UNIVERSIDADE FEDERAL DE SANTA CATARINA

PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

ESCALONAMENTO DE TAREFAS *JOB-SHOP* REALISTAS UTILIZANDO ALGORITMOS GENÉTICOS EM *MATLAB*

Dissertação submetida à
Universidade Federal de Santa Catarina
como parte dos requisitos para a
obtenção do grau de Mestre em Engenharia Elétrica

FELIPE LUÍS BECK

Florianópolis, Novembro de 2000.

ESCALONAMENTO DE TAREFAS *JOB-SHOP* REALISTAS UTILIZANDO ALGORITMOS GENÉTICOS EM *MATLAB*

Felipe Luís Beck

'Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Engenharia Elétrica, Área de Concentração em Controle, Automação, e Informática Industrial, e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina.'

Prof. Cristoph Steffan Thomalla, Dr.
Orientador

Prof. Aguinaldo Silveira e Silva, Ph. D.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

Prof. Cristoph Steffan Thomalla, Dr.
Presidente

Prof. Guilherme Bittencourt, Dr.

Prof. Marcelo Ricardo Stemmer, Dr.

Profa. Jacqueline Gisele Rolim, Dra.

Dedico este trabalho a meus pais,

e a Periappa e Periamma.

Agradecimentos

Em primeiro lugar, eu gostaria de agradecer à minha família. É difícil encontrar famílias com três gerações de curso superior. Todo o empenho e esforço necessários para tanto precisa ser reconhecido. Agradeço, pois, ao vovô Beck (*in memorian*), por ter dado início a esta caminhada. Agradeço ao meu pai e minha mãe, por terem feito todo o possível (e o impossível) para me manterem na escola, apesar das freqüentes ameaças, e por nunca terem me deixado pensar que já havia feito o suficiente. Agradeço ao meu irmão André, por ter me convencido tão habilmente a vir para Florianópolis. As fotos da Praia Mole e seus *bundafish* certamente ajudaram. Agradeço à minha irmã Ana e meu irmão Mateus pelo companheirismo, e por serem loucos o suficiente para comerem moqueca no verão, apenas para dar cabo de quilos de peixes doados por pescadores da Armação. Agradeço, ainda, a todo o resto desta gigantesca família, por serem, no final das contas, uma Grande Família.

Agradeço à minha segunda família, o professor Branson e sua esposa Judy, que me acolheram durante cinco longos anos nos Estados Unidos. As lágrimas de saudade sempre pareceram mais poéticas quando vistas por trás de um copo de *rum-and-coke*.

Agradeço ao Dr. Thomalla, meu orientador e a todos os professores do LCMI que, de uma forma ou de outra, sempre estiveram dispostos a ajudar. Agradeço, em especial, aos membros desta Banca, por não terem dificultado muito o meu trabalho. Isto, é claro, se eles aprovarem esta dissertação. Além destes, gostaria de agradecer os professores que me receberam tão bem no gelado Wisconsin. Entre eles, destaco o Dr. Robert Johnson e o Dr. Angus Menuge. Também agradeço aos professores Brune e Sporket, do antigo Colégio Sinodal, por terem acreditado e confiado, e por terem me dado pontapés em número quase que suficiente.

Agradeço à Silvana e Oscar Lehenbauer, por terem me dado um teto sob o qual viver.

Agradeço à minha Neguinha Magali, sua família, e seu computador. Sem ela, eles, e ele, os últimos dois anos teriam sido tristes, sem graça, e cinco anos, respectivamente.

Agradeço também pela paciência da Rubelise, a voz chiada mas encantadora da Karina, a amizade e risada da Michelle, o sotaque da Karen, a ajuda do Antônio, as discussões com o Alex, as besteiras do Vallim, do Hlima, do Fábio, e do resto do pessoal do laboratório, do futebol, e dos churrascos. E como esquecer o Rodrigo e a Ana, apesar de eu sempre ter que estar bêbado para poder me comunicar na linguagem das Letras. Agradeço, e muito, ao Alemão Blauth, ao Herton Poco, ao Breno Cupido, Rafael Pato Choco, e Gabriel Tatu, por todos os anos, cervejas, piadas, brigas, e correspondências... Agradeço, ainda, a todos a quem eu não agradeci.

Por fim, agradeço ao governo, através de CAPES, pela bolsa dedicada. Espero que as greves destes últimos anos tenham surtido algum efeito, para que estas bolsas não se tornem tão irreais quanto o salário mínimo!

Resumo da Dissertação apresentada à UFSC como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica

ESCALONAMENTO DE TAREFAS *JOB-SHOP* REALISTAS UTILIZANDO ALGORITMOS GENÉTICOS EM *MATLAB*

Felipe Luís Beck

Novembro / 2000

Orientador: Cristoph Steffan Thomalla, Dr.
Área de Concentração: Controle, Automação e Informática Industrial.
Palavras-chave: escalonamento *job-shop* realista, MATLAB, algoritmos genéticos.
Número de Páginas: 106.

Esta dissertação aborda o desenvolvimento e a implementação de um sistema para escalonamento de tarefas do tipo *job-shop*. Com o sistema que foi desenvolvido, é possível o tratamento não apenas de problemas clássicos, como também de problemas realistas, que levam em conta um grande número de características reais, ignoradas em problemas clássicos de escalonamento. Estes problemas realistas possuem um elevado número de operações e máquinas, se comparados a problemas clássicos, além de recursos adicionais, multiplicidade de máquinas do mesmo tipo, e vários modos possíveis de execução, entre outras características. Com o sistema desenvolvido é, também, possível o tratamento de problemas de escalonamento dos tipos *flow* e *open-shop* e, inclusive, de problemas que mesclem estes três tipos.

Tendo sido desenvolvido utilizando MATLAB e uma *toolbox* genética, mostra-se que um sistema simples, construído a partir de software amplamente disponível, é satisfatório para a utilização em problemas de escalonamento. Os resultados obtidos em testes comprovam a eficiência do sistema criado no tratamento de problemas de escalonamento clássicos, e também a sua boa performance quando são tratados problemas realistas.

Abstract of Dissertation presented to UFSC as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

REALISTIC JOB-SHOP SCHEDULING USING GENETIC ALGORITHMS IN *MATLAB*

Felipe Luís Beck

November / 2000

Advisor: Cristoph Steffan Thomalla, Dr.
Area of Concentration: Industrial Automation.
Keywords: realistic job-shop scheduling, MATLAB, genetic algorithms.
Number of pages: 106.

This dissertation involves the design and implementation of a job-shop scheduling system. The developed system is able to tackle not only classical scheduling problems, but more realistic scheduling problems as well. These realistic problems require a great deal of additional effort for successful scheduling, since they include a great many restrictions and characteristics not present in traditional scheduling problems. As an example, realistic concerns taken into account include multiplicity of machines, additional resources to be used, multiple execution modes for each task, as well as other characteristics neglected by classical problems. The developed system is also able to successfully work with flow and open shop scheduling problems, and even problems which mix these three variations.

Having been developed for use with MATLAB, a readily available and inexpensive system, and making use of a simple genetic toolbox, it shows that no large computer systems are required for successful scheduling, even with increasing problem sizes. The system's efficacy in tackling large scheduling problems is attested by the test results included.

Sumário

Lista de Figuras

Lista de Tabelas

1. Introdução

Problemas de escalonamento ocorrem em praticamente todas as áreas, nas mais variadas formas, e sua resolução sempre é muito importante. Não se poderiam fazer cirurgias em um hospital sem que se decidisse, anteriormente, em que salas seriam feitas as operações. Não seriam obtidos os melhores resultados se, em um sistema de computadores, não houvesse forma de decidir a tarefa a ser executada, quando, e por quanto tempo. Podemos ver, com estes simples exemplos, que problemas de escalonamento aparecem em muitas áreas e de formas bastante variadas. Sua resolução, contudo, não é uma tarefa trivial. Por exemplo, a tarefa de alocar 250 salas de hospital aos mais variados tipos de pacientes, enfermeiros, aparelhagem, e assim por diante, não é fácil. Não é simples manter 50 engenheiros de software trabalhando juntos em um projeto comum. Da mesma forma, não é trivial a tarefa de decidir, em uma fábrica, que operações devem ser executadas em quais máquinas, quais os recursos a serem utilizados e em que ordem, para a construção de um determinado número de partes desejadas. Escalonamento requer que se leve em conta um grande número de informações a respeito do problema a ser tratado. Ainda no exemplo da fábrica, é preciso saber das relações entre as variadas operações a serem executadas para uma parte a ser construída, as relações entre as variadas partes que compõem o produto final, os processos a que cada parte deve ser submetida, que máquinas são necessárias para cada operação, quando estas máquinas precisam parar para manutenção, que recursos são necessários e estão disponíveis para a realização de cada etapa do projeto, e assim por diante. Também não é difícil concluir que é da maior importância que se resolvam estes problemas de forma prática e eficiente. Um erro qualquer, por menor que seja, pode ser a diferença entre um bom e um péssimo resultado. Se o paciente foi mandado para uma determinada sala para que fosse feita a cirurgia, enquanto o médico foi mandado a outra, é bastante óbvio que o resultado não seria dos melhores.

Nesta dissertação, um tipo específico de escalonamento será tratado: o escalonamento do tipo *job-shop*. Poder-se-ia traduzir isto como "chão de fábrica", mas com algum receio de ser uma interpretação muito vaga, pois *job-shop*, na verdade, é um tipo específico de chão de fábrica, conforme será visto no próximo capítulo. Além de escalonamento, isto é, a decisão do *que* será feito, *onde*, e *quando*, precisa-se tomar outras decisões, por exemplo, *como* algo deve ser feito (planejamento). Neste trabalho, abordar-se-á especificamente o problema de escalonamento de tarefas. Planejamento também será feito, mas de uma forma menos específica.

Muito já foi pesquisado com relação a este tipo de escalonamento, ou variações deste. De acordo com WALL [1], métodos de planejamento ou escalonamento já vêm sendo pesquisados desde, no mínimo, 1950. Uma grande variedade de métodos já foi utilizada em tentativas de se criar

sistemas de escalonamento condizentes com a dificuldade do problema. Em alguns casos, garante-se que uma solução ótima será obtida, mas não garante-se que isto ocorrerá com a necessária rapidez (métodos ótimos). Em outros casos, pode-se garantir que uma resposta será alcançada dentro de um espaço limite de tempo, mas não se tem garantias de que seja uma solução ótima (métodos de aproximação) [2]. Em muitos casos, inclusive, achar uma solução apenas viável já é uma tarefa das mais difíceis [1]. Isto porque problemas de escalonamento são NP-difíceis. Em outras palavras, ainda não se desenvolveu um algoritmo capaz de encontrar soluções ótimas em tempo polinomial. Um pequeno exemplo de FRENCH [3] é o de quatro rapazes tentando achar uma maneira de todos lerem quatro jornais diferentes, dadas uma série de restrições (do tipo *Pessoa A só pode ler o Jornal 2 depois que ler o Jornal 1*, e assim por diante). Para estas quatro pessoas, mais de 330 mil possibilidades diferentes existem. Se um computador pudesse verificar 1000 destas possibilidades por segundo, o que é, até certo ponto, otimista, o tempo necessário seria torno de cinco minutos e meio. Parece fácil, mas se o problema fosse modificado para incluir apenas mais um leitor, o tempo que o computador levaria para checar as possibilidades ultrapassaria 57 horas. Como comparação, os problemas clássicos de escalonamento *job-shop* normalmente estudados têm mais de 10 máquinas e 10 tarefas. Qualquer problema mais realista, como os que serão tratados nesta dissertação, facilmente extrapola estes números (um dos problemas realistas que será tratado aqui inclui mais de 500 operações a serem executadas em mais de 80 unidades de máquinas e recursos adicionais). Métodos de procura do tipo *força-bruta* estão, obviamente, descartados. Já métodos heurísticos de procura funcionam bem em problemas de otimização quando a complexidade é menor, mas também falham quando o espaço de procura tem muitas dimensões ou é muito vasto [4].

Algoritmos genéticos são ideais para a resolução de problemas NP-difíceis, por serem um método ao mesmo tempo *forte* e *fraco* [5]. *Fracos* no sentido de serem amplamente aplicáveis, isto é, aplicáveis, sem grandes modificações, a uma grande variedade de problemas, nas mais variadas áreas. Por outro lado, sistemas *fortes* são, geralmente, aqueles que necessitam de um grande número de informações específicas sobre o problema e o domínio a serem tratados, isto dando-lhes mais eficiência em seus métodos de busca. Algoritmos genéticos, por demandarem que o problema a ser resolvido seja "traduzido" para uma forma com a qual estes possam lidar, atingem a força e a ampla aplicabilidade desejáveis em problemas de escalonamento. Eles têm sido usados em problemas de escalonamento desde 1985, quando DAVIS [6] publicou o primeiro artigo unindo teoria de escalonamento *job-shop* e um algoritmo genético básico.

Uma infinidade de artigos publicados mostra que vasta pesquisa tem sido feita na aplicação de algoritmos genéticos a problemas de escalonamento, algumas demonstrando mais sucesso que outras. Em muitos casos, algoritmos genéticos são usados juntamente com outros métodos para melhorar a qualidade das soluções [2, 7, 8, 9]. Porém, a grande maioria destas abordagens limita-se

a tratar problemas de escalonamento clássicos que, senão irreais, são demasiadamente simples devido ao grande número de restrições impostas. DORNDORF e PESCH [10] sugerem que, para problemas de escalonamento, os algoritmos genéticos sejam usados em sistemas híbridos, incorporando métodos alternativos de busca local para que se obtenham soluções de maior qualidade [10]. Ao mesmo tempo que não se discute a validade desta afirmação, nesta dissertação tentar-se-á mostrar que um algoritmo genético puro obtém resultados satisfatórios para problemas de escalonamento, tanto clássicos como reais. Tendo sido usado software matemático amplamente disponível (MATLAB®), mostra-se que não são necessários sistemas computacionais altamente especializados para resolução de problemas de escalonamento com algoritmos genéticos. Para tal fim foram implementadas, sobre uma *toolbox* genética já existente, uma série de modificações e rotinas adicionais, de forma que possam ser tratados tanto os problemas clássicos, como alguns problemas reais, propostos por CÂNDIDO [2].

A dissertação está organizada da seguinte forma: no segundo capítulo será caracterizado, com maior rigor, o problema de escalonamento *job-shop*. Também serão mostrados, com um pouco mais de detalhe, métodos já utilizados na resolução destes problemas. No final deste mesmo capítulo, os problemas reais propostos serão analisados No terceiro capítulo, os algoritmos genéticos e suas peculiaridades serão descritos, enquanto o quarto capítulo apresentará o sistema proposto. O quinto capítulo trará os resultados obtidos em testes com os problemas clássicos e reais. No sexto capítulo, fazem-se as conclusões e considerações finais. Nos anexos estão disponibilizados os diagramas de fluxo das rotinas criadas e utilizadas, além das descrições dos problemas tratados.

2. Escalonamento

2.1. Definição do Problema Clássico

O problema de escalonamento, em geral, é descrito como sendo a alocação de *recursos* a *operações* (pertencentes a certas *tarefas*) durante um dado espaço de tempo, respeitando uma série de *restrições*, para que se alcance um *objetivo* final. Esta definição faz uso dos termos mais importantes em se falando de escalonamento, apesar de ser bastante vaga e pouco descritiva. Evoluindo a partir dela, chegaremos a uma definição bastante mais precisa e descritiva. *Recursos* são, geralmente, máquinas, materiais, aparelhos, salas, pessoas, e qualquer outra coisa que precise ser usada para que se complete uma dada *operação*. As *tarefas* são compostas de uma série de operações que, juntas, completam alguma parte daquilo que precisa ser feito. As *restrições* dizem respeito a todo o tipo de limitações que são impostas à execução destas operações, como, por exemplo, o fato de duas operações pertencentes à mesma tarefa não poderem ser executadas ao mesmo tempo, ou o fato de uma máquina só poder processar uma operação por vez. Finalmente, o *objetivo* diz respeito à meta que se pretende atingir, por exemplo, que recursos de um certo tipo sejam minimamente utilizados, ou que se minimize o tempo total de funcionamento do sistema.

Tendo esta vaga noção do problema a ser tratado, podemos começar a definir com mais precisão o que é escalonamento, e, mais especificamente, o que é escalonamento do tipo *job-shop*. Começaremos com a definição dos problemas clássicos, para depois tratarmos dos casos mais realistas a serem abordados nesta dissertação. FRENCH [3] lista as imposições que normalmente são utilizadas no tratamento de problemas de escalonamento. Muitos autores usam variações destas regras [11, 12], algumas das quais incluímos na lista a seguir.

1. Cada tarefa é uma entidade independente, isto é, duas operações pertencentes à uma mesma tarefa não podem ser executadas simultaneamente, e não existem relações de precedência entre operações de tarefas diferentes.
2. Uma vez iniciada uma operação, esta precisa ser executada até que termine: o sistema é não-preemptivo.
3. Cada tarefa tem o mesmo número de operações, sendo uma operação por máquina. Em outras palavras, não se permite que duas operações de uma mesma tarefa sejam executadas na mesma máquina, e não se permite a possibilidade de uma tarefa não ter nenhuma operação em alguma máquina.
4. Não existe cancelamento. Todas as tarefas precisam ser executadas até que estejam completas.

5. Os tempos de processamento das operações independem das operações previamente escalonadas. Ou seja, os tempos de preparo de máquina (*setup*) são independentes da seqüência de operações, e tempos de transporte entre máquinas são ignorados.
6. Tarefas não precisam ser executadas continuamente, isto é, não é necessário que todas as operações de uma tarefa sejam executadas sem intervalo entre elas.
7. Não existem restrições quanto ao horário de início de cada operação, ou seja, respeitadas as relações de precedência, todas as operações podem ser executadas a partir do início, quando o processo de escalonamento inicia.
8. Só existe uma máquina de cada tipo.
9. Máquinas podem ficar ociosas.
10. Máquinas não podem processar mais de uma operação por vez.
11. Máquinas nunca quebram e ficam disponíveis durante todo o processo.
12. Limitações tecnológicas são conhecidas de antemão e imutáveis.
13. Todo o sistema é determinístico. O número de tarefas é conhecido e constante, o número de máquinas é conhecido e constante, os tempos de processamento são conhecidos e constantes. Além disto, todas as outras quantias que definem o problema são conhecidas e constantes.

Não é difícil ver como muitas destas restrições são irreais, ou, no mínimo, muito vagas em se tratando de problemas minimamente realistas. Um sem-número de possibilidades são ignoradas com estas restrições. Por exemplo, a possibilidade de se fazer serviços de montagem, onde duas partes são construídas simultaneamente e depois montadas junto. Máquinas que nunca quebram ou que nunca tenham que ser submetidas a serviços de manutenção são obviamente desejáveis, mas absolutamente irreais. Estas são algumas das restrições que não serão observadas nos casos realistas a serem analisados mais adiante neste trabalho.

Matematicamente, a definição do problema clássico de escalonamento do tipo *job-shop* para minimização do tempo total de execução das tarefas pode ser feita conforme HUSBANDS [13]. São dados um conjunto J de tarefas, um conjunto M de máquinas, e um conjunto O de operações. Para cada operação $p \in O$ existe uma tarefa $j_p \in J$ à qual esta operação pertence, e uma máquina $m_p \in M$ na qual esta operação precisa ser processada por um tempo $t_p \in N$. Existe também uma relação binária $\rightarrow$ de ordenamento temporal sobre O que decompõe o conjunto em redes de ordenamento parcial correspondente às tarefas. Isto é, se $x \rightarrow y$, então $j_x = j_y$, e não existe um $z \neq x$, $z \neq y$, que faça $x \rightarrow z$ ou $z \rightarrow y$. Em outras palavras, o operador $\rightarrow$ determina a seqüência a ser seguida na execução das operações de uma tarefa. Utilizando a função objetivo de minimização do tempo necessário para completar a execução de todas as tarefas, o problema consiste em achar um tempo de início s_p para cada operação $p \in O$ de forma que

$$max_{p \in O}(s_p + t_p)$$

seja minimizado sujeito às seguintes restrições:

$$t_p \geq 0, \ \forall p \in O$$

$$s_x - s_y \geq t_y, \ se \ y \rightarrow x, \ x,y \in O$$

$$(s_i - s_j \geq t_j) \vee (s_j - s_i \geq t_i), \ se \ m_i = m_j, \ i,j \in O$$

Além do escalonamento do tipo *job-shop* definido acima, muitas variações deste são abordadas na literatura. Sendo o escalonamento *job-shop* o mais geral e mais complexo de todos os problemas tradicionais de escalonamento [13], muitas vezes usam-se casos particulares deste em trabalhos. Os casos mais comumente tratados são os de escalonamento *open-shop* [12] e *flow-shop* [14, 15, 16].

Em escalonamento do tipo *open-shop*, a grande diferença é que não existe uma ordem que precise ser respeitada na execução das operações de tarefas. Não existe, isto é, qualquer relação de precedência entre as operações de uma mesma tarefa. Este tipo de escalonamento não é tão estudado como os outros dois casos, apesar de ser muito importante, pois modela um grande número de problemas comuns[1]. Em se tratando de escalonamento do tipo *flow-shop*, a diferença é que as operações precisam ser executadas na mesma seqüência para todas as tarefas. Ou seja, as relações de precedência entre operações são as mesmas para todas as tarefas. Um exemplo disto é o procedimento de pouso para aviões: todos os aviões que precisam pousar em um determinado local devem executar exatamente o mesmo procedimento. Apesar destes problemas também serem significativos, apenas serão abordados nesta dissertação os problemas de escalonamento do tipo *job-shop*, justamente pelo fato dos outros dois serem casos especiais deste.

Como já foi dito, os problemas clássicos fazem uso de uma série de suposições que acabam por torná-los demasiadamente simples. Eles não são representativos da grande maioria de problemas de escalonamento que ocorrem na realidade. Apesar disto, serão tratados nesta dissertação como forma de avaliação do desempenho deste trabalho. Isto se faz necessário porque a grande maioria dos trabalhos de pesquisa aborda apenas os problemas clássicos. Nesta dissertação, serão apresentados tanto algoritmos que lidam com problemas clássicos quanto algoritmos para o tratamento de problemas mais realistas, sugeridos por CÂNDIDO [2] como problemas a serem utilizados para testes de sistemas que tratam de escalonamento realístico. Muitos problemas clássicos, dos mais variados tipos e tamanhos, dentre os quais serão selecionados alguns para

[1] Como exemplos, FANG *et al.* [12] cita operações de manutenção ou consertos em automóveis, ou tarefas de *upgrade* em computadores, onde as operações podem ser executadas de forma independente uma da outra.

avaliação deste trabalho, estão compilados em uma biblioteca virtual para pesquisa operacional mantida por BEASLEY [65].

Na definição matemática dada acima, considerou-se que o objetivo a ser atingindo era a minimização do tempo necessário para execução das tarefas. Apesar deste objetivo ser o que mais freqüentemente aparece na literatura, é preciso que se definam outros, pois, muitas vezes, combinações de objetivos são usadas. Por exemplo, pode-se querer minimizar o tempo de execução e também o custo associado a esta execução. Na seção seguinte, serão apresentados os modelos de avaliação mais freqüentemente utilizados.

2.2. Critérios de Avaliação

Aqui serão apenas listadas as métricas de avaliação mais comuns e sem o rigor matemático que pode ser encontrado em [3]. Vale a pena ressaltar que estas métricas de avaliação são aplicáveis aos três tipos de problemas de escalonamento descritos na seção anterior. Basicamente, existem três tipos diferentes de critérios de avaliação para escalas. Os critérios podem ser divididos entre aqueles baseados em tempo de término, prazos de entrega, ou custos de utilização e inventário. Dentro de cada uma destas categorias, as medidas (critérios) podem ser ainda divididas entre regulares ou não. Nos bastará mantermos clara a idéia de que uma medida regular de performance não pode ser melhorada atrasando-se a execução de alguma tarefa[2].

Dentre os critérios baseados em *tempos de término* de operações ou tarefas, destacam-se quatro. O tempo máximo que uma tarefa leva para completar execução (*maximum flow-time*), isto é, o tempo decorrido desde que a tarefa está pronta para ser processada até seu término, considera que o custo do processo está diretamente relacionado com o tempo que se leva para executar a tarefa mais longa. Sem se levar em conta os prazos de liberação das tarefas (restrição número sete da seção anterior), esta medida é igual ao tempo total de produção, o tempo decorrido entre o início do processo e o término da última tarefa (*maximum completion-time* ou *make-span*). Igualmente, pode-se levar em consideração as médias destas duas medidas. Em outras palavras, considera-se que o custo do processo está diretamente relacionado ao tempo médio que as tarefas levam desde que são liberadas até seu término (*mean flow-time*), ou o tempo médio que as tarefas precisam para estarem concluídas (*mean completion-time* ou *make-span*). Estas quatro medidas de performance são medidas regulares.

Em muitos casos, as tarefas têm prazos de entrega pré-definidos (*due-dates*), ou seja, um prazo dentro do qual elas precisam estar concluídas, geralmente sendo punidas por atraso. Medidas óbvias de avaliação neste caso levam em conta a diferença de tempo entre o prazo de entrega das tarefas e seus tempos de término. É também óbvio que, em muitos casos, uma tarefa pode acarretar

prejuízos se completada antes do seu prazo de entrega[3]. Por isto, duas medidas diferentes podem ser consideradas aqui: uma que somente leva em consideração o atraso (entrega estritamente após o prazo definido), e outra que leva em conta também o adiantamento de uma tarefa (entrega antes do prazo definido, o que pode ser bom ou ruim). Estas tanto podem ser quantitativas, isto é, variar de acordo com o *quanto* se atrasou, quanto qualitativas, onde um atraso de um segundo ou de um século é igual (um exemplo seria um avião que, não pousando antes do prazo de término de combustível, tanto faz se pousa uma hora ou dez dias atrasado, as conseqüências são as mesmas). Neste caso, pode-se apenas levar em conta o número de tarefas atrasadas, sem se considerar o quanto cada uma atrasou.

Já em se tratando de custos associados à *utilização* ou *inventário*, pode-se levar em conta o número de tarefas ociosas (aquelas que não tenham nenhuma operação sendo executada em um dado momento), à espera de que máquinas sejam liberadas para continuarem o processamento; ou o número de tarefas não completadas em um determinado momento, o que afetaria custos de se manter inventário durante o processo. Pode-se escolher, também, maximizar o número de tarefas em operação em um determinado momento, para reduzir o tempo ocioso das máquinas, possivelmente melhorando a produtividade.

