizado da própria função matemática. A normalização dos valores assegura que estes estejam na faixa [0,1].

Seja *fit*(*i*) a aptidão do cromossoma *i.* A forma padrão de normalização dos valores de aptidão em problemas de maximização é dada pelo seguinte procedimento:

1. Calcule o valor da função objetivo (fo) de cada cromossoma *i* da população.
2. Faça F_t igual à soma dos valores da função objetivo dos cromossomas.
3. Faça *fit*(*i*)=fo(*i*)/F_t, onde *fit*(*i*) é a aptidão do cromossoma *i.*

Se o problema a ser resolvido é de minimização, o passo 3 do procedimento acima poderia ser substituído por:

3. Faça $fit(i) = \frac{(F_t - f_O(i))}{\sum_{i=1}^{m} F_t - f_O(i)}$, onde $fit(i)$ é a aptidão do cromossoma i e m é o número de cromossomas na população.

Como em qualquer heurística, a função utilizada para designar os valores de aptidão aos membros da população deve ser tal que dê como resultado uma boa distinção entre eles. Considere um problema de minimização e uma população de 5 cromossomas com os seguintes valores de função objetivo: 12, 9, 8, 11 e 10. Suponha também que soluções com valor 12 sejam consideradas de baixa qualidade e que soluções com valor 8 sejam consideradas de alta qualidade. Aplicando a normalização padrão, obtém-se:

$fit(1)$= 0,19; $fit\ t(2)$= 0,205; $fit\ t(3)$= 0,21; $fit\ t(4)$= 0,195; fit (5)= 0,2.

Note que esta função de aptidão não fornece uma boa distinção entre os cromossomas. Se estes valores de *fit* fossem usados, seria necessário muito tempo para que os descendentes do melhor indivíduo influam mais que os descendentes do pior indivíduo. Nesse sentido, uma forma mais apropriada de normalização poderia consistir no seguinte:

$fit(i) = \frac{f_O(máximo) - f_O(i)}{\sum_{i=1}^{m} f_O(máximo) - f_O(i)}$, onde $f_o(máximo)$ é o maior (pior) valor de função objetivo dos indivíduos da população.

Com isso, obtém-se valores de *fit* mais distintos que ressaltam a contribuição dos melhores cromossomas:

$fit(1)$= 0; $fit(2)$= 0,3; $fit(3)$= 0,4; $fit(4)$= 0,1; $fit(5)$= 0,2

$f_o(máximo)$ pode também ser substituído por outro valor de forma a não reduzir a função de aptidão (e portanto, a probabilidade de seleção) do pior indivíduo a zero.

Para o problema de designação de tarefas em multiprocessadores foi utilizada a seguinte função de aptidão, encontrada em vários trabalhos correlatos (AHMAD & DHODHI, 1996):

$$fit(i) = \frac{atraso\_máximo - atraso(i)}{\sum_{j=1}^{j=m} (atraso\_máximo - atraso(j))}$$

onde: *atraso_máximo* é o atraso máximo de um cromossoma da população corrente, *atraso* (*i*) é o atraso do cromossoma *i* e *m* é o tamanho da população.

Note que os valores da função acima se encontram na faixa [0,1). O pior cromossoma da população tem valor de aptidão 0. Uma possibilidade mais sofisticada de função de aptidão para o problema consiste em incorporar um termo adicional (também normalizado) que valorizaria cromossomas com menor número de processadores. Deve-se ressaltar que tal critério é secundário para os propósitos deste trabalho, devendo ser utilizado para desempate entre cromossomas com o mesmo valor de atraso.

3.1.4 Operadores genéticos

Os operadores genéticos representam o núcleo de um AG. O objetivo básico de um operador genético é produzir novos cromossomas que possuam propriedades genéticas superiores às encontradas nos pais. Em algoritmos genéticos convencionais, os operadores clássicos são a mutação e o cruzamento. Os melhores operadores de mutação e cruzamento dependem da aplicação, requerendo, portanto estudos empíricos para sua determinação.

O operador de mutação é necessário para a introdução e manutenção da diversidade genética da população, alterando arbitrariamente um ou mais componentes de uma estrutura escolhida. Desta maneira, ele fornece meios para introdução de novos elementos na população. Considere o cromossoma com representação binária da FIGURA 3.5:

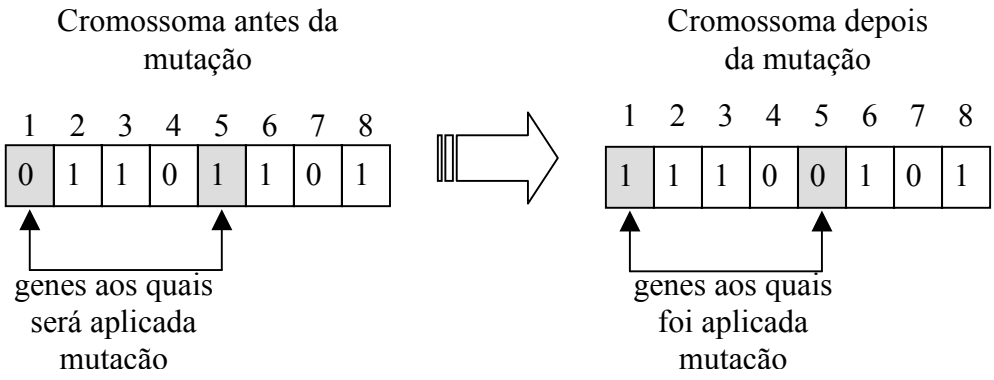


FIGURA 3.5- Operador de mutação em um cromossoma com representação binária.

Os genes 1 e 5 do cromossoma foram escolhidos (aleatoriamente) para mutação, a qual consistiu em alterar valores dos genes de 0 para 1 (gene 1) e 1 para 0 (gene 5).

O operador de mutação é aplicado a cada gene de um dado indivíduo com probabilidade dada pela taxa de mutação p_m. Geralmente utiliza-se uma taxa de mutação pequena, pois é um operador genético secundário, mas a escolha da taxa de mutação pode variar dependendo do problema.

Para o problema de designação de tarefas em multiprocessadores foram utilizados dois tipos de operação de mutação. O primeiro tipo (**retirada/inserção**) corresponde à seleção e retirada de uma tarefa de um processador e sua redesignação a um novo processador. O segundo tipo de mutação (**troca**) corresponde à troca de tarefas em processadores distintos. É fácil perceber que se estas decisões forem aleatórias, a solução resultante pode ser infactível em relação à capacidade do processador que recebe a tarefa. A forma de lidar com tal situação será descrita no passo 6.2.4 da seção 4.2 no capítulo 4.

Outro operador genético é o cruzamento, responsável pela combinação de características dos pais durante a reprodução de forma que as próximas gerações herdem essas características. Ele é considerado o operador genético predominante, por isso é aplicado com probabilidade dada pela taxa de cruzamento *pc*, maior que a taxa de mutação. Existem vários tipos de operadores de cruzamento. Os mais utilizados são:

- Um ponto: Um ponto de ruptura nos cromossomas é escolhido aleatoriamente. A partir deste ponto as informações genéticas dos pais são trocadas. As informações anteriores a este ponto em um dos pais são ligadas às informações posteriores a este ponto no outro pai, conforme ilustrado na FIGURA 3.6.

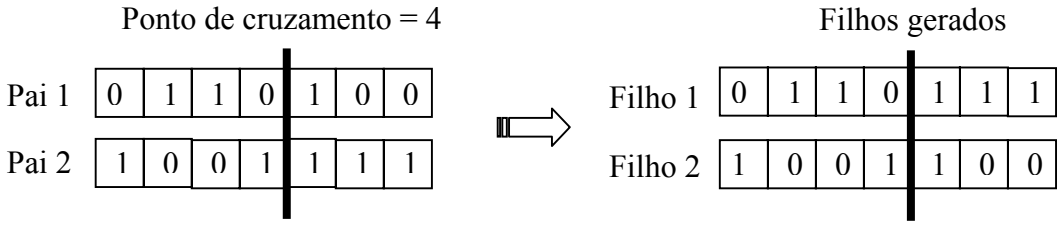


FIGURA 3.6 - Cruzamento em um ponto.

Observa-se que o filho 1 teve os 4 primeiros genes (4 é devido ao ponto de cruzamento) copiados do pai 1 e os outros genes copiados do pai 2. O filho 2 teve os 4 primeiros genes copiados do pai 2 e os outros genes copiados do pai 1.

- Multi-pontos: É uma generalização da idéia do cruzamento em um ponto, onde mais pontos de cruzamento podem ser utilizados. Na FIGURA 3.7 é mostrado um cruzamento em dois pontos.

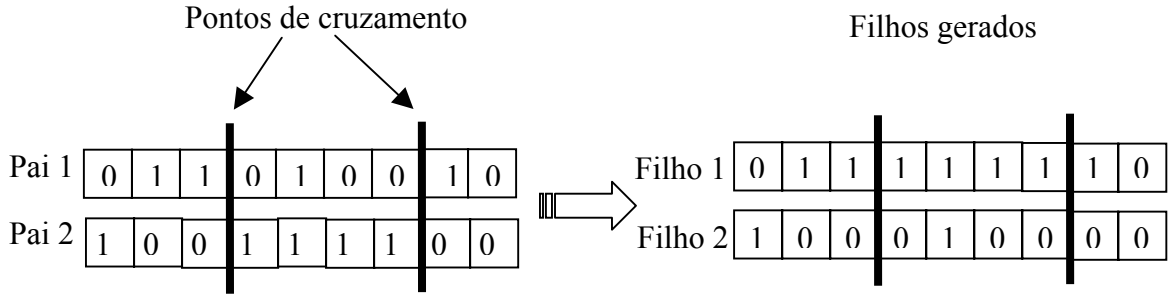


FIGURA 3.7 - Cruzamento em dois pontos.

No cruzamento da FIGURA 3.7 os dois pontos de cruzamento ocorrem no gene 3 e no gene 7. Dessa forma o filho 1 teve os 3 primeiros genes (3 é devido ao primeiro ponto de cruzamento) copiados do pai 1, os genes de 4 a 7 (7 é devido ao segundo ponto de cruzamento) copiados do pai 2 e os demais genes copiados novamente do pai 1. O filho 2 teve os 3 primeiros genes copiados do pai 2, os genes de 4 a 7 copiados do pai 1 e os demais genes copiados do pai 2.

- Uniforme: Não utiliza pontos de cruzamento, mas determina aleatoriamente qual dos pais fornecerá o valor de cada gene dos cromossomas filhos. Em representações binárias, isso é feito da seguinte maneira. Começando pelo primeiro *bit*, é aleatoriamente selecionado um pai para que contribua com seu primeiro *bit* ao primeiro filho, enquanto que o segundo filho recebe o *bit* do segundo pai. Este processo continua até que todos os *bits* tenham sido designados. Na FIGURA 3.8 é ilustrado o cruzamento uniforme. Os genes marcados são os genes que são copiados do pai 1 para o filho 1 e do pai 2 para o filho 2; os demais genes são copiados do outro pai.

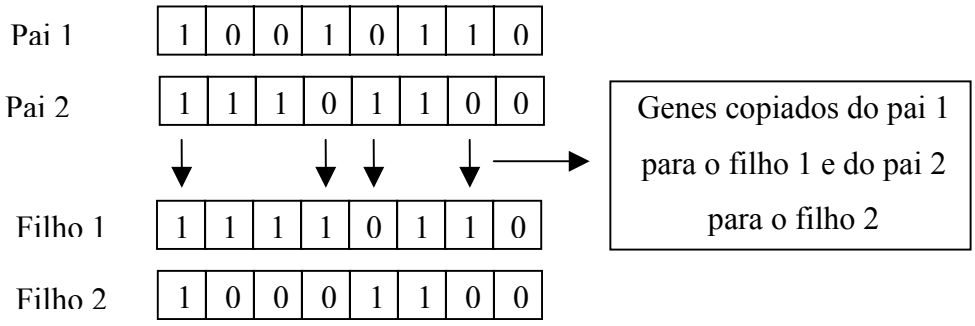


FIGURA 3.8 - Cruzamento uniforme.

Existem vários outros tipos de operadores de cruzamento, alguns projetados para problemas específicos (MICHALEWICZ, 1996).

Para o problema de designação de tarefas em multiprocessadores, foram investigados inicialmente os operadores genéricos cruzamento em um ponto (**1P**) e cruzamento em dois pontos (**2P**). Assim como na mutação, a solução resultante pode ser infactível em relação à capacidade de um ou mais processadores dos cromossomas filhos resultantes. A forma de lidar com tal situação é descrita na seção 3.1.5 abaixo e ilustrada no capítulo 4. Posteriormente, foi elaborado um novo operador de cruzamento, descrito na seção 3.1.6.

3.1.5 Tratamento de indivíduos infactíveis

Em muitos problemas é possível projetar codificações e operadores genéticos que garantem a factibilidade dos cromossomas gerados. Caso isso não seja feito, não é possível estabelecer uma correspondência entre os pontos do domínio do problema e o conjunto de variáveis binárias (ou outra representação utilizada). Como conseqüência, nem todos os cromossomas codificam indivíduos válidos do espaço de busca. Nesses casos devem ser feitos procedimentos para torná-los factíveis ou para penalizá-los. Por exemplo, no problema e implementação aqui tratados, quando se aplica o cruzamento em um ponto (1P) ou em dois pontos (2P) a dois cromossomas, geralmente são gerados cromossomas filhos com processadores com capacidade violada (seção 3.1.4). Como exemplo, considere na TABELA 3.1, 7 tarefas com seus respectivos requerimentos; na TABELA 3.2 tem-se os 2 cromossomas pais que farão cruzamento com as cargas dos processadores onde estão designadas as tarefas. Suponha

a aplicação de cruzamento de 1 ponto como mostrado na FIGURA 3.9, resultando em dois cromossomas filhos infactíveis conforme ilustrado na TABELA 3.3.

TABELA 3.1- Tarefas e requerimentos

Tarefas	Requerimentos (tp, md, mp)
1	50, 50, 50
2	70, 50, 50
3	40, 30, 20
4	80, 20, 20
5	20, 10, 10
6	30, 10, 20
7	50, 40, 60

Pai 1

TABELA 3.2- Cromossomas pais

Processador	tp, md, mp
0	90, 80, 70
1	100, 60, 70
2	100, 30, 30
3	50, 40, 60

Pai 2

Processador	tp, md, mp
0	90, 80, 70
1	90, 60, 60
2	80, 50, 80
3	80, 20, 20

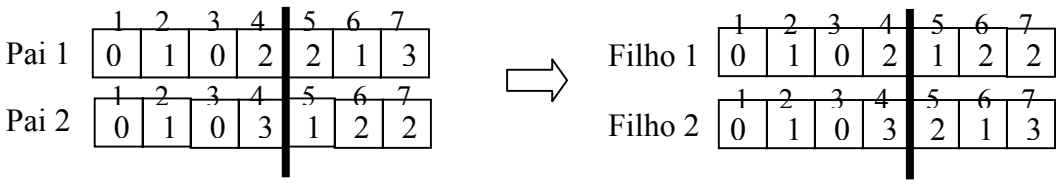


FIGURA 3.9 - Cruzamento 1P.

TABELA 3.3 - Cromossomas filhos.