Pode-se ver que as formas de se avaliar uma ordem de processamento são muitas e que avaliam diferentes partes do processo. Em muitos casos, por esta razão, utilizam-se combinações destas medidas, para que se possa ter uma idéia mais ampla do que está sendo feito no sistema. Por exemplo, pode-se tentar reduzir o tempo máximo de conclusão das tarefas ao mesmo tempo em que se procura maximizar a utilização de uma dada máquina. Também não é raro encontrarem-se casos em que estas medidas de avaliação possuem pesos de acordo com a importância de cada uma (isto é, cada uma das medidas é apenas uma parte do custo total final do processo); ou em que cada uma das tarefas possui um peso diferente na avaliação do processo.

2.3. Métodos de Resolução

A literatura disponível deixa claro que problemas de escalonamento, tanto clássicos como realistas, já foram abordados com uma variedade de métodos, alguns dos quais mostrando melhores resultados que outros. Conforme já foi dito, o problema vem sendo estudado, mais detalhadamente, desde os anos 60 [1]. Métodos matemáticos, regras de despacho (*dispatching rules*), métodos baseados em simulações, métodos baseados em inteligência artificial, heurísticas, e ainda, recentemente, paradigmas multi-agente de inteligência artificial distribuída são algumas das formas

---

[2] Esta propriedade de alguns critérios de avaliação será útil para limitarmos o espaço de busca do algoritmo genético, conforme será visto mais adiante.

utilizadas para a resolução de problemas de escalonamento em geral [7]. Aqui serão apresentadas algumas destas abordagens, divididas de acordo com o tipo de método utilizado, a saber, métodos exatos ou heurísticos. Os métodos heurísticos podem ser, ainda, subdivididos em estocásticos e determinísticos [1]. No final desta seção, apesar de ainda não termos entrado em detalhes sobre algoritmos genéticos (capítulo 3), apresentaremos algumas justificativas para o uso destes em problemas de escalonamento. Vale a pena mencionar que aqui não será feito um estudo exaustivo dos métodos de resolução utilizados. O leitor interessado pode utilizar as referências mencionadas, ou [2, 3, 9, 17, 18, 19, 20, 21, 22] para um estudo mais detalhado de alguns destes métodos.

2.3.1. Métodos Exatos

Métodos exatos são aqueles que sempre acham alguma solução (ótima) para o problema proposto, ou, caso não exista uma solução, geralmente dão alguma indicação disto [1]. Para escalonamento, os métodos mais comumente utilizados são o *critical-path method* (CPM), programação linear ou de inteiros, e métodos enumerativos. Também podem ser utilizados algoritmos construtivos, que compõem soluções ótimas a partir das informações do problema e seguindo um número de regras pré-determinadas [3]. O método CPM sempre acha o menor prazo de execução para um determinado número de tarefas a serem escalonadas, mas não leva em consideração restrições temporais ou materiais, limitando, assim, sua aplicabilidade [1]. Já os algoritmos construtivos são aplicáveis a uma quantidade mínima de problemas, geralmente pequenos e bastante restritivos quanto a tempos de processamento de tarefas [3].

Métodos de programação linear ou de inteiros também requerem que muitas simplificações sejam adotadas nos problemas para que estes possam ser resolvidos. Além disto, não são facilmente adaptáveis a variações no tamanho dos problemas, sendo utilizados apenas em problemas de pequeno porte [1]. DAVIS [6] já havia mencionado que problemas de escalonamento reais, com suas muitas restrições e imposições, não são de fácil resolução por métodos matemáticos utilizados em pesquisa operacional. Conforme FRENCH [3], apesar destes métodos (de programação de inteiros, por exemplo) parecerem bastante promissores no tratamento de problemas de escalonamento, na realidade eles não são. Geralmente, faz-se uma tradução do problema de escalonamento para um formato que possa ser resolvido através de programas matemáticos. Contudo, e isto é que faz a diferença, este tratamento por métodos matemáticos não é, em nada, mais fácil que a resolução do problema original [3].

Entre os métodos enumerativos destacam-se as árvores de busca e técnicas de programação dinâmica. Como seu nome deixa claro, métodos enumerativos resolvem os problemas listando todas

---

[3] Um exemplo disto é o método de trabalho de algumas fábricas de computadores que apenas montam os computadores para entrega direta ao consumidor final, evitando que tenham gastos com armazenamento de produtos prontos.

as possibilidades e eliminando as opções não-ótimas. Em métodos de enumeração explícita, todas as possibilidades são listadas e depois se faz o cancelamento daquelas que não são apropriadas. Procedimentos de busca em árvores, assim como alguns dos métodos apresentados acima, não são aplicáveis a problemas de grande porte. O aumento desenfreado do tamanho da árvore de busca com pequenas adições ao problema torna o método muito demorado. Geralmente utilizam-se heurísticas para limitar o tamanho destas árvores, permitindo, assim, a resolução de problemas um pouco maiores [1] (mas ainda limitados se comparados a problemas reais de escalonamento). Outro método enumerativo, mas implícito (isto é, que não explora, explicitamente todas as possibilidades), é o chamado *branch and bound*. Este método utiliza uma estratégia de procura especializada, que pode determinar que um certo ponto da árvore seja cortado por não apresentar características desejáveis, limitando, assim, o tempo necessário para terminar a busca. Contudo, é importante notar que, nos piores casos, esta enumeração implícita se torna uma simples enumeração explícita [3]. Mesmo nos melhores casos, onde grande parte da árvore de busca é podada, o tempo necessário para execução é bastante grande, devido à grande quantidade de variações nas escalas geradas que precisam ser analisadas [12].

Programação dinâmica, se comparada à enumeração completa, é muito mais rápida. A maior diferença é causada pelo simples fato de métodos de enumeração completa gerarem toda a árvore de busca, para só depois começar a eliminar *galhos*. Já os métodos de programação dinâmica vão eliminando possibilidades à medida que a árvore é construída. [3]. Isto faz com que o tempo de resolução do problema diminua bastante. Contudo, métodos dinâmicos apresentam outro problema. Como estes métodos precisam guardar, em memória, um grande número de valores intermediários usados nos cálculos, a necessidade de memória computacional aumenta. É óbvio que, hoje em dia, a memória de computadores pode ser considerada infinita, mas, para que esta memória seja infinita, é preciso que se utilizem meios com métodos de acesso lentos (se comparados com métodos de acesso de memória interna), nos remetendo de volta ao problema de métodos de enumeração completa (tempo). FRENCH [3] considera 25 como sendo o número máximo de tarefas em um problema de escalonamento para que este possa ser resolvido por métodos de programação dinâmica.

2.3.2. Métodos Heurísticos

A palavra *heurística* significa "algo que serve para encontrar", o que não é uma definição muito precisa. Em muitos casos, a palavra é usada como o oposto de algorítmico[4], e, em se tratando de métodos de busca, para descrever qualquer função que estime o custo de uma solução [20]. Heurísticas, ou métodos de aproximação, podem ser descritas como sendo regras de escolha que devem ser seguidas para a decisão de qual tarefa deve ser escalonada em um dado momento. Ao

contrário dos métodos exatos, os métodos baseados em heurísticas não garantem que uma solução ótima será encontrada, mas sim uma solução boa [36]. Estes métodos heurísticos podem ser divididos em dois grupos: os estocásticos, que utilizam operações probabilistas (de forma que podem nunca operar da mesma forma em um problema), e os determinísticos, que operam sempre da mesma forma para cada problema. Também são utilizados métodos híbridos, com resultados bastante variados [7, 8, 9, 23].

Em se falado de métodos determinísticos, geralmente a heurística é definida como uma regra de prioridade a ser utilizada para a escolha de uma dentre um conjunto de operações ainda não escalonadas [11]. Heurísticas de escalonamento determinam quando uma tarefa (ou uma operação de uma tarefa) deve ser executada. Heurísticas de seqüenciamento determinam a ordem em que as tarefas devem ser executadas. Estas são geralmente utilizadas em conjunto com árvores de busca e determinam que partes da árvore devem ser ignoradas e que partes da árvore devem ser expandidas. Já heurísticas hierárquicas consideram vários níveis de prioridade para poderem atingir vários objetivos ao mesmo tempo [1]. O exemplo mais famoso do uso de heurísticas determinísticas é o *Shifting Bottleneck Procedure* [18]. Neste procedimento, a cada iteração, um sub-problema (que assume que o problema é composto de apenas uma máquina) é resolvido otimamente, sendo depois re-otimizadas as seqüências de operações determinadas em iterações anteriores. HOLTHAUS [19] apresenta um estudo onde, de acordo com a meta a ser atingida (o critério a ser considerado para avaliação), são criadas combinações de regras de prioridade que tendem a melhorar os resultados. Já CASKEY e STORCH [17] concluem que a maioria das regras de prioridade comumente utilizadas são satisfatórias para minimização de atrasos, sendo que combinações de várias regras não demonstraram surtir grande efeito nos resultados. Em [22], um algoritmo heurístico é proposto para a resolução de problemas de escalonamento com tempos de preparo dependentes da seqüência escalonada. KUMAR e SRINIVASAN [24] fazem um estudo utilizando algoritmos genéticos e regras de prioridade. As seis regras de prioridade mais estudadas na literatura (mais de 100 já foram estudadas) são apresentadas a seguir [3].

- SPT - ***Shortest Processing Time*** - seleciona aquela operação que tem o menor tempo de processamento;
- FCFS - ***First Come First Served*** - seleciona as operações a serem escalonadas na ordem em que elas entraram no sistema;
- MWKR - ***Most Work Remaining*** - seleciona operações da tarefa que tem o maior tempo de processamento total restante;

---

[4] Apesar disto não estar inteiramente correto, uma vez que mesmo um método heurístico pode utilizar passos algorítmicos (não randômicos) para atingir suas soluções [20].

- LWKR - *Least Work Remaining* - seleciona operações da tarefa que tem o menor tempo de processamento total restante;
- MOPNR - *Most Operations Remaining* - seleciona operações da tarefa que tem o maior número de operações a serem escalonadas;
- RANDOM - seleciona operações aleatoriamente.

Estes métodos determinísticos são bastante efetivos na resolução de problemas combinatórios de menor complexidade, mas ainda assim falham com grandes espaços de busca ou quando estes são multi-dimensionados. Os problemas precisam, então, ser simplificados, confinando o espaço de busca e tornando o sistema menos flexível [4].

Métodos heurísticos também incluem métodos estocásticos, dentre os quais se destacam métodos que utilizam inteligência artificial, algoritmos genéticos, simulações de processos físicos, entre outros. Métodos como sistemas especialistas, ou sistemas baseados em conhecimento, são altamente especializados e requerem uma grande quantidade de informações a respeito do problema a ser resolvido, de forma que se tornam bastante específicos para um dado tipo de problema [1].

Métodos de procura baseados em melhorias iterativas incluem os procedimentos conhecidos como *hill-climbing* e *simulated annealing* (têmpera simulada). O procedimento de *hill-climbing*[5] incorpora um sistema de laços de programação que continuamente evoluem na direção de um valor crescente (melhorando, a cada iteração, o resultado desejado). São três as maiores limitações destes métodos, de acordo com RUSSEL e NORVIG [20]. Máximos locais param o algoritmo, fazendo com que se interrompa a procura em posições que podem estar bastante distantes do desejado. Platôs são áreas do espaço de busca onde os valores da função de avaliação formam um plano. Neste caso, a procura se torna igual a uma busca aleatória. Picos são locais onde uma mudança de posição, por menor que seja, pode significar estar do outro lado da curva, fazendo com que se perca muito tempo sem evolução para valores melhores. Com estas limitações, o algoritmo chega a um ponto em que não se obtém maior rendimento. Neste caso, pode-se fazer duas coisas: reiniciar de um ponto diferente, ou permitir que se desça um pouco, para depois continuar a procura (piorar para melhorar). Esta descida é que define o processo conhecido como simulação de *annealing* (têmpera, ou recozimento). Permite-se que o algoritmo piore a função de avaliação (mudando de posição no espaço de procura), para depois continuar procurando a partir deste novo ponto.

2.3.3. Algoritmos Genéticos

No capítulo seguinte, os algoritmos genéticos serão descritos em detalhe. Nesta seção, apenas justificaremos as opções pelas quais eles foram escolhidos para este trabalho. Conforme já foi dito, escalonamento do tipo *job-shop* é um dos mais difíceis dos problemas de escalonamento.

---

[5] Escalada de colina, numa tradução literal.

Além disto, grande parte dos problemas de escalonamento são considerados NP-difíceis [3]. Isto descarta a possibilidade de se utilizar métodos exatos na resolução de problemas minimamente complexos. Na seção anterior vimos alguns métodos que, por um motivo ou outro, poderiam ser utilizados na resolução destes problemas, especialmente os métodos heurísticos estocásticos. Técnicas de inteligência artificial são mais úteis que métodos convencionais quando não se pode representar o domínio do problema utilizando métodos matemáticos [4], justamente o caso dos problemas de escalonamento [6]. Técnicas de busca que fazem uso de heurísticas para limitar o tamanho do espaço de busca não têm, sequer, a garantia de que encontrarão soluções aceitáveis, quanto mais boas ou ótimas. Técnicas que incorporam conhecimento específico a respeito do problema se tornam aplicáveis somente a uma determinada classe, limitando sua aplicabilidade [25].

Problemas combinatórios similares, como o problema do caixeiro viajante, já foram resolvidos com sucesso através de algoritmos genéticos [26]. Algoritmos genéticos são bastante flexíveis no aspecto de representação, pois o conhecimento a respeito do sistema que é preciso para orientar a busca pode ser mantido separado da implementação do algoritmo propriamente dito. Isto faz com que, apesar de ser um método geralmente descrito como *fraco*, isto é, um método que tem ampla aplicabilidade devido à generalidade, ele possa se tornar um método *forte*, isto é, bastante específico [27]. Não se sacrifica, em outras palavras, ampla aplicabilidade por poder de busca, e nem vice-versa. Eles unem procura direcionada (fazendo uso de soluções boas já encontradas) e procura estocástica [28], varrendo o espaço de busca aleatoriamente (mas não de forma cega, sem direção [29]) e eficientemente [30]. Ao contrário de métodos como *hill-climbing* e *simulated annealing* (que utilizam uma única solução durante a busca), os algoritmos genéticos fazem uso de uma população de possíveis soluções, aumentando a rapidez e a eficiência da busca [31], e permitindo que se explore as características desejáveis comuns a boas soluções. GOLDBERG [29] resume as quatro peculiaridades dos algoritmos genéticos:

1. Algoritmos genéticos trabalham com uma codificação do conjunto de parâmetros e não diretamente com os parâmetros.
2. Algoritmos genéticos fazem a busca com uma população de pontos e não com um único ponto.
3. Algoritmos genéticos usam informação de recompensa (*payoff*), a função-objetivo, e não derivadas destas, ou conhecimento auxiliar.
4. Algoritmos genéticos usam regras de transição probabilísticas e não determinísticas.

A força de busca dos algoritmos genéticos está em seus esquemas, partes do cromossoma que são comuns às soluções boas [32], e na maneira como estes esquemas tendem a crescer à medida que sua aptidão melhora. Já os esquemas que demonstrarem menor aptidão tendem a desaparecer, em detrimento dos mais aptos. Para um pouco mais de detalhes na aplicabilidade de algoritmos

genéticos, HUSBANDS [13] faz um levantamento de usos de algoritmos genéticos em problemas de escalonamento, e TADEI *et al.* [21] faz uma comparação entre vários métodos utilizados para resolução de problemas de escalonamento. Além disto, referências a uma série de artigos utilizando algoritmos genéticos com escalonamento ou problemas combinatórios estão listados na bibliografia [2, 4, 6, 7, 8, 9, 10, 11, 12, 14, 15, 16, 22, 23, 24, 27, 30, 31, 33, 34, 35, 36, 37, 38, 39, 40, 63, 64].

2.4. Construção de Escalas

O objetivo nesta seção não é apresentar um estudo compreensivo de métodos de construção de escala. O objetivo, isto sim, é aprofundar um pouco o conhecimento a respeito das características de alguns algoritmos de construção de escala para poder, a partir daí, caracterizar algumas escolhas que foram feitas para o projeto. Especificamente, trataremos de descrever três tipos de escalas: escalas ativas (*active schedules*), escalas sem-atraso (*non-delay*), e escalas semi-ativas (*semi-active*). As escalas sem-atraso são uma sub-classe das escalas ativas que, por sua vez, são uma sub-classe das escalas semi-ativas.

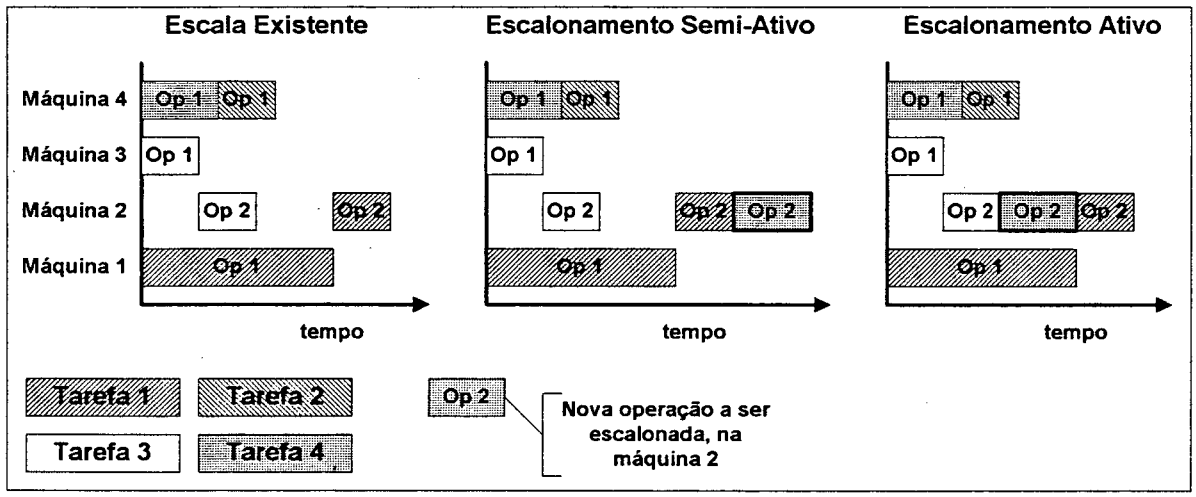

Figura 2.1 - Escalonamento Ativo e Semi-Ativo

As escalas semi-ativas são aquelas em que todas as operações são escalonadas o mais cedo possível *após* as operações que já estão escalonadas, obedecendo, necessariamente, a todas as restrições tecnológicas e a seqüência de processamento. Já escalas ativas são aquelas que garantem que nenhuma operação pode ser iniciada mais cedo sem que isto atrase outra operação já escalonada, ou viole as restrições do problema. Em outras palavras, toda operação pode ser escalonada numa posição *anterior* às tarefas já escalonadas (respeitando as restrições), desde que isto não cause um atraso em alguma destas operações já escalonadas. A figura 2.1 mostra isto com maior clareza. Já escalas sem-atraso são aquelas em que nenhuma máquina permanece ociosa quando poderia estar processando alguma operação. Em se tratando de medidas regulares de performance, apenas as escalas semi-ativas precisam ser analisadas para a obtenção de uma escala ótima. Contudo,

FRENCH [3] reduz esta necessidade, mostrando que ao menos uma solução ótima, e ativa, sempre existe[6]. Por este motivo, na construção das escalas (capítulo 4), apenas serão consideradas as escalas ativas.

2.5. Problemas Realistas

Conforme já dissemos no início deste capítulo, a grande maioria dos artigos e trabalhos que utilizam algoritmos genéticos para escalonamento considera problemas clássicos que, nem de perto, modelam características reais encontradas normalmente. Por este motivo usaremos, além dos problemas clássicos, problemas propostos por CÂNDIDO [2] que modelam problemas realistas. Nesta seção, descreveremos estes problemas um pouco mais em detalhe, em especial com relação às restrições clássicas (apresentadas anteriormente neste capítulo) que não serão observadas. As listagens completas destes problemas (assim como as listagens dos problemas clássicos usados) estão disponíveis no anexo A.

A primeira restrição examinada era a de que cada tarefa seria uma entidade independente, com relações de precedência estritas e invioláveis entre operações de uma mesma tarefa (sendo que duas operações de uma mesma tarefa não podem ser executadas simultaneamente). Nos novos problemas, cada tarefa continua sendo uma entidade independente, mas composta de partes. As operações de cada uma destas partes podem, ou não, ter relações de precedência com as operações de outras partes da mesma tarefa. Isto pode ser visto claramente na figura 2.2. As operações das partes **0** e **2** podem ser realizadas independentemente das operações da parte **1**, mas as operações da parte **3** requerem que todas as operações das partes **0**, **1**, e **2** já estejam completas. Isto permite modelar atividades de montagem de peças, por exemplo[7].

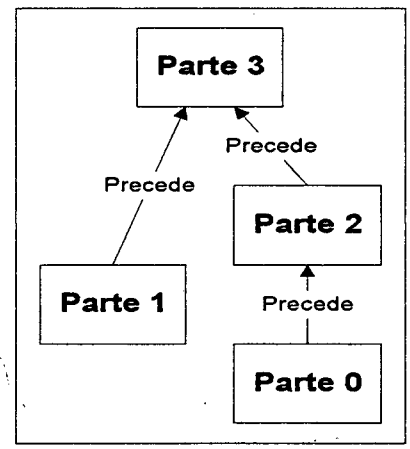

Figura 2.2 - Possível Relação de Precedência em Problemas Realistas

[6] O que não significa que não possa haver outras soluções ótimas que não sejam ativas.

[7] A segunda restrição, que diz respeito à operação contínua e ininterrupta, será mantida, assim como a quarta restrição (impossibilidade de cancelamento). Não é difícil ver como estas restrições podem ser bastante realistas.

Em nível de partes, as operações ainda terão relações de precedência entre si. Contudo, a terceira restrição mostrada na página 4 também será relaxada. As tarefas terão números variados de operações, em um número qualquer de máquinas. Além disso, dentro de uma parte poderemos ter várias formas de se completar o mesmo processo, isto é, vários modos de execução. Por exemplo, como na figura 2.3, pode-se completar o processo por várias rotas diferentes, cada uma com seus requisitos: máquinas, recursos adicionais, tempos de processamento, e assim por diante[8]. Para tanto, utiliza-se a noção de subprocessos e rotas. Cada parte é composta de um ou mais subprocessos, e cada um destes é composto de uma ou mais rotas (uma seqüência estrita de operações). Apenas uma rota precisa ser executada dentro de cada subprocesso, e uma vez iniciada uma rota, esta precisa ser seguida até o término do subprocesso a que ela pertence.

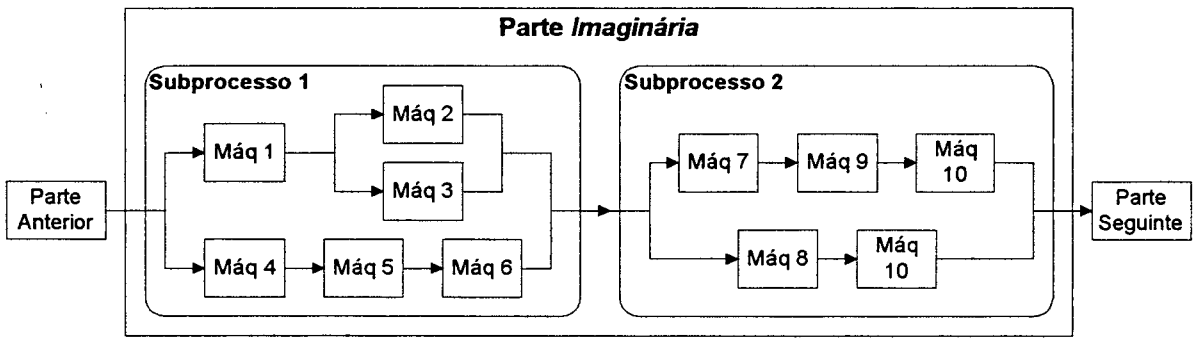

Figura 2.3 - Possíveis Modos de Execução de Tarefas

A quinta característica de problemas clássicos (tempos de processamento independentes da seqüência de operações) também será relaxada. Em se tratando de máquinas, tempos de preparo de máquina serão, em alguns casos, dependentes das operações previamente escalonadas na máquina em questão. Todas as máquinas terão um tempo de preparo regular (de acordo com, e definido por cada operação que necessita esta máquina) que será utilizado, caso não exista um tempo de preparo dependente da seqüência de operações aplicável ao caso. Tempos de preparo, contudo, não serão aplicados a recursos adicionais[9]. Vale a pena ressaltar que este *setup* pode ser executado ao mesmo tempo que a execução de outra operação da mesma parte, independentemente de relações de precedência. Isto equivale a dizer que a máquina onde será realizada a próxima operação da parte pode estar sendo preparada ao mesmo tempo que a operação anterior desta mesma parte está sendo executada em outra máquina. A sexta restrição será mantida (não é necessário que todas as operações de uma parte sejam executadas continuamente).

---

[8] Na verdade, esta opção (de processos diferentes) não pertence à etapa de escalonamento, mas sim de planejamento [1, 28]. Neste trabalho, portanto, será feito planejamento também, haja visto que caberá ao algoritmo genético a decisão sobre que caminho tomar, conforme veremos no capítulo 5.

[9] Recursos adicionais precisarão estar disponíveis durante todo o tempo de processamento, mas também não precisarão estar disponíveis durante o tempo de *setup*.

Haverá casos onde a execução das operações de uma parte só poderá ser iniciada após um dado momento. Isto é, a sétima restrição tampouco será válida. Um exemplo deste tipo de requerimento é o caso onde se depende de algum fornecedor que só pode disponibilizar as peças a partir de um dado momento. A oitava restrição (apenas uma máquina de cada tipo) será relaxada. Alguns tipos de máquinas serão únicos, enquanto outros poderão ter várias unidades disponíveis. O mesmo vale para recursos adicionais. Apenas uma unidade de máquina será necessária para cada operação, mas recursos adicionais podem ser requeridos em qualquer número (inclusive múltiplas unidades de cada recurso).

A décima restrição, que diz respeito à capacidade de processamento das máquinas (apenas uma operação pode ser processada de cada vez), será mantida, enquanto a décima-primeira será apenas relaxada. Apesar do sistema continuar determinístico (isto, é, todas as quantidades são conhecidas de antemão), todos os recursos ficarão indisponíveis durante determinados momentos (paradas para manutenção preventiva). É óbvio que isto não é realista, pois nossas máquinas continuam inquebráveis, mas o objetivo desta dissertação é tratar casos determinísticos. Para casos dinâmicos[10] o leitor interessado pode procurar [23, 37].