Filho 1

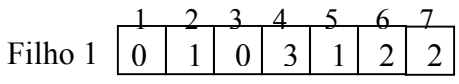
Processador	tp, md, mp
0	90, 80, 70
1	90, 60, 60
2	160, 70, 100

Capacidade violada

Filho 2

Processador	tp, md, mp
0	90, 80, 70
1	100, 60, 70
2	80, 50, 80
3	80, 20, 20

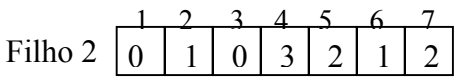
Filho 1 e Filho 2 tiveram respectivamente o processador 2 e o processador 3 com a capacidade violada. Primeiramente repara-se o Filho 1. As tarefas que estão designadas para o processador 2 são: 4, 6 e 7. Uma forma de factibilizar este cromossoma consiste em colocar alguma(s) de sua(s) tarefa(s) em outro(s) processador(es) de forma que ao final, as capacidades de todos os processadores sejam respeitadas. Suponha que a tarefa 4 foi designada para ser retirada do processador 2. Como não há possibilidade de designá-la em outro processador existente pois as capacidades seriam violadas, cria-se um novo processador (processador 3) para o qual é designada a tarefa 4 (FIGURA 3.10).



Processador	tp, md, mp
0	90, 80, 70
1	90, 60, 60
2	80, 50, 80
3	80, 20, 20

FIGURA 3.10 - Filho 1 depois da reparação.

O mesmo procedimento pode ser aplicado a Filho 2. As tarefas que estão no processador 3 com capacidade violada são: 4 e 7. Escolhe-se entre as duas tarefas, uma para ser retirada do processador 3. Suponha que tenha sido escolhido a tarefa 7. Entre os processadores existentes, a tarefa 7 pode ser designada somente no processador 2 sem violação de sua capacidade. Designa-se, portanto, a tarefa 7 ao processador 2, conforme a FIGURA 3.11.



Processador	tp, md, mp
0	90, 80, 70
1	100, 60, 70
2	70, 50, 70
3	80, 20, 20

FIGURA 3.11- Filho 2 depois da reparação.

Para o problema de designação de tarefas em multiprocessadores, permitiu-se a geração de cromossomas infactíveis (aplicável após cruzamento 1P e 2P e mutação), seguida de sua factibilização. A escolha das tarefas a serem retiradas dos processadores com capacidade violada e a forma de designação destas tarefas são discutidas no capítulo 4.

3.1.6 Operador de cruzamento 1PC

Conforme mencionado nas seções 3.1.4 e 3.1.5, os operadores de cruzamento 1P e 2P não garantem a factibilidade dos cromossomos filhos gerados com respeito aos limites de memória de dados, memória de programa e tempo de processamento (capacidade) dos processadores. De fato, tais cruzamentos quase sempre resultam filhos infactíveis. Por essa razão, um mecanismo de reparação das soluções (descrito e ilustrado com detalhes na seção 3.1.5) foi utilizado após cada cruzamento. Em termos gerais, tal mecanismo consiste em selecionar tarefas em processadores com capacidade violada e redesigná-las a outros processadores.

Uma grande desvantagem da reparação é que dada a freqüência com que é aplicada, acaba representando uma parcela considerável do tempo de execução do algoritmo. Por esta razão, o operador de cruzamento *1 ponto com construção* (1PC) foi elaborado com o objetivo de eliminar a necessidade de reparação. A descrição do operador é dada a seguir:

- Dados dois cromossomas pais, define-se aleatoriamente um ponto de quebra nos mesmos, resultando em duas subtiras. Os dois cromossomas filhos copiam a 1ª subtira dos pais e a partir do ponto de quebra, a 2ª subtira é construída selecionando-se o

processador cujos limites de memória de dados, memória de programa e tempo de processamento dos processadores não sejam excedidos com a designação da tarefa, e que resulte na maior redução do atraso.

Por exemplo, considere o diagrama de tarefas da FIGURA 3.12 que está com os nós ordenados topologicamente, isto é qualquer tarefa sucessora de uma dada tarefa i estará sempre em uma posição $j > i$ do vetor:

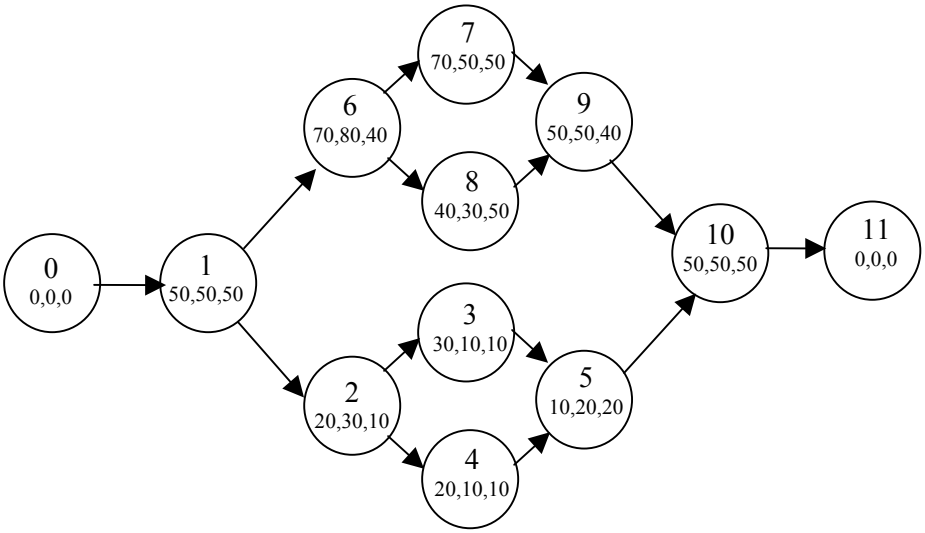
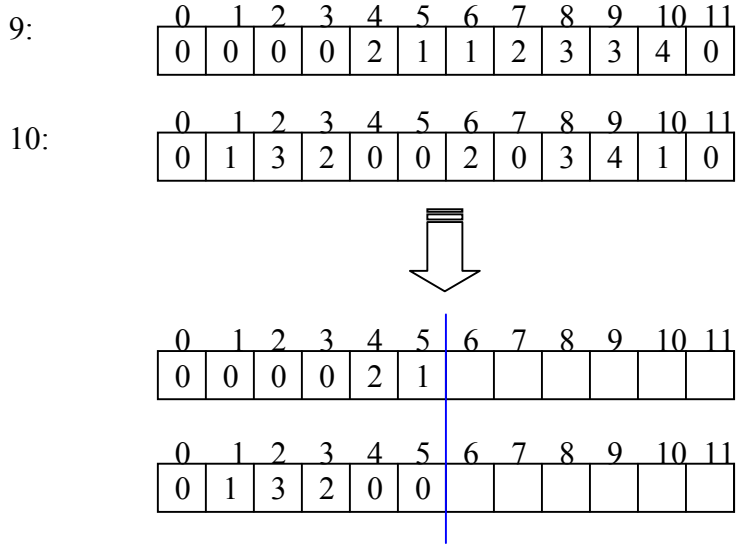


FIGURA 3.12 - Diagrama de tarefas com os nós ordenados topologicamente

Considere que 2 cromossomos tenham sido selecionados para cruzamento no ponto de cruzamento 5. A 1ª subtira do 1º pai (cromossoma 9) é copiada para o primeiro filho e a 1ª subtira do 2º pai (cromossoma 10) é copiada para o 2º filho, conforme ilustrado abaixo:



9`:

10`:

Após as cópias das subtiras terem sido realizadas até o ponto de cruzamento 5, começa-se a construir a 2ª subtira dos filhos. Considere o filho 9':

Tarefa a ser analisada: 6

- Processadores que podem receber tarefa 6: 1, 2 e 3.
- A designação da tarefa em qualquer um dos processadores resulta em um mesmo atraso acumulado até a tarefa 6. Seleciona-se aleatoriamente o processador 2 para designação desta tarefa, obtendo-se:

9´:

0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	2	1	2					

Tarefa a ser analisada: 7

- Processadores que podem receber a tarefa 7: 1 e 3.
- A designação da tarefa no processador 3 resulta em menor atraso acumulado até a tarefa 7. Designa-se esta tarefa ao processador 3.

9´:

0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	2	1	2	3				

Tarefa a ser analisada: 8

- Processadores que podem receber tarefa 8: 1 e 4.
- A designação da tarefa em qualquer um dos processadores resulta em mesmo atraso acumulado até a tarefa 8. Seleciona-se aleatoriamente o processador 1 para designação desta tarefa, obtendo-se:

9´:

0	1	2	3	4	5	6	7	8	9	10	11
0	0	0	0	2	1	2	3	1			

Tarefa a ser analisada: 9

- Processadores que podem receber tarefa 9: 4.
- Tarefa 9 designada ao processador 4.

	0	1	2	3	4	5	6	7	8	9	10	11
9′:	0	0	0	0	2	1	2	3	1	4		

Tarefa a ser analisada: 10

- Processadores que podem receber tarefa 10: 4 e 5.
- A designação da tarefa 10 ao processador 4 resulta em menor atraso acumulado até a tarefa 10. Designa-se esta tarefa ao processador 4, obtendo-se:

	0	1	2	3	4	5	6	7	8	9	10	11
9′:	0	0	0	0	2	1	2	3	1	4	4	

Tarefa a ser analisada: 11 (tarefa *output*).

- A tarefa *output* é sempre designada ao processador 0.

	0	1	2	3	4	5	6	7	8	9	10	11
9′:	0	0	0	0	2	1	1	2	3	3	4	0

O cromossoma filho 9’ está completo e as capacidades dos processadores respeitadas. Esse procedimento é aplicado para o filho 10’, que após todas as designações torna-se:

	0	1	2	3	4	5	6	7	8	9	10	11
10′:	0	1	3	2	0	0	2	3	0	4	1	0

O operador de cruzamento 1PC sempre resulta em filhos factiveis além de simultaneamente promover designações com redução do atraso.

3.1.7 Tamanho da população, probabilidade de aplicação de operadores genéticos e outros parâmetros.

3.1.7.1 Tamanho da população

O tamanho da população determina o número de cromossomas presentes na população e é uma das mais importantes decisões na elaboração de um AG. Com uma população pequena o AG pode convergir rápida e prematuramente, enquanto que com uma população muito grande o AG pode "desperdiçar" recursos computacionais e o tempo de espera por uma melhoria deve ser longo. É importante manter a diversidade da população, mas deve-se levar também em conta uma pressão na direção de soluções de alta qualidade. O tamanho da população deve buscar o equilíbrio entre esses dois aspectos. Alguns pesquisadores (MICHALEWICZ, 1996) sugerem o uso de população com tamanho variável: em diferentes estágios do processo, diferentes tamanhos da população podem ser mais adequados. Por exemplo, em etapas em que ocorram melhorias substanciais na qualidade das soluções, a diminuição do tamanho da população causaria maior pressão de convergência a soluções de alta qualidade, enquanto que o aumento do tamanho da população pode trazer maior diversificação quando a busca parece estar convergindo para uma população homogênea. Estes conceitos guardam similaridades com estratégias de intensificação e diversificação em Busca Tabu (GLOVER & LAGUNA, 1997).

3.1.7.2 Taxa de cruzamento

A taxa de cruzamento (ou probabilidade de cruzamento, p_c) determina o número esperado de cromossomas na população que sofrerão cruzamento. Seja n o número de cromossomas na população, então o número esperado de cromossomas que sofrerão cruzamento é $n \, . p_c$.

3.1.7.3 Taxa de mutação

A taxa de mutação (ou probabilidade de mutação, p_m) determina o número esperado de genes que sofrerão a mutação. Cada gene (em todos cromossomas na população) tem probabilidade igual de sofrer mutação. Seja n o número de

cromossomas na população e g o número de genes em cada cromossoma, então o número de genes que sofrerão mutação é $n \cdot g \cdot p_m$.

3.1.7.4 Critério de parada

Em implementações de AG devem ser determinadas condições de parada da busca heurística. Em particular foram adotados os seguintes critérios padrão:

- Atingiu-se um número limite de gerações.
- Gerou-se uma população homogênea, isto é, os indivíduos tornaram-se todos iguais.

3.1.7.5 Mecanismos de amostragem de populações

A seleção dos cromossomas que comporão a próxima geração pode ser feita de variadas formas. Dentre estas, destacam-se três grupos principais segundo o grau de influência da aleatoriedade no processo:

- **Amostragem direta**: seleção de um subconjunto de cromossomas da população mediante um critério fixo, por exemplo "os n melhores cromossomas".
- **Amostragem aleatória simples ou equiprovável**: atribui-se a mesma probabilidade de seleção para todos os elementos da população.
- **Amostragem estocástica**: atribui-se probabilidades de seleção com base na função de aptidão. Existem vários mecanismos de amostragem estocástica, de acordo com a aplicação, dos quais são mostrados dois:
- **Roleta** (GOLDBERG,1989): consiste em criar uma roleta na qual cada cromossoma possui um segmento proporcional à sua aptidão. Considere um problema de maximização e uma

população de 4 cromossomas cuja aptidão é dada pela função objetivo (TABELA 3.4). Os valores normalizados da função de aptidão foram obtidos segundo o procedimento apresentado na seção 3.1.3.

TABELA 3.4 – Ilustração de seleção por roleta.

Cromossoma	**Fo**	***fit***
1	**45**	**16**
2	**89**	**31,8**
3	**125**	**44,7**
4	**21**	**7,5**
Σ=	**280**	**100**

Com os valores percentuais de *fit* da TABELA 3.4, elaborou-se a roleta da FIGURA 3.13. Ela é girada 4 vezes para efetuar a seleção dos indivíduos que formarão a nova população; indivíduos com maior área na roleta têm maiores chances de serem selecionados mais vezes que os indivíduos menos aptos.

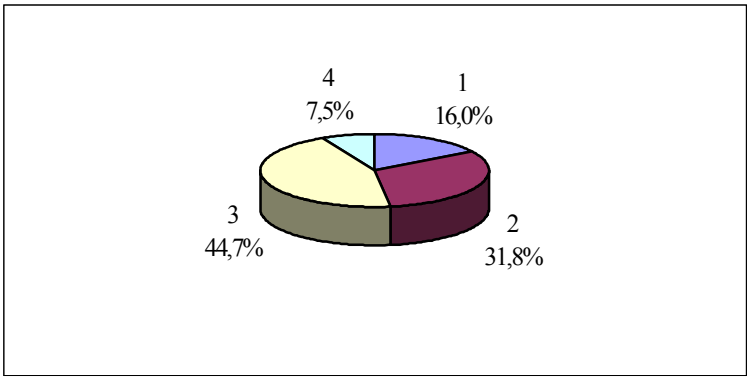


FIGURA 3.13 - Representação gráfica da roleta.

- **Torneio**: cada elemento da nova população é selecionado elegendo-se o melhor indivíduo de um conjunto de *z* elementos tomados aleatoriamente na população atual. Este procedimento é repetido o número de vezes necessário para completar a nova população. O valor do parâmetro *z* costuma ser um valor inteiro pequeno em relação ao tamanho total da população atual, em geral 2 ou 3.

Neste trabalho, utilizou-se seleção por roleta. Detalhes da implementação computacional da geração da roleta são apresentados no capítulo 4.