[10] Onde tarefas podem ser mudadas ou canceladas durante execução do escalonamento (casos preemptivos), máquinas podem quebrar, e assim por diante.

3. Algoritmos Genéticos

3.1. O Que São?

Algoritmos genéticos, desenvolvidos por John Holland, são algoritmos de busca baseados na mecânica de seleção natural [29]. Através de estratégias como reprodução, mutação e seleção, procura-se aumentar a aptidão (qualidade) dos *seres* de uma população, *seres* estes que representam soluções para algum problema. A busca genética inicia com a criação de uma população inicial. A partir daí, a cada geração, indivíduos são selecionados e, de alguma maneira, reproduzem-se combinando material genético, e, em alguns casos, são submetidos a mutações, para que se mantenha um mínimo de diversidade na população. Quando os indivíduos resultantes da reprodução superam, em qualidade, outros membros da população, estes são substituídos. Na figura 3.1, pode-se ver o funcionamento básico dos algoritmos genéticos.

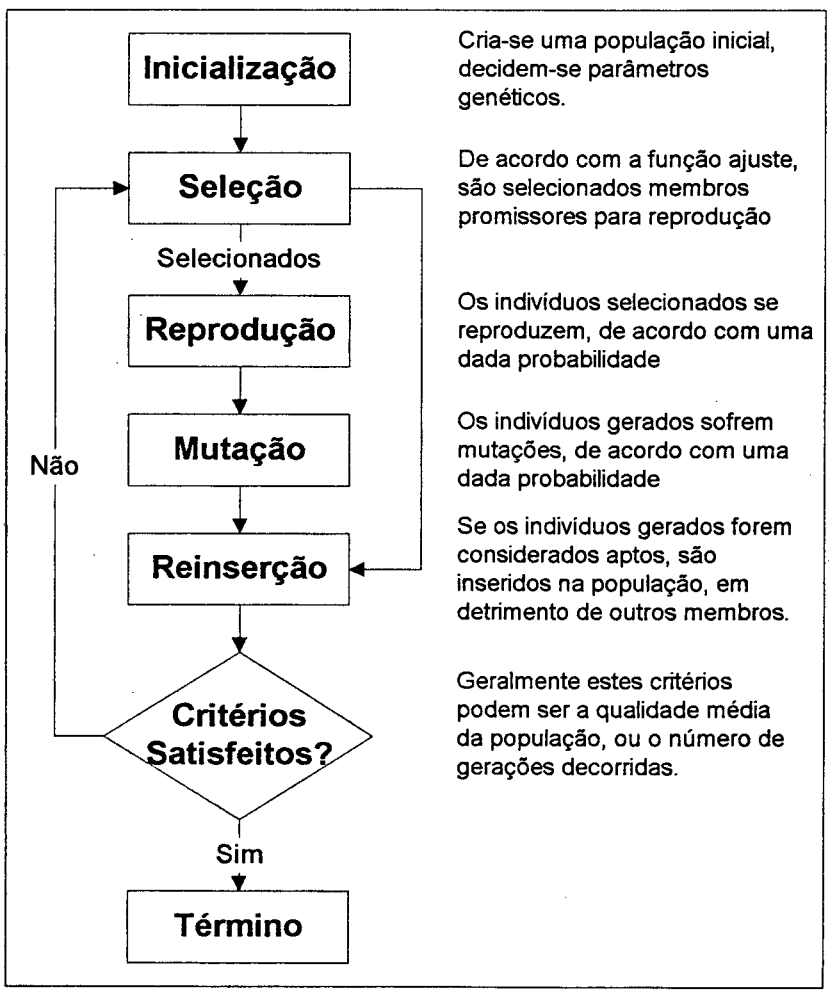

Figura 3.1 - Funcionamento Básico dos Algoritmos Genéticos

Neste capítulo, serão descritos em maiores detalhes os algoritmos genéticos. Muitas das qualidades dos algoritmos genéticos já foram apresentadas no capítulo anterior, portanto, aqui nos

limitaremos a descrever a funcionalidade intrínseca dos algoritmos genéticos. Também descreveremos o ambiente genético da *toolbox* utilizada para a produção deste trabalho.

Quando se trabalha com algoritmos genéticos, ou quando se tenta aplicá-los a algum problema, quatro são as perguntas que precisam ser respondidas [20]:

1. Como avaliaremos o desempenho de cada indivíduo, e como este desempenho será comparado ao desempenho do restante da população?
2. Como os indivíduos serão representados?
3. Como os indivíduos serão selecionados para reprodução?
4. Como será feita a reprodução dos indivíduos?

Para responder estas perguntas, primeiramente precisamos saber mais sobre cada uma das perguntas e quais as opções que existem. Precisamos saber de que formas pode-se fazer a avaliação da população. Precisamos saber que representações podem ser usadas para cada tipo de problema, ou que representações já foram utilizadas anteriormente e com que resultados. Precisamos estar cientes da forma como a reprodução afeta a população e os indivíduos, entre outras coisas. Este é o objetivo deste capítulo, ao passo que o próximo capítulo responderá às quatro perguntas propriamente ditas para problemas de escalonamento. A seguir descrevemos, em detalhes, cada um destes pontos.

3.2. Representação

Apesar dos algoritmos genéticos serem um método heurístico, a busca realizada por eles não opera no espaço definido pelo problema em si (o espaço definido pelas possíveis soluções do problema), mas sim no espaço definido pela representação escolhida [1] (uma *tradução* destas soluções). Uma conclusão óbvia disto é que se pode escolher a representação que melhor modelar as soluções, de forma a simplificar o problema. Para problemas combinatórios (caixeiro viajante, ou o próprio problema do escalonamento), a representação clássica (binária) de algoritmos genéticos tem se mostrado pouco apropriada [25, 34]. Várias representações diferentes já foram utilizadas nestes casos, geralmente envolvendo alfabetos de maior cardinalidade e, em muitos casos, elevados graus de *epístase*[5] [41]. Algumas destas representações serão descritas a seguir. Estudos mais detalhados podem ser encontrados em [11] e [13], pois não caberá aqui uma análise detalhada sobre as qualidades de cada uma destas representações. Para descrever as que serão apresentadas, serão analisados três aspectos: o mapeamento entre os espaços da representação e da solução[6], a forma

---

[5] *Epístase* é definida como um alto grau de interação entre os genes de um cromossoma. Como exemplo, BEASLEY *et al.* [44] cita o sistema de localização de morcegos por eco. A capacidade de produzir ruídos ultrasônicos só é útil se também estiver presente a capacidade de ouvi-los.

[6] De preferência, a relação entre os espaços deve ser de 1-1 (cromossomas diferentes representam soluções diferentes). Quando isto não é possível, n-1 (várias formas de representar a mesma solução) é preferível a

como este mapeamento é feito[7], e a *Lamarkianidade*[8] da representação. Uma representação deveria atender uma quarta característica, a de impedir cromossomas ilegais (isto é, aqueles que não representam uma solução) de aparecem, restringindo o espaço de busca às soluções válidas. Isto facilita a busca ao mesmo tempo em que não requer que se criem maneiras de corrigir ou destruir cromossomas inválidos [33, 43]. Mas, assumindo que a população inicial só inclui indivíduos válidos (conforme será o caso neste trabalho), isto será examinado mais a fundo na seção seguinte, quando tratarmos dos operadores genéticos.

Representações baseadas em tarefas, geralmente, listam a ordem em que as operações de cada tarefa devem ser escalonadas. Os cromossomas são, então, uma simples lista de todas as tarefas, representando as prioridades de cada uma (escalonam-se todas as operações da tarefa com maior prioridade, depois todas as operações da tarefa seguinte, e assim por diante). Um simples mapeamento é necessário para converter cromossomas em soluções e a representação é *Lamarkiana* e pouco redundante. Mas é uma representação incompleta (incapaz de gerar todo o espaço de busca) [11], e não garante que as escalas geradas sejam ativas. Uma variação desta representação foi usada por UCKUN *et al.* [4], utilizando três níveis de representação e incluindo, em cada um, uma quantidade maior de informações específicas ao problema.

Representações que se baseiam em números de operações (algumas vezes chamadas de permutações com repetição) listam cada tarefa um número de vezes igual ao número total de operações pertencentes àquela tarefa. Desta forma, qualquer permutação dos genes sempre produz uma solução válida. É medianamente redundante (n-1), meio-*Lamarkiana*, e requer um simples mapeamento para conversão (mas, a garantia de escalas ativas serem produzidas depende deste procedimento de conversão). Representações baseadas em listas de preferência são compostas de blocos que representam as máquinas do problema[9]. Cada bloco lista as tarefas e suas relativas preferências para cada máquina [34]. Também é medianamente redundante, meio-*Lamarkiana*, e uma heurística simples é suficiente para decodificar cromossomas.

Além destas, várias representações já foram testadas, algumas com resultados mais promissores que outras. Representações baseadas em tempos de término [1], listas de regras de prioridade [10, 11, 24], e listas de posições [14, 16, 27] são alguns exemplos.

1-n, apesar disto significar falsa competição entre cromossomas. Mais detalhes podem ser encontrados em [11].

[7] Quatro níveis são possíveis: desde sem necessidade de mapeamento (representação direta das soluções), até uma heurística complexa, passando por uma relação simples de mapeamento e uma heurística simples. Geralmente, quanto mais simples a representação, mais complicado é o mapeamento, e vice-versa [10].

[8] *Lamarkianidade* é definida como uma qualidade que indica o grau a que cromossomas conseguem passar seus méritos às gerações subsequentes. A utilização de uma representação não-*Lamarkiana* pode significar uma busca cega, inteiramente aleatória [11].

[9] Não é muito difícil ver como esta representação é imprópria para casos mais realistas, onde, geralmente, várias unidades de recursos (máquinas) são necessárias para se processar as operações.

3.3. Função Objetivo e Função de Aptidão

Depois de escolhida uma maneira de se representar as soluções de tal forma que o algoritmo genético possa trabalhar, é preciso definir de que modo cada indivíduo da população será avaliado. Isto é, é preciso que se defina quais cromossomas representam melhores soluções, para que estes possam ser utilizados para futuras reproduções. É bastante claro por que a avaliação é importante em um algoritmo genético, afinal, juntamente com a decodificação dos cromossomas para soluções, a avaliação de cada um destes é a principal ligação entre o espaço utilizado na busca (o espaço da representação) e o espaço das soluções. Freqüentemente os termos função objetivo e função de aptidão (*fitness function*) são usados com o mesmo significado [44]. O valor da função objetivo é usado, muitas vezes, diretamente como o valor desta função de aptidão. Para tratar soluções possivelmente inválidas são, então, utilizadas funções de penalização, que tentam dar uma idéia de quão errada uma solução está. Como estaremos evitando que soluções inválidas estejam presentes nas populações, não utilizaremos funções de penalização, mas sim uma distinção entre a função objetivo e a função de aptidão.

Especialmente em se tratando de problemas de otimização combinatória, muitas vezes a função objetivo não é usada diretamente para avaliação dos membros da população relativamente ao restante da população[10]. Ao invés disto, utiliza-se a função de ajuste para determinar quão boa uma solução é quando comparada com as outras soluções (dos outros membros da população), utilizando o valor da função objetivo apenas como um valor intermediário para que se faça esta comparação [45]. Métodos geralmente utilizados para se fazer isto são comparação proporcional[11] e *rankeamento*[12] dos indivíduos.

3.4. Seleção para Reprodução

A seleção dos indivíduos mais aptos (determinados pela função de ajuste) para reprodução é um processo probabilístico, onde cada indivíduo tem uma probabilidade de ser escolhido de acordo com o valor da função de aptidão. Estes indivíduos escolhidos são, então, submetidos aos processos de recombinação genética. Uma variedade de métodos para seleção existe, cada um deles influenciando, de maneira diferente, a exploração do espaço de busca [46]. Isto ocorre devido à chamada pressão para seleção, que é, em termos, o grau em que os melhores indivíduos são favorecidos nesta seleção em detrimento dos outros indivíduos [47]. Quanto maior for a pressão para seleção, mais são favorecidos os melhores indivíduos. Não é difícil ver como esta pressão para

---

[10] A função de ajuste também pode ser entendida como determinante do número esperado de vezes que um indivíduo será selecionado para (possivelmente) reproduzir [45].

[11] Neste caso, o valor da função objetivo de cada membro é dividido pelo valor total (a soma dos valores da função objetivo de todos indivíduos).

seleção pode afetar a busca. Uma pressão muito alta favorecerá demasiadamente os melhores indivíduos, fazendo com que o algoritmo (possivelmente) venha a convergir prematuramente. Já uma pressão para seleção muito baixa fará com que o algoritmo leve mais tempo que o necessário para encontrar alguma solução aceitável.

Os métodos mais conhecidos para seleção são divididos em dois grandes grupos: aqueles que fazem seleção proporcional e aqueles que se valem da posição de cada indivíduo dentro da população (isto é, o *ranking* de cada um) para a seleção. Em seleção proporcional, cada indivíduo tem chances proporcionais à comparação de sua medida de ajuste com a medida de aptidão do restante da população. Já para os métodos baseados em ordinais (posições), só importa a colocação absoluta de cada membro dentro da população. Os métodos mais conhecidos para se fazer seleção proporcional são: seleção proporcional, seleção estocástica com resto e seleção estocástica universal. Esta classe de métodos inclui também a famosa seleção por roleta, onde o resultado da avaliação de cada indivíduo é transformado em uma abertura angular em uma roleta [32], dando aos indivíduos mais aptos uma maior chance de serem selecionados. Já os métodos ordinais mais conhecidos são seleção por torneio, seleção μ-λ, seleção truncada, e *ranking* linear. Estudos detalhados sobre cada um destes métodos de seleção podem ser encontrados em [46, 47, 48, 49, 50, 62].

Uma questão importante a ser colocada em se falando de seleção para reprodução é a quantidade de indivíduos que serão selecionados, reproduzirão, e serão re-inseridos na população, a chamada *generation-gap*. Na maioria das vezes esta questão não é sequer abordada e, simplesmente, a cada geração, toda a população é substituída por novos indivíduos [44]. Em alguns algoritmos, chamados de estado estacionário (*steady-state*) ou regime permanente, apenas um ou dois elementos da população são escolhidos para serem substituídos pelos novos elementos. Apesar das vantagens e desvantagens de se fazer a substituição de uma das formas citadas ainda não estarem determinadas com precisão [51], a possibilidade de se fazer uma evolução mais semelhante à que ocorre na natureza (pais convivendo e competindo com seus próprios filhos) gera algumas questões. Por exemplo, como serão selecionados os indivíduos que devem *morrer* a cada geração? Esta seleção deve seguir algum tipo de norma (por exemplo, ser o *inverso* da seleção para reprodução[13]), ou deve-se fazer substituição aleatória? Para este trabalho, basta mantermos em mente que, se nem toda a população precisa ser substituída, economiza-se recursos computacionais, pois não é preciso que se recalcule valores objetivos de uma população inteira a cada geração.

---

[12] Os indivíduos simplesmente recebem valores (não necessariamente proporcionais) de acordo com seu valor objetivo, indicando qual indivíduo teve o valor mais baixo, qual foi o segundo menor, e assim por diante.

[13] Por exemplo, utilizando-se a mesma função para seleção, e considerando-se aqueles com pior valor.

3.5. Reprodução e Mutação

Nesta seção trataremos das duas formas mais comumente utilizadas como técnicas de alteração do código genético, isto é, operadores de recombinação e de mutação. Estas técnicas atuam sobre os pais (aqueles indivíduos selecionados para reprodução) e sobre os filhos (os indivíduos gerados na reprodução), respectivamente. Suas funções são específicas e cada uma atua de forma contrária à outra. Ao passo que recombinação tende a fazer com que a população venha a convergir (de preferência para valores ótimos), mutação tem como objetivo manter um certo grau de diversidade na população (isto é, impedir que a população se torne convergente rapidamente demais). Recombinação tem o objetivo específico de construir novos indivíduos que contenham algumas das características de seus progenitores (esquemas), enquanto que mutação tem o objetivo de *atrapalhar* esta construção [52], mudando aleatoriamente valores dos genes dentro dos indivíduos criados. A idéia é que mutação pode, assim, evitar a perda definitiva de genes.

Três princípios a serem seguidos por qualquer operador de recombinação são apresentados em [53]. Os operadores devem *respeitar* os progenitores, *sortir apropriadamente* e *transmitir estritamente* as características dos progenitores. Estas características são um pouco complexas de se demonstrar para uma certa representação, mas facilmente exemplificadas: se ambos os pais têm olhos azuis, *respeito* significa que a cria deverá, necessariamente, ter olhos azuis. Se um progenitor tem olhos azuis e o outro tem cabelos castanhos, a cria gerada deverá poder ter olhos azuis ou cabelos castanhos, ou ainda, ambos (as características dos dois progenitores *sortidas*). Já a *transmissão estrita* das características diz que se um progenitor tem olhos azuis e o outro tem olhos castanhos, a cria só poderá ter olhos azuis ou castanhos. É difícil de comprovar que estas características estão presentes em um esquema de recombinação, mas elas estão, geralmente, presentes nos operadores clássicos que atuam em representações binárias [53].

Um grande número de operadores para reprodução já foi criado e utilizado, com diferentes graus de sucesso, e uma variedade de problemas. Como grande parte das representações utilizadas em problemas combinatórios requer que valores não possam ser repetidos em cromossomas (como, por exemplo, o fato de o caixeiro-viajante não poder passar duas vezes pela mesma cidade), estes operadores tendem a evitar que isto aconteça.

Um dos primeiros operadores especializados para problemas combinatórios foi o PMX (*Partially-Mapped Crossover*), criado por GOLDBERG [40], e exemplificado na figura 3.2. POON e CARTER [27] e MURATA *et al.* [14] fazem um estudo sobre vários tipos de operadores de recombinação para diferentes representações, ao passo que GOLDBERG [25] explica o funcionamento de vários operadores diferentes Dois destes operadores estão descritos na figura 3.3.

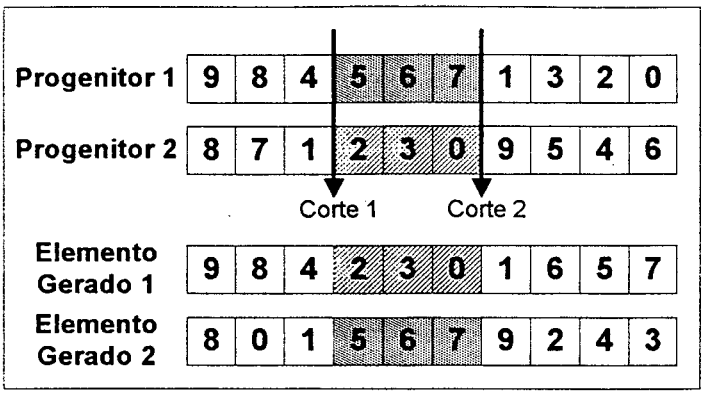

Figura 3.2 - Operador PMX de Goldberg

Operadores multi-ponto [25, 54, 55], operadores específicos para problemas de escalonamento [30] e seqüenciamento [26], operadores uniformes [56] e operadores auto-adaptáveis [57] são alguns exemplos de operadores de reprodução, mas uma descrição mais completa destes não será feita aqui.

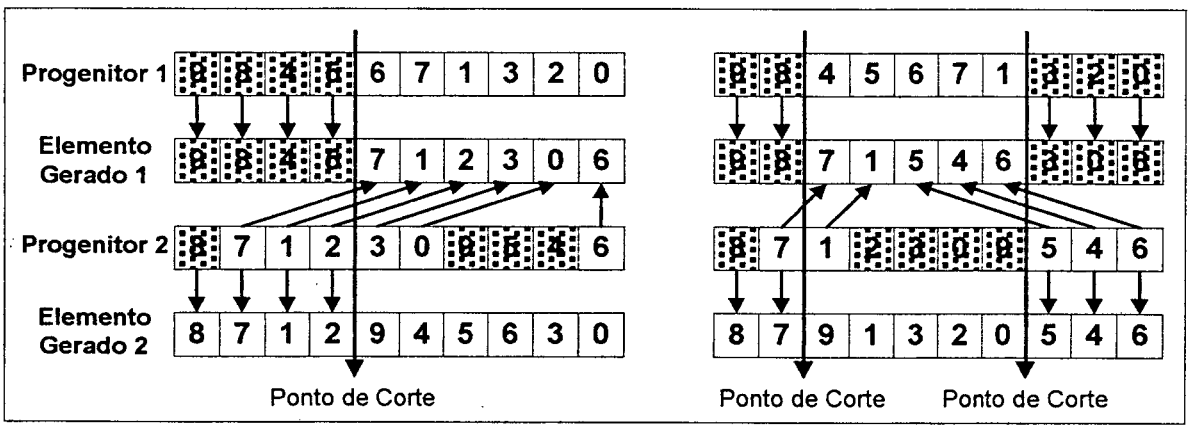

Figura 3.3 - *Crossover* de 1 e 2 Pontos

Já para os operadores de mutação, não existem muitas regras a serem seguidas, e eles são, inclusive, muitas vezes considerados operadores secundários [25]. Contudo, não se pode negar sua importância na evolução de uma população. Como métodos de recombinação geralmente misturam características de dois progenitores, uma quantidade de informação pode ficar perdida. Mutação pode, neste sentido, re-estabelecer ou recuperar material genético perdido (ou ainda não explorado), evitando assim que a população venha a convergir para um valor sub-ótimo [58]. Inclusive, em alguns casos, a mutação, sozinha, pode proporcionar uma busca melhor do que com operadores de recombinação [59]. Duas formas de se implementar mutação são mostradas na figura 3.3, conforme MURATA *et al.* [14].

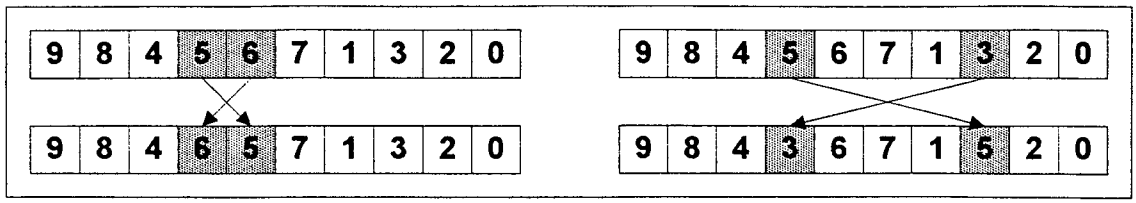

Figura 3.4 - Mutação Adjacente e Arbitrária

3.6. A Toolbox Genética para MATLAB®

MATLAB® (MATtrix LABoratory) foi criado no final de 1970 para cursos de teoria matricial, álgebra linear e análise numérica e, hoje em dia, tornou-se um ambiente de computação técnica da maior importância [60]. Dada a versatilidade de sua linguagem de programação de alto nível, problemas podem ser escritos em arquivos especiais (*m-files*) em uma fração do tempo que seria necessário para criar programas em C ou Fortran com o mesmo objetivo. É justamente deste modo que foi composta a *toolbox* genética para utilização em MATLAB® [45]. A semelhança entre o único tipo de dados com que MATLAB® trabalha (matrizes multi-dimensionais) e as estruturas de dados em um algoritmo genético ajudam a fazer de MATLAB® um ambiente muito propício para o desenvolvimento de estruturas genéticas. Os cromossomas de um algoritmo genético são facilmente representáveis através de uma matriz bi-dimensional, onde cada linha da matriz representa um indivíduo e cada coluna representa um gene. Após decodificados, cada indivíduo é submetido à avaliação da função objetivo, donde resulta uma matriz de largura um, com tantas linhas quanto membros na população. Subpopulações (para algoritmos genéticos paralelos) são facilmente tratáveis através de separações virtuais dentro de uma matriz *população*. Não é difícil ver como a habilidade de MATLAB® em trabalhar com matrizes pode ajudar no tratamento das unidades genéticas.

A *toolbox* utilizada neste trabalho é composta de uma série de arquivos *m* que fazem uso destas funções para tratamento matricial de MATLAB® para implementar uma série de métodos para algoritmos genéticos [45]. São, aí, implementadas as mais importantes funções de algoritmos genéticos e, sobre a funcionalidade desta *toolbox*, foram implementadas mais rotinas para tratamento específico de problemas de escalonamento clássicos geralmente abordados na literatura, e problemas de escalonamento mais realistas. No próximo capítulo será tratada, em detalhes, a questão da implementação destas rotinas adicionais, assim como serão descritas, em maior detalhe, as rotinas da *toolbox* que serão utilizadas. Nos anexos estão os diagramas de fluxo das funções implementadas.

4. O Sistema Desenvolvido

O sistema proposto nesta dissertação foi desenvolvido sobre uma plataforma PC rodando Windows 95, utilizando MATLAB® versão 5.2 [**MATLAB**] e uma *toolbox* genética desenvolvida na Universidade de Sheffield [45]. Uma parte da funcionalidade genética necessária para a implementação do sistema proposto está prontamente disponível nesta *toolbox*, mas uma série de modificações foram necessárias para adaptá-la ao trabalho com representações genéticas menos convencionais. Apesar da *toolbox* apresentar um número de funções bastante úteis para o tratamento de representações com números reais, binários, ou mesmo inteiros, ela não dispõe de métodos específicos para tratamento de representações baseadas em ordem. Toda a parte de avaliação do algoritmo genético, especificamente para tratar problemas de escalonamento, também teve que ser implementada. Neste capítulo será descrita, em maiores detalhes, a implementação do sistema. As funções da *toolbox* que foram utilizadas sem modificações também serão descritas, mas em menor grau de detalhes. Vale a pena ressaltar que a funcionalidade da *toolbox* não se limita às funções utilizadas. Uma descrição mais completa desta *toolbox* pode ser encontrada em [45]. Tampouco será descrito o ambiente MATLAB®, haja visto que, sendo um software amplamente disponível e utilizado, é mais fácil e melhor que se procure informações a respeito em qualquer manual de instruções ou guia de usuários. Uma boa referência para iniciantes é [60]. Neste capítulo serão descritos os detalhes da implementação seguindo a ordem de execução normal de um algoritmo genético. Os diagramas de fluxo para as funções criadas estão no anexo B.