3.1.7.6 Critérios de substituição

Os critérios utilizados para selecionar os indivíduos que formarão a população para aplicação do cruzamento e mutação não têm necessariamente que ser os mesmos usados para selecionar os indivíduos sobreviventes após o cruzamento e a mutação. A substituição da população pode ser feita das seguintes formas (YEPES, 2001):

- **Substituição Imediata**: os descendentes substituem os progenitores, isto é, os cromossomas resultantes do cruzamento, substituirão seus pais.
- **Substituição por fator cheio**: os descendentes substituem aqueles membros da população mais parecidos com ele.

- **Substituição por inserção**: seleciona-se *s* membros da população (geralmente os *s* piores) para serem substituídos pelos descendentes.
- **Substituição por inclusão**: os *s* descendentes são somados aos *n* progenitores em uma única população e nesta, são selecionados os *n* melhores.

Neste trabalho, utilizou-se o critério de substituição imediata.

3.2 Motivação para o uso da metodologia

Algoritmos genéticos têm sido aplicados com sucesso na resolução de vários problemas de otimização, incluindo otimização numérica e problemas de otimização combinatória como o TSP assimétrico, o problema de empacotamento industrial bidimensional, o problema de programação em uma máquina e o problema de alocação em máquinas unidimensional. Exemplos recentes de aplicações são encontrados em MURATA *et al*. (1996), GONG *et al.* (1999), SHARMA *et al*. (1997), HOPPER & TURTON (1999), WANG *et al*., (1999), BURIOL (2000) e FRANÇA *et al.* (2001). Aplicações de algoritmos genéticos em problemas de engenharia de produção podem ser encontradas em GEN & CHENG (1997, 1999).

Os AG também vêm sendo utilizados em outras situações práticas com resultados bastante satisfatórios. Em LAUDON & LAUDON (1999) tem-se um exemplo interessante onde um sistema baseado em algoritmos genéticos foi elaborado para auxiliar testemunhas a identificar suspeitos criminais. Muitas vezes as testemunhas não conseguem descrever as características individuais de um suspeito, porém se saem muito melhor no reconhecimento de rostos. O *software* funciona por meio de ilustrações aleatórias de faces em uma tela de computador, combinando e recombinando características até surgir a melhor descrição. O sistema capacita a testemunha de um crime a selecionar a figura mais exata de um suspeito, um passo mais perto da seleção natural, a partir de um grupo de imagens composto por características escolhidas aleatoriamente. O *software* então combina a melhor figura com outras faces construídas aleatoriamente e repete o processo seguidamente enquanto elimina as características improváveis. Em um determinado momento, uma foto do suspeito é formada. Enquanto

um ser humano pode se cansar depois de realizar uma dúzia de tentativas, o *software* pode pesquisar um enorme "espaço de faces" muito rapidamente, tentando milhões de possibilidades.

No próximo capítulo é apresentada a implementação de um AG para o problema de designação de tarefas, e descritas com maior detalhe suas características.

4 ALGORITMOS GENÉTICOS PARA O PROBLEMA DE DESIGNAÇÃO DE TAREFAS A MULTIPROCESSADORES DIGITAIS DE SINAL

Em termos gerais, o algoritmo base segue o fluxograma apresentado na FIGURA 3.3, capítulo 3. Uma descrição mais detalhada destes e de outros passos envolvidos é apresentada na próxima seção.

4.1 Passos do algoritmo base

Passo 1: Leitura do arquivo

Nesta etapa, é lido o arquivo onde estão armazenados para cada tarefa:

- Identificador ou nome da tarefa.
- Requerimentos da tarefa: memória de programa, memória de dados e tempo de processamento.
- Predecessores e os sucessores da tarefa.

São criadas duas tarefas fantasmas: tarefa *input* que precede as tarefas sem predecessores e tarefa *output* que sucede as tarefas sem sucessores. As tarefas fantasmas têm requerimentos nulos e são permanentemente designadas ao processador 0 (mestre). Estas tarefas são necessárias para o cálculo do atraso uma vez que a porta do sinal de entrada e do sinal de saída é o processador 0.

Passo 2: Ordenação topológica

As tarefas são ordenadas topologicamente em virtude do procedimento utilizado para o cálculo do atraso (passo 5). A ordenação topológica garante que o atraso até uma dada tarefa *i* não seja calculado antes dos atrasos de todas as suas predecessoras. É importante ressaltar que, para o cálculo do atraso, é necessária uma estrutura computacional onde as tarefas estejam ordenadas topologicamente, enquanto que na representação computacional do cromossoma para aplicação dos operadores de cruzamento e mutação, as tarefas podem estar ou não ordenadas topologicamente (ver seção 3.1.1).

Passo 3: Inicialização dos processadores

São criados *n* processadores inicialmente vazios (*n* é o número total de tarefas, isto é, as tarefas do arquivo e as duas tarefas fantasmas).

Passo 4: Inicialização da população

Conforme mencionado na seção 3.1.2, a população inicial pode ser totalmente gerada de forma aleatória ou pode-se incluir soluções geradas por um outro método heurístico. A geração aleatória de uma população de m soluções é feita da seguinte maneira. Para cada tarefa associa-se um número aleatório entre 0 e 1; os números resultantes são então ordenados pelo algoritmo *QuickSort*. As tarefas associadas a cada número da lista ordenada são designadas uma a uma aos processadores a partir do processador 0. Se a designação da tarefa *i* ultrapassar a capacidade do processador corrente *p*, esta é designada ao processador *p*+1. Desse modo, obtém-se uma população de soluções factíveis. Após a inicialização da população calcula-se o atraso de cada solução.

Passo 5: Cálculo do atraso

O atraso é calculado por programação dinâmica com recursividade progressiva. Considere o grafo com as tarefas 1,2,3,4 da FIGURA 4.1. As tarefas *input* e *output* são tarefas fantasmas criadas no passo 1. O número em cada caixa é o identificador da tarefa e o número sobre cada caixa é o processador atual da tarefa.

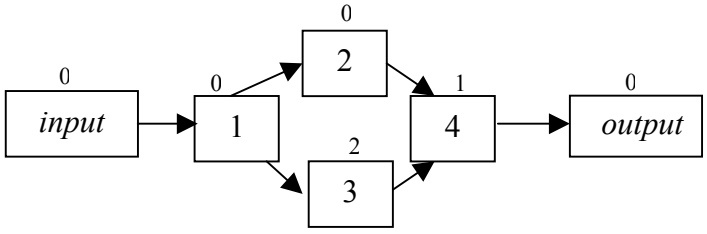
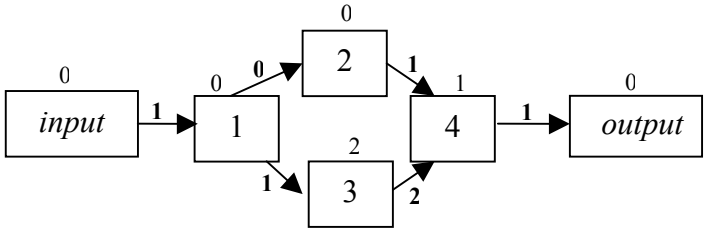


FIGURA 4.1- Diagrama de tarefas.

- Cálculo dos pesos dos arcos (que nada mais é que o tempo de comunicação entre as tarefas, conforme seção 2.3 do capítulo 2).



- Cálculo do Atraso Total Acumulado até cada tarefa *i* (ATA(*i*)):

  Seguindo a ordenação topológica e considerando a designação realizada, é calculado o ciclo em que cada tarefa *i* é executada. Seja *P*, o conjunto de predecessoras imediatas da tarefa *i*.

  ATA(*i*) = Max $_{j \in P}$\{ ATA(*j*) + tempo de comunicação entre *j* e *i*\}

  Por definição, a tarefa *input* (designada ao processador 0) tem ATA(*input*) = 1.

  - ✓ Tarefa 1 (processador 0):

    Predecessora: Tarefa *input* (processador 0).

    ATA(1) = ATA(*input*) + 0 = 1 + 0 = 1.
  - ✓ Tarefa 2 (processador 0):

    Predecessora: Tarefa 1 (processador 0).

    ATA(2) = ATA(1) + 0 = 1 + 0 = 1.
  - ✓ Tarefa 3 (processador 2):

    Predecessora: Tarefa 1 (processador 0).

    ATA(3)= ATA(1) + 1 = 1 + 1 = 2
  - ✓ Tarefa 4 (processador 1):

    Predecessoras: Tarefas 2 e 3.

    $$\text{ATA (4)} = \text{Max} \begin{cases} \text{ATA(2)} + 1 = 1 + 1 = 2 \\ \text{ATA(3)} + 2 = 2 + 2 = 4 \end{cases}$$

    ATA (4) = 4.
  - ✓ Tarefa *output* (processador 0) - fantasma:

    Predecessora: Tarefa 4 (processador 1).

    ATA(*output*)= ATA(4) + 1 = 4 + 1 = 5.

  O atraso total corresponde ao atraso da tarefa *output* (5 ciclos).

Passo 6: Busca com algoritmo genético

O algoritmo genético base consiste das seguintes etapas:

6.1 Obtenha uma população inicial com m cromossomas.

6.2 Enquanto (geração corrente ≤ n^o. máximo de gerações) ou (população não homogênea) faça:

6.2.1 Calcule a aptidão de cada cromossoma da população.

6.2.2 Selecione m cromossomas para a próxima geração.

6.2.3 Aplique cruzamento com probabilidade p_c entre pares de cromossomas selecionados no passo anterior (cromossomas filhos substituem os pais). Faça reparação dos cromossomas infactíveis (se houver).

6.2.4 Aplique mutação com probabilidade p_m em cada tarefa dos cromossomas da geração corrente.

6.3 Retorne o cromossoma com menor valor de atraso obtido no processo.

Foram elaboradas 6 implementações iniciais de algoritmos genéticos. Para estes algoritmos, a população inicial foi gerada aleatoriamente e não foram realizadas buscas locais. As diferenças entre as 6 versões consistem de: (a) tipo de operador de cruzamento- 1P ou 2P; (b) tipo de operador de mutação - retirada/inserção ou troca; e (c) nível de informação na tomada de decisões durante a operação de mutação e durante a reparação dos cromossomas infactíveis que sucede a aplicação de cruzamento. Em relação ao item (c), o primeiro algoritmo (algoritmo puro) faz todas as escolhas de forma aleatória. O segundo algoritmo (inteligente 1) faz algumas escolhas de forma determinística, ou seja, visando a geração de soluções com menor atraso. O terceiro, quarto, quinto e sexto algoritmos (inteligente 2, 3, 4 e 5) fazem ainda mais escolhas de forma determinística que o inteligente 1. Como os cinco últimos algoritmos correspondem a modificações do primeiro, a seção 4.2 descreve uma iteração do algoritmo puro para ilustrar o funcionamento geral do algoritmo. As particularidades dos algoritmos inteligente 1, 2, 3, 4 e 5 são então esclarecidas nas seções 4.3 a 4.7.

4.2 Uma iteração do algoritmo genético PURO

Considere o problema de 10 tarefas apresentado na FIGURA 4.2:

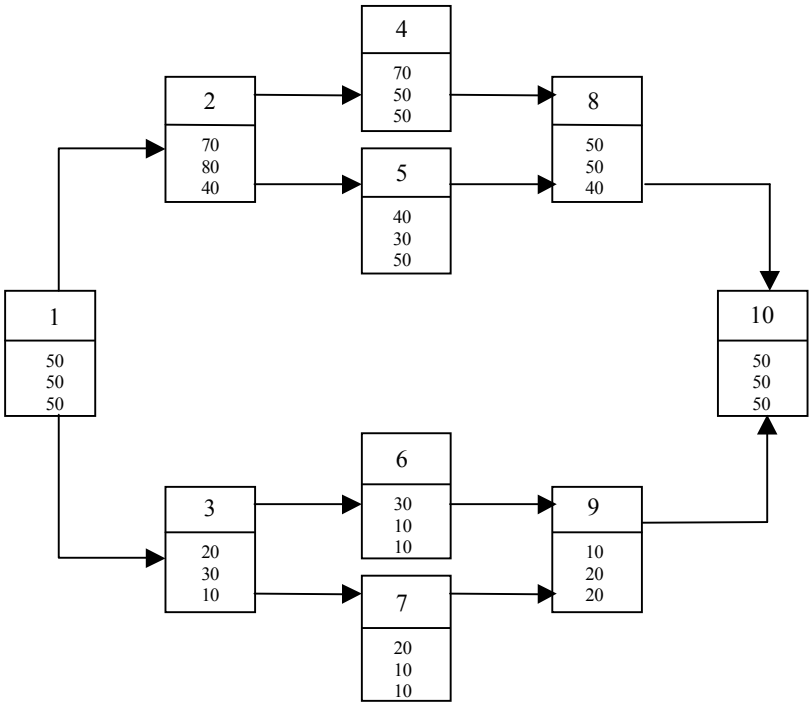


FIGURA 4.2 - Diagrama de 10 tarefas.

Conforme definido na seção 3.1.1, capítulo 3, a representação dos cromossomas consiste em um vetor de números inteiros de n posições onde em cada posição *i* é armazenado o número do processador da tarefa *i*. (FIGURA 4.3). Note que as *n* tarefas do diagrama estão ordenadas topologicamente e as tarefas fantasmas mencionadas no passo 1 da seção 4.1 foram excluídas da representação pois, apesar de necessárias para o calculo do atraso, não fazem parte do diagrama em si.

Tarefas ⇨	1	2	3	4	5	6	7	8	9	10
Processadores ⇨	1	1	2	3	2	0	0	3	2	3

FIGURA 4.3- Representação de um cromossoma.

Passo 6.1 (Obtenção da população inicial) : Admitindo tamanho da população m = 20, obtém-se:

Cromossoma 1: 0 4 1 2 3 1 1 3 1 0 Atraso: 6

Cromossoma 2: 3 0 1 4 1 1 0 2 1 2 Atraso: 8

Cromossoma 3: 1 2 0 4 0 1 0 3 0 3 Atraso: 8
Cromossoma 4: 4 3 0 1 0 1 0 2 0 2 Atraso: 8
Cromossoma 5: 2 1 0 3 0 1 0 4 0 2 Atraso: 10
Cromossoma 6: 3 1 2 2 0 4 1 0 0 3 Atraso: 9
Cromossoma 7: 2 1 0 4 3 1 3 2 0 0 Atraso: 9
Cromossoma 8: 3 0 2 2 4 1 0 1 1 3 Atraso: 9
Cromossoma 9: 1 2 3 3 0 0 0 1 0 4 Atraso: 9
Cromossoma 10: 1 3 2 2 4 3 0 1 0 0 Atraso: 9
Cromossoma 11: 0 2 0 1 3 0 1 3 1 4 Atraso: 8
Cromossoma 12: 0 4 1 1 3 0 0 2 1 2 Atraso: 7
Cromossoma 13: 0 1 0 2 3 0 2 3 1 4 Atraso: 7
Cromossoma 14: 4 3 2 1 0 2 1 0 0 2 Atraso: 9
Cromossoma 15: 4 2 3 0 3 0 1 4 1 1 Atraso: 9
Cromossoma 16: 1 2 3 0 3 2 0 4 0 1 Atraso: 9
Cromossoma 17: 1 4 2 0 3 2 0 2 0 1 Atraso: 11
Cromossoma 18: 4 3 0 2 0 2 0 1 0 1 Atraso: 9
Cromossoma 19: 2 1 0 0 3 3 1 2 0 4 Atraso: 11
Cromossoma 20: 4 3 0 1 2 3 0 0 1 2 Atraso: 9

O melhor cromossoma é o primeiro com atraso 6.

Passo 6.2.1 (Cálculo da aptidão): Conforme definido na seção 3.1.3, capítulo 3, a aptidão de cada cromossoma é calculada por meio da função:

$$fit(i) = \frac{atraso\_máximo - atraso(i)}{\sum_{j=1}^{j=m} (atraso\_máximo - atraso(j))}$$

onde *atraso_máximo* é o atraso máximo de um cromossoma da população corrente, *atraso*(*i*) é o atraso do cromossoma *i* e *m* é o tamanho da população.

O atraso máximo da população é 11. Assim:

Fit(1)=0,108696
Fit(2)=0,065217
Fit(3)=0,065217
Fit(4)=0,065217
Fit(5)=0,021739
Fit(6)=0,043478
Fit(7)=0,043478
Fit(8)=0,043478
Fit(9)=0,043478
Fit(10)=0,043478
Fit(11)=0,065217
Fit(12)=0,086957
Fit(13)=0,086957
Fit(14)=0,043478
Fit(17)=0,000000
Fit(18)=0,043478

Fit(15)=0,043478	*Fit*(19)=0,000000
Fit(16)=0,043478	*Fit*(20)=0,043478

Passo 6.2.2 (Seleção dos cromossomas para a próxima geração): Admitindo que a população seja totalmente gerada de maneira aleatória, utilizou-se geração de roleta para seleção dos cromossomas que comporão a próxima geração. A implementação consiste inicialmente no cálculo da probabilidade acumulada q_i de cada cromossoma i (i = 1,...,20). Ou seja, $q_1 = fit_1$ e $q_{i+1} = q_i + fit_{i+1}$. Desta forma, obtém-se:

q(1)=0,108696	q(11)=0,608696
q(2)=0,173913	q(12)=0,695652
q(3)=0,239130	q(13)=0,782609
q(4)=0,304348	q(14)=0,826087
q(5)=0,326087	q(15)=0,869565
q(6)=0,369565	q(16)=0,913043
q(7)=0,413043	q(17)=0,913043
q(8)=0,456522	q(18)=0,956522
q(9)=0,500000	q(19)=0,956522
q(10)=0,543478	q(20)=1,0000

São gerados m números aleatórios entre 0 e 1 (m é o tamanho da população). Seja v um destes números. Se $q(j-1) < v \leq q(j)$, então j é selecionado para a nova população. Suponha que os 20 números aleatórios tenham sido:

0.360882	0.621693
0.363946	0.780021
0.457001	0.267468
0.205352	0.295457
0.698214	0.702830
0.837232	0.456934
0.751804	0.964296
0.901830	0.188492
0.186460	0.012273
0.775525	0.224511

A nova população consistirá dos seguintes cromossomas:

Cromossoma 1: 3 1 2 2 0 4 1 0 0 3 (cromossoma 6) *Fit*: 0.04348

Cromossoma 2: 3 1 2 2 0 4 1 0 0 3 (cromossoma 6) *Fit*: 0.04348

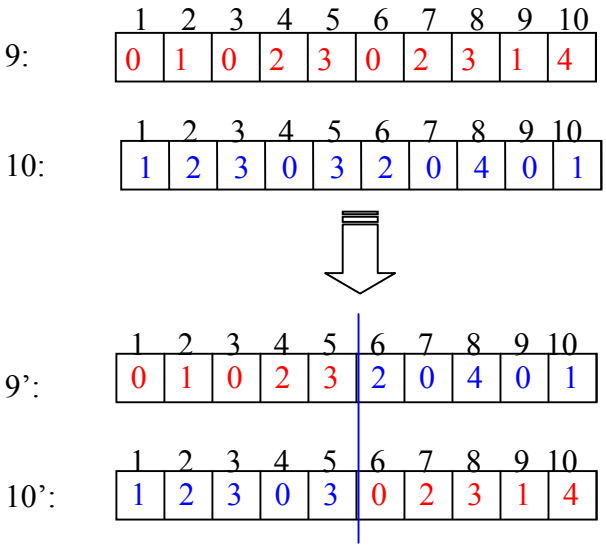
Cromossoma 3: 0 4 1 1 3 0 0 2 1 2 (cromossoma 12) *Fit*: 0.08696
Cromossoma 4: 0 1 0 2 3 0 2 3 1 4 (cromossoma 13) *Fit*: 0.08696
Cromossoma 5: 1 2 3 3 0 0 0 1 0 4 (cromossoma 9) *Fit*: 0.04348
Cromossoma 6: 1 2 0 4 0 1 0 3 0 3 (cromossoma 3) *Fit*: 0.06522
Cromossoma 7: 0 1 0 2 3 0 2 3 1 4 (cromossoma 13) *Fit*: 0.08696
Cromossoma 8: 4 2 3 0 3 0 1 4 1 1 (cromossoma 15) *Fit*: 0.04348
Cromossoma 9: 0 1 0 2 3 0 2 3 1 4 (cromossoma 13) *Fit*: 0.08696
Cromossoma 10: 1 2 3 0 3 2 0 4 0 1 (cromossoma 16) *Fit*: 0.04348
Cromossoma 11: 1 2 0 4 0 1 0 3 0 3 (cromossoma 3) *Fit*: 0.06522
Cromossoma 12: 0 1 0 2 3 0 2 3 1 4 (cromossoma 13) *Fit*: 0.08696
Cromossoma 13: 4 3 0 1 0 1 0 2 0 2 (cromossoma 4) *Fit*: 0.06522
Cromossoma 14: 4 3 0 1 0 1 0 2 0 2 (cromossoma 4) *Fit*: 0.06522
Cromossoma 15: 0 1 0 2 3 0 2 3 1 4 (cromossoma 13) *Fit*: 0.08696
Cromossoma 16: 1 2 3 3 0 0 0 1 0 4 (cromossoma 9) *Fit*: 0.04348
Cromossoma 17: 4 3 0 1 2 3 0 0 1 2 (cromossoma 20) *Fit*: 0.04348
Cromossoma 18: 1 2 0 4 0 1 0 3 0 3 (cromossoma 3) *Fit*: 0.06522
Cromossoma 19: 0 4 1 2 3 1 1 3 1 0 (cromossoma 1) *Fit*: 0.10870
Cromossoma 20: 1 2 0 4 0 1 0 3 0 3 (cromossoma 3) *Fit*: 0.06522

Passo 6.2.3 (Cruzamento dos cromossomas selecionados no passo anterior): Considere a probabilidade de cruzamento (p_c) igual a 0,25, ou seja, espera-se que 25% dos cromossomas sofram cruzamento. Para cada cromossoma é associado um número aleatório; se esse número for menor que o p_c o cromossoma é selecionado para cruzamento. Suponha que os números aleatórios sejam:

Cromossoma 1: 0.899110
Cromossoma 2: 0.322802
Cromossoma 3: 0.862941
Cromossoma 4: 0.763110
Cromossoma 5: 0.966640
Cromossoma 6: 0.274332
Cromossoma 7: 0.396042
Cromossoma 8: 0.836142
Cromossoma 9: 0.020803
Cromossoma 10: 0.246993
Cromossoma 11: 0.052760
Cromossoma 12: 0.507681
Cromossoma 13: 0.422738
Cromossoma 14: 0.000334
Cromossoma 15: 0.150121
Cromossoma 16: 0.709295
Cromossoma 17: 0.312469
Cromossoma 18: 0.395377
Cromossoma 19: 0.547683
Cromossoma 20:0.260987

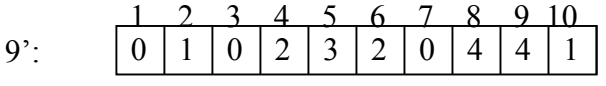
Então, os cromossomas selecionados para cruzamento são 9, 10, 11, 14 e 15. Como o número de cromossomas para cruzamento é ímpar, escolhe-se

aleatoriamente um cromossoma para completar os pares. Considere que o cromossoma 20 seja selecionado. Os cromossomas são então emparelhados aleatoriamente e para cada par será gerado um número inteiro aleatório no intervalo [1..9] (10 é o comprimento total – número de tarefas excluindo as tarefas fantasmas – em um cromossoma) correspondente ao ponto de cruzamento. Considere que o primeiro par de cromossomas seja 9 e 10 e que o ponto de cruzamento seja 5. Então:



Após o cruzamento, se a capacidade de algum processador tiver sido violada, faz-se a reparação nos cromossomas, retirando alguma tarefa e designando-a a outro processador. **Todas estas decisões são feitas de forma aleatória**. Por exemplo:

- Cromossoma 9': processador 0 com a capacidade violada.
- Tarefas no processador 0: 1, 3, 7, 9. Seleciona-se a tarefa 9 para retirada do processador 0.
- Processadores que podem receber tarefa 9: 3 e 4. Seleciona-se processador 4.

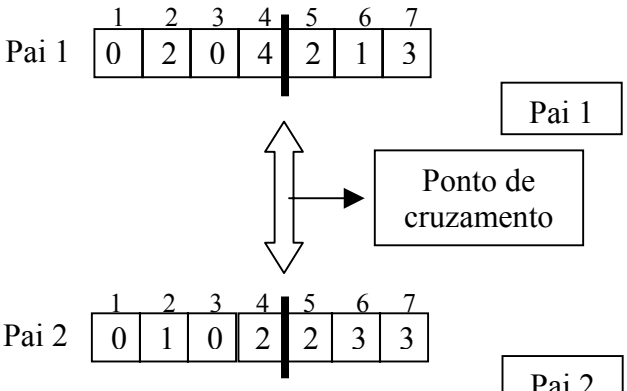


- Cromossoma 9': processador 1 com a capacidade violada.
- Tarefas no processador 1: 2 e 10. Seleciona-se tarefa 2 para retirada do processador 1.
- Processadores que podem receber tarefa 2: nenhum.
- Cria-se um novo processador (processador 5) para receber tarefa 2.

	1	2	3	4	5	6	7	8	9	10
9':	0	5	0	2	3	2	0	4	4	1

O cromossoma 9' foi factibilizado, ou seja, as capacidades de todos os seus processadores foram respeitadas. O cruzamento e reparação são realizados para todos os pares de cromossomas que participam do cruzamento.

Após a reparação do cruzamento, verifica-se a existência de processadores vazios. Como estas situações podem resultar em atrasos adicionais desnecessários, estes processadores são preenchidos com as tarefas dos processadores imediatamente subseqüentes, os quais, por sua vez, tornam-se vazios. Aplicando este processo de forma consecutiva, os últimos processadores tornam-se vazios, e são então eliminados. Este procedimento não implica em prejuízo à qualidade da solução. Considere os dados da TABELA 3.1 do capítulo 3 e os cromossomas pais da FIGURA 4.4:



Pai 1

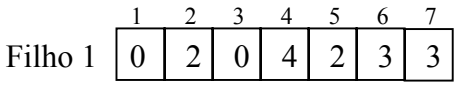
Processador	tp, md, mp
0	90, 80, 70
1	30, 10, 20
2	90, 60, 60
3	50, 40, 60
4	80, 20, 20

Pai 2

Processador	tp, md, mp
0	90, 80, 70
1	70, 50, 50
2	100, 30, 30
3	80, 50, 80

FIGURA 4.4 - Cruzamento em um ponto.

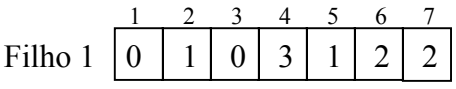
Com este cruzamento, Filho 1 corresponde ao cromossoma da FIGURA 4.5, onde o processador 1 está vazio.



Processador	tp, md, mp
0	90, 80, 70
1	0, 0, 0
2	90, 60, 60
3	80, 50, 80
4	80, 20, 20

FIGURA 4.5 - Cromossoma filho com processador vazio.

O processador 1 vazio é preenchido com as tarefas do processador imediatamente subseqüente, os quais, por sua vez, tornam-se vazios. Aplicando este processo de forma consecutiva, o último processador torna-se vazio, e é então eliminado conforme mostra a FIGURA 4.6.



Processador	tp, md, mp
0	90, 80, 70
1	90, 60, 60
2	80, 50, 80
3	80, 20, 20

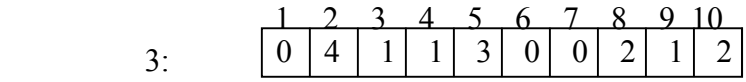
FIGURA 4.6 - Cromossoma filho após reparação.

Feitas as factibilizações necessárias, temos a seguinte população após o cruzamento:

Cromossoma 1: 3 1 2 2 0 4 1 0 0 3 0
Cromossoma 2: 3 1 2 2 0 4 1 0 0 3 0
Cromossoma 3: 0 4 1 1 3 0 0 2 1 2 0
Cromossoma 4: 0 1 0 2 3 0 2 3 1 4 0
Cromossoma 5: 1 2 3 3 0 0 0 1 0 4 0
Cromossoma 6: 1 2 0 4 0 1 0 3 0 3 0
Cromossoma 7: 0 1 0 2 3 0 2 3 1 4 0
Cromossoma 8: 4 2 3 0 3 0 1 4 1 1 0
Cromossoma 9: 0 5 0 2 3 2 0 4 4 1 0
Cromossoma 10: 1 2 3 0 3 0 2 4 1 4 0
Cromossoma 11: 1 3 0 4 0 1 0 2 0 2 0
Cromossoma 12: 0 1 0 2 3 0 2 3 1 4 0
Cromossoma 13: 4 3 0 1 0 1 0 2 0 2 0
Cromossoma 14: 1 2 0 4 0 1 0 3 0 3 0
Cromossoma 15: 4 1 0 2 0 1 2 3 0 3 0
Cromossoma 16: 1 2 3 3 0 0 0 1 0 4 0
Cromossoma 17: 4 3 0 1 2 3 0 0 1 2 0
Cromossoma 18: 1 2 0 4 0 1 0 3 0 3 0
Cromossoma 19: 0 4 1 2 3 1 1 3 1 0 0
Cromossoma 20: 1 2 0 4 3 0 2 3 1 0 0

Passo 6.2.4 (Mutação dos cromossomas): Considere que a probabilidade de mutação (p_m) adotada seja 0.02, ou seja, espera-se 2% das tarefas sofram mutação. Para cada tarefa na população, é gerado um número aleatório *r*. Se este número for menor que 0.02 a tarefa associada sofrerá mutação.

Suponha que uma das tarefas selecionadas para sofrer mutação seja a tarefa 9 do cromossoma 3.



A tarefa deve ser retirada do processador corrente e designada a um outro processador. As **decisões são feitas aleatoriamente**. Entretanto, algumas regras foram definidas:

- Se a tarefa está no último processador da linha, verifica-se se esta pode ser designada a algum outro processador existente. Se não puder ser designada, a mutação não é aplicada.
- Se a tarefa está em um processador que não seja o último (no exemplo acima, o último seria o processador 4) e esse processador ficar vazio com a retirada, a mutação não é aplicada.