4.1. Representação

Conforme foi visto no capítulo anterior, uma das primeiras considerações a ser levada em conta é a questão da representação. Isto é, como se *traduzirá* o problema para uma forma com a qual o algoritmo genético possa trabalhar. Além disso, como foi visto no capítulo anterior, uma série de representações diferentes da representação clássica (binária) já foram utilizadas, algumas com maior sucesso que outras. Neste trabalho, faremos uso de uma representação que, apesar de redundante[1], tem algumas características que justificam seu uso, especialmente pelo fato de ser aplicável tanto aos problemas clássicos, como aos problemas realistas que serão tratados. Uma variação desta representação foi usada em [12]. Neste trabalho, cada cromossoma era uma lista de tamanho $j \times m$, onde j é o número de tarefas (*jobs*) e m é o número de máquinas do problema. Cada posição do cromossoma (gene) poderia conter o valor de qualquer tarefa. A primeira tarefa que aparece no cromossoma tem sua primeira operação escalonada o mais cedo possível, a segunda

tarefa que aparece no cromossoma também tem sua primeira operação escalonada o mais cedo possível, resguardadas as relações de precedência e exclusão. Aparecendo a mesma tarefa novamente, esta tem a sua segunda operação escalonada, e assim por diante. Qualquer escala gerada desta forma é, garantidamente, uma escala ativa [11]. Não é difícil ver como esta representação é bastante apropriada para problemas clássicos, onde as relações de exclusão entre operações de uma mesma tarefa impedem que duas operações possam ser executadas simultaneamente. Já se isto fosse possível, haveria que ser definida alguma forma de se decidir qual operação seria escalonada antes. É também óbvio que qualquer permutação do cromossoma sempre gera uma escala válida [11]. Contudo, para reprodução sexuada (havendo interação entre dois progenitores para a geração de um novo elemento), são necessários operadores especializados para impedir que se criem elementos inválidos. Considere, por exemplo, o caso da figura 4.1. Ali, os elementos criados são inválidos, pois faltam alguns números de tarefas e outros aparecem mais que deviam. De que forma se consertaria estes elementos? Como definir uma forma para decidir qual dos genes substituir e por que valor este deveria ser substituído? Se esta substituição fosse feita aleatoriamente, perder-se-ia algum poder do algoritmo genético. Se, por outro lado, fossem modificados os operadores genéticos para, por exemplo, impedir cromossomas inválidos de aparecerem, estes operadores teriam que utilizar conhecimento adicional sobre o problema (por exemplo, o número de operações em cada tarefa, para saber quantas vezes cada tarefa deveria aparecer no cromossoma[2]), complicando seu funcionamento e limitando sua aplicabilidade.

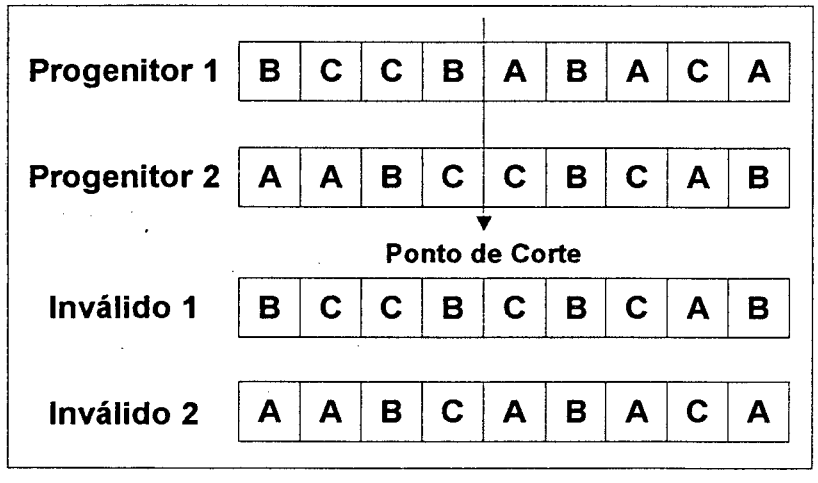

Figura 4.1 - Geração de Elemento Inválido

Por estes motivos adotou-se uma representação similar. Ao invés dos cromossomas listarem números de tarefas, eles listarão, diretamente, os valores das operações de cada tarefa,

---

[1] Várias formas diferentes existem de se representar a mesma solução (mapeamento n para 1), dando origem a falsa competição entre vários cromossomas que, na verdade, representam a mesma solução.

[2] Isto seria ainda mais complicado em se tratando de problemas realistas, onde o número de operações necessárias para cada tarefa muda de acordo com as diferentes rotas que podem ser percorridas. Além disto, como decidir que rotas seriam seguidas em cada caso?

contadas continuamente. Em outras palavras, para um caso onde $j = 10$, e $m = 10$, os valores a serem utilizados podem variam dentro do intervalo fechado $[0, (10 \times 10) - 1]$. Não é difícil de se imaginar de que forma se descobriria a que tarefa pertence uma operação listada no cromossoma. Para casos clássicos, através de simples operações aritméticas como divisões, arredondamentos, e cálculos de resto, identifica-se a tarefa apropriada[3]. Mas, esta representação também apresenta três problemas, bastante similares aos problemas da representação anterior, mas de mais fácil resolução. Esta representação também pode gerar cromossomas inviáveis[4], pois estes podem desrespeitar as relações de precedência entre operações de uma mesma tarefa. Contudo, como estas relações de precedência são estritamente definidas, é bastante simples resolver qualquer quebra de precedência, simplesmente escalonando-se a operação apropriada[5], fazendo com que qualquer permutação das operações gere uma escala válida. Em alguns trabalhos é defendida a idéia de se permitir que cromossomas inviáveis apareçam [42], ao invés de impedi-los de aparecerem. Estes seriam, então, avaliados de acordo com suas propriedades boas, para, de certa forma, ajudarem na busca. A justificativa para tal tese é que, muitas vezes, especialmente em espaços de busca multi-dimensionais, uma solução ótima pode estar no limiar entre uma região viável e uma região inviável. Em se permitindo que cromossomas inviáveis sejam tratados, pode-se "cercar" estas soluções ótimas por vários lados, aumentando a probabilidade de se atingi-la.

O fato de se permitir que cromossomas inviáveis apareçam ajuda a responder outra pergunta: que rotas seguir em cada subprocesso? Como será visto mais adiante, a posição das operações no cromossoma será usada para definir uma espécie de prioridade entre as alternativas possíveis. Desta maneira, não é preciso que se faça uso de métodos aleatórios para decisão. Permitindo-se cromossomas inviáveis, dá-se igual chance a todas as rotas de serem a rota preferencial para cada subprocesso. Este processo será definido com maior rigor neste mesmo capítulo.

Outro problema desta representação é a possibilidade de serem duplicados números de operações após uma reprodução sexuada, em detrimento de alguma outra operação que fique faltando. Mas, a resolução deste problema também é bastante fácil, conforme será visto mais adiante. Basta, por enquanto, saber que não será necessária nenhuma heurística adicional para decidir como corrigir cromossomas inválidos, uma vez que estes serão impedidos de aparecerem. Operadores genéticos apropriados terão que ser desenvolvidos, mas estes não precisarão fazer uso de

---

[3] É claro que, neste caso está se assumindo que todas as tarefas têm o mesmo número de operações. E no caso realista, onde isto não ocorre? Os problemas de reprodução da representação anterior não se repetem? Como veremos a seguir, não.

[4] Cromossomas inviáveis não são inválidos. Cromossomas inválidos não representam uma solução, ao passo que cromossomas inviáveis representam uma solução que não é correta.

[5] No final das contas, é isto que faz com que a representação seja redundante, mas as outras vantagens desta representação tornam esta redundância aceitável.

conhecimentos específicos sobre o problema para operarem, simplificando sua atuação. Além disto, a exemplo da variação mostrada, utilizando-se uma forma apropriada de construção de escala pode-se, também, garantir que todas as escalas geradas serão ativas.

4.2. *Inicialização*

Tendo sido definida a representação a ser utilizada no algoritmo genético, o próximo passo é decidir de que forma se fará a criação da população inicial. Vários métodos já foram testados na literatura, incluindo a utilização de algoritmos determinísticos para a criação de uma população inicial de forma não-aleatória. Para problemas de escalonamento, LEE e CHOI [16] criam a população inicial inteira utilizando regras de prioridade para as operações, enquanto CHEN *et al.* [61] criam apenas metade desta população inicial utilizando métodos heurísticos. LOUIS e XU [23], por sua vez, determinaram que, para seu caso específico, 5 a 10% da população inicial deveria ser criada de forma não-aleatória, ao passo que REEVES [43] achou melhor criar apenas um dos membros da população inicial através de uma heurística construtiva. Como pode-se ver, não existe consenso quanto à criação da população inicial. Sabe-se, isto sim, que, quando a criação da população inicial faz uso de conhecimentos específicos do problema a ser resolvido, a taxa de mutação deve ser aumentada, de forma a impedir convergência prematura e que, na maioria das vezes, o importante é, simplesmente, manter uma população inicial bastante diversa [61].

Foi decidido que, nesta implementação, a população inicial será gerada de forma aleatória[6], utilizando a função **crtbp** da *toolbox*. Especificando-se o número de indivíduos da população desejada, o tamanho de cada indivíduo e a base (valor máximo) de cada gene, a função cria uma população aleatoriamente. A figura 4.2 ilustra este processo. Contudo, a função não impede que, em um mesmo cromossoma, valores apareçam duplicados e, tampouco, que todas as operações estejam presentes nos cromossomas. Para corrigir estes dois problemas, foi criada uma função adicional (**tiebreak**) que opera sobre esta população inicial, corrigindo-a.

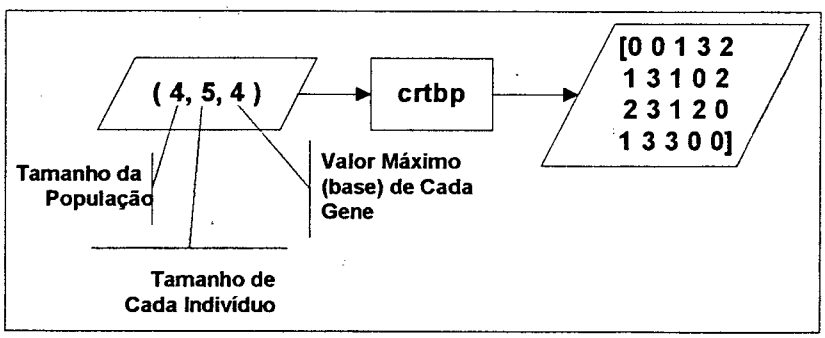

Figura 4.2 - Função CRTBP

Para a implementação da função **tiebreak** foi utilizada uma variação de um operador de *crossover* criado para ser utilizado em problemas combinatórios, a primeira versão do *tie-breaking*

crossover [27]. Um pouco modificado, excluindo-se a funcionalidade específica para realização de recombinação entre os dois progenitores e criação de novos indivíduos, o algoritmo se torna uma maneira eficiente de garantir que todos os cromossomas da população inicial sejam válidos, não repetindo e nem ignorando operações. Além disto, aplicado à população inicial, esta continua tão aleatória quanto antes. O funcionamento básico desta função pode ser visto na figura 4.3.

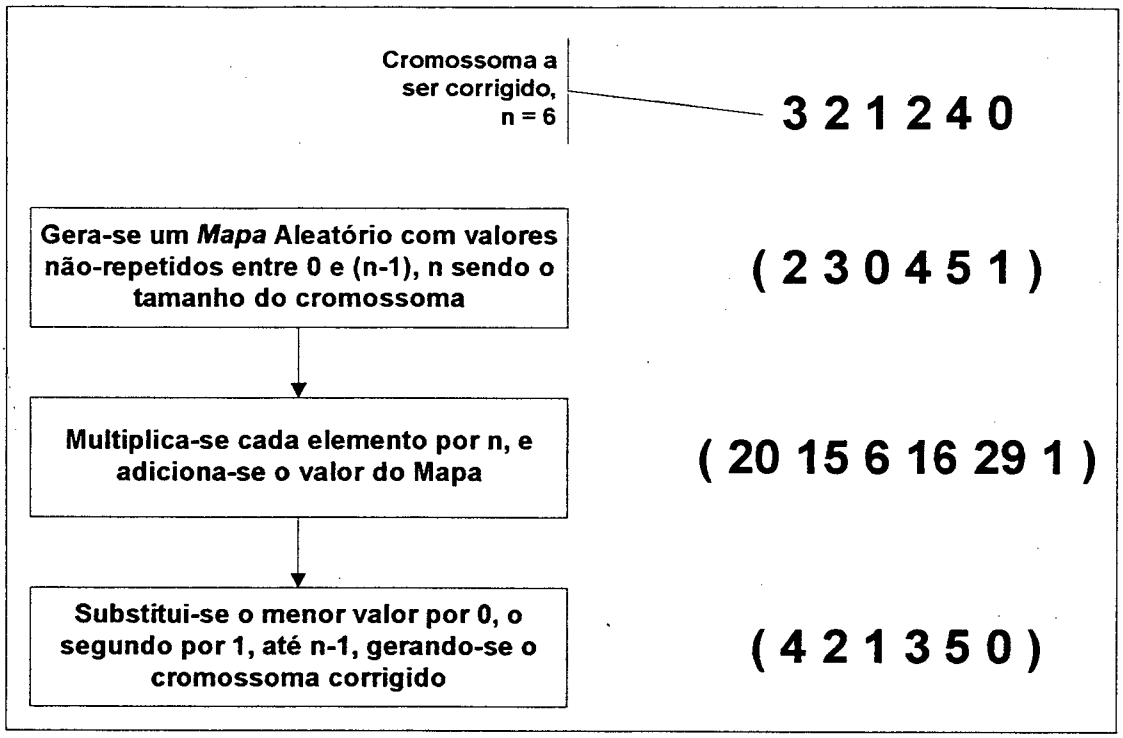

Figura 4.3 - Função TIEBREAK

4.3. Definição e Avaliação dos Problemas

Antes que se possa fazer a seleção dos indivíduos mais aptos para serem utilizados na geração de novos elementos, é preciso que se defina de que forma estes serão avaliados (primeira pergunta feita no capítulo anterior). É também aqui que entra a questão do mapeamento entre cromossomas (representação das soluções) e escalas (as soluções propriamente ditas). Para se fazer esta *tradução*, será necessário que se faça uma distinção entre os problemas clássicos e os problemas realistas a serem abordados. Afinal, apesar de ambos estarem sendo considerados neste trabalho, algumas pequenas diferenças precisarão ser levadas em conta. Por exemplo, para os problemas clássicos, apenas consideram-se tarefas (compostas de operações) e máquinas. Já para os problemas realistas, conforme foi dito no capítulo 2, cada tarefa é composta de uma ou mais partes, cada uma destas incluindo vários subprocessos, sendo estes compostos das operações propriamente ditas agrupadas em rotas e, além de máquinas, recursos adicionais também serão considerados. Por este motivo, duas formas de se fazer a construção das escalas serão utilizadas.

[6] Mas fica aqui a sugestão para desenvolvimentos futuros sobre este trabalho.

Ambas são bastante similares em seu funcionamento, mas levam em conta as peculiaridades de cada tipo de problema. Antes disto, porém, será explicada a forma de definição de cada tipo de problema, em arquivos utilizados para entrada de dados.

4.3.1. Definição do Problema Clássico

Para se fazer a definição dos problemas clássicos, dois arquivos de entrada de dados serão utilizados. O primeiro destes define as tarefas e as máquinas onde cada operação precisa ser executada, além dos tempos de execução de cada operação. Na figura 4.4 aparece um exemplo para um caso simples de três tarefas e três máquinas. Já o segundo arquivo para definição do problema clássico é um que define as relações de precedência e exclusão entre as operações de cada tarefa. Para o mesmo caso da figura 4.4 cada tarefa teria suas relações de precedência definidas em uma matriz 3x3.

	Op 1- Màquina	Duração	Op 2- Màquina	Duração	Op 3- Màquina	Duração
Tarefa 1	1	5	2	6	3	1
Tarefa 2	2	8	3	2	1	7
Tarefa 3	1	4	3	3	2	5

Figura 4.4 - Definição do Problema Clássico

A definição das relações de precedência entre operações de uma mesma tarefa se faz da seguinte forma: qualquer valor $v = 1$ na posição ***(linha, coluna)*** significa que a operação ***linha*** precisa ser completada antes que a operação ***coluna*** possa ser iniciada, ao passo que um valor $v = 0$ indica que não existem relações diretas[7] de precedência entre as operações. A diagonal principal (***linha*** = ***coluna***) deve ter todos os elementos **1**. Mais adiante se verá o porquê desta necessidade. É fácil ver, creio, que todo e qualquer tipo de construção de precedências pode ser representada desta forma. Dois exemplos são mostrados na figura 4.5. Mantendo-se os valores abaixo da diagonal principal (***linha*** > ***coluna***) sempre iguais a **0**, evita-se a criação de precedências circulares (por exemplo, operação 2 tendo que ser executada antes da operação 4; e operação 4 tendo que ser executada antes da operação 2). Adiantando-se, e já pensando nos problemas mais realistas, caso uma operação precise ser executada mais de uma vez, qualquer valor $v > 1$ indicaria a quantidade de vezes que se precisa realizar esta operação antes que a próxima operação possa ser escalonada.

[7] É obvio que podem existir relações indiretas de precedência entre elas. Por exemplo, o caso das operações 2 e 4 no primeiro exemplo da figura 4.5: a operação 2 não precede diretamente a operação 4 mas, devido às outras restrições, a operação 4 nunca poderá ser executada antes (ou ao mesmo tempo) da operação 2.

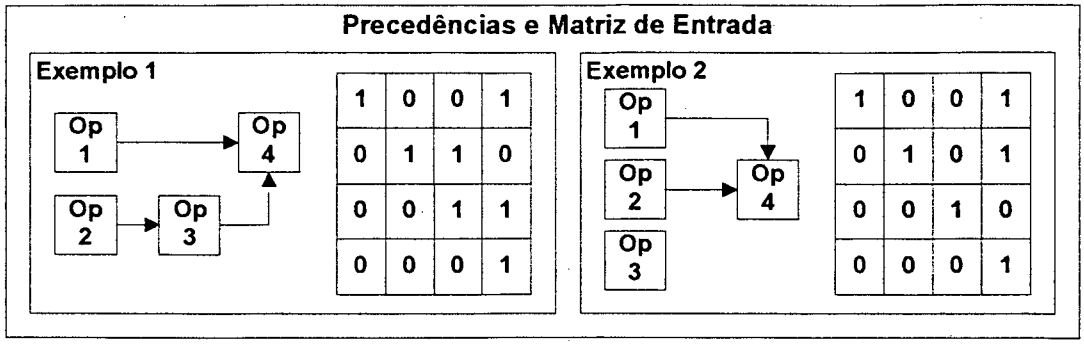

Figura 4.5 - Definição das Relações de Precedência

Uma vez lido este arquivo, as matrizes de precedência são unidas em uma matriz tridimensional, para que a sua passagem, como parâmetro entre funções, seja facilitada[8]. Cada página da matriz tri-dimensional será a matriz de precedências para cada tarefa (figura 4.6).

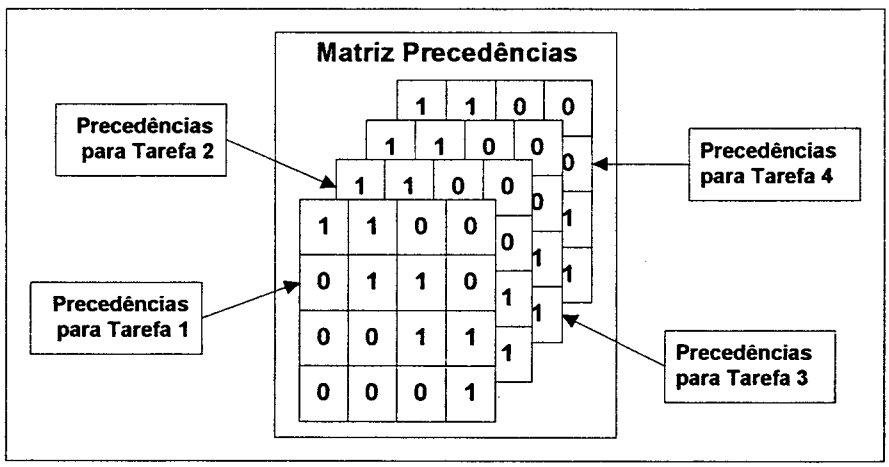

Figura 4.6 - Organização da Matriz de Precedências 3D

4.3.2. Avaliação dos Casos Clássicos

Para avaliação dos problemas clássicos, foi criada a função **evalpop** (*evaluate population*). Antes de se fazer a avaliação de cada cromossoma de uma população, é preciso que se construa a escala correspondente a cada um destes cromossomas. Na verdade, duas escalas serão criadas para que se possa manter correção quanto a tempos de execução. Para cada cromossoma, cria-se uma escala pertinente às tarefas e outra escala pertinente às máquinas. Isto é feito para que se facilite a procura por vagas (ou, espaços ociosos) no período de execução da máquina necessária para execução da tarefa. O diagrama de fluxo da função dá uma boa idéia do que é feito para o escalonamento de cada operação. Aqui será feita apenas uma descrição sucinta.

---

[8] Apesar da *toolbox* ter sido desenvolvida para MATLAB® versão 4.x (que não trabalha com matrizes tridimensionais), estas matrizes serão processadas apenas por funções criadas para esta dissertação, sem serem processadas por funções intrínsecas à *toolbox*, evitando qualquer problema causado pela diferença entre as versões.

Antes que se possa começar a processar os cromossomas, examinam-se os parâmetros de entrada da função para que se descubra se existem relações de precedência. Relembrando do capítulo dois, caso não existam relações de precedência entre as operações, o problema sendo resolvido é um de *open-shop*, e, neste caso, o vetor de referência (para as páginas da matriz tri-dimensional de precedências) é zerado[9]. Cria-se também um vetor coluna com o mesmo número de linhas que o número de indivíduos na população. Este vetor coluna será utilizado para armazenamento dos valores da função objetivo para cada cromossoma. Então, para cada cromossoma, são criadas duas escalas, uma para as tarefas e outra para as máquinas. Valendo-se das restrições descritas no capítulo dois, sabe-se que cada tarefa terá, sempre, uma operação em cada máquina. Portanto, o tamanho da escala para cada tarefa é conhecido. Nesta escala, listam-se apenas os tempos de início e término das operações. Não é listado o número das operações escalonadas em cada período. Para se manter consistência na ordem das operações escalonadas, o horário de término de cada operação é gravado, negativado, em uma cópia da página apropriada da matriz de precedências, na linha correspondente à operação, em todas as colunas onde o valor não seja zero, e servirá para determinar o prazo de término de cada operação. Desta forma pode-se facilmente saber a partir de que momento uma operação subsequente poderá ser escalonada. Se não houver relações de precedência, não é necessário que se considere o horário de término, pois as operações podem ser escalonadas em qualquer ordem, desde que respeitando a exclusão mútua entre operações da mesma tarefa.

Esta cópia da página de precedências também será utilizada na busca (quando necessário) da operação a ser escalonada. Esta busca será feita na função **chkprcdc** (***check precedences***). Para cada operação do cromossoma busca-se, nesta matriz de precedências, saber se esta operação já pode ser escalonada. Caso ela não possa ser escalonada (se nem todas as operações precedentes estiverem completas), faz-se uma busca nesta matriz para se determinar o número da operação que pode ser escalonada. Caso a operação já tenha sido escalonada, a busca é no sentido de se encontrar alguma operação sucessora. Retorna-se, sempre, o número da operação que pode ser escalonada, e o maior (mais tarde) tempo de término das operações imediatamente anteriores a esta. Caso nenhuma operação seja encontrada, um sinal de erro é enviado, parando execução do programa. Isto é feito pois sabe-se que, no caso clássico, o número de operações é o mesmo para todas as tarefas. Quando ocorrer de uma operação estar listada no cromossoma fora da ordem de precedência, outra operação da mesma tarefa é escalonada. Portanto sempre aparecerá, no mesmo cromossoma, alguma operação desta tarefa que já tenha sido escalonada. Como sempre escalona-se uma operação, independentemente do número desta estar correto, deverá haver alguma operação que

[9] Não é obrigatório que todas as tarefas tenham relações de precedência. É permitido, por exemplo, que algumas tarefas sejam do tipo *open-shop* (sem precedências), ao passo que as outras sejam do tipo *job* ou

ainda não tenha sido escalonada. Se isto não ocorrer é porque deve ter acontecido algum erro de execução, impossibilitando que se continue o processamento sem mais erros.

Feito isto, encontra-se a linha apropriada nas escalas das tarefas e das máquinas (a máquina necessária para escalonamento da operação). A função **findidle** acha, em cada uma das escalas, todos os períodos ociosos disponíveis, levando em conta, é claro, o prazo de término da operação anterior e o tempo de execução requisitado pela operação a ser escalonada. Já **insertat**, por sua vez, tenta achar os períodos ociosos combinados (entre máquina e tarefa) em que a operação caiba. Aquele que iniciar mais cedo é retornado. A função **insertat** está um pouco mais detalhada a seguir.

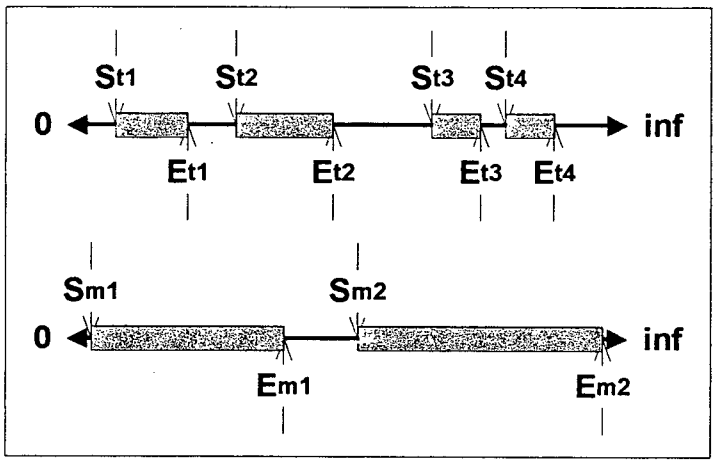

Figura 4.7 - Períodos Ociosos da Máquina e Tarefa

Considere duas listas de períodos ociosos, cada um com um horário de início S e término E, em ordem crescente, conforme a figura 4.7 (na verdade, nas escalas do programa, não serão listados os horários de início e término, mas sim o início e duração de cada período ocioso), e o tempo de execução *Exec* da operação a ser escalonada. Os períodos ociosos podem ser iguais ou maiores que *Exec*. O menor horário de início é calculado a partir destas listas. São verificadas todas as combinações possíveis entre os períodos ociosos da máquina e da tarefa, levando-se em conta que $(S_{mx}, E_{mx}) \cap (S_{ty}, E_{ty}) = (S_{ty}, E_{ty}) \cap (S_{mx}, E_{mx})$. Sempre que $(S_t, E_t) \cap (S_m, E_m) \geq Exec$, esta interseção é listada como uma possível posição para escalonamento da operação, desde que a precedência entre operações de uma mesma tarefa seja respeitada. Contudo, nem todas as combinações precisam ser checadas. Se, por exemplo, $(S_{tx}, E_{tx}) \cap (S_{my}, E_{my})$ for um intervalo possível, por que verificar os outros períodos ociosos da tarefa? Afinal, as comparações futuras seriam com (S_{tz}, E_{tz}), $z > x$. Sabendo-se que $S_1 < E_1 < S_2 < E_2 < \ldots < S_n < E_n$, isto implica que a próxima interseção a ser encontrada iniciaria, necessariamente, mais tarde. Portanto, assim que se encontra uma combinação de períodos ociosos viável, para-se de procurar[10].