Note que em operações de mutação retirada/inserção, um processador vazio surge quando a tarefa a ser retirada é a única designada a este processador. Portanto, nestas operações de mutação é fácil prever a ocorrência de processadores vazios. Caso a mutação provoque o esvaziamento do último processador da linha, a mutação é aplicada uma vez que o processador vazio pode ser eliminado imediatamente após a mutação da tarefa a um processador admissível, sem prejuízo à qualidade da solução. Se não houver processadores admissíveis para designação da tarefa, a mutação não é aplicada, pois a tarefa seria redesignada a um novo processador no fim da linha (onde a tarefa já está). Se o processador a ficar vazio não for o último, a mutação simplesmente não é aplicada para evitar o procedimento de eliminação de processadores. Em operações de cruzamento não é possível prever a geração de processadores vazios, daí a necessidade de eliminá-los após a realização do cruzamento.

Para o exemplo acima:

- Processador da tarefa 9 : processador 1 $\Rightarrow$ não é o último e caso a tarefa seja retirada, não ficará vazio.
- Processadores que podem receber a tarefa 9: 3 e 4. Seleciona-se o processador 3.

3':

1	2	3	4	5	6	7	8	9	10
0	4	1	1	3	0	0	2	3	2

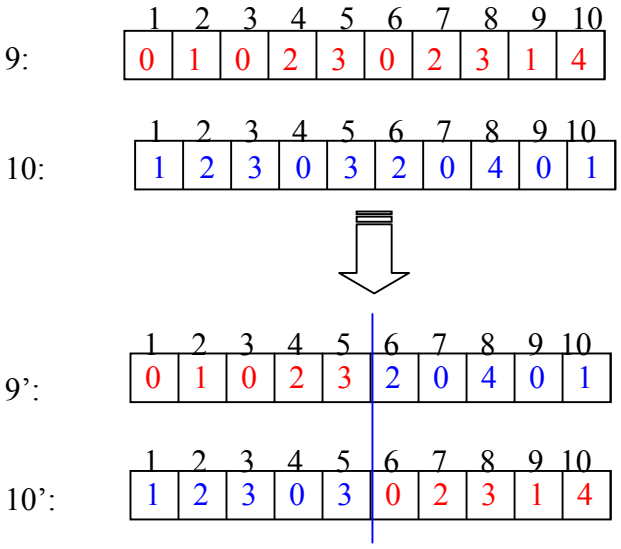
Para as outras tarefas faz-se a mutação seguindo o mesmo procedimento.

Após estes passos, têm-se a 1ª Geração para a qual se repetirá o processo descrito:

Cromossoma 1: 3 1 2 2 0 4 1 0 0 3 Atraso: 9
Cromossoma 2: 3 1 2 2 0 4 1 0 0 3 Atraso: 9
Cromossoma 3: 0 4 1 1 3 0 0 2 3 2 Atraso: 7
Cromossoma 4: 0 1 0 2 3 0 2 3 1 4 Atraso: 7
Cromossoma 5: 1 2 3 3 0 0 0 1 0 4 Atraso: 9
Cromossoma 6: 1 2 0 4 0 1 0 3 0 3 Atraso: 8
Cromossoma 7: 0 1 0 2 3 0 2 3 1 4 Atraso: 7
Cromossoma 8: 4 2 3 0 3 0 1 4 1 1 Atraso: 9
Cromossoma 9: 0 5 0 2 3 2 0 4 4 1 Atraso: 9
Cromossoma 10: 1 2 3 5 3 0 2 4 1 4 Atraso: 11
Cromossoma 11: 1 3 0 4 0 1 0 2 0 2 Atraso: 8
Cromossoma 12: 0 1 0 2 3 0 2 3 1 4 Atraso: 7
Cromossoma 13: 4 3 0 1 0 1 0 2 0 2 Atraso: 8
Cromossoma 14: 1 2 0 4 0 1 0 3 0 3 Atraso: 8
Cromossoma 15: 4 1 0 2 0 1 2 3 0 3 Atraso: 7
Cromossoma 16: 1 2 3 3 0 0 0 1 0 4 Atraso: 9
Cromossoma 17: 4 3 0 1 2 3 0 0 1 2 Atraso: 9
Cromossoma 18: 1 2 0 4 0 1 0 3 0 3 Atraso: 8
Cromossoma 19: 0 4 1 2 3 1 1 5 1 0 Atraso: 7
Cromossoma 20: 1 2 0 4 3 0 2 3 1 0 Atraso: 8

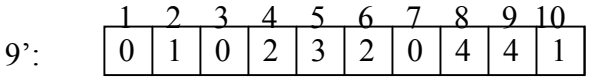
4.3 Algoritmo INTELIGENTE 1 – cruzamento e mutação

Passo 6.2.3 (Cruzamento dos cromossomas): Da mesma forma que na iteração ilustrativa do algoritmo puro, considere que o primeiro par de cromossomas seja 9 e 10 e que o ponto de cruzamento seja 5. Então:



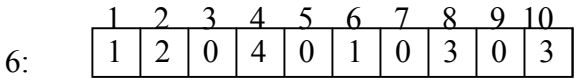
Após o cruzamento, se a capacidade de algum processador tiver sido violada, faz-se a reparação nos cromossomas retirando aleatoriamente alguma tarefa e designando-a, desta vez, dentre os processadores que podem receber a tarefa**, àquele que resulta no menor atraso possível**. Por exemplo:

- Cromossoma 9: processador 0 com a capacidade violada.
- Tarefas no processador 0: 1, 3, 7, 9. Seleciona-se a tarefa 9 para retirada do processador 0.
- Processadores que podem receber tarefa 9: 3 e 4.
- A designação da tarefa tanto ao processador 3 como ao processador 4, resulta em atraso de 8. Seleciona-se processador 4 (empates são decididos aleatoriamente).



Passo 6.2.4 (Mutação dos cromossomas)

Suponha que uma das tarefas selecionadas para sofrer mutação seja a tarefa 3 do cromossoma 6.



A tarefa deve ser retirada do processador corrente, e designada, dentre os processadores que podem recebê-la**, àquele que resulta no menor atraso possível**. As regras do algoritmo puro em relação ao esvaziamento de processadores são mantidas.

Para o exemplo acima:

- Processador da tarefa 3 : processador 0 $\Rightarrow$ não é o último e caso a tarefa seja retirada, não ficará vazio.
- Processadores que podem receber a tarefa 3: 1 e 4.
- A designação da tarefa ao processador 1 resulta em atraso de 8. A designação da tarefa ao processador 4 resulta em atraso de 9. Seleciona-se processador 1 (empates são resolvidos aleatoriamente).

	1	2	3	4	5	6	7	8	9	10
6':	1	2	1	4	0	1	0	3	0	3

4.4 Algoritmo INTELIGENTE 2 - cruzamento e mutação

Passo 6.2.3 (Cruzamento dos cromossomas): Mais uma vez, considere que o primeiro par de cromossomas seja 9 e 10 e que o ponto de cruzamento seja 5.:

	1	2	3	4	5	6	7	8	9	10
9:	0	1	0	2	3	0	2	3	1	4
10:	1	2	3	0	3	2	0	4	0	1

	1	2	3	4	5	6	7	8	9	10
9':	0	1	0	2	3	2	0	4	0	1
10':	1	2	3	0	3	0	2	3	1	4

Após o cruzamento, considere que a capacidade de um dado processador foi violada. Dentre as tarefas deste processador e dentre os processadores que podem receber cada uma delas, a reparação do cromossoma é feita **obtendo o par (tarefa, novo processador) que resulta no menor atraso possível**. Por exemplo:

- Cromossoma 9: processador 0 com a capacidade violada.
- Tarefas no processador 0: 1, 3, 7, 9.
- Tarefa 1 - Processadores que podem recebê-la: 3 e 4. A designação da tarefa 1 tanto ao processador 3 como ao processador 4, resulta em atraso de 10. O processador preferencial para designação é 4 (empates resolvidos aleatoriamente).
- Tarefa 3 - Processadores que podem recebê-la: 3 e 4. A designação da tarefa 1 tanto ao processador 3 como ao processador 4, resulta em atraso de 8. O processador preferencial para designação é 4.
- Tarefa 7 - Processadores que podem recebê-la: 1 e 4. A designação da tarefa 7 ao processador 1 resulta em atraso de 7. A designação da tarefa 7 ao processador 4 resulta em atraso de 8. O processador preferencial para designação é 1.
- Tarefa 9 - Processadores que podem recebê-la: 3 e 4. A designação da tarefa 9 tanto ao processador 3 como ao processador 4, resulta em atraso de 8. O processador preferencial para designação é 3.

O par selecionado para reparação é (7,1), uma vez que tal designação resulta no menor atraso total.

9':

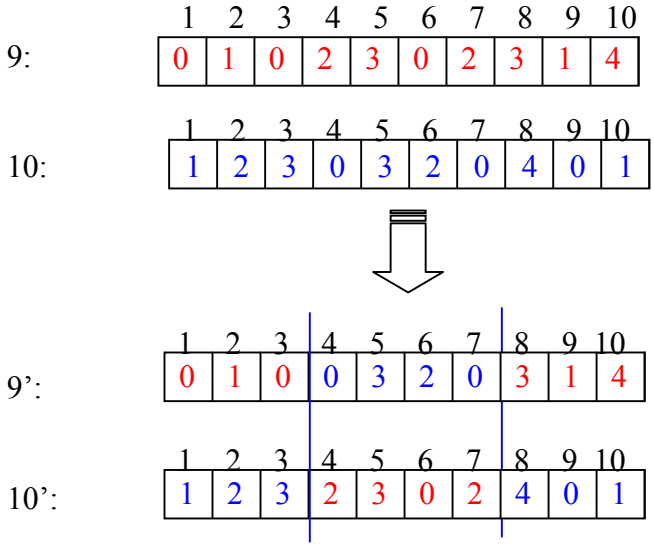
1	2	3	4	5	6	7	8	9	10
0	1	0	2	3	2	1	4	0	1

Passo 6.2.4 (Mutação dos cromossomas): A mutação é feita como descrito no algoritmo inteligente 1.

4.5 Algoritmo INTELIGENTE 3 - cruzamento e mutação

Este algoritmo difere de inteligente 2 no operador de cruzamento. Ao invés de utilizar cruzamento em um ponto, utilizou-se cruzamento em dois pontos, isto é, dados dois cromossomas pais, define-se aleatoriamente dois pontos de quebra nos mesmos, resultando em 3 subtiras. A 1ª e a 3ª subtira de um dos pais são intercaladas com a 2ª subtira do outro pai gerando assim dois cromossomas filhos.

Passo 6.2.3 (Cruzamento dos cromossomas): Mais uma vez, considere que o primeiro par de cromossomas seja 9 e 10 e que os pontos de cruzamento sejam 3 e 7. Então:



Após o cruzamento, é feita a reparação como no inteligente 2, isto é, **obtendo o par (tarefa, novo processador) que resulta no menor atraso possível**.

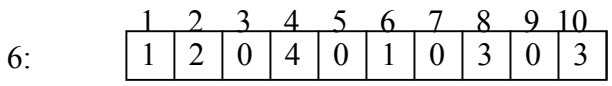
Passo 6.2.4 (Mutação dos cromossomas): A mutação é feita como descrito no algoritmo inteligente 1.

4.6 Algoritmo INTELIGENTE 4 - cruzamento e mutação

Passo 6.2.3 (Cruzamento dos cromossomas): O cruzamento e sua reparação é feita de forma similar ao descrito no algoritmo inteligente 2, isto é, cruzamento em um ponto e a reparação **obtendo o par (tarefa, novo processador) que resulta no menor atraso possível**.

Passo 6.2.4 (Mutação dos cromossomas): Ao invés de utilizar mutação retirada/inserção como nos algoritmos anteriores, é utilizada mutação troca, ou seja, a troca de tarefas que estão em processadores distintos.

Suponha que uma das tarefas selecionadas para sofrer mutação seja a tarefa 3 do cromossoma 6.



A tarefa selecionada para mutação deve ser retirada do processador corrente, e trocada com outra tarefa que esteja em um processador diferente desde que a troca não viole a capacidade de nenhum dos dois processadores. Dentre as tarefas possíveis para serem trocadas com a tarefa selecionada, escolhe-se aquela cuja troca resulte na solução de menor atraso.

Para o exemplo acima:

- Tarefas que podem ser trocadas com a tarefa 3: 1 e 6.
- Troca da tarefa 3 com a tarefa 1 resulta em um atraso de 7 ciclos.
- Troca da tarefa 3 com a tarefa 6 resulta em um atraso de 8 ciclos.

Assim, a tarefa 3 é designada ao processador 1 e a tarefa 1 é designada ao processador 0.

	1	2	3	4	5	6	7	8	9	10
6':	0	2	1	4	0	1	0	3	0	3

4.7 Algoritmo INTELIGENTE 5 - cruzamento e mutação

Passo 6.2.3 (Cruzamento dos cromossomas): O cruzamento e sua reparação é feita de forma similar ao descrito no algoritmo inteligente 3, isto é, cruzamento em dois pontos e a reparação **obtendo o par (tarefa, novo processador) que resulta no menor atraso possível**.

Passo 6.2.4 (Mutação dos cromossomas): A mutação é feita como descrito no algoritmo inteligente 4, isto é, troca entre tarefas.

4.8 Experimentos computacionais

Em todos os experimentos descritos neste capítulo, os algoritmos genéticos foram implementados em Delphi 5.0 e executados em um microcomputador Pentium II, 333 MHZ, 64 Mbytes de RAM. Os experimentos compreendem um conjunto de diagramas de tarefas com diferentes níveis de conectividade e topologias, obtidos em CHINNECK *et al.*(2003) e através de um gerador aleatório de grafos de

tarefas (LAVOIE,1999). Foi realizada a ordenação topológica das tarefas antes da definição do vetor de representação dos cromossomas. As alturas, larguras, grau das tarefas (vértices) dos diagramas, etc., foram gerados aleatoriamente em faixas especificadas. A especificação dos parâmetros é dada abaixo:

- [Número mínimo de vértices, Número máximo de vértices] = [5,100] ou [5, 200], dependendo da bateria de experimentos.
- [Altura mínima, Altura máxima] = [10,20] ou [20,30] ou [30, 40] ou [40, 50].
- [Largura mínima, Largura máxima] = [5,10].
- Número de vértices em cada nível do diagrama = 0,75*Largura.
- Grau máximo dos vértices = 0,5*Largura.
- [Grau mínimo, Grau máximo] = [1, Grau máximo]

Os valores fixados dos parâmetros dos algoritmos genéticos são:

- Tamanho da população (*m*) = 20 cromossomas.
- N^{o} máximo de gerações (*maxger*) = 2000.

Para efeitos de comparação de desempenho, o conjunto de diagramas foi também resolvido pelo algoritmo de CHINNECK *et* al. (2003), cuja implementação (em linguagem C) foi disponibilizada pelos autores. Utilizou-se o compilador Lcc-Win32 e as execuções se deram no mesmo computador utilizado nos experimentos com os algoritmos genéticos.

4.8.1 Primeira bateria de experimentos – Algoritmos PURO e INTELIGENTE 1 a 5

Inicialmente foram realizados experimentos com 10 diagramas-teste (5 a 96) tarefas a fim de determinar os valores de probabilidade de cruzamento (p_c) e probabilidade de mutação (p_m) mais adequados. Esses diagramas foram divididos em 5 grupos: Grupo 1 (5 a 10 tarefas), Grupo 2 (11 a 15 tarefas), Grupo 3 (16 a 25 tarefas), Grupo 4 (26 a 50 tarefas) e Grupo 5 (51 a 100 tarefas). A divisão em grupos buscou analisar a relação entre os valores de p_c e p_m e o tamanho do problema (número de

tarefas). Os algoritmos foram aplicados com cada variação paramétrica de p_m entre 0,01 e 0,05 (incrementos de 0,01), e p_c entre 0,1 e 1,0 (incrementos de 0,1). Concluiu-se que para todos os algoritmos, pequeníssimas probabilidades de mutação (p_m) resultam nas melhores performances, e que quanto maior o número de tarefas menor deve ser o valor deste parâmetro. Com base nesses resultados, selecionou-se p_m = 0,01 para os algoritmos puro, inteligente 4 e inteligente 5, e p_m=0,03 para todos os outros algoritmos nos experimentos posteriores. No que diz respeito à probabilidade de cruzamento p_c, selecionou-se p_c=0,4 (algoritmo puro), p_c=0,3 (inteligente 1), p_c=0,8 (inteligente 2, inteligente 3 e inteligente 5) e p_c=1,0 (inteligente 4). A TABELA 4.1 resume as características dos algoritmos propostos.

TABELA 4.1 – Características dos algoritmos propostos.

ALGORITMO	CRUZAMENTO	MUTAÇÃO	PROBABILIDADE DE CRUZAMENTO	PROBABILIDADE DE MUTAÇÃO
PURO	1 ponto	Inserção	0,4	0,01
INTELIGENTE 1	1 ponto	Inserção	0,3	0,03
INTELIGENTE 2	1 ponto	Inserção	0,8	0,03
INTELIGENTE 3	2 pontos	Inserção	0,8	0,03
INTELIGENTE 4	1 ponto	Troca	1,0	0,01
INTELIGENTE 5	2 pontos	troca	0,8	0,01

Com estas probabilidades de cruzamento e mutação, foram então realizados experimentos com outros 14 diagramas de 5 a 96 tarefas. A TABELA 4.2 apresenta os resultados dos algoritmos e do algoritmo de CHINNECK *et* al. (2003). Também foram incluídas as melhores soluções obtidas pelo algoritmo inteligente 2 com outros valores de parâmetros (inteligente 2'). A coluna $\overline{Atraso}$ apresenta o atraso médio e $\overline{\#\,proc}$ indica o número médio de processadores das soluções obtidas pelos algoritmos para os 24 diagramas. $\Delta\overline{Atraso}$ é a diferença entre $\overline{Atraso}$ de cada algoritmo e o atraso médio das soluções obtidas com o algoritmo de CHINNECK *et* al.(2003). Similarmente, $\Delta\overline{\#\,proc}$ é a diferença entre $\overline{\#\,proc}$ de cada algoritmo e o número médio de

processadores das soluções geradas pelo algoritmo de CHINNECK *et* al. (2003). $\overline{T}$ é o tempo computacional médio para obtenção da melhor solução da busca. As três últimas colunas da tabela indicam o número de problemas em que os algoritmos obtiveram soluções com o mesmo valor de atraso (coluna E), menor valor de atraso (coluna G) e maior valor de atraso (coluna P) que o algoritmo de CHINNECK *et al* (2003).

TABELA 4.2 - Resultados computacionais – 24 diagramas (5 a 96 tarefas).

Algoritmo	$\overline{Atraso}$	$\overline{\#\ proc}$	$\Delta\overline{Atraso}$	$\Delta\overline{\#\ proc}$	$\overline{T}$ (seg)	E	G	P
Puro	11,71	10,04	+1,54	-0,67	1,62	8	2	14
Inteligente 1	10,88	10,29	+0,71	-0,42	14,67	12	5	7
Inteligente 2	10,29	9,96	+0,13	-0,75	37,20	15	5	4
Inteligente 3	10,29	10,25	+0,13	-0,46	155,18	13	6	5
Inteligente 4	9,92	10,04	-0.25	-0,67	247,34	16	6	2
Inteligente 5	10,17	9,71	0	-1,00	126,82	16	4	4
Inteligente 2´	9,71	9,88	-0,46	-0,83	155,69	15	8	1
Chinneck *et* al.	10,17	10,71	-	-	< 0.1			

Na coluna $\overline{\#\ proc}$ da TABELA 4.2, observa-se que todos os 6 algoritmos genéticos geraram soluções com um número médio de processadores inferior ao obtido com a heurística de CHINNECK *et al.*(2003). A heurística construtiva realiza as designações seqüencialmente e não possui nenhum mecanismo de refinamento posterior das soluções com vistas à redução do número de processadores.

Em relação ao atraso, os algoritmos não conseguiram superar, de forma geral, o desempenho da heurística de CHINNECK *et al.* (2003). (coluna $\Delta\overline{Atraso}$). Como esperado, o algoritmo genético puro teve a pior performance. Os melhores resultados foram obtidos com os algoritmos inteligente 2, inteligente 3, inteligente 4 e inteligente 5. O algoritmo inteligente 5, em particular, foi o único que gerou atrasos em média inferiores aos obtidos por CHINNECK *et al.* (2003). Entretanto, sua aplicabilidade é reduzida em virtude dos tempos computacionais excessivos. O algoritmo inteligente 2, apesar de não obter um atraso médio inferior ao da heurística de CHINNECK *et al* .(2003), encontrou soluções com um menor valor de atraso em 21% dos exemplos, a um tempo computacional que pode ser considerado razoável (37 segundos) para muitas aplicações PDS. A aplicação deste algoritmo com outros valores de parâmetros genéticos p_c e p_m (inteligente 2') resultou em soluções com atraso igual

ou inferior aos obtidos pela heurística de CHINNECK *et al.* (2003) em 23 dos 24 exemplos.

O impacto da utilização de 2 pontos de cruzamento não ficou estabelecido; aumentou o tempo computacional mantendo o atraso das soluções com o algoritmo inteligente 3 e diminuiu o tempo computacional aumentando o atraso das soluções com o algoritmo inteligente 5. A utilização de mutação troca resultou no aumento do tempo computacional com melhoria do atraso das soluções (inteligente 4).

As conclusões principais destes experimentos são descritas a seguir:

- Quanto mais determinísticas as escolhas no procedimento de reparação, maior é a qualidade das soluções geradas e maior o tempo computacional.
- A utilização do cruzamento 1P e mutação retirada/inserção resultou no melhor *trade-off* entre tempo computacional e qualidade das soluções.
- A ordenação topológica das tarefas antes da definição do vetor de representação dos cromossomas (seção 3.1.1) não trouxe ganhos em termos de qualidade da solução. Entretanto, verificou-se reduções de tempo computacional de até 60% em relação às versões dos algoritmos que não a utilizaram. A aplicação da ordenação topológica reduz o tempo dispendido em etapas de reparação de soluções. Como exemplo, o algoritmo inteligente 2 sem ordenação topológica utilizou em média 66,83% do tempo computacional com reparação de soluções. Com a ordenação, este tempo diminuiu para 50,23%.

4.8.2. Segunda bateria de experimentos – Algoritmos INTELIGENTE 6 a 9

Uma das questões mais importantes ao se utilizar meta-heurísticas consiste na elaboração de implementações cujos parâmetros sejam claramente estabelecidos e adaptáveis a problemas com diferentes características. O fato do melhor algoritmo da primeira bateria de experimentos ter gerado para a maioria dos exemplos soluções com qualidade igual ou inferior ao do algoritmo de CHINNECK *et al.* (2003), não é de grande valia, pois essas soluções foram obtidas com diferentes valores de p_c e

p_m. Em particular, o valor mais adequado do parâmetro p_m pareceu ser o mais sensível a variações do tamanho (definido como o número de tarefas) do diagrama.

Com base nessas observações, foi elaborada uma nova implementação (algoritmo inteligente 6). Neste algoritmo, a probabilidade de mutação é obtida por uma função do número de tarefas do diagrama. A função foi determinada da seguinte forma: para os 24 arquivos da primeira bateria, verificou-se que em média os melhores resultados eram obtidos quando se aplicava a mutação a 30 tarefas. Ou seja, independentemente do número de tarefas do diagrama, parece interessante que 30 delas sofram mutação. Portanto:

$$p_m \times n \times m = 30$$

onde m é o tamanho da população e n é o número de tarefas.

Como o tamanho da população utilizado é 20, então p_m é dado por:

$$p_m = \frac{30}{n \times 20} = \frac{1{,}5}{n}$$

Além dos valores de p_m obtidos com essa função, o algoritmo inteligente 6 utiliza:

- Operador de mutação retirada/inserção. A tarefa selecionada é designada ao processador admissível que resulta na maior redução do atraso.
- Operador de cruzamento de 1 ponto com construção (1PC) com p_c = 0,8. Conforme mencionado na seção 3.1.6, o operador 1PC não requer a aplicação de procedimentos de reparação (o que implica na redução do tempo de execução) combinada à vantagem de promover designações com menor atraso. O valor de p_c foi baseado nos experimentos anteriores.
- População inicial gerada aleatoriamente.

A TABELA 4.3 apresenta os resultados do algoritmo inteligente 6 e do algoritmo de CHINNECK *et al.* (2003) para os 24 diagramas.

TABELA 4.3 - Resultados computacionais – 24 diagramas (5 a 96 tarefas).

Algoritmo	$\overline{Atraso}$	$\overline{\#\ proc}$	$\Delta\overline{Atraso}$	$\Delta\overline{\#\ proc}$	$\overline{T}$ (seg)	E	G	P
Inteligente 6	9,58	10,21	-0,59	-0,5	5,6	15	9	0
Chinneck *et al.*	10,17	10,71	-	-	0,31	-	-	-

Para todos os 24 diagramas, o algoritmo inteligente 6 obteve atrasos iguais ou menores que as obtidos com o algoritmo de CHINNECK *et al.* (2003). Este resultado melhora o atraso médio do melhor algoritmo da primeira bateria de testes em 6,9% e o tempo computacional requerido em 85%. Ou seja, resultou em menor atraso médio, em menor tempo. Além disso, o algoritmo inteligente 6 superou as soluções de CHINNECK *et al.* (2003) em 37,5% dos problemas.

Com base nesses resultados promissores, o algoritmo inteligente 6 foi testado para outros 93 diagramas de 5 a 176 tarefas, também gerados aleatoriamente. A TABELA 4.4 apresenta os resultados com estes e os 24 diagramas da primeira bateria, totalizando 117 diagramas. A TABELA 4.5 apresenta esses resultados em 5 grupos distintos definidos por uma faixa de número de tarefas, onde $\overline{\#\ tarefas}$ é o número médio de tarefas de cada grupo e # problemas é o número de problemas de cada grupo.

TABELA 4.4 - Resultados computacionais gerais– 117 diagramas (5 a 176 tarefas).

Algoritmo	$\overline{Atraso}$	$\overline{\#\ proc}$	$\Delta\overline{Atraso}$	$\Delta\overline{\#\ proc}$	$\overline{T}$ (seg)	E	G	P
Inteligente 6	25,42	27,11	-1,08	-2,66	48,88	29	77	11
Chinneck *et al.*	26,50	29,77	-	-	1,56	-	-	-

TABELA 4.5 - Resultados computacionais para 5 grupos de diagramas.

Grupo	$\overline{\#\ tarefas}$	# problemas	Chinneck *et al.* $\overline{Atraso}$ / $\overline{\#\ proc}$	Algoritmo Inteligente 6 $\overline{Atraso}$ / $\overline{\#\ proc}$	$\Delta\overline{Atraso}$	$\Delta\overline{\#\ proc}$	$\overline{T}$ (seg)	E	G	P
1	15,9	23	6,96/6,57	6,22/6,13	-0,74	-0,43	0,6	13	10	0
2	32,4	24	13,38/14,17	12,00/13,21	-1,38	-0,96	5,6	5	19	0
3	69,3	26	27,88/31,04	25,50/28,58	-2,38	-2,46	28	2	24	0
4	89,3	26	34,08/39,65	32,62/36,08	-1,46	-3,58	62	4	18	4
5	143,6	18	56,06/64,11	57,33/57,39	+1,28	-6,72	180	5	6	7

A TABELA 4.5 indica que o algoritmo inteligente 6 obteve valores de atraso igual ou inferior aos obtidos por CHINNECK *et al.* (2003) em todos os

problemas dos grupos 1 a 3 (até 80 tarefas). O desempenho do algoritmo inteligente 6 decaiu com os grupos 4 e 5 (mais de 80 tarefas).

Em 67 dos 117 diagramas, o algoritmo inteligente 6 encontrou a melhor solução em até 20 segundos. Nos 50 diagramas restantes estão incluídos 10 dos 11 diagramas para os quais o algoritmo inteligente 6 não conseguiu superar a solução encontrada pelo algoritmo de CHINNECK *et al.* (2003). Apenas um desses 11 diagramas possuem 46 tarefas enquanto os demais possuem mais de 80 tarefas.

Para esses 50 diagramas (considerados difíceis), foram elaboradas outras 3 implementações (algoritmos inteligente 7, 8 e 9) a partir do algoritmo inteligente 6, que visam investigar possíveis melhorias metodológicas. As características desses algoritmos são dadas na seção 4.8.2.1, a seguir.

4.8.2.1 População inicial mista e aplicação de buscas locais

Em alguns algoritmos, foi investigada a inclusão de soluções geradas determininisticamente na população inicial e a aplicação de buscas locais em cromossomas selecionados. No primeiro caso, parte da população é composta por soluções que correspondem à solução padrão de CHINNECK *et al.* (2003) e perturbações do mesmo que alteram pouco a qualidade da solução. As buscas locais, por sua vez, foram aplicadas sempre que tenha transcorrido um número pré-especificado de gerações sem melhoria do atraso, em um pequeno conjunto de cromossomas de menor atraso encontrados em todas as gerações até o momento. Para isso é feita uma lista das melhores soluções, por sua vez atualizada a cada intervalo de gerações sem melhoria. A vizinhança da busca local é descrita como se segue:

- Para cada um dos cromossomas selecionados, são computadas todas as possíveis *trocas* de tarefas entre processadores, e *inserçõe*s de uma tarefa em um novo processador (vizinhança da busca), e selecionada a troca ou inserção que resultar na maior diminuição do atraso.
- O procedimento é repetido a partir do cromossoma resultante até que um ótimo local tenha sido obtido.

A busca é realizada apenas para as soluções do conjunto que não tenham sido encontradas previamente. Sua aplicação em cromossomas mais promissores tende a

acelerar a convergência a soluções de maior qualidade. As características dos algoritmos são dadas a seguir:

- Algoritmo inteligente 7: algoritmo inteligente 6 com buscas locais. As buscas locais são feitas a partir das 5 soluções (25% da população) com menor atraso encontrados em todas as gerações até o momento, a cada 200 gerações sem melhoria (10% do número total de gerações). Os 5 ótimos locais substituem as 5 soluções de maior atraso da população atual.
- Algoritmo inteligente 8: algoritmo inteligente 6 com 6 soluções do algoritmo de CHINNECK *et al.* (2003) na população inicial. Estas soluções correspondem à solução padrão e 5 soluções decorrentes de perturbações do algoritmo, todas distintas.
- Algoritmo inteligente 9: algoritmo inteligente 6 com 6 soluções do algoritmo de CHINNECK *et al*. (2003) na população inicial e buscas locais.