---

flow-shop.

[10] Isto significa a utilização de *breaks* dentro dos laços necessários, o que não é muito recomendável. Fica a sugestão para melhorias.

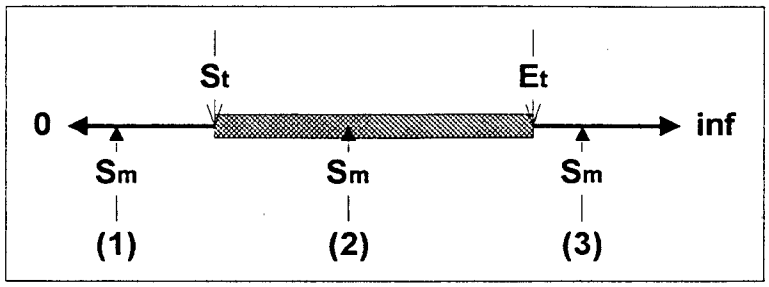

Figura 4.8 - Combinações Possíveis de Inícios de Períodos Ociosos

Como determinar se a interseção entre dois períodos ociosos é ou não viável será explicado a seguir. Considere apenas um período ocioso, que poderia ser tanto da máquina quanto da tarefa. Considerêmo-lo como sendo da tarefa. O início de qualquer período ocioso da máquina pode ser, relativamente ao início e término do período ocioso da tarefa, uma de três possibilidades: $S_m < S_t$ (chamemos esta possibilidade de **1**), ou $S_t \leq S_m < E_t$ (idem, **2**), ou $S_m \geq E_t$ (**3**), conforme a figura 4.8. Isto, claro, assumindo que $S_t > 0$. Caso contrário, restam apenas as duas últimas possibilidades. Mas, consideremos $S_t > 0$. Para cada uma das três possibilidades, existe um número de possibilidades para E_m, também com relação a S_t e E_t. Se (**1**), as possibilidades são $E_m \leq S_t$ (**1.1**), $S_t < E_m \leq E_t$ (**1.2**), ou $E_m > E_t$ (**1.3**). Se (**2**), $E_m \leq E_t$ (**2.1**) ou $E_m > E_t$ (**2.2**). Já se (**3**), apenas uma possibilidade existe, $E_m > E_t$. Os casos (**1.1**) e (**3**) aparecem na figura 4.9. É bastante óbvio que a interseção dos períodos ociosos não é viável.

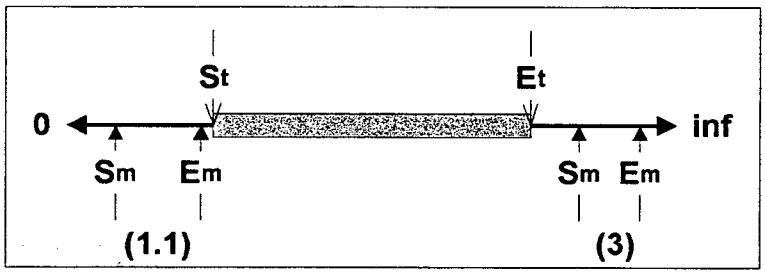

Figura 4.9 - Interseções Inviáveis de Períodos Ociosos

Os casos (**1.3**) e (**2.1**) são diagramados na figura 4.10. Ambos os casos representam interseções viáveis, conforme é mostrado. Os casos (**1.2**) e (**2.2**) são especiais, uma vez que podem, ou não, ser viáveis, dependendo de *Exec*, e estão mostrados na figura 4.11.

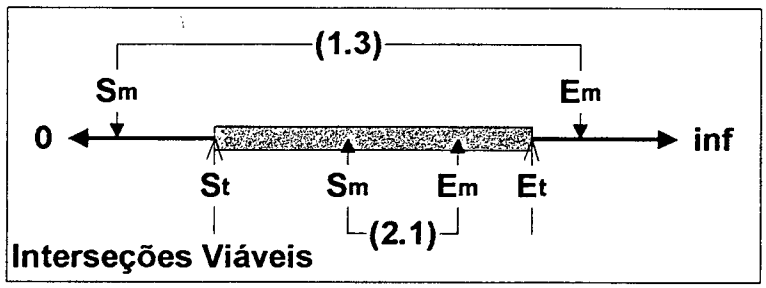

Figura 4.10 - Interseções Viáveis de Períodos Ociosos

Baseado no horário de início do menor período ocioso viável encontrado, calcula-se o prazo de término da operação e escalona-se esta operação na máquina e na escala da tarefa (função **insert**). Escalonando-se cada operação o mais cedo possível, garante-se que a escala gerada será ativa. Caso exista uma página de precedências, modifica-se esta, adicionando-se o tempo de término da operação nas posições apropriadas. À medida que cada cromossoma é decodificado, calcula-se o maior tempo de término dentre as tarefas (ou seja, o tempo total que foi necessário para que a última operação fosse completada) e coloca-se este valor no vetor objetivo. Esta será a medida da função objetivo[11], utilizada para avaliar o desempenho de cada cromossoma. Depois disto, o desempenho de cada cromossoma relativo ao restante da população será determinado.

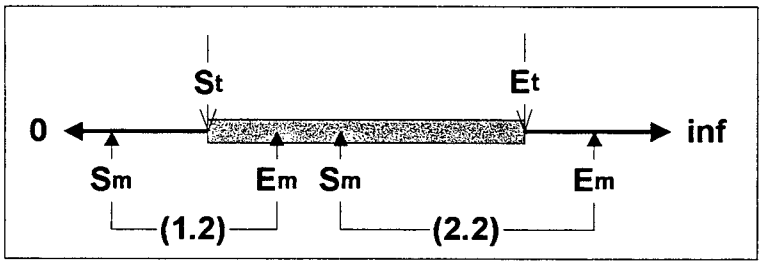

Figura 4.11 - Interseções Dependentes do Tempo de Execução

4.3.3. Definição do Problema Realista

Para os casos de tratamento de problemas realistas, são necessárias uma série de modificações na forma de definir o problema. Serão necessários, não dois, mas quatro arquivos para a definição de cada caso. O primeiro arquivo, que define as relações de precedência entre as partes pertencentes a cada tarefa, é similar ao arquivo que define relações de precedência entre operações no caso clássico. Contudo, algumas modificações são necessárias.

Supõe-se que não existem relações de precedência entre as tarefas. Já a definição de relações de precedência entre as partes que compõe uma tarefa serão feitas da mesma forma que no caso clássico, mas as matrizes para cada tarefa deverão estar na diagonal de uma segunda matriz, de acordo com a figura 4.12. Isto se faz necessário pois, neste caso, é possível que existam quantidades diferentes de partes em cada operação (a tarefa **0**, na figura, tem quatro partes, mas a tarefa **2** é composta de apenas três partes)[12]. Os valores negativos dentro da matriz de precedências serão

[11] Outra sugestão para futuras melhorias seria a incorporação de outras funções objetivo, incluindo, possivelmente, funções multi-objetivos.

[12] MATLAB® requer que todas as páginas de uma matriz multi-dimensional sejam do mesmo tamanho. Portanto, neste caso, não se pode construir uma matriz multi-dimensional a partir das matrizes individuais.

transformados em valores NaN[13] quando o arquivo é lido, para que sirvam como *barreiras*, definindo os limiares entre as matrizes de cada tarefa.

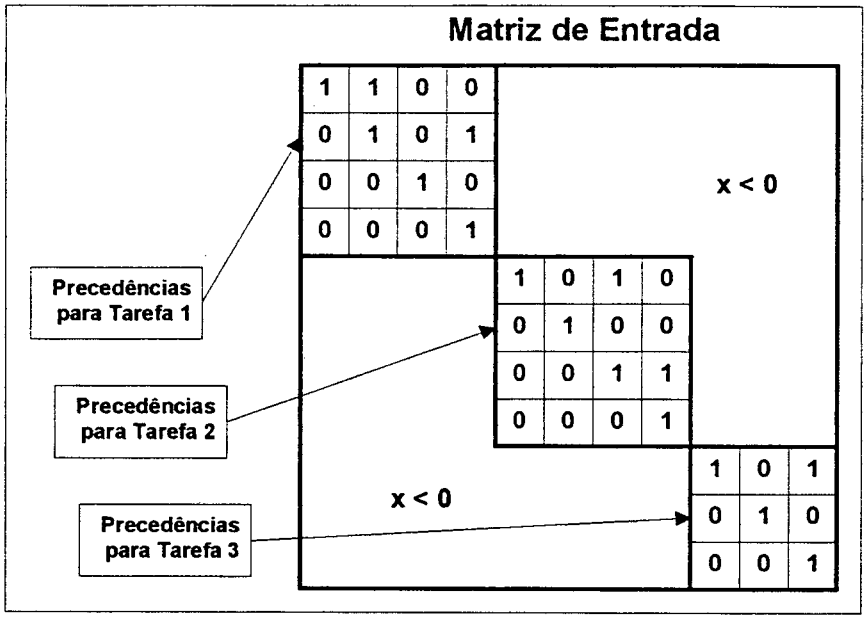

Figura 4.12 - Matrizes de Precedência para Problemas Realistas

O segundo arquivo para definição fornece informações a respeito dos recursos (tanto máquinas quanto recursos adicionais possivelmente necessários) disponíveis no sistema. Um exemplo é mostrado na figura 4.13. Vale a pena ressaltar que podem existir múltiplas unidades de um determinado número de recursos. Para cada unidade de cada recurso são listadas as paradas obrigatórias para manutenção, definindo-se o início e a duração destas paradas. Presume-se que todas as unidades, de todos os recursos, tenham, no mínimo, uma parada obrigatória para manutenção[14]. Neste arquivo também vale a regra de números negativos serem transformados em valores NaN, para garantirem matrizes retangulares.

[13] Valores NaN (*Not a Number*) podem ser utilizados em cálculos em MATLAB®, sem que causem erros durante a execução dos programas. Geralmente, o próprio NaN é o resultado de qualquer operação que o envolva.

[14] Esta obrigatoriedade facilitará a criação das escalas quando da avaliação de cada indivíduo.

			Paradas para Manutenção			
	Tipo	Unid.	Início	Duração	Início	Duração
	0	1	335	56	1781	22
	1	1	506	54	1630	36
	2	1	1053	1	-1	-1
Máquinas	4	1	528	1	1499	18
	5	1	291	36	1717	6
	5	2	936	2	-1	-1
	5	3	223	30	1663	36
	6	1	673	30	-1	-1
	6	2	1269	14	-1	-1
Outros	7	1	480	2	1602	1
Recursos	12	3	59	48	1237	48
	13	1	527	1	1713	24
	13	2	510	26	1686	18

Figura 4.13 - Definição de Recursos e Máquinas

O terceiro (e maior) arquivo é aquele que define as características individuais de cada uma das operações (figura 4.14). São definidas a tarefa, a parte, o subprocesso e a rota a que cada operação pertence. Também são listados os tempos de execução unitários de cada operação, o tempo de preparo (*setup*) da máquina necessária para execução (neste arquivo, apenas o tempo de preparo que independe da seqüência de operações na máquina é listado), o tipo de máquina requisitado, além dos tipos e quantidades de cada recurso adicional necessário para a execução da tarefa. Além disto, em alguns casos, lista-se o horário de disponibilidade (*ready time*) da operação (isto é, o horário antes do qual a operação não pode ser executada). Para deixar a matriz de entrada retangular, valores negativos são utilizados.

									Recursos Adicionais Necessários									
Op	Tarefa	Parte	Subp	Rota	Setup	Exec	Liberação	Máquina	Tipo	Quant	Tipo	Quant	Tipo	Quant	Tipo	Quant	Tipo	Quant
0	0	0	0	0	6	2	0	2	8	1	-1	-1	-1	-1	-1	-1	-1	-1
1	0	0	0	0	6	3	0	3	6	1	-1	-1	-1	-1	-1	-1	-1	-1
2	0	0	0	0	7	1	0	2	6	1	7	1	13	1	-1	-1	-1	-1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
25	1	4	7	12	3	5	0	4	7	1	-1	-1	-1	-1	-1	-1	-1	-1
26	1	4	8	13	9	4	98	2	6	2	7	1	8	1	10	1	11	1
27	1	4	8	14	6	1	0	3	10	1	13	1	-1	-1	-1	-1	-1	-1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
133	4	17	49	70	7	4	0	2	6	1	10	1	-1	-1	-1	-1	-1	-1
134	4	17	50	71	5	3	0	3	11	1	-1	-1	-1	-1	-1	-1	-1	-1
135	4	17	50	71	3	1	0	4	7	2	10	1	12	1	-1	-1	-1	-1

Figura 4.14 - Definição das Operações

A definição dos tempos de preparo (*setup*) dependentes da seqüência de operações já escalonadas é feita no quarto arquivo de entrada de dados. Para cada operação são listadas as

operações que, sendo escalonadas antes desta, mudam o tempo de preparo necessário e os tempos que se aplicariam a cada caso (figura 4.15). A matriz também deve ser retangular e, para isto, usam-se valores negativos.

Com estes quatro arquivos, pode-se definir completamente um problema nos moldes das restrições que foram relaxadas na seção 2.5. Em [2] foram sugeridos seis problemas para serem utilizados no teste de sistemas de escalonamento de problemas reais. Dois destes serão tratados neste trabalho, juntamente com cinco problemas clássicos geralmente utilizados para teste e validação de métodos de escalonamento. As listagens completas destes sete problemas estão no apêndice B.

	Tempos de Setup																				
Operação	Op. Anterior	Tempo	Op. Anterior	Tempo	Op. Anterior	Tempo	Op. Anterior	Tempo	Op. Anterior	Tempo	Op. Anterior	Tempo	Op. Anterior	Tempo	Op. Anterior	Tempo	Op. Anterior	Tempo	Op. Anterior	Tempo	Op. Anterior	Tempo
6	15	7	17	5	33	5	41	5	68	6	76	8	80	6	110	5	124	9	-1	-1	-1	-1
14	24	4	48	1	50	3	91	3	98	6	100	6	109	4	112	4	116	4	-1	-1	-1	-1
15	6	9	17	3	68	5	76	7	110	5	124	3	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
17	6	7	15	10	33	4	41	4	68	3	76	6	80	6	124	3	-1	-1	-1	-1	-1	-1
24	14	7	29	5	48	7	50	7	91	0	98	3	112	7	116	7	-1	-1	-1	-1	-1	-1
100	14	6	24	2	29	6	48	3	50	4	91	5	109	5	116	4	120	5	-1	-1	-1	-1
109	14	5	24	5	48	6	50	7	91	5	98	0	110	10	112	5	116	2	120	5	-1	-1
120	14	6	24	7	29	2	48	8	50	7	91	9	98	6	100	4	109	6	112	4	116	7
124	17	3	41	5	68	4	80	4	110	9	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Figura 4.15 Definição dos Tempos de Preparo Dependentes da Seqüência

4.3.4. Avaliação dos Casos Realistas

Como não poderia deixar de ser, a função criada para a construção das escalas e avaliação do desempenho de cada cromossoma para os problemas reais é bastante similar àquela criada para o caso clássico, inclusive no nome. A função **evalpop2**, contudo, é um tanto mais complicada, o que, de forma alguma, vem como uma surpresa, haja visto as características dos problemas realistas.

O primeiro passo na avaliação é a construção da escala dos recursos. Como dissemos, todas as unidades de recursos precisam ter, no mínimo, uma parada obrigatória de manutenção. Estas paradas são utilizadas como inicializadoras das escalas. Diferentemente do problema clássico, neste as escalas incluirão os números das operações escalonadas em cada período. Para as paradas de manutenção, usa-se, simplesmente, o horário de início negativado (para designar o número da operação). Desta forma, impede-se que uma parada de manutenção venha a causar confusões quanto a mudanças em tempos de preparo de máquinas, e assim por diante. Isto poderá ser visto com maior clareza na descrição da função **schdlop** (*schedule operation*). A escala das partes também é inicializada (neste caso, usar-se-á uma matriz com tantas linhas quanto forem as partes no

problema e colunas em múltiplos de três, para listar número, início e término das operações). Após isto ser feito, inicia-se a decodificação de cada cromossoma.

Para cada cromossoma, constrói-se a seqüência preferencial das rotas. De acordo com a posição de cada operação no cromossoma, inclui-se a rota a que esta operação pertence em um vetor. Este processo é demonstrado na figura 4.16. Os número das rotas não são repetidos e este vetor será utilizado para decidir que rota tomar quando se iniciar um novo subprocesso.

Após isto, determinam-se as informações a respeito da operação listada (parte a que pertence, tarefa a que esta parte pertence). Com isto, pode-se localizar a parte apropriada para a operação na matriz de precedências. Lembrando, neste caso a matriz de precedências lista as relações entre as partes de uma tarefa e não entre as operações de uma tarefa. Desta forma, **chkprcdc2** determina que parte da tarefa em questão deve ser trabalhada, seguindo, basicamente, a mesma lógica de **chkprcdc**. Contudo, neste caso, não se retorna um erro quando não se encontra nenhuma parte ainda não escalonada. Isto porque as várias rotas que podem ser percorridas podem mudar o número total de operações necessárias para execução completa de uma parte. Ou seja, em alguns casos poderemos ter mais operações listadas no cromossoma do que são necessárias para completar a execução, sem que isto seja um erro. Caso a parte já tenha sido completada, passa-se a verificar a próxima operação listada. Caso contrário, obtém-se as informações pertinentes à parte que precisa ser trabalhada (subprocessos, rotas disponíveis, entre outras).

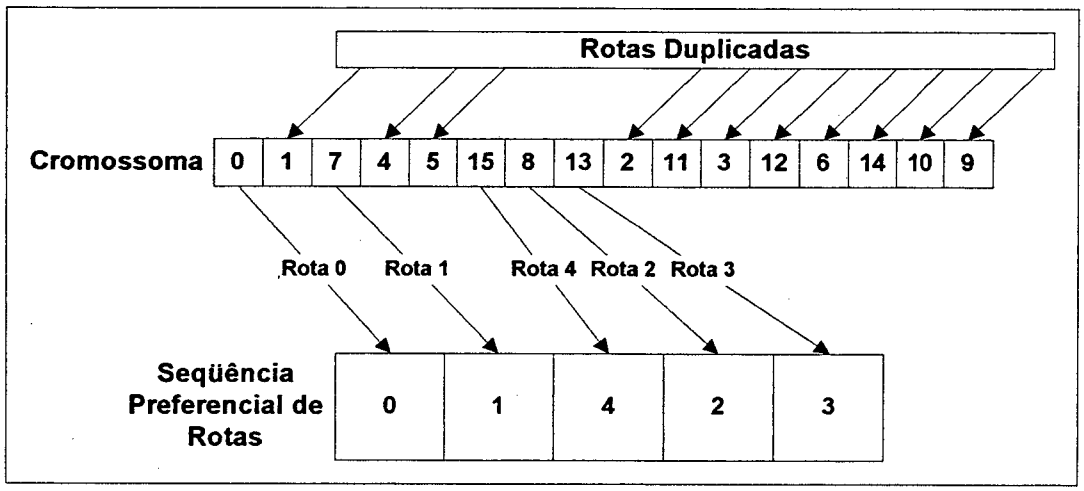

Figura 4.16 - Construção da Seqüência Preferencial de Rotas

Estas informações são repassadas a **chkschdl** (***check schedule***), que determinará qual a próxima operação a ser escalonada. Em **chkschdl**, caso seja necessário, também será determinada a rota que deverá ser seguida. Para tanto, a função **startnewsubp** (***start new subprocess***) utiliza a lista de rotas preferenciais criada a partir de cada cromossoma. Além disto, a rota à qual pertence a operação listada no cromossoma (independentemente de esta estar pronta ou não para ser escalonada) é usada. Caso esta rota seja uma das rotas do subprocesso a ser iniciado, esta rota é

escolhida e retorna-se a sua primeira operação. Caso contrário, verifica-se a lista preferencial de rotas e a primeira operação da primeira rota pertencente ao subprocesso é retornada para escalonamento. Além disso, retorna-se o tempo após o qual a operação pode ser escalonada. Se a operação que se achou como pronta para ser escalonada for a última operação de uma parte (isto é, a última operação do último subprocesso da parte), retorna-se também esta informação (a linha da matriz de precedências é retornada se a matriz tiver que ser modificada, caso contrário, retorna-se este número negativado) para que, após escalonada a operação, se possa modificar os valores na matriz de precedência apropriadamente, com o tempo de término da parte.

Feito isto, o escalonamento da operação apropriada na escala das máquinas é feito pela função **schdlop** (*schedule operation*). É nesta função que serão calculados os períodos ociosos das máquinas e da tarefa, assim como os tempos de preparo de máquina dependentes da seqüência de operações já escalonadas. Como as operações serão escalonadas de forma a garantir escalas ativas (isto é, o mais cedo possível), haverá casos onde uma operação será escalonada entre duas outras operações previamente escalonadas. Neste caso, é preciso que se leve em consideração tanto o tempo de preparo da máquina para a operação a ser escalonada, como a mudança no tempo de preparo da máquina para a operação seguinte. Como se pode ver na figura 4.17, os tempos de preparo de duas operações podem mudar sempre que se escalona uma operação em um período ocioso.

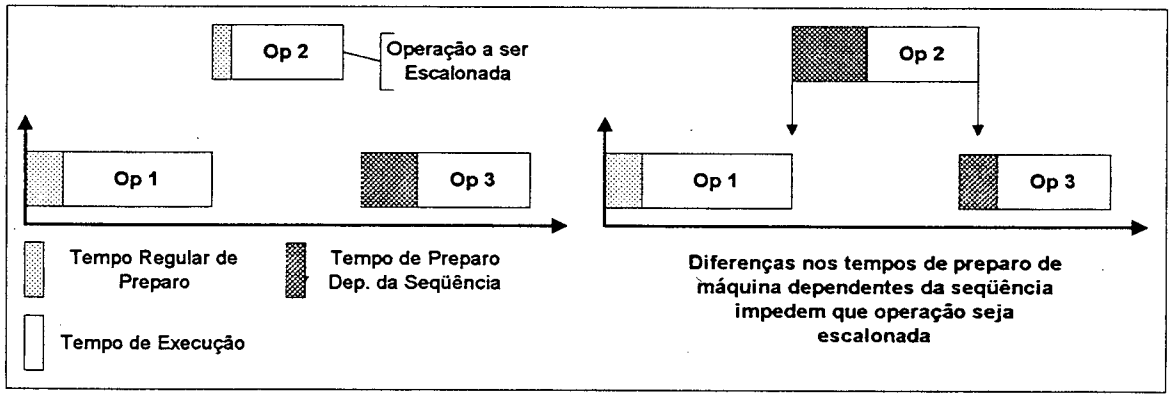

Figura 4.17 - Mudanças em Tempos de Preparo de Máquina

Como se vê na figura 4.17, é necessário que se verifique estas mudanças em tempos de preparo de máquinas, uma vez que eles podem mudar o tempo disponível e o tempo necessário para escalonamento. Na primeira parte da figura, observa-se que as operações **1** e **3** estão escalonadas em seqüência. Neste caso, a operação **3** tem um tempo de preparo de máquina dependente da operação previamente escalonada (**1**). Considerando-se apenas o tempo regular de preparo da operação **2**, esta operação caberia neste período. Contudo sua inserção, mostrada na segunda parte da figura) mudaria tanto o seu próprio tempo de preparo (pois este, agora, dependeria da operação **1**), quanto o tempo de preparo da operação **3**, que agora dependerá da operação **2**. Levando-se esta

mudanças em conta, a operação **2** não mais poderia ser escalonada nesta posição. E, o inverso também seria possível, onde as mudanças em tempos de preparo permitiriam que uma operação fosse escalonada em uma posição antes imprópria.

Baseada nestas mudanças em tempos de preparo, os quais mudam o período ocioso real existente, a função **findidle2** encontra todos os períodos ociosos (tanto nos recursos como nas tarefas) disponíveis[15]. Vale a pena mencionar que estes tempos de preparo não aparecem nas escalas que são criadas. Eles são, isto sim, calculados sempre que uma nova operação precise ser escalonada. Mesmo que fossem incluídos os tempos de preparo de máquina nas escalas, seria necessário revê-los, uma vez que eles podem mudar a cada operação escalonada. É óbvio que este tempo de preparo necessário é descontado (ou adicionado, caso o tempo de preparo da operação diminua) do período ocioso real disponível nas máquinas.

Tendo disponíveis os períodos ociosos existentes na máquina e nos recursos solicitados, assim como na tarefa à qual pertence a operação, pode-se achar o melhor lugar para se escalonar a operação. Antes disto, contudo, uma vez que múltiplas unidades de recursos adicionais podem ser requeridas pelas operações, é preciso que se ache a melhor combinação entre os períodos ociosos disponíveis em todas as unidades requeridas. Isto é feito na função **combine**, onde todas as possibilidades válidas (a combinação de tantas unidades quanto forem necessárias) são verificadas. Duas listas (matrizes) são o resultado desta função. Uma é a lista dos recursos encontrados e outra é a lista das unidades de recurso e dos períodos ociosos encontrados[16]. Depois disto, encontra-se o menor tempo disponível (aquele que inicia mais cedo), e escalona-se a operação no período ocioso. Sendo esta operação a última da parte, modifica-se a matriz de precedências apropriadamente. Os valores objetivos são calculados da mesma forma que foram calculados para os casos clássicos, mencionados na seção anterior.

4.4. Cálculo da Função de Aptidão

Tanto para os casos clássicos quanto para os casos realistas a serem tratados, também é necessário fazer uma comparação de cada indivíduo de uma população com o restante desta. Nem que isto signifique, simplesmente, normalizar-se os valores da função objetivo, para que o operador de seleção (que será visto mais adiante) possa entendê-los. Afinal, este valor da função de ajuste será utilizado como medida da qualidade de cada indivíduo, para que tenha mais ou menos chances

---

[15] É bom lembrar que os tempos de preparo somente se aplicam às máquinas. Eles não incorrem sobre (e portanto não mudam) o tempo de processamento nas tarefas e nos recursos adicionais requeridos pelas operações.