A TABELA 4.6 apresenta os resultados obtidos pelos algoritmos inteligente 6, 7, 8 e 9. Assim como na TABELA 4.1, os valores apresentados nas colunas 4 e 5 representam o desempenho relativo de cada algoritmo ao da heurística de CHINNECK *et al.* (2003).

TABELA 4.6 - Resultados computacionais (problemas difíceis - 46 a 176 tarefas).

Algoritmo	$\overline{Atraso}$	$\overline{\#\ proc}$	$\Delta\overline{Atraso}$	$\Delta\overline{\#\ proc}$	$\overline{T}$ (seg)	E	G	P
Inteligente 6	38,88	40,84	-0,74	-4,16	107,22	8	32	10
Inteligente 7	36,42	41,36	-3,20	-3,64	220,29	3	45	2
Inteligente 8	37,76	42,62	-1,86	-2,38	58,17	11	39	0
Inteligente 9	35,74	42,74	-3,88	-2,26	191,23	1	49	0
Chinneck *et al.*	39,62	45,00	-	-	2,53	-	-	-

A TABELA 4.6 indica que os menores atrasos médios foram obtidos com o algoritmo inteligente 9 onde foram incorporadas algumas soluções iniciais de boa qualidade e busca local, a um custo computacional maior. O tempo médio gasto na busca local correspondeu a 24,4% do tempo total.

O algoritmo inteligente 8, que por sua vez incorpora algumas soluções iniciais de boa qualidade e nenhuma busca local, ofereceu o melhor *trade-off* entre

qualidade de solução e tempo computacional. Os algoritmos inteligente 8 e 9 obtiveram valores de atraso menor ou igual aos obtidos por CHINNECK *et al.* (2003) em todos os 50 diagramas. A eficácia dos algoritmos é observada no fato de ambos incluírem a solução padrão de CHINNECK *et al.* (2003) na população inicial, e ainda assim obterem melhorias com relação a estas soluções em uma parcela substancial do grupo de problemas.

As principais conclusões desses experimentos são as seguintes:

- para problemas até 80 tarefas, a combinação do operador de cruzamento 1PC com $p_m = \frac{1,5}{n}$ resulta em melhores soluções tanto em termos de atraso como de número de processadores, a um tempo computacional consideravelmente menor, em relação aos operadores 1P e 2P com p_m fixo.

Para esta combinação e problemas com mais de 80 tarefas:

- a inclusão de buscas locais resulta em melhores soluções tanto em termos de atraso como de número de processadores, a um tempo computacional maior.
- a inclusão de soluções de boa qualidade na população inicial resulta em melhores soluções tanto em termos de atraso como de número de processadores, em menor tempo computacional.

Com vistas a melhorias adicionais, o algoritmo que resultou em menores atrasos médios (inteligente 9) foi testado com outros 5 intervalos de gerações sem melhoria para realização das buscas locais. O intervalo de 50 gerações sem melhoria foi o que apresentou melhorias, ainda que pequenas, na qualidade da solução. Com este valor de parâmetro, o algoritmo 9 foi então aplicado a todos os 117 diagramas. Outro algoritmo que também apresentou bons resultados e o melhor *trade-off* entre qualidade de solução e tempo computacional - inteligente 8 - também foi aplicado a todos os 117 diagramas. Os resultados dos algoritmos inteligentes 8 e 9 foram comparados com o algoritmo de CHINNECK *et al*. (2003) e com um algoritmo de busca local *multiple starts*. Os experimentos e resultados são discutidos na próxima seção.

4.8.3 Terceira bateria de experimentos – Algoritmos INTELIGENTE 8 e 9

Até o momento, os resultados sugerem que para o problema aqui tratado, versões mais elaboradas de algoritmos genéticos apresentam uma melhor performance em termos de qualidade de solução em relação a uma heurística construtiva sofisticada. Uma outra questão que merece investigação é se esta superioridade é mantida em relação a outros algoritmos de busca, ou seja, algoritmos que, ao contrário da heurística de CHINNECK *et al*. (2003), permitem a melhoria das soluções iniciais.

Neste sentido, foi elaborado um algoritmo do tipo *multiple starts*, o qual consiste de iterativamente, gerar uma solução inicial seguida por um procedimento de busca. O procedimento de busca consiste em analisar todas as possíveis trocas de tarefas entre processadores e inserções de uma tarefa em um processador distinto (vizinhança) a cada iteração, e selecionar a troca ou inserção que resulta na maior diminuição do atraso. As soluções iniciais utilizadas pelo *multiple starts* são as 6 soluções geradas pelo algoritmo de CHINNECK *et al.* (2003) e perturbações do mesmo, e soluções geradas aleatoriamente. Para uma comparação mais justa, o critério de parada do algoritmo *multiple starts* corresponde ao tempo utilizado pelo algoritmo inteligente 9 para encontrar a melhor solução. O resultado do algoritmo é o melhor ótimo local encontrado em todas as execuções.

A TABELA 4.7 apresenta os resultados do algoritmo inteligente 8, do algoritmo inteligente 9 (com aplicação de busca local a cada 50 gerações sem melhoria), do algoritmo *multiple starts*, e da heurística de CHINNECK *et al*. (2003) para os 117 diagramas. Assim como na TABELA 4.6, os valores de $\Delta\overline{Atraso}$ e $\Delta\overline{\#proc}$ representam o desempenho relativo dos três primeiros algoritmos ao da heurística de CHINNECK *et al.* (2003) A TABELA 4.8 apresenta os resultados da heurística de CHINNECK *et al.* (2003) e do algoritmo inteligente 8, divididos em grupos de acordo com o número de tarefas, a TABELA 4.9 apresenta os resultados da heurística de CHINNECK *et al.* (2003) e do algoritmo inteligente 9, divididos em grupos de acordo com o número de tarefas, a TABELA 4.10 apresenta os resultados do algoritmo *multiple starts* e do algoritmo inteligente 8, divididos em grupos de acordo com o número de tarefas e a TABELA 4.11, por sua vez, apresenta os resultados do algoritmo *multiple*

starts e do algoritmo inteligente 9, também divididos em grupos de acordo com o número de tarefas. Nas TABELAS 4.10 e 4.11 as colunas $\Delta\overline{Atraso}$ e $\Delta\overline{\#\ proc}$ indicam respectivamente a diferença entre os atrasos médios e número médio de processadores obtidos com o algoritmo inteligente 8 (TABELA 4.10) e inteligente 9 (TABELA 4.11) com o algoritmo *multiple starts*.

TABELA 4.7 - Resultados computacionais gerais– 117 diagramas (5 a 176 tarefas).

Algoritmo	$\overline{Atraso}$	$\overline{\#\ proc}$	$\Delta\overline{Atraso}$	$\Delta\overline{\#\ proc}$	$\overline{T}$ (seg)	E	G	P
Inteligente 8	24,91	27,95	-1,62	-1,82	33,84	29	88	0
Inteligente 9	23,79	28,09	-2,71	-1,68	97,21	15	102	0
Multiple Starts	24,88	29,50	-1,62	-0,27	7,16	40	77	0
Chinneck *et al.*	26,50	29,77	-	-	1,56	-	-	-

TABELA 4.8 - Resultados computacionais para 5 grupos de diagramas – algoritmo INTELIGENTE 8 e Chinneck *et al.*

Grupo	$\overline{\#\ tarefas}$	# problemas	Chinneck *et al.* $\overline{Atraso}$ / $\overline{\#\ proc}$	Algoritmo Inteligente 8 $\overline{Atraso}$ / $\overline{\#\ proc}$	$\Delta\overline{Atraso}$	$\Delta\overline{\#\ proc}$	$\overline{T}$ (seg)	E	G	P
1	15,9	23	6,96/6,57	6,17/6,17	-0,83	-0,39	2,52	11	12	0
2	32,4	24	13,38/14,17	12,04/13,17	-1,33	-1,00	10,01	5	19	0
3	69,3	26	27,88/31,04	25,38/28,81	-2,50	-2,23	30,21	2	24	0
4	89,3	26	34,08/39,65	32,12/37,31	-1,96	-2,35	53,83	5	21	0
5	143,6	18	56,06/64,11	54,89/60,72	-1,17	-3,39	82,02	7	11	0

TABELA 4.9 - Resultados computacionais para 5 grupos de diagramas – algoritmo INTELIGENTE 9 e Chinneck *et al.*

Grupo	$\overline{\#\ tarefas}$	# problemas	Chinneck *et al.* $\overline{Atraso}$ / $\overline{\#\ proc}$	Algoritmo Inteligente 9 $\overline{Atraso}$ / $\overline{\#\ proc}$	$\Delta\overline{Atraso}$	$\Delta\overline{\#\ proc}$	$\overline{T}$ (seg)	E	G	P
1	15,9	23	6,96/6,57	6,13/6,17	-0,87	-0,39	3,59	10	13	0
2	32,4	24	13,38/14,17	11,79/13,04	-1,58	-1,13	11,54	3	21	0
3	69,3	26	27,88/31,04	24,46/28,81	-3,42	-2,23	71,07	0	26	0
4	89,3	26	34,08/39,65	30,92/37,35	-3,15	-2,31	122,15	0	26	0
5	143,6	18	56,06/64,11	51,00/61,78	-5,06	-2,33	332,77	0	18	0

TABELA 4.10 - Resultados computacionais para os 5 grupos de diagramas – algoritmo INTELIGENTE 8 e *mutliple starts*.

Grupo	$\overline{\#\text{ tarefas}}$	# problemas	*Multiple Starts* $\overline{Atraso}$ / $\overline{\#\ proc}$	Algoritmo Inteligente 8 $\overline{Atraso}$ / $\overline{\#\ proc}$	$\Delta\overline{Atraso}$	$\Delta\overline{\#\ proc}$	E	G	P
1	15,9	23	6,48/6,13	6,17/6,17	-0,31	0,04	16	6	1
2	32,4	24	12,88/13,96	12,04/13,17	-0,84	-0,79	8	16	0
3	69,3	26	26,04/30,85	25,38/28,81	-0,65	-2,04	5	16	5
4	89,3	26	32,19/39,58	32,12/37,31	-0,07	-2,27	10	7	9
5	143,6	18	52,17/63,56	54,89/60,72	+2,72	-2,84	2	0	16

TABELA 4.11 - Resultados computacionais para os 5 grupos de diagramas – algoritmo INTELIGENTE 9 e *mutliple starts*.

Grupo	$\overline{\#\text{ tarefas}}$	# problemas	*Multiple Starts* $\overline{Atraso}$ / $\overline{\#\ proc}$	Algoritmo Inteligente 9 $\overline{Atraso}$ / $\overline{\#\ proc}$	$\Delta\overline{Atraso}$	$\Delta\overline{\#\ proc}$	E	G	P
1	15,9	23	6,48/6,13	6,13/6,17	-0,35	0,04	17	6	0
2	32,4	24	12,88/13,96	11,79/13,04	-1,08	-0,92	4	20	0
3	69,3	26	26,04/30,85	24,46/28,81	-1,58	-2,04	5	21	0
4	89,3	26	32,19/39,58	30,92/37,35	-1,27	-2,23	5	20	1
5	143,6	18	52,17/63,56	51,00/61,78	-1,17	-1,78	4	13	1

Como já observado nas baterias de experimentos anteriores, os algoritmos 8 e 9 obtiveram ganhos expressivos em relação aos resultados de CHINNECK *et al.*(TABELA 4.7). O algoritmo inteligente 9 superou os resultados do algoritmo inteligente 8 em relação à qualidade das soluções, porém requerendo maior custo computacional. Com relação ao algoritmo inteligente 9, para os grupos com menor número de tarefas (grupos 1 e 2 - TABELA 4.9) a diferença em relação ao atraso é pequena, porém conforme aumenta o número de tarefas, a diferença de atraso médio também aumenta. O algoritmo inteligente 9 apresenta uma diferença de atraso de até 9 ciclos para o diagrama de 176 tarefas, e em média obtém uma diferença de 5,06 ciclos para o grupo 5 que contém diagramas com mais de 100 tarefas.

Na TABELA 4.7, observa-se que o algoritmo *multiple starts* produz, em média, soluções de qualidade superior às obtidas pela heurística de CHINNECK *et al.* (2003). Os tempos médios de execução até o melhor ótimo local de *multiple starts* foram bastante reduzidos, apresentando uma pequena variação - 5 a 20 segundos - entre os grupos de tarefas.

O algoritmo inteligente 8 obteve um desempenho considerado competitivo em relação ao algoritmo *multiple starts*. Apesar de apresentar atrasos

menores em 38,5% do total de diagramas (contra 26,5% de exemplos com atrasos maiores), o algoritmo 8 não conseguiu superar as soluções de *multiple starts* em nenhum dos diagramas do grupo 5 (veja TABELA 4.10). De fato, em 89% dos diagramas deste grupo, o algoritmo 8 foi superado por *multiple starts*. Concluiu-se, portanto, que o desempenho do algoritmo 8 em relação a *multiple starts* diminui com o aumento do número de tarefas.

Por outro lado, observa-se na TABELA 4.11 que o algoritmo inteligente 9 obteve soluções com atraso igual (35 diagramas) ou inferior (80 diagramas) às obtidas por *multiple starts*. Os 2 diagramas em que o algoritmo inteligente 9 obteve soluções de pior qualidade que as apresentadas pelo algoritmo *multiple starts* pertencem aos grupos 4 e 5, os quais contêm mais de 78 tarefas. Como as soluções iniciais geradas pelo algoritmo de CHINNECK *et al.* (2003) são utilizadas pelos dois algoritmos e o tempo de execução é o mesmo para ambos, esses resultados sugerem que o algoritmo *multiple starts* obtém boas soluções rapidamente, porém possibilidades de melhorias adicionais são também rapidamente esgotadas. De fato, os melhores ótimos locais foram obtidos a partir das soluções de CHINNECK *et al*. (as primeiras as serem utilizadas), e a aplicação de buscas locais nas demais soluções – geradas aleatoriamente – se revelaram infrutíferas.

O algoritmo inteligente 9, por outro lado, mostrou-se capaz de obter melhorias significativas em estágios avançados da busca a partir de uma população mista. Pode-se dizer que uma das razões de seu melhor desempenho seja o resultado da combinação de decisões aleatórias que conferem diversidade à população e de boas decisões determinísticas que intensificam a busca em regiões atraentes. Entretanto, o maior custo computacional requerido pode não ser desprezível.