[16] A matriz dos períodos ociosos é organizada de forma a listar todas as unidades achadas e apenas um período ocioso por linha. Isto é feito para permitir que a procura seja feita dentro de um laço, que executa tantas vezes quantas são as unidades requeridas de um recurso.

de ser selecionado para reprodução. Para casos onde a minimização da função objetivo é desejada (como é o caso aqui), sempre é necessário que se faça esta conversão [45].

A *toolbox* apresenta dois métodos de se comparar os indivíduos. A função **ranking** simplesmente cria um *ranking* (que pode ser linear ou não) dos indivíduos de acordo com o valor objetivo de cada um, e de acordo com uma *pressão para seleção* (vista no capítulo 3) determinada pelo usuário. Já a função **scaling** converte os valores objetivos para uma medida de ajuste com um limite superior determinado pelo usuário, de acordo com o algoritmo proposto por GOLDBERG [25]. Contudo, este método não será utilizado nesta implementação.

4.5. Seleção

Quanto à seleção dos cromossomas a serem utilizados como progenitores para reprodução, a funcionalidade pronta da *toolbox* será utilizada. Duas maneiras de se fazer seleção são implementadas na *toolbox*, a saber, seleção por roda de roleta, também chamada de seleção estocástica com substituição, e seleção estocástica universal. De acordo com CHIPPERFIELD *et al.* [45], a seleção estocástica universal tem complexidade $O(N)$, ao passo que a seleção por roleta pode ser implementada com complexidade $O(NLogN)$, onde N é o tamanho da população. Apenas a opção de seleção estocástica universal será utilizada nos testes.

4.6. Reprodução e Mutação

Outra pergunta que deve ser respondida, conforme RUSSEL e NORVIG [20], é a forma como será feita a reprodução dos indivíduos selecionados para tal. Já vimos que uma série de operadores especializados foram utilizados para problemas de otimização combinatória. Neste trabalho foram implementadas as duas versões apresentadas no capítulo anterior, a saber, o *crossover* de um e dois pontos, apresentados por MURATA *et al.* [14]. Estes operadores evitam a criação de cromossomas ilegais, uma vez que impedem a duplicação de valores em detrimento de outros. Eles também apresentam a vantagem de explorar tanto o posicionamento absoluto dos genes dentro do cromossoma, quanto o posicionamento relativo entre estes, a exemplo do PMX criado por GOLDBERG [40], mas de uma forma mais simplificada. Uma porção do cromossoma é copiada diretamente ao elemento criado (explorando posicionamento absoluto no cromossoma), enquanto a porção restante recebe uma seqüência de acordo com o posicionamento relativo dos genes dentro do cromossoma. O operador de mutação também será implementado conforme MURATA *et al.* [14], e já foi demonstrado no capítulo anterior. Ambos os operadores (*crossover* e mutação) foram implementados sobre os operadores já disponíveis do *toolbox*. Para o operador de mutação, apenas uma implementação foi feita, uma vez que o operador de mutação adjacente é um caso especial do operador de mutação arbitrário. Desta forma, ambos os operadores podem agir durante a mesma execução do algoritmo genético. Para o operador de *crossover*, apenas uma implementação foi feita,

mas uma que permite que se especifique o número de pontos a serem utilizados. Assim, apenas o operador especificado será utilizado para cada oportunidade. Diagramas de fluxo e as listagens destes operadores podem ser encontrados no anexo B.

Uma ressalva importante a se fazer é sobre as probabilidades destes operadores genéticos, mais especificamente sobre a do operador de mutação. Geralmente, na literatura, são encontrados (e recomendados) valores para a probabilidade de mutação de 5% ou abaixo disto. Ou seja, cada gene de um cromossoma teria uma chance em vinte de ser mutado. Mas, para o operador genético implementado isto não é válido, uma vez que o operador de mutação não opera sobre cada gene, e sim sobre o cromossoma inteiro. Portanto, taxas de mutação baixas (como são normalmente utilizadas) se mostrarão insuficientes. Para levar isto em conta, a probabilidade de mutação foi aumentada, conforme será visto no próximo capítulo. O leitor desavisado certamente se surpreenderia com as probabilidades utilizadas.

4.7. Reinserção

Após serem selecionados os melhores indivíduos, e estes serem submetidos à reprodução, é preciso que se insira os elementos criados na população. Também é preciso saber que porcentagem da população antiga será substituída pelos novo elementos. Caso a população inteira deva ser substituída, a primeira pergunta perde seu sentido. Afinal, simplesmente destrói-se a população anterior e cria-se uma nova, apenas com os elementos gerados. Caso uma porcentagem menor que 100% da população tenha que ser substituída, é necessário, antes disso, que se faça a avaliação dos elementos criados. Para tanto, utiliza-se a mesma função **evalpop** (ou, **evalpop2**). É importante lembrar que, caso não seja necessário substituir toda a população, economiza-se tempo de computação. Afinal, a cada geração, apenas são calculados os valores objetivo daqueles elementos criados. Desta forma, permite-se uma evolução mais rápida do algoritmo.

A *toolbox* apresenta uma função para inserção de elementos criados na população. A função **reins**, através dos parâmetros de entrada, permite que se escolha o modo de substituição dos elementos. Tanto substituição aleatória uniforme (onde os indivíduos são substituidos aleatoriamente), como substituição baseada no valor da função de ajuste dos elementos (onde os elementos com pior valor de ajuste são substituídos pelos novos elementos criados) são possíveis. A função também permite que se faça substituição de um número menor de elementos do que foram criados, mas esta opção não será utilizada.

4.8. Término do Algoritmo Genético

Uma série de métodos podem ser utilizados para se definir quando a execução de um algoritmo genético deve ser interrompida. Em alguns casos, calcula-se uma função representativa da média dos valores da função objetivo para cada geração. Quando esta função não é

significativamente melhorada em um dado número de gerações, pára-se o algoritmo. Contudo, este método pode não ser o mais indicado, uma vez que a população pode não ter sua média aumentada, mas isto não significa que o algoritmo tenha trancado em algum mínimo local. Como ter certeza que, com mais uma, ou duas, ou três gerações, não se acharia a saída para outras regiões do espaço de busca? Portanto, nesta implementação será utilizado um número limite de gerações. Independentemente da evolução do algoritmo, chegando-se neste limiar, interrompe-se a execução. A mesma pergunta anterior pode ser feita, mas, neste caso, sempre pode-se reiniciar a busca utilizando a última população como ponto de partida para mais um tanto de gerações.

5. Testes e Avaliação do Sistema

A fim de avaliar o desempenho do sistema proposto e criado nesta dissertação, uma série de testes foram realizados, tanto com problemas clássicos normalmente abordados na literatura, quanto com problemas realistas de escalonamento propostos em [2]. Uma série de combinações de parâmetros genéticos foram utilizados, mas um estudo aprofundado sobre o efeito de cada um sobre a evolução nos casos de teste não foi feita. Tampouco foram feitas verificações mais completas com todas as combinações possíveis de parâmetros, uma vez que esta verificação levaria mais tempo que o disponível. Para cada combinação de tamanho de população, probabilidade de *crossover* e mutação, e *generation gap*, 32 variações dos outros parâmetros são possíveis. Se considerássemos apenas três variações para cada um dos quatro parâmetros citados, seriam mais 81 combinações. No total, teríamos 2592 combinações de parâmetros para cada problema.

Inicialmente o sistema foi testado para problemas clássicos de escalonamento. Como existe uma infinidade destes problemas, somente quatro foram escolhidos. Estes quatro problemas são normalmente abordados na literatura, e por isto, podem ser utilizados para uma comparação entre várias abordagens. Os problemas estão descritos, em maiores detalhes, no apêndice A. São eles ABZ6 (10x10), CAR4 (14x4), LA22 (15x10) e MT10 (10x10).

Como já foi dito, uma série de combinações de parâmetros genéticos foram testados, a fim de se descobrir que combinações poderiam dar os melhores resultados. A probabilidade de mutação utilizada nos vários casos variou entre 0,5, 0,7, 0,85 e 1,0. Já a probabilidade de recombinação (*crossover*) variou entre 0,7 e 1,0, em incrementos de 0,1. Também foram testados dois métodos de substituição de membros da população por novos elementos gerados, a saber, um método de substituição uniforme e um baseado no valor da função ajuste dos elementos (*fitness*). Além destes parâmetros, também variou-se a porcentagem da população a ser substituída em cada geração. Três valores foram utilizados: 0,5 (50% da população substituída em cada geração), 0,75, e 1,0 (quando toda a população antiga é substituída pelos novos elementos gerados). Assim, 96 casos de teste foram utilizados com os problemas clássicos. Outros parâmetros poderiam ter sido variados[1] para a obtenção de resultados mais específicos sobre o efeito de cada operador genético na evolução. Contudo, já existem estudos específicos sobre o efeito de alguns destes operadores na evolução de algoritmos genéticos [**referência**]. Apesar desta análise mais aprofundada estar fora do âmbito desta dissertação, isto não significa que um estudo desta ordem deixe de ser importante.

---

[1] Como, por exemplo, o tamanho da população, o número de gerações, o número de pontos a serem utilizados na aplicação de *crossover*, o método de seleção de elementos para reprodução, o método de rankeamento dos elementos da população para substituição por novos elementos, entre outros.

Além dos parâmetros combinados descritos acima, em todos os casos de teste que serão listados aqui foram utilizados os seguintes parâmetros (fixos): o tamanho da população foi fixado em 50, o número limite de gerações foi fixado em 100, sempre utilizou-se o método de *crossover* de dois pontos, assim como seleção estocástica universal para determinação dos indivíduos a serem utilizados na reprodução, *rankeamento* linear e a maior pressão de seleção possível para a *toolbox* (2).

Cada um dos problemas clássicos foi resolvido cinco vezes com os parâmetros descritos. As 17 melhores combinações foram selecionadas e estão descritas nos gráficos que seguem. É importante ressaltar que, muitas vezes, os resultados podem não ser significativos para um certo problema, mas a combinação ainda assim está listada por ter apresentado resultados promissores em outros dos problemas analisados. As 17 combinações que apresentaram os melhores resultados estão listadas na tabela 5.1.

	Probabilidade Xover	Probabilidade Mutação	Generation Gap	Método de Substituição
Combinação 1	0,7	0,5	0,5	uniforme
Combinação 2	0,7	0,7	0,75	uniforme
Combinação 3	0,7	0,85	0,75	uniforme
Combinação 4	1	0,5	0,75	uniforme
Combinação 5	1	0,85	0,75	uniforme
Combinação 6	0,7	1	1	uniforme
Combinação 7	0,9	0,5	1	uniforme
Combinação 8	0,9	0,7	1	uniforme
Combinação 9	0,9	0,85	1	uniforme
Combinação 10	0,9	1	1	uniforme
Combinação 11	1	0,7	1	uniforme
Combinação 12	1	1	1	uniforme
Combinação 13	0,7	1	0,5	fitness
Combinação 14	0,7	1	0,75	fitness
Combinação 15	0,7	0,85	1	fitness
Combinação 16	0,7	1	1	fitness
Combinação 17	0,8	0,85	1	fitness

Tabela 5.1 - Melhores Combinações de Parâmetros

Os gráficos descritivos das melhores combinações de parâmetros utilizados listam os melhores valores obtidos nas cinco execuções, juntamente com a média dos melhores valores encontrados. Ao invés de serem descritos em valores absolutos, os resultados foram calculados em termos da diferença entre o melhor valor obtido e a melhor solução conhecida para cada problema, expressa em porcentagem. Acredita-se que, desta forma, é mais fácil a comparação com outras abordagens. Os valores absolutos das melhores soluções conhecidas para cada problema também estão disponíveis no anexo A.

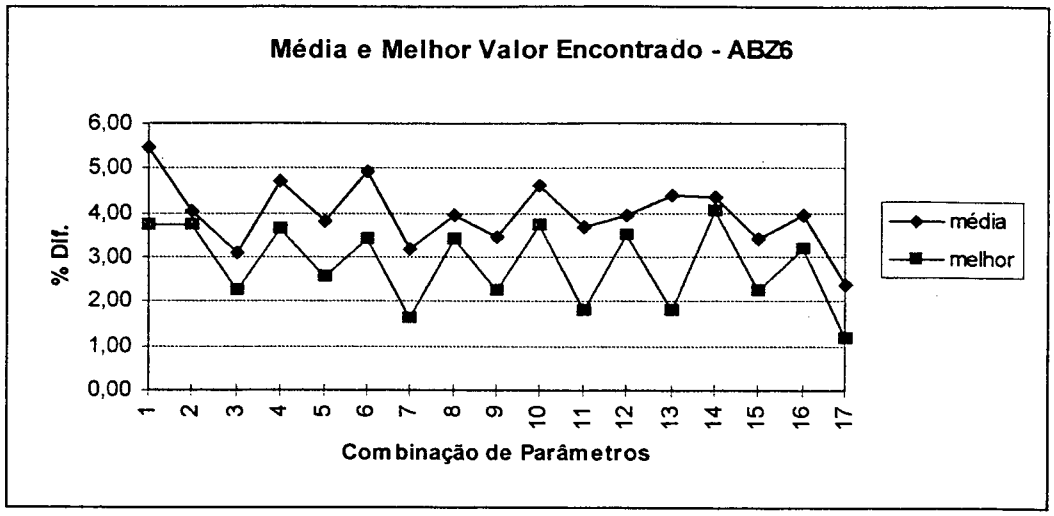

Figura 5.1 - Resultados ABZ6

Como se pode ver, os melhores valores encontrados para ABZ6 foram nos casos 7 e 17, onde o melhor valor foi de 1,59% e 1,17% acima da melhor solução conhecida, respectivamente. Pode-se também notar que o valor médio acompanha, na maior parte dos casos, o melhor valor encontrado. Isto indica, de certa forma, a melhor aptidão de algumas combinações de parâmetros para este problema especificamente. Em outra palavras, isto poderia indicar uma "tendência ao sucesso" nas execuções com estes parâmetros. É bastante claro que, quanto menor for o melhor valor, mais a média dos valores será "puxada" para baixo, mas, observando-se os resultados de cada uma das execuções, nota-se que, nestas combinações, valores baixos são a regra, ao passo que, em outras, a quantidade de valores altos e baixos é parelha (ou, com mais valores "ruins" do que "bons", como o caso 13). Seria interessante se, com um maior número de repetições, esta tendência fosse, ou não, confirmada.

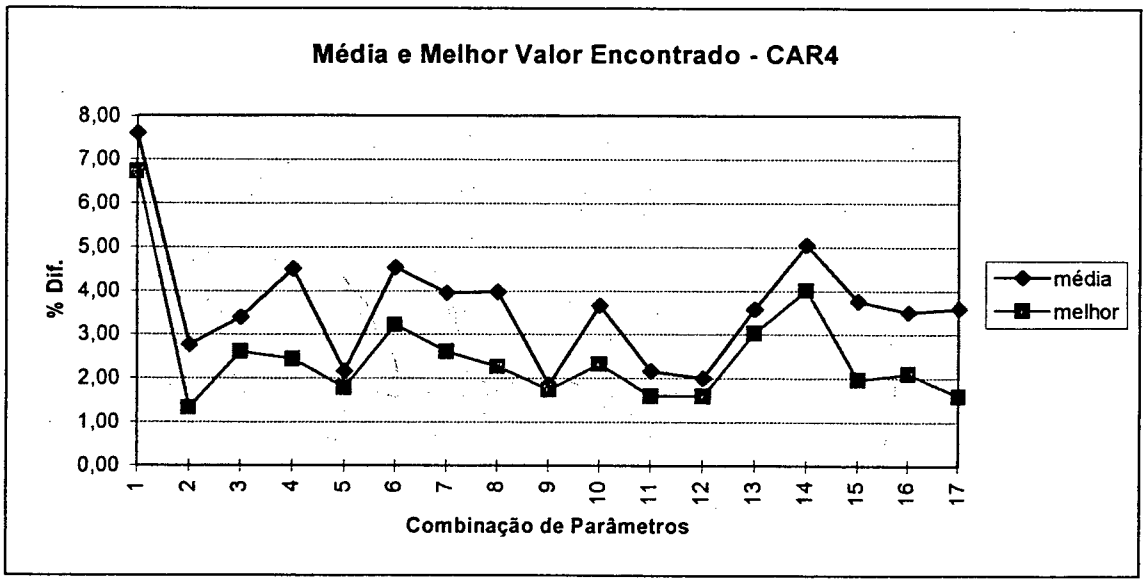

Figura 5.2 - Resultados CAR4

No gráfico da figura 5.2 fica bastante evidente a ocorrência de um caso em que, em todas as execuções, valores bastante semelhantes (e baixos) foram encontrados. Apesar de não serem os menores valores encontrados, a nona combinação de parâmetros sempre encontrou valores similares. Neste caso, fica clara a boa atuação do algoritmo genético. Apesar de tratar populações iniciais bastante diferentes, bons valores foram obtidos em todas as execuções. Neste gráfico nota-se, também, a combinação 17, onde a média dos melhores valores não acompanha o menor valor, o oposto do ocorrido com ABZ6, com a mesma combinação.

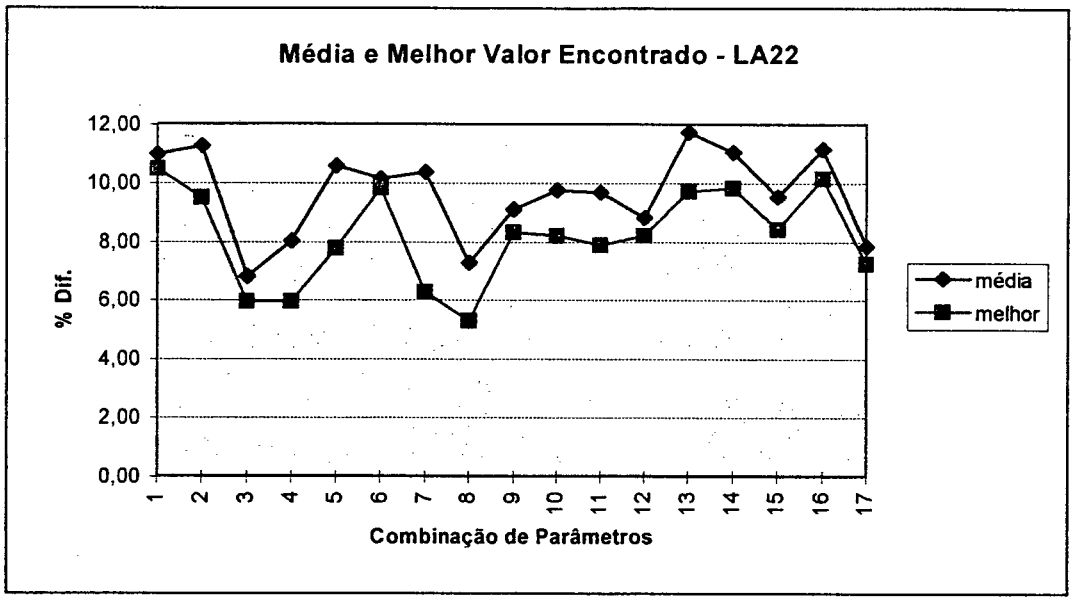

Figura 5.3 - Resultados LA22

No gráfico dos resultados de LA22, pode-se observar, além dos óbvios valores mais elevados encontrados, o oposto do que se percebeu anteriormente. No caso 6, em todas as execuções, os valores encontrados foram similares e mais elevados do que os valores encontrados na maioria dos outros casos. Pode-se ver que, mesmo em problemas envolvendo 150 operações, o sistema implementado encontra valores bem abaixo de 10%.

O último problema clássico a ser analisado foi MT10. Os resultados obtidos neste problema podem ser vistos na figura 5.4. Ali sobressaem-se os casos 6 e 12, onde, em todas as execuções, bons valores foram encontrados. Já no caso 7, que para ABZ6 havia apresentado bons resultados, percebe-se que a melhor solução encontrada é um caso isolado, o que se confirma ao analisar-se os resultados obtidos em cada uma das execuções.

A conclusão a que se chega, na análise destes resultados, é que as combinações de parâmetros têm efeitos variados em problemas diferentes. Não é possível dizer que uma dada combinação de parâmetros obterá bons resultados em todas as execuções, com todos os problemas. Com certeza, cinco execuções não bastam para que se possa fazer uma afirmação incisiva quanto ao

desempenho de cada uma das combinações[2]. Porém, pode-se já ter uma idéia de como seria o desempenho de cada uma destas combinações. Além disso, pode-se concluir que, independente dos parâmetros utilizados, o sistema desenvolvido apresenta um desempenho bastante bom para os problemas analisados, levando-se em conta o baixo número de indivíduos analisados. Como termo de comparação, PARK e PARK [39] e FANG *et al.* [12] utilizam populações de 500 indivíduos e 300 gerações, totalizando 150.000 indivíduos analisados. Em nosso caso, com uma população de 50 indivíduos e 100 gerações, nos casos em que usamos 100% de substituição das populações, apenas 5000 indivíduos são testados, com resultados apenas um pouco mais elevados em nosso caso.

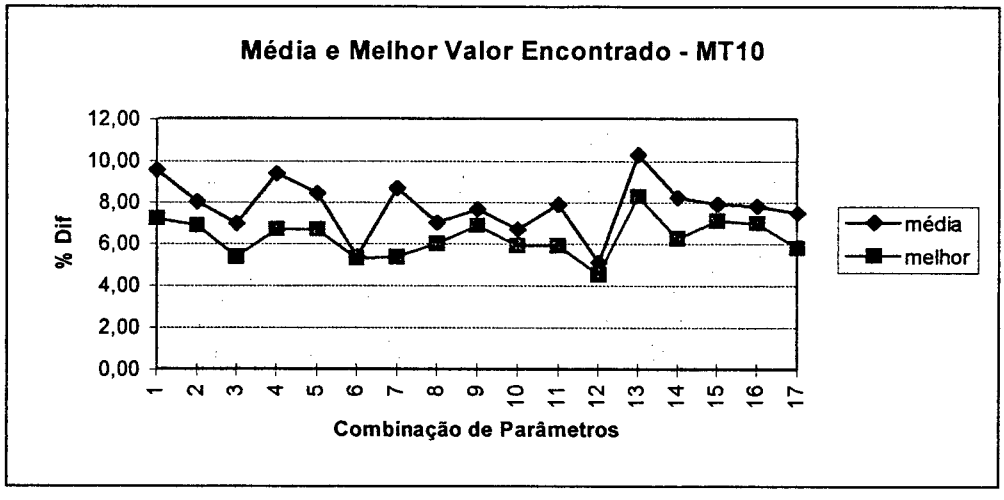

Figura 5.4 - Resultados MT10

A fim de testar o sistema desenvolvido com os problemas realistas, foi utilizada uma combinação de parâmetros que demonstrou bons resultados para os casos clássicos analisados. É necessário ressaltar, conforme já foi dito, que parâmetros que dão bons resultados em um problema podem não ser os ideais para outro. Contudo, não seria possível testar todas as combinações com os problemas realistas, pois o tamanho dos problemas acarreta um maior tempo de execução. O teste de múltiplas combinações torna-se, pois, inviável. Portanto, decidiu-se que o caso 3 seria utilizado. Esta combinação de parâmetros foi escolhida por ter apresentado resultados bons em todos os casos, com exceção de CAR4, e, mesmo neste, não sendo dos piores casos.

Os problemas realistas que foram utilizados para testes estão descritos no apêndice A. Eles são uma variação dos problemas criados por [2] e levam em conta um grande número de características realistas, conforme a descrição apresentada nos capítulos anteriores. Como nos problemas clássicos, os problemas realistas foram executados cinco vezes cada, com os parâmetros do caso três. Como a intenção desta dissertação não é a de fazer uma comparação específica com a implementação de CÂNDIDO [2], os resultados apresentados não se baseiam em uma diferença entre valores obtidos e melhores soluções conhecidas. Estão, isto sim, representados em valores

[2] Outras implementações utilizam 20, 30, ou até mais execuções [12, 33, 39].

absolutos, para que se possa melhor visualizar o funcionamento do sistema implementado. O gráfico que segue descreve a média das cinco execuções para o primeiro problema realista.

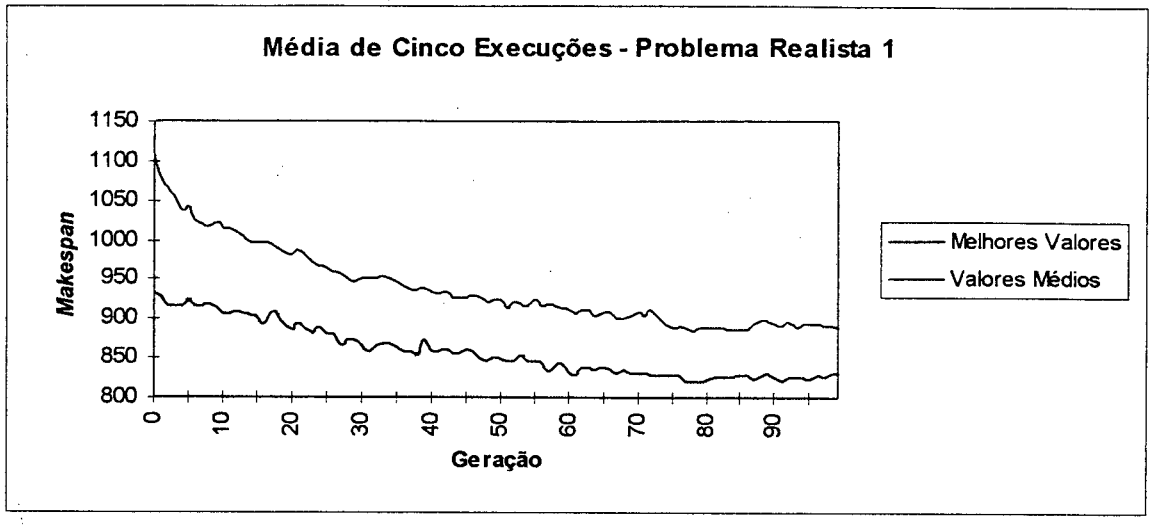

Figura 5.5 - Resultados Problema Realista 1

Estão descritos, no gráfico da figura 5.5, a média dos melhores valores encontrados (em cada execução) por geração, assim como a média dos valores médios das populações. Nesta última, pode-se perceber, claramente, a curva que normalmente descreve a operação de um algoritmo genético, com rápida evolução nas gerações iniciais e pouca mudança nas últimas. O fato da evolução nas gerações iniciais não ser tão acentuada como se esperaria, deve-se ao alto grau de redundância da representação utilizada. Também é visível que, nas últimas gerações, o algoritmo tende a ter valores mais elevados que anteriormente. Contudo, como vê-se em torno da geração 40, estas elevações são, sempre, seguidas por valores ainda menores, o que também é bastante descritivo da boa atuação de um algoritmo genético. Com apenas 3800 indivíduos analisados (38 elementos novos por geração, em 100 gerações), é evidente a boa atuação do sistema criado para este problema realista.

A figura 5.6 mostra os resultados obtidos nas cinco execuções do segundo problema realista que foi estudado. Pode-se ver, a exemplo do gráfico anterior, a curva descritiva da atuação do algoritmo genético, apesar desta apresentar consideravelmante mais variação, o que pode ser explicado pelo fato de, neste problema, haver grande diferença entre possíveis rotas para a execução das tarefas. A evolução, observa-se, é mais rápida que no primeiro caso. Além disto, observa-se que, no final do gráfico, ocorre uma grande melhora nos melhores valores encontrados, sem que esta queda seja acompanhada pela média das populações. Ao se analisar os dados individuais de cada execução, vê-se que este salto ocorreu apenas em uma delas. Como apenas 100 gerações foram utilizadas, acredita-se que, com um número maior de elementos testados, seja possível a obtenção de resultados melhores também nas outras execuções.

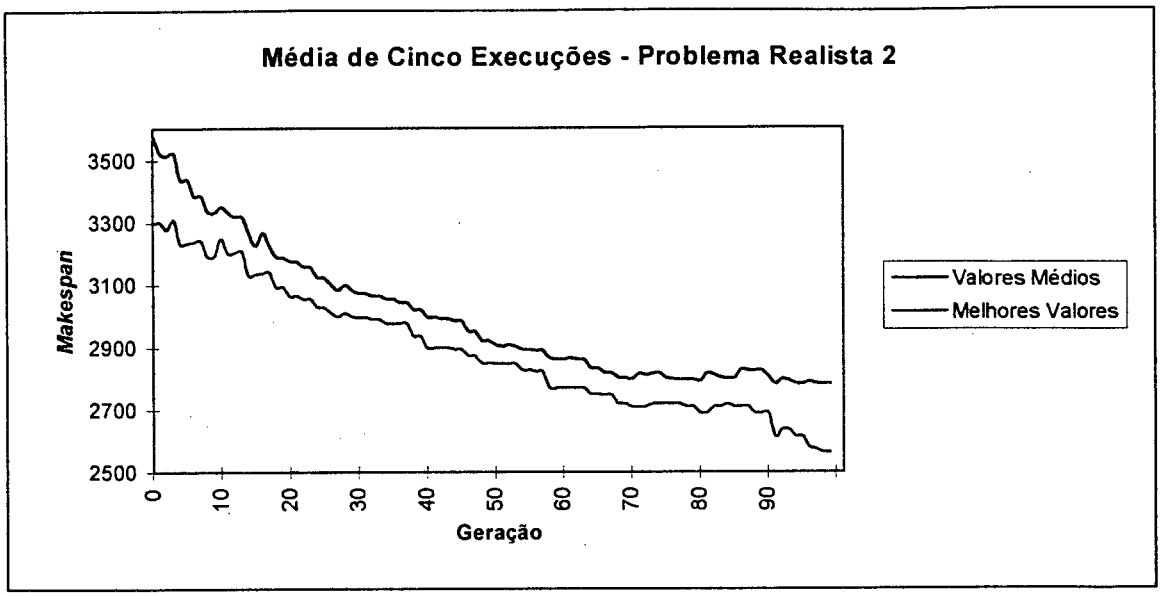

Figura 5.6 - Resultados Problema Realista 2

Apesar do sistema desenvolvido ter demonstrado sua capacidade de tratar problemas de escalonamento clássicos e reais, foram constatados alguns problemas durante sua execução. Em se tratando dos problemas clássicos, a performance não deixa nada a desejar, uma vez que resultados bons são obtidos em pouco tempo de execução. O tempo necessário para a avaliação de cada população com os problemas clássicos era da ordem de minutos. Contudo, para os problemas realistas, esta era a ordem do tempo necessário para a avaliação de cada indivíduo. Limitando-se o número de recursos adicionais necessários para cada operação a um máximo de três[3], foi possível reduzir-se o tempo necessário para avaliação. Isto porque, algumas vezes, a execução da função **combine** (que gera todas as combinações de períodos ociosos para a máquina e recursos adicionais) gerava matrizes de até 200.000 elementos. Apesar de MATLAB ser especializado no tratamento de matrizes, não é surpresa que, com matrizes deste tamanho, a execução seja demorada. A limitação do número de recursos permitidos diminuiu o tempo necessário, mas não satisfatoriamente. Maiores análises para a identificação de outros pontos que poderiam acarretar melhoras no tempo de execução não foram feitas.

[3] O número máximo em [2] é de cinco.

6. Considerações Finais e Sugestões para Trabalhos Futuros

Como foi visto, o sistema desenvolvido pode, com sucesso, tratar problemas de escalonamento realistas e clássicos, sejam eles do tipo *flow-shop*, *open-shop*, ou *job-shop*. Para os casos clássicos, as funções criadas tratam os três tipos de problemas sem que nenhuma modificação precise ser feita, inclusive permitindo que se trabalhe com os três tipos no mesmo problema. A mudança de um tipo de problema para o outro se faz, simplesmente, na forma de definição deste problema. Já para tratar problemas realistas, é necessário que se faça uso de uma série de funções específicas, mesmo porque, estas precisam levar em conta um sem-número de características adicionais. Para problemas clássicos, a performance mostrou-se satisfatória tanto em termos de resultados obtidos quanto em termos de tempo de execução. Já para os problemas realistas estudados, os resultados obtidos são, igualmente, satisfatórios, mas com um tempo de execução demasiadamente alto.

As características dos problemas realistas abordados aproximam a teoria de escalonamento à prática e, desta forma, a presente dissertação representa um passo importante. Ao abordar, de forma simples, problemas realistas, este trabalho representa um avanço significativo na utilização de algoritmos genéticos puros neste tipo de problemas, ao mesmo tempo em que não ignora a teoria, por tratar, de forma similar, problemas clássicos de escalonamento. Tendo sido desenvolvido com uma linguagem de programação amplamente disponível e fácil de usar, demonstra não serem necessários sistemas altamente especializados para o tratamento deste tipo de problema, apesar da grande demanda computacional necessária.

Ao longo do trabalho, foram feitas uma série de considerações sobre melhorias que podem ser feitas ao sistema, algumas das quais são resumidas aqui. Em primeiro lugar, seria interessante identificar que funções demandam maior tempo para execução e de que forma estas poderiam ser modificadas para que, nos casos realistas, resultados possam ser obtidos em menor tempo.

A inclusão de uma função para definição dos critérios a serem utilizados para a avaliação das escalas geradas representaria uma grande melhoria. Desta forma, não somente a comparação com outras abordagens seria facilitada, como também daria maior flexibilidade ao sistema, por permitir que a métrica de avaliação seja definida pelo usuário. A possibilidade de se fazer avaliações multi-objetivo também traria benefícios.

A inclusão de características dinâmicas, e o tratamento adequado pelas funções, representaria uma melhora significativa para a abordagem de problemas realistas, apesar da grande dificuldade de se tratar não-determinismo. Uma vez que o sistema desenvolvido só trata casos determinísticos, para que ele pudesse ser, efetivamente, aplicado em casos reais, precisaria poder

tratar também estas características não-determinísticas (por exemplo, máquinas quebradas). Além disto, um maior número de características realistas poderia ser incorporado, pois as características que foram levadas em conta nos problemas realistas representam apenas uma pequena parcela destas.

A modificação da função de criação da população inicial também poderia trazer vantagens. A inclusão de alguns membros gerados de forma não aleatória, poder-se-ia, talvez, chegar a resultados ainda melhores nos casos clássicos e realistas.

Por fim, seria interessante a realização de uma série adicional de testes, tanto com os problemas clássicos como os realistas, a fim de se saber o efeito de cada operador genético na evolução com o sistema desenvolvido. Possivelmente, melhores resultados poderiam ser obtidos desta forma.

Anexo A - Problemas Utilizados na Avaliação

Problemas Clássicos Utilizados

Os problemas clássicos utilizados na avaliação do sistema criado para esta dissertação, além de um sem-número de outros problemas de pesquisa operacional, estão disponíveis na biblioteca de pesquisa operacional mantida por BEASLEY [65]. Todos os casos serão listados em forma matricial, com cada tarefa sendo definida em uma linha da matriz, listando número da máquina onde a operação deve ser executada, e o tempo de execução. Também é listado o valor da melhor solução conhecida para cada problema, tomando-se o tempo de término (*makespan*) da última operação como métrica.

ABZ6 - Adams and Zawack 10x10 (Tabela 1, exemplo 6)

7	62	8	24	5	25	3	84	4	47	6	38	2	82	0	93	9	24	1	66
5	47	2	97	8	92	9	22	1	93	4	29	7	56	3	80	0	78	6	67
1	45	7	46	6	22	2	26	9	38	0	69	4	40	3	33	8	75	5	96
4	85	8	76	5	68	9	88	3	36	6	75	2	56	1	35	0	77	7	85
8	60	9	20	7	25	3	63	4	81	0	52	1	30	5	98	6	54	2	86
3	87	9	73	5	51	2	95	4	65	1	86	6	22	8	58	0	80	7	65
5	81	2	53	7	57	6	71	9	81	0	43	4	26	8	54	3	58	1	69
4	20	6	86	5	21	8	79	9	62	2	34	0	27	1	81	7	30	3	46
9	68	6	66	5	98	8	86	7	66	0	56	3	82	1	95	4	47	2	78
0	30	3	50	7	34	2	58	1	77	5	34	8	84	4	40	9	46	6	44

Tabela Anexo A .1 - Definição de ABZ6

Melhor solução conhecida: 973;

CAR4 - Carlier 14x4 exemplo 4

0	456	1	856	2	963	3	696
0	789	1	930	2	21	3	320
0	630	1	214	2	475	3	142
0	214	1	257	2	320	3	753
0	573	1	896	2	124	3	214
0	218	1	532	2	752	3	528
0	653	1	142	2	147	3	653
0	214	1	547	2	532	3	214
0	204	1	865	2	145	3	527
0	785	1	321	2	763	3	536
0	696	1	124	2	214	3	214
0	532	1	12	2	257	3	528
0	12	1	345	2	854	3	888
0	457	1	678	2	123	3	999

Tabela Anexo A .2 - Definição de CAR4

Melhor solução conhecida: 8003;

LA22 - Lawrence 15x10 (Tabela 7, exemplo 2); também chamado de setb2, ou B2

9	66	5	91	4	87	2	94	7	21	3	92	1	7	0	12	8	11	6	19
3	13	2	20	4	7	1	14	9	66	0	75	6	77	5	16	7	95	8	7
8	77	7	20	2	34	0	15	9	88	5	89	6	53	3	6	1	45	4	76
3	27	2	74	6	88	4	62	7	52	8	69	5	9	9	98	0	52	1	88
4	88	6	15	1	52	2	61	7	54	0	62	8	59	5	9	3	90	9	5
6	71	0	41	4	38	3	53	7	91	8	68	1	50	5	78	2	23	9	72
3	95	9	36	6	66	5	52	0	45	8	30	4	23	2	25	7	17	1	6
4	65	1	8	8	85	0	71	7	65	6	28	5	88	3	76	9	27	2	95
9	37	1	37	4	28	3	51	8	86	2	9	6	55	0	73	7	51	5	90
3	39	2	15	6	83	9	44	7	53	0	16	4	46	5	24	1	25	8	82
1	72	4	48	0	87	2	66	9	5	6	54	7	39	8	35	5	95	3	60
1	46	3	20	0	97	2	21	9	46	7	37	8	19	4	59	6	34	5	55
5	23	3	25	6	78	1	24	0	28	7	83	8	28	9	5	2	73	4	45
1	37	0	53	7	87	4	38	3	71	5	29	9	12	8	33	6	55	2	12
4	90	8	17	2	49	3	83	1	40	6	23	7	65	9	27	5	7	0	48

Tabela Anexo A .3 - Definição de LA22

Melhor solução conhecida: 927;

MT10 - Fisher and Thompson 10x10; também chamado ft10

0	29	1	78	2	9	3	36	4	49	5	11	6	62	7	56	8	44	9	21
0	43	2	90	4	75	9	11	3	69	1	28	6	46	5	46	7	72	8	30
1	91	0	85	3	39	2	74	8	90	5	10	7	12	6	89	9	45	4	33
1	81	2	95	0	71	4	99	6	9	8	52	7	85	3	98	9	22	5	43
2	14	0	6	1	22	5	61	3	26	4	69	8	21	7	49	9	72	6	53
2	84	1	2	5	52	3	95	8	48	9	72	0	47	6	65	4	6	7	25
1	46	0	37	3	61	2	13	6	32	5	21	9	32	8	89	7	30	4	55
2	31	0	86	1	46	5	74	4	32	6	88	8	19	9	48	7	36	3	79
0	76	1	69	3	76	5	51	2	85	9	11	6	40	7	89	4	26	8	74
1	85	0	13	2	61	6	7	8	64	9	76	5	47	3	52	4	90	7	45

Tabela Anexo A .4 - Definição de MT10

Melhor solução conhecida: 930;

Problemas Realistas Utilizados

Os problemas realistas que foram utilizados para testes nesta dissertação, conforme já foi dito, são variações daqueles apresentados em [2]. Estão listadas, nesta ordem, as relações de precedência entre as partes que compõe cada tarefa, as máquinas e recursos adicionais que podem ser requisitados por cada tarefa, assim como as paradas para manutenção preventiva, a definição de cada operação (incluindo tempo de execução, subprocesso, rota, máquina e recursos adicionais necessários, e assim por diante), e, por fim, os tempos de preparo de máquinas dependentes da seqüência de operações escalonadas, organizados por número de operação.

PROBLEMA 1

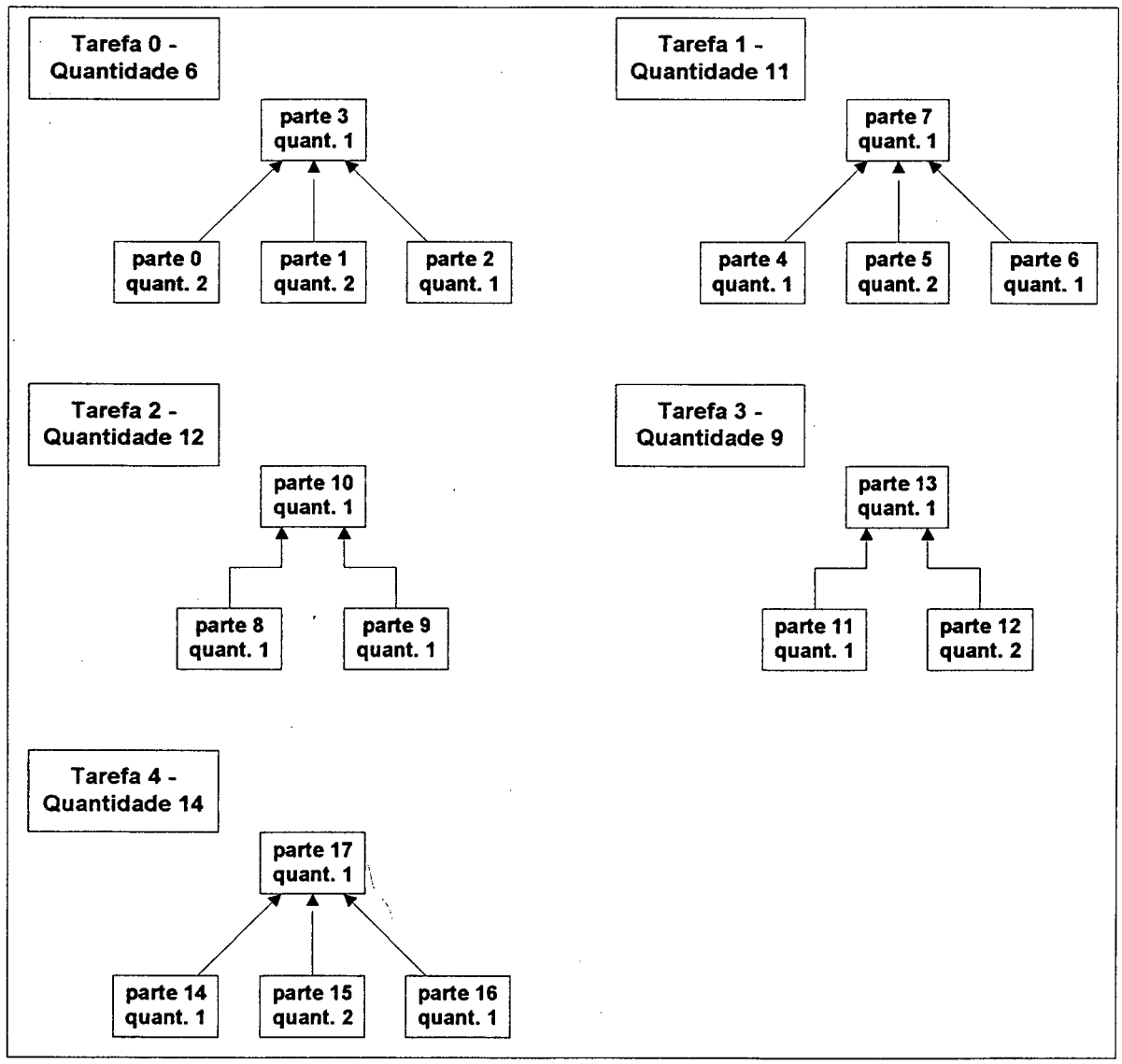

Figura Anexo A .1 - Relações de Precedência entre Partes - Problema 1

	Tipo	Unidade	Início Parada 1	Duração	Início Parada 2	Duração
Máquinas	0	1	335	56	1781	22
	1	1	506	54	1630	36
	2	1	1053	1		
	3	1	905	1		
	3	2	612	14		
	4	1	528	1	1499	18
	5	1	291	36	1717	6
	5	2	936	2		
	5	3	223	30	1663	36
	5	4	1181	78		
Outros Recursos	6	1	673	30		
	6	2	1269	14		
	6	3	113	36	1539	66
	6	4	311	34	1545	48
	7	1	480	2	1602	1
	7	2	426	50	1616	26
	7	3	767	16		
	7	4	1238	46		
	8	1	955	32		
	8	2	403	34	1597	14
	9	1	281	58	1359	32
	9	2	842	40		
	10	1	714	30		
	10	2	1156	48		
	10	3	866	40		
	10	4	771	36		
	11	1	124	1	1185	50
	11	2	905	54		
	11	3	127	2	1379	28
	11	4	1041	20		
	12	1	244	16	1380	36
	12	2	378	24	1682	26
	12	3	59	48	1237	48
	12	4	149	1	1400	50
	13	1	527	1	1713	24
	13	2	510	26	1686	18
	13	3	964	14		
	13	4	181	6	1557	46

Tabela Anexo A .5 - Máquinas e Recursos Adicionais - Problema 1

Operação	Tarefa	Parte	Subprocesso	Rota	Setup	Execução Unitária	Horário Liberação	Máquina	Rec. Adcnl.	Quant.	Rec. Adcnl.	Quant.	Rec. Adcnl.	Quant.
0	0	0	0	0	6	2	0	2	8	1				
1	0	0	0	0	6	3	0	3	6	1				
2	0	0	0	0	7	1	0	2	6	1	7	1	13	1
3	0	0	1	1	3	2	0	3	8	2				
4	0	0	1	1	3	4	0	5						
5	0	1	2	2	3	1	0	5	10	1				
6	0	1	2	2	10	1	0	0						
7	0	1	3	3	4	3	0	1	9	2				
8	0	1	3	4	6	3	0	4						
9	0	1	3	4	4	1	0	5						
10	0	1	3	4	6	1	0	2	10	1	12	2		
11	0	2	4	5	6	2	0	4	13	1				
12	0	2	4	5	6	3	0	5	7	2	10	1	13	1
13	0	2	4	6	3	3	0	5	7	1	10	1		
14	0	2	4	6	5	1	0	1						
15	0	3	5	7	4	2	0	0	8	1				
16	0	3	5	7	8	1	0	3	7	1	8	1	10	1
17	0	3	5	8	5	5	0	0	8	1	10	1		
18	0	3	5	8	6	1	0	4						
19	0	3	6	9	5	4	0	5	7	1	10	1		
20	0	3	6	10	6	3	0	2	7	1				
21	0	3	6	10	6	4	0	3	6	2	11	1	12	1
22	1	4	7	11	7	4	0	3						
23	1	4	7	11	5	2	0	5						
24	1	4	7	12	4	4	0	1	6	1	7	1	11	1
25	1	4	7	12	3	5	0	4	7	1				
26	1	4	8	13	9	4	98	2	7	1	10	1		
27	1	4	8	14	6	1	0	3	10	1	13	1		
28	1	4	9	15	4	1	0	4	13	1				
29	1	4	9	16	6	4	0	1						
30	1	4	9	16	4	2	0	3	6	1				
31	1	4	10	17	5	5	0	2	8	1	10	1		
32	1	4	10	17	4	1	0	3						
33	1	4	10	17	4	5	0	0	10	1				
34	1	5	11	18	2	2	0	0	6	2	9	1		
35	1	5	12	19	8	1	0	4	11	1	12	1		
36	1	5	12	19	3	4	0	3	6	2	7	1		
37	1	5	12	20	4	5	0	5	6	1				
38	1	5	13	21	2	5	0	5	11	1				
39	1	5	13	21	3	4	0	2						
40	1	6	14	22	8	5	0	5	6	1	10	1		
41	1	6	14	22	4	2	0	0						
42	1	6	14	22	4	1	0	3	7	1	11	1		
43	1	6	15	23	3	3	0	2						
44	1	6	15	23	8	3	0	4						

45	1	6	15	23	6	3	0	2						
46	1	6	15	24	1	3	0	5						
47	1	6	15	24	8	5	0	4	7	1	11	1		
48	1	6	15	24	5	1	0	1						
49	1	6	16	25	2	1	0	4	12	2	13	1		
50	1	7	17	26	2	4	0	1	6	1				
51	1	7	18	27	7	4	0	4	13	1				
52	1	7	18	28	8	6	0	4						
53	1	7	18	28	5	1	0	3	7	1	10	1		
54	1	7	19	29	1	5	0	2	8	1	13	1		
55	1	7	19	29	5	3	0	5	7	1	10	1		
56	1	7	19	29	3	5	0	2						
57	2	8	20	30	8	2	0	5	7	1	10	1		
58	2	8	20	30	4	3	0	2	6	1				
59	2	8	20	31	6	1	0	2	7	1	10	1		
60	2	8	20	31	6	3	0	5	7	1	10	1		
61	2	8	21	32	6	5	0	3	10	1				
62	2	8	21	32	7	5	0	4						
63	2	8	21	33	5	2	0	4	7	1				
64	2	9	22	34	8	2	0	2	6	1	8	2		
65	2	9	22	35	5	4	0	3	11	1				
66	2	9	22	35	6	5	0	2	7	1				
67	2	9	22	35	6	3	0	3	11	1	12	1		
68	2	9	23	36	5	4	0	0	13	1				
69	2	9	23	36	3	5	0	4						
70	2	9	24	37	6	1	0	2	10	1	13	2		
71	2	10	25	38	6	2	0	5	6	1				
72	2	10	25	38	7	2	0	2	9	1	11	1		
73	2	10	26	39	5	3	0	3	7	1				
74	2	10	26	39	7	2	0	5						
75	2	10	27	40	7	3	0	3	10	1	13	1		
76	2	10	27	40	7	2	0	0	12	1				
77	2	10	27	41	3	4	0	3	7	1	10	1		
78	2	10	27	41	5	1	0	2	7	1				
79	3	11	28	42	8	2	0	4	13	1				
80	3	11	28	42	6	5	0	0						
81	3	11	28	42	1	2	0	3	10	1				
82	3	11	28	43	5	2	0	4	10	1				
83	3	11	28	43	8	3	0	0	6	1				
84	3	11	29	44	6	1	0	1	13	1				
85	3	12	30	45	1	3	0	5	6	1				
86	3	12	31	46	6	2	0	3						
87	3	12	31	46	5	1	0	2	6	1				
88	3	12	31	47	3	1	0	2	6	1	12	1		
89	3	12	32	48	8	5	0	3	6	2	7	1		
90	3	12	32	48	9	2	0	5						
91	3	12	32	49	4	1	0	1	6	1				
92	3	12	32	49	7	2	0	4	10	1	12	1		
93	3	13	33	50	2	1	0	3						
94	3	13	33	50	10	2	0	2	10	1				

95	3	13	33	50	4	6	0	5	10	1	12	1		
96	3	13	34	51	9	4	0	4	8	2				
97	3	13	34	51	6	4	0	3	6	1				
98	3	13	35	52	5	2	0	1	7	1	8	1	9	2
99	3	13	35	52	4	1	0	4	13	1				
100	3	13	36	53	7	4	0	1						
101	3	13	36	53	3	4	0	5	7	2				
102	4	14	37	54	3	3	0	5	6	2	7	1	8	1
103	4	14	38	55	2	6	0	3						
104	4	14	38	55	5	5	0	5						
105	4	14	38	56	8	2	0	3	11	1				
106	4	14	38	56	3	4	0	5						
107	4	14	39	57	6	2	0	2	8	1	11	1		
108	4	14	40	58	3	3	0	5	7	1	11	1	13	1
109	4	14	40	58	3	5	0	1						
110	4	14	41	59	5	1	0	0						
111	4	14	41	59	5	1	0	4						
112	4	14	41	59	6	2	0	1	11	1				
113	4	14	41	60	3	5	0	0						
114	4	14	41	60	6	1	0	2	7	1				
115	4	14	41	60	3	1	0	5						
116	4	15	42	61	4	3	0	1	11	1				
117	4	15	42	61	4	1	0	5	11	1				
118	4	15	42	62	6	2	0	4						
119	4	15	43	63	8	2	0	3	6	1				
120	4	15	43	63	5	1	0	1	6	2	13	1		
121	4	15	43	64	5	1	0	3	11	1				
122	4	15	44	65	1	1	0	5	6	1	7	1	13	1
123	4	16	45	66	3	3	0	2	12	2				
124	4	16	46	67	6	2	0	0						
125	4	16	46	67	4	1	0	3	10	2				
126	4	16	46	67	7	1	0	1	11	1				
127	4	17	47	68	9	1	0	2	8	1	10	1	11	1
128	4	17	47	68	4	6	0	5	7	1	8	1	13	1
129	4	17	48	69	7	3	0	2						
130	4	17	48	69	5	5	0	5						
131	4	17	48	69	3	2	0	2	10	1	11	1	12	1
132	4	17	49	70	5	2	0	4						
133	4	17	49	70	7	4	0	2	6	1	10	1		
134	4	17	50	71	5	3	0	3	11	1				
135	4	17	50	71	3	1	0	4	7	2	10	1	12	1