5 CONCLUSÕES E PERSPECTIVAS

Neste trabalho, foi abordada a utilização de algoritmos genéticos/populacionais para a resolução do problema de designação de tarefas em multiprocessadores digitais de sinal com tempos de comunicação entre os processadores. Dez implementações caracterizadas por diferentes operadores de cruzamento, mutação, população inicial e incorporação ou não de buscas locais foram desenvolvidas para uma arquitetura particular de processamento digital de sinais. Em particular, foi elaborado um operador de cruzamento (1PC) específico para o problema. Resultados computacionais com 117 diagramas de tarefas foram apresentados e comparados entre os algoritmos, com a heurística construtiva de CHINNECK *et al.* (2003) e com um algoritmo de busca *multiple starts.*

Todos os 10 algoritmos obtiveram resultados expressivos na redução do número de processadores em relação aos obtidos por CHINNECK *et al.* (2003). Dentre os algoritmos Puro e inteligente 1 a 6, caracterizados por população inicial gerada aleatoriamente, operadores de cruzamento de propósito geral e procedimento de reparação de soluções infactíveis, observou-se que quanto mais determinísticas as escolhas no procedimento de reparação, maior é a qualidade das soluções geradas e maior o tempo computacional para obtê-las. Deste grupo, o algoritmo inteligente 6 é a implementação com melhor *trade-off* entre qualidade da solução (menor atraso) e tempo computacional, melhorando os valores de atraso da heurística construtiva em 65% dos casos. Dentre os algoritmos inteligente 7 a 9, caracterizados pelo uso do operador de cruzamento específico 1PC, observou-se que a adoção de busca local e população inicial mista resulta em soluções de maior qualidade a um maior tempo computacional. Deste grupo, o algoritmo inteligente 8 (população inicial mista) é o que apresentou o melhor *trade-off* entre qualidade da solução e tempo computacional. Do conjunto total de diagramas, inteligente 8 melhorou em 74% dos exemplos, os valores de atraso da heurística de CHINNECK *et al*. (2003), e em 38,5% dos exemplos, os valores de atraso do algoritmo *multiple starts*. O algoritmo inteligente 9 (população inicial mista e buscas locais), melhorou em 88% dos exemplos, os valores de atraso da heurística de CHINNECK *et al.*(2003), e em 68% dos exemplos, os valores de atraso do algoritmo *multiple starts*.

A maior limitação quanto à aplicabilidade dos algoritmos propostos diz respeito ao tempo computacional requerido, uma vez que este supera em várias ordens de magnitude, os tempos de execução da heurística construtiva e do algoritmo *multiple starts*. Para o algoritmo 8, os tempos médios de obtenção da melhor solução representaram um acréscimo de 2069% em relação à heurística de CHINNECK *et al*. (2003), e 373% em relação ao algoritmo *multiple starts*. Para o algoritmo 9, esses tempos representaram um acréscimo de 6131% em relação à heurística de CHINNECK *et al*. (2003), e 1257% em relação ao algoritmo *multiple starts*. Isso inviabiliza sua utilização em aplicações que exigem que a designação das tarefas aos processadores seja feita em tempo real. Neste trabalho, procurou-se diminuir os tempos de execução otimizando-se a etapa do cálculo do atraso. Especificamente, o atraso de um cromossoma filho foi obtido tomando-se os atrasos parciais até o ponto de cruzamento dos pais e calculando-se o atraso total a partir daí. De forma similar, pode-se obter diminuições adicionais de tempos de execução em outras etapas dos algoritmos. Uma outra possibilidade a ser investigada, é a redução da vizinhança na busca local. A troca de tarefas entre processadores poderia ser aplicada somente entre tarefas onde pelo menos uma delas pertença ao caminho crítico. Como a designação das tarefas dos caminhos críticos é que define o valor do atraso total, é possível que tal redução da vizinhança venha a diminuir o tempo computacional sem detrimento à qualidade da solução.

Uma das perspectivas de continuidade deste trabalho é a extensão dos algoritmos mais promissores (inteligente 6, inteligente 8 e inteligente 9) a outras arquiteturas de processamento paralelo. Sob esta perspectiva, torna-se necessário investigar mudanças das regras de designação especificadas para a atual arquitetura e que foram adotadas nos algoritmos genéticos propostos.

Uma das perspectivas de continuidade deste trabalho é a extensão dos algoritmos mais promissores (inteligente 6, inteligente 8 e inteligente 9) a outras arquiteturas de processamento paralelo. Sob esta perspectiva, torna-se necessário investigar mudanças das regras de designação especificadas para a atual arquitetura e que foram adotadas nos algoritmos genéticos propostos.

REFERÊNCIAS BIBLIOGRÁFICAS

AHMAD, I.; DHODHI, M. K. **Multiprocessor Scheduling in a Genetic Paradigm**, Parallel Computing Vol. 22, pp. 395-406, 1996.

AMDAHL, G. **Validity of the single processor approach to achieving large scale computing capability**, Proceedings of the on AFIPS Sprint Joint Computer Conference (Resto, Va.), AFIPS Press, Arlington, VA, pp. 483-485, 1967.

BELL, C. E.; HAN, J. **A New Heuristic Solution Method In Resource-Constrained Project Scheduling**. Naval Research Logistics. Vol. 38, pp. 315-331. 1991.

BELLMAN, R.; ESOGBUE, A.O.; NABESHIMA, I. **Mathematical Aspects Of Scheduling And Applications**. Pergamon Press. 1982.

BILLIONNET, A. **Allocating Tree Structured Programs in a Distributed System with Uniform Communication Costs**, IEEE Transaction on Parallel and Distributed Systems, Vol. 5, nº 4, Abril de 1994.

BURIOL, L. S. **Algoritmo Memético para o Problema do Caixeiro Viajante Assimétrico como Parte de um *Framework* para Algoritmos Evolutivos**, Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação da Unicamp, como requisito parcial para a obtenção do título de Mestre em Engenharia Elétrica, 2000.

CHINNECK, J. W.; PUREZA, V.; GOUBRAN, R. A.; KARAM, G. M.; LAVOIE, M. **A Fast Task-to-processor Assignment Heuristic for Real-Time Multiprocessor DSP Applications**, Computers and Operations Research, Vol. 30, no. 5, pp. 643-570, 2003 .

CHOUDHARY, A. N.; NARAHARI, B.; NICOL, D. M.; SIMHA, R. **Optimal Processor Assignment for a Class of Pipelined Computations**, IEEE Transaction on Parallel and Distributed Systems, Vol. 5, nº 4, pp. 439-445, 1994.

CHU, W.; LAN, M. T.; HELLERSTEIN, J. **Estimation of intermodule communication (IMC) and its applications in distributed processing systems**, IEEE Transaction Computers, Vol. C-33, nº 8, pp. 691-699, Agosto 1984.

CORRÊA, R. C.; FERREIRA A.; REBREYEND, P. **Scheduling Multiprocessor Tasks with Genetic Algorithms**, IEEE Transactions on Parallel and Distribuited Systems, Vol. 10, nº 8, Agosto de 1999.

DIAZ, A.; GLOVER, F.; GHAZIRI, H. M.; GONZÁLEZ, J. L.; LAGUNA, M.; MOSCATO, P.; TSENG, F. T. **Optimización Heurística y Redes Neuronales en Dirección de Operaciones e Ingeniería**, Editorial Paraninfo, 1ª edição, 1996.

FEO, T.; REZENDE, M. **A greed randomized adaptative search procedure for maximum independent set**. Operations Research, nº 42, pp. 860-879,1994.

FRANÇA P. M.; MENDES A.; MOSCATO P. **A memetic algorithm for the total tardiness single machine scheduling problem**. European Journal of Operational Research, Vol. 132, pp. 224-242, 2001.

GAJSKI, D. D.; PIER, J. **Essential issues in multiprocessor**, IEEE Computer, Vol.18, nº 6, Junho 1985.

GEN, M.; CHENG, R. **Genetic Algorithms and Engineering Design**, Edition Hardcover, 1ª Edição, 1997.

GEN, M.; CHENG, R. **Genetic Algorithms and Engineering Optimization**, Interscience, Dezembro de 1999.

GLOVER, F.; LAGUNA, M. **Tabu Search**, Kluwer Academic Publishers, 1ª edição, 1997.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization, and Machine Learning**, Reading, Massachusetts: Addison-Wesley, 1989.

GONG, D.; YAMAZAKI, G.; GEN, M.; XU, W. **A Genetic Algorithm method for one-dimensional machine location problems**. International Journal of Production Economics, Vol. 60-61, pp. 337-348, 1999.

GOUBRAN, R. A.; HAFEZ H. M.; SHEIKH, A. U. H. **Implementation of a Real-Time Mobile Channel Simulator Using a DSP Chip**, IEEE Transaction on Instrumentation and Measurement, Vol. 40, nº 4, Agosto de 1991.

HAX, A.; CANDEA, D. **Production and Inventory Management,** Englewood Cliffs, N.J., Prentice-Hall, 1984.

HOPPER, E.; TURTON B. **A Genetic Algorithm for a 2 D Industrial Packing Problem**, Computer & Industrial Engineering, Vol. 37, pp. 375-378, 1999.

HOU, E. S. H.; ANSARI, N.; REN, H. **A Genetic Algorithm for Multiprocessor Scheduling**, IEEE Transactions on Parallel and Distribuited Systems, Vol. 5, nº 2, Fevereiro de 1994.

HU, T. C. **Parallel Sequencing and Assembly Line Problems**, Operations Research, Vol. 9, nº 6, pp. 841-848, 1961.

HWANG, K. **Advanced Computer Architecture: Parallelism, Scalability, Programmability**, McGraw-Hill, Inc., New York, NY, 1993.

Introduction to Digital Signal Processing. Disponível em: http://www.dsptutor.freeuk.com. Acesso em: Setembro de 2002.

JAIN, K. K.; RAJARAMAN, V. **Lower and Upper Bounds on Time for Multiprocessor Optimal Schedules**, IEEE Transactions on Parallel and Distributed Systems, Vol. 5, nº 8, pp. 879-886, Agosto 1994.

KASAHARA, H.; NARITA, S. **Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing**, IEEE Transactions on Computers, Vol. C-33, nº 11, pp. 1023-1029, Novembro 1984.

KIRKPATRICK, S.; Gelatt, C. D.; Vecchi, M. P. **Optimization by Simulated Annealing**, Science, nº 220, pp. 671-680, 1983.

KUMAR, V.; GRAMA, A.; GUPTA, A.; KARYPIS, G. **Introduction to Paralell Computing: Design and Analysis of Algorithms**, Benjamin/Cummings, Redwood City, CA, 1994.

KWOK, Y.K.; AHMAD, I. **Static Scheduling for Allocating Directed Task Graphs to Multiprocessors**, ACM Computing Surveys, Vol. 31, nº 4, Dezembro de 1999.

LAUDON, K. C.; LAUDON, J. P. **Sistemas de Informação com Internet**, LTC, 4ª edição, 1999.

LAVOIE, M., **Task Assignment in a DSP Multiprocessor Environment**, Master's Thesis, Tech. Rep. SCE-90-18, Department of Systems and Computer Engineering,Carleton University, Ottawa, Canada, Agosto 1990.

LEE, E. A.; MESSERSCHMITT, D. G. **Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing**, IEEE Transactions on Computers, Vol. C-36, nº 1, pp. 24-35, Janeiro 1987.

LEE, E. A.; MESSERSCHMITT, D. G. **Synchronous Data Flow**, Proceedings of the IEEE, Vol. 75, nº 9, pp. 1235-1245, Setembro 1987.

LEE, E. A.; HO, W. H.; GOEI, E. E.; BIER, J. C.; BHATTACHARYYA, S. **Gabriel: A Design Environment for DSP**, IEEE Transactions on Acoustics, Speech, and Signal Processing, Vol. 37, nº 11, pp. 1751-1762, Novembro 1989.

LEWIS, T. G.; EL-REWINI, H. **Parallax: A tool for parallel program scheduling**, IEEE Parallel Distributed Technology, Vol. 1, nº 2, pp. 64-76, Maio 1993.

LO, V. M.; RAJOPADHYE, S.; GUPTA, S.; KELDSEN, D.; MOHAMED, M. A., NITZBERG, B.; TELLE, J. A.; ZHONG, X. **OREGAMI: Tools for mapping parallel computations to parallel architectures**, International Journal Parallel Program, Vol. 20, nº 3, pp. 237-270, 1991.

LORD, R. E.; KOWALIK, J. S.; KUMAR, S. P. **Solving linear algebraic equations on an MIMD computer**, J. ACM, Vol. 30, nº 1, pp. 103-117, Janeiro 1983.

MICHALEWICZ, Z. **Genetic Algorithms+Data Structures = Evolutions Programs**, Springer, 3a. Edição, 1996.

MURATA, T.; ISHIBUCHI, H.; TANAKA H. **Multiple-objective genetic algorithm and its applications to flowshop scheduling**. Computer and Industrial Engineering, Vol. 30, nº 4, pp. 957-968, 1996.

NORMAN, M. G.; THANISCH, P. **Models of Machines and Computations for Mapping in Multicomputers**, ACM Computing Surveys, Vol. 25, nº 3, pp. 263-302, Setembro 1993.

PALAZZO, L. A. M. **Algoritmos para Computação Evolutiva – Relatório Técnico – Pelotas: Grupo de Pesquisa em Inteligência Artificial- Universidade Católica de Pelotas**, 1997.

Programing – Basic Theory for the Unwary. Disponível em: <http://users.actcom.co.il/~chool/lupg/tutorials/parallel-programing-theory/parallel-programing-theory.html. Acesso em Setembro de 2002.

QUINN, M. J. **Parallel computing: theory and practice**. McGraw- Hill, Inc., New York, NY, 1994.

SHARMA, R.; BALACHANDER, T.; McCORD, C.; ANAND, S.; ZHANG, Q. **Genetic Algorithms for the Single-Sheet and Multi-Sheet Non-convex Cutting Stock problem**. Computer Aided Manufacturing Laboratory, Industrial Engineering program, University of Cincinnatti, PO Box 210116, Cincinnati, OH- 45221-0116, Janeiro, 1997.

SILVA, N. C. **Implementação de redes Parallel – Self Organizing Map para mapear imagens meteorológicas de radar**. Disponível em: http://www.inf.ufrgs.br/gpesquisa/procpar/paral.html. Acesso em: Setembro de 2002.

SMITH, S. W. **The Scientist and Engineer's Guide to Digital Signal Processing.** Disponível em: http://www.dspguide.com. Acesso em: Setembro de 2002.

STEPHENS R. **The Next Digital Wave: Real-Time Processing** . Disponível em: http://www.ti.com/corp/docs/investor/speeches/steph97/stephens97.shtml. Acesso em Junho de 2003.

WANG, D.; GEN, M.; CHENG, R. **Scheduling grouped jobs on single machine with genetic algorithm**. Computer and Industrial Engineering, nº 36, pp. 309-324, 1999.

WU, M. Y.; GAJSKI, D. D. **Hypertool: A programming aid for message- passing systems**. IEEE Transaction Parallel Distributed Systems, Vol. 1, nº 3, pp. 330-343, 1990.

YANG, T.; GERASOULIS, A. **PYRROS: Static task scheduling and code generation for message passing multiprocessors**. In Proceedings of the 1992 international conference on Supercomputing (ICS'92, Washington, DC, July 19-23, 1992), K. Kennedy and C. D. Polychronopoulos, ACM Press, New York, NY, 428-437, 1992.

YEPES, I. **Sistemas Inteligentes**. Disponível em: <http://www.geocities.com/igoryepes/index.htm>. Acesso em Abril de 2001.

ZOMAYA, A. Y.; WARD S.; MACEY B. **Genetic Scheduling for Parallel Processor Systems: Comparative Studies and Performance Issues**, IEEE Transactions on Parallel and Distribuited Systems, Vol. 10, nº 8, Agosto de 1999.