Tabela Anexo A .6 - Definição das Operações - Problema 1

Operação	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup
6	15	7	17	5	33	5	41	5	68	6	76	8	80	6	110	5	124	9				
14	24	4	48	1	50	3	91	3	98	6	100	6	109	4	112	4	116	4				
15	6	9	17	3	68	5	76	7	110	5	124	3										
17	6	7	15	10	33	4	41	4	68	3	76	6	80	6	124	3						
24	14	7	29	5	48	7	50	7	91	0	98	3	112	7	116	7						
29	14	4	48	5	50	7	98	5	100	4	109	5	112	5	116	9	120	3				
33	6	6	17	5	41	5	68	6	76	6	80	3	110	7	124	2						
41	6	7	17	2	33	6	68	8	76	5	80	2	110	7	124	6						
48	14	2	24	4	50	4	91	2	98	6	109	4	112	8	116	4	120	1				
50	24	4	29	7	48	4	98	5	100	6	109	5	116	5	120	4						
68	6	6	15	6	17	0	33	7	41	3	76	3	80	3	110	7	124	8				
76	6	4	15	3	17	3	33	4	41	3	68	5	80	5	110	2	124	3				
80	6	5	15	3	17	5	41	8	68	7	110	3										
91	14	7	24	2	29	7	50	3	98	6	116	6	120	6								
98	14	4	24	4	29	4	48	6	50	2	91	3	100	3	109	8	112	7	116	7		
100	14	6	24	2	29	6	48	3	50	4	91	5	109	5	116	4	120	5				
109	14	5	24	5	48	6	50	7	91	5	98	0	110	10	112	5	116	2	120	5		
110	6	8	17	3	33	6	41	4	68	6	76	3	80	4	124	8						
112	14	4	24	2	29	4	4	8	50	3	109	3	116	8	120	2						
116	24	6	29	3	48	10	50	5	98	2	100	3	109	7	112	3	120	2				
120	14	6	24	7	29	2	48	8	50	7	91	9	98	6	100	4	109	6	112	4	116	7
124	17	3	41	5	68	4	80	4	110	9												

Tabela Anexo A .7 - Definição dos Tempos de Preparo Dependentes da Seqüência - Problema 1

PROBLEMA 2

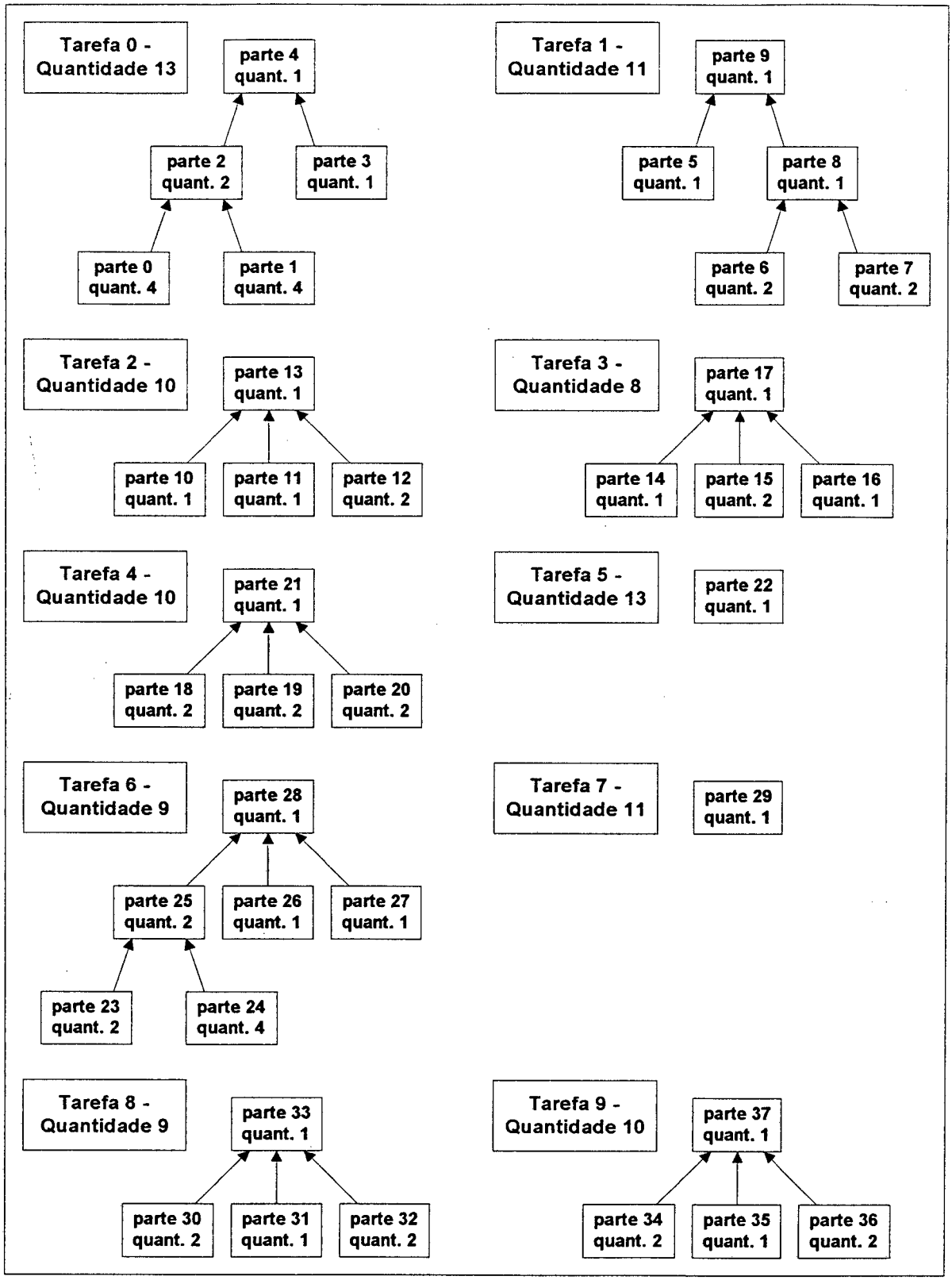

Figura Anexo A .2 - Relações de Precedência entre Partes - Problema 2

	Tipo	Unidade	Início Parada 1	Duração	Início Parada 2	Duração
Máquinas	0	0	1667	27		
	1	0	1235	30	4235	31
	2	0	2499	17		
	3	0	133	25	3071	127
	3	1	1823	32		
	4	0	832	37	3779	29
	4	1	2981	34		
	5	0	1032	14	4316	30
	6	0	2074	27		
	6	1	1310	42	4562	20
	6	2	2590	40		
	7	0	1680	41		
	7	1	2881	10		
	7	2	56	31	3147	29
	8	0	2278	19		
	8	1	887	43	4080	13
	8	2	2970	38		
	9	0	2331	43		
	9	1	1654	24		
	9	2	1174	32	3996	22
	10	0	2399	45		
	10	1	1082	24	3806	39
Outros Recursos	11	0	179	30	3629	20
	11	1	2079	27		
	11	2	297	40	2827	25
	11	3	2266	23		
	12	0	928	27	3775	26
	12	1	2576	28		
	12	2	932	39	3371	37
	12	3	1363	45	4828	38
	13	0	589	22	3971	1
	13	1	2764	23		
	13	2	106	18	3124	18
	13	3	27	42	3639	29
	14	0	27	32	3209	35
	14	1	1779	31		
	15	0	857	14	4591	13
	15	1	2622	37		
	16	0	2479	24		
	16	1	1987	36		
	16	2	668	40	4278	24
	16	3	1426	29		
	17	0	2889	51		
	17	1	186	19	3355	20
	17	2	2245	29		
	17	3	551	28	3729	35
	18	0	2772	28		

	18	1	1294	33	4627	1
	19	0	886	13	4199	19
	19	1	278	51	3599	22
	19	2	2587	43		
	19	3	759	7	3736	41
	20	0	642	18	4020	37
	20	1	673	34	3707	22
	20	2	1134	19	4003	39
	20	3	686	17	3763	25
	21	0	2348	34		
	21	1	1166	21	4127	21
	21	2	867	26	4103	26
	21	3	1048	25	3953	35
	22	0	1908	27		
	22	1	1815	12		
	22	2	2309	27		
	22	3	2035	29		
	23	0	2234	39		
	23	1	0	34	3574	12
	23	2	2403	49		
	23	3	349	1	3230	48
	24	0	781	28	3209	25
	24	1	2346	44		
	25	0	364	13	3707	27
	25	1	204	30	2694	13
	25	2	2203	38		
	25	3	2496	36		
	26	0	2260	27		
	26	1	2939	23		
	27	0	441	48	3609	29
	27	1	2228	16		
	27	2	2399	19		
	27	3	168	39	3297	35
	28	0	1953	7		
	28	1	2013	29		
	29	0	1705	39		
	29	1	1565	40		
	30	0	2184	21		
	30	1	941	29	4330	39

Tabela Anexo A .8 - Máquinas e Recursos Adicionais - Problema 2

Operação	Tarefa	Parte	Subpocesso	Rota	Setup	Execução Unitária	Horário Liberação	Máquina	Rec. Adcnl.	Quant.	Rec. Adcnl.	Quant.	Rec. Adcnl.	Quant.
0	0	0	0	0	7	1	0	6						
1	0	0	0	0	7	4	0	9						
2	0	0	0	0	3	4	0	7	19	1	22	1		
3	0	0	0	0	1	4	0	8						
4	0	0	0	0	4	1	0	10	20	1	21	1		
5	0	0	0	0	4	3	0	8						
6	0	0	1	1	7	2	0	10	17	1	22	1		
7	0	0	1	1	7	1	0	2						
8	0	0	1	2	6	1	0	1	16	1				
9	0	0	1	2	5	3	0	4						
10	0	0	1	2	9	1	0	5	25	1				
11	0	0	2	3	4	1	0	4	22	1				
12	0	0	2	3	6	1	0	5						
13	0	0	2	3	6	2	0	9	16	1				
14	0	1	3	4	5	1	0	10	11	1	18	1	24	1
15	0	1	3	4	7	3	0	6	17	1				
16	0	1	4	5	3	3	0	8	19	1				
17	0	1	4	5	7	3	0	5	17	1	25	1		
18	0	1	4	5	6	3	0	3						
19	0	1	4	5	8	5	0	6	27	1				
20	0	1	4	5	4	2	0	8						
21	0	1	4	6	4	3	0	0						
22	0	1	4	6	5	1	0	10	22	1				
23	0	1	5	7	5	1	0	7	13	1	21	1		
24	0	1	5	7	3	1	0	10						
25	0	1	5	8	7	1	0	6	28	1				
26	0	1	5	8	3	1	0	7	11	1	25	1		
27	0	2	6	9	7	4	0	6						
28	0	2	6	9	3	4	0	4	13	1	16	1	23	1
29	0	2	6	9	5	3	0	8	13	2				
30	0	2	6	9	7	1	0	6	11	1				
31	0	2	7	10	6	4	0	9	20	1	25	1		
32	0	2	7	10	3	1	0	7	18	1	27	1		
33	0	2	7	10	7	5	0	10	25	1				
34	0	2	7	10	5	2	0	9						
35	0	2	7	10	8	1	0	7						
36	0	2	8	11	4	5	0	10						
37	0	2	8	11	6	1	0	4						
38	0	2	8	11	8	1	0	5	23	1				
39	0	2	8	11	4	1	0	9	21	1				
40	0	2	8	11	3	2	0	6	16	1				
41	0	2	8	11	4	4	0	3						
42	0	2	9	12	4	4	0	7	24	1				
43	0	2	9	12	4	2	0	2						
44	0	2	9	12	4	3	0	0	16	2	18	1		

45	0	2	10	13	3	4	0	4						
46	0	2	10	13	4	2	0	9	13	1	25	1		
47	0	2	10	14	0	2	0	9	17	1	20	1		
48	0	2	10	14	3	4	0	2	20	1	22	1		
49	0	2	10	14	5	2	0	1						
50	0	2	10	14	5	3	0	3						
51	0	3	11	15	5	2	0	3						
52	0	3	11	15	7	4	13	10	19	1				
53	0	3	11	16	3	1	0	7						
54	0	3	12	17	3	1	0	5	17	1				
55	0	3	12	17	7	2	0	7	30	1				
56	0	3	12	17	2	1	0	4	19	1	29	1		
57	0	3	13	18	2	3	0	1	16	2	20	1		
58	0	3	13	18	1	1	0	3						
59	0	3	13	18	8	1	0	4	16	1	22	1		
60	0	3	13	18	5	1	0	6	27	1				
61	0	3	13	18	6	2	0	9	22	1				
62	0	3	14	19	3	1	0	0						
63	0	3	14	19	6	3	0	3						
64	0	3	14	19	3	4	0	6	23	1				
65	0	3	14	19	0	1	0	0	16	1	28	1		
66	0	3	14	19	2	3	0	4						
67	0	3	14	20	6	2	0	8	21	1	29	1		
68	0	3	15	21	7	3	0	10						
69	0	3	15	21	7	6	0	4	16	1				
70	0	3	15	21	5	3	0	5	16	2	25	1		
71	0	3	15	22	5	3	0	1	16	1				
72	0	3	15	22	3	2	0	10						
73	0	3	16	23	5	2	0	7	16	1				
74	0	4	16	23	6	4	0	10	20	1	21	1	22	1
75	0	4	16	23	3	5	0	1	20	1	22	1	27	1
76	0	4	17	24	5	3	0	7	16	1	25	1		
77	0	4	17	24	2	3	0	8	17	1	22	1	27	1
78	0	4	17	24	8	2	0	2	16	1	19	1	20	1
79	0	4	17	24	6	1	0	3						
80	0	4	17	25	6	6	0	8						
81	0	4	17	25	6	4	0	9	16	1	19	1		
82	0	4	17	25	5	5	0	10						
83	0	4	17	25	3	4	0	9	20	1	25	1		
84	0	4	17	25	5	4	0	10	17	1				
85	0	4	18	26	7	4	0	6	16	1				
86	0	4	18	26	7	1	0	10	21	1				
87	0	4	18	26	7	1	0	5	11	1	13	1		
88	0	4	18	26	4	1	0	0	20	2				
89	0	4	18	26	7	3	0	5						
90	0	4	19	27	4	2	0	6	22	1	29	1		
91	0	4	19	27	4	3	0	3	15	1	21	1		
92	0	4	19	27	5	3	0	10	16	1	17	1	19	1
93	0	4	19	28	2	1	0	10	16	1				
94	0	4	19	28	4	4	0	8						

95	0	4	20	29	5	5	0	7	22	1				
96	0	4	20	29	2	2	0	5	12	1				
97	0	4	20	29	3	4	0	10	12	1	15	1		
98	0	4	20	30	4	3	0	7	17	1	21	1		
99	0	4	20	30	4	3	0	2						
100	0	4	20	30	8	4	0	10	16	1				
101	0	4	20	30	6	5	0	5	25	1				
102	0	4	20	30	4	2	0	3						
103	0	4	20	30	6	4	0	10	20	1				
104	1	5	21	31	2	2	0	7	22	1				
105	1	5	21	31	3	1	0	3						
106	1	5	21	31	7	2	0	6	16	1	21	1	22	1
107	1	5	21	31	2	4	0	4						
108	1	5	21	31	7	5	0	5	12	1	22	1		
109	1	5	21	32	4	4	0	7	13	1				
110	1	5	21	32	8	3	0	10	16	1				
111	1	5	21	32	6	4	0	4	23	2				
112	1	5	21	32	7	3	0	7						
113	1	5	21	32	3	1	0	4						
114	1	5	22	33	6	4	0	8	12	1	16	1	27	1
115	1	5	22	33	3	4	0	4						
116	1	5	22	33	0	2	0	5	16	1				
117	1	5	22	33	3	3	0	10	16	2				
118	1	6	23	34	6	4	0	10	17	1	22	1	25	1
119	1	6	23	34	6	2	0	6	21	1	23	1		
120	1	6	23	35	5	3	0	5						
121	1	6	23	35	1	3	0	10	22	2				
122	1	6	23	35	7	2	0	3						
123	1	6	23	35	6	2	0	8	16	2	17	1	22	1
124	1	6	24	36	7	3	0	10	22	1				
125	1	6	24	36	5	4	0	7	16	1	25	1		
126	1	6	24	36	0	3	0	10	22	1	23	1		
127	1	6	24	36	3	3	0	8						
128	1	6	25	37	4	1	0	9	26	1				
129	1	6	25	37	4	1	0	3						
130	1	6	25	37	7	3	0	10	12	1	22	1		
131	1	6	25	37	4	3	0	8						
132	1	6	25	37	10	1	0	7	25	1				
133	1	6	26	38	4	1	0	9	21	1				
134	1	7	27	39	6	2	0	4	12	1	13	1		
135	1	7	28	40	5	4	0	7						
136	1	7	28	40	7	1	0	10						
137	1	7	28	40	3	2	0	3						
138	1	7	28	40	5	6	0	9						
139	1	7	28	40	3	4	0	8	26	1				
140	1	7	28	40	9	1	0	2						
141	1	7	28	41	6	2	0	9	22	1				
142	1	7	28	41	6	3	0	3	21	2				
143	1	7	28	41	5	3	0	8	16	1				
144	1	8	29	42	1	3	0	7	21	1				

145	1	8	29	42	5	3	0	10	12	1	26	1		
146	1	8	30	43	4	1	0	8						
147	1	8	30	43	5	4	0	7	20	2	23	1	25	1
148	1	8	30	43	4	2	0	10	11	1				
149	1	8	30	43	6	4	0	8						
150	1	8	30	43	5	1	0	7	21	1				
151	1	8	30	44	6	5	0	6	22	1				
152	1	8	30	44	6	1	0	0						
153	1	8	30	44	7	3	0	7	20	1	22	1		
154	1	8	31	45	6	2	0	5	16	1				
155	1	8	31	45	4	3	0	8						
156	1	8	31	45	4	3	0	4	20	1				
157	1	8	32	46	8	1	0	9						
158	1	8	32	46	3	4	0	10						
159	1	8	32	46	6	4	0	4	25	2				
160	1	8	32	46	9	1	0	6						
161	1	8	32	46	6	4	0	10	11	1	20	1		
162	1	8	32	46	6	3	0	1						
163	1	9	33	47	2	1	0	0						
164	1	9	33	47	4	1	0	7						
165	1	9	33	47	7	4	0	4	16	1				
166	1	9	33	47	6	3	0	8						
167	1	9	33	47	6	3	0	1						
168	1	9	34	48	1	2	0	6	13	2				
169	1	9	34	48	7	2	0	5	11	1	22	2		
170	1	9	34	48	6	3	0	10	20	1	29	1		
171	1	9	34	48	6	3	0	9	12	1				
172	1	9	35	49	6	4	0	0	16	2	20	1		
173	1	9	35	49	7	2	0	8						
174	1	9	35	49	5	2	0	9	17	1				
175	1	9	35	49	4	2	0	4	21	1	25	1		
176	1	9	35	49	9	2	0	3						
177	2	10	36	50	4	3	0	4						
178	2	10	36	50	5	1	0	9						
179	2	10	37	51	8	2	0	5	26	1				
180	2	10	37	51	0	2	0	8						
181	2	10	37	51	9	4	0	10	11	1	21	1		
182	2	10	38	52	5	1	0	3						
183	2	10	38	52	4	2	0	2	25	1				
184	2	10	38	52	8	4	0	6	16	1	19	1	21	1
185	2	10	38	52	4	2	0	3						
186	2	10	38	52	4	3	36	7	15	2				
187	2	10	39	53	4	3	0	8						
188	2	10	39	53	5	3	0	3						
189	2	10	39	53	5	5	0	7	19	1				
190	2	10	39	53	6	4	0	5						
191	2	11	40	54	7	4	0	10	13	1				
192	2	11	40	54	4	1	0	2	22	1				
193	2	11	40	54	8	2	0	0	19	1				
194	2	11	40	54	5	3	0	9	12	1	25	1		

195	2	11	41	55	6	4	0	3						
196	2	11	41	55	7	4	0	9	21	1	25	1		
197	2	11	41	55	7	1	0	7	20	1				
198	2	11	41	55	5	2	0	8	11	1	16	1	24	1
199	2	11	41	55	7	3	0	0	20	1				
200	2	12	42	56	6	1	0	5						
201	2	12	42	56	3	1	0	2						
202	2	12	42	56	7	2	0	5						
203	2	12	42	56	2	3	0	9						
204	2	12	42	56	3	1	0	6						
205	2	12	43	57	3	3	0	9	22	2				
206	2	12	43	57	4	2	0	5						
207	2	12	43	57	7	4	0	10						
208	2	12	43	57	7	5	0	8						
209	2	12	43	57	4	1	0	9						
210	2	13	44	58	0	1	0	7	20	1				
211	2	13	44	58	6	4	0	5						
212	2	13	44	58	6	3	0	2						
213	2	13	44	58	4	1	0	8						
214	2	13	45	59	5	5	0	10	25	1				
215	2	13	45	59	1	1	0	8	20	2				
216	2	13	45	59	3	3	0	2	20	1	23	1		
217	2	13	45	59	4	3	0	9	11	1				
218	3	14	46	60	7	1	0	8						
219	3	14	46	60	6	4	0	5						
220	3	14	46	60	7	1	0	9						
221	3	14	46	60	7	2	0	1						
222	3	14	46	60	7	1	0	2						
223	3	14	46	61	9	3	0	3						
224	3	14	47	62	3	1	0	8	22	2	25	1		
225	3	14	47	62	3	1	0	4	11	1				
226	3	14	47	62	3	4	0	9	17	1	20	1		
227	3	14	47	62	6	2	0	1	20	1				
228	3	14	48	63	8	2	0	6						
229	3	14	48	63	6	2	0	5						
230	3	14	48	63	3	3	0	10	11	1	25	2		
231	3	14	48	63	4	3	0	3	27	1				
232	3	14	48	63	2	1	0	9						
233	3	14	49	64	6	1	0	1	27	1	29	1		
234	3	14	49	64	6	6	0	7						
235	3	14	49	64	7	3	0	5	21	2				
236	3	14	49	64	6	3	0	10						
237	3	14	49	64	4	3	0	6	29	1				
238	3	14	50	65	7	2	0	4						
239	3	14	50	65	5	3	0	3						
240	3	14	50	65	3	3	0	9						
241	3	15	51	66	4	4	0	5						
242	3	15	51	66	5	3	0	3	16	1				
243	3	15	51	66	4	1	0	5	14	1				
244	3	15	51	66	6	2	0	8						

245	3	16	52	67	3	4	0	5	17	1	20	1	25	2
246	3	16	52	67	4	1	0	7	29	2				
247	3	16	52	67	6	1	0	2						
248	3	16	52	67	2	2	0	8	12	1	25	1		
249	3	16	53	68	3	3	0	3	25	1				
250	3	16	53	68	7	4	0	5	16	1				
251	3	16	53	68	3	6	0	7	19	1				
252	3	16	53	68	3	3	0	8	20	2	25	1		
253	3	16	53	68	2	3	0	10						
254	3	17	54	69	1	2	0	7	22	1				
255	3	17	55	70	5	1	0	3	16	1	23	1		
256	3	17	55	70	5	4	0	8	22	1				
257	3	17	56	71	6	2	0	4	20	1	25	1		
258	3	17	56	71	3	4	0	3	19	1	24	1		
259	3	17	56	71	5	1	0	10	11	1	17	1	22	1
260	3	17	56	71	7	3	0	3	25	1				
261	3	17	56	71	5	3	0	1	16	1				
262	3	17	56	71	8	2	0	8	19	1	20	1		
263	3	17	56	72	5	2	0	5	20	1				
264	3	17	57	73	8	1	0	6						
265	3	17	57	73	5	5	0	2						
266	3	17	57	74	4	3	0	6	22	1				
267	3	17	57	74	1	4	0	8						
268	3	17	57	74	3	1	0	2						
269	3	17	57	74	6	2	0	5						
270	4	18	58	75	4	2	0	5	12	1				
271	4	18	58	75	3	3	0	10	22	1	25	2		
272	4	18	59	76	4	2	0	5						
273	4	18	59	76	7	1	0	1	29	1				
274	4	18	59	76	5	2	0	9						
275	4	18	59	76	6	1	0	10	16	1	21	1		
276	4	18	59	76	4	2	0	7						
277	4	18	59	76	4	2	0	10						
278	4	19	60	77	9	1	0	9	20	1	25	1		
279	4	19	60	77	7	2	0	6	13	1	20	1	21	1
280	4	19	60	77	2	1	0	5						
281	4	19	60	77	3	3	0	10	13	1	14	1	27	1
282	4	19	60	77	8	6	0	9						
283	4	19	60	77	4	2	0	6	27	1	29	1		
284	4	19	61	78	8	4	0	5	26	1				
285	4	19	61	78	6	1	0	4						
286	4	19	61	78	0	1	0	3	25	1				
287	4	20	62	79	6	1	0	4	22	1				
288	4	20	62	79	6	1	0	9						
289	4	20	63	80	7	3	0	6	13	1	19	1	25	1
290	4	20	63	80	3	4	0	8	16	1				
291	4	20	63	80	1	4	0	5	17	2	22	1		
292	4	20	64	81	6	2	0	7	22	1				
293	4	20	65	82	6	4	0	3						
294	4	20	65	82	6	5	0	7	19	1				

295	4	21	66	83	3	3	0	6	21	1				
296	4	21	66	83	5	3	0	9						
297	4	21	66	83	5	1	0	0	27	1				
298	4	21	66	83	7	1	0	9						
299	4	21	66	83	6	2	0	6	18	1	25	1		
300	4	21	66	84	6	1	0	1						
301	4	21	66	84	8	3	0	0	16	1				
302	4	21	66	84	7	6	0	2	11	1	25	1		
303	4	21	66	84	5	3	0	1						
304	4	21	66	84	6	3	0	0	13	2				
305	4	21	66	84	4	4	0	10	22	1				
306	4	21	67	85	4	4	0	5	21	1				
307	4	21	67	85	7	6	0	8						
308	4	21	67	85	5	2	0	10	17	1	19	1	22	1
309	4	21	67	85	10	4	0	7	22	1				
310	4	21	67	85	5	2	0	10	26	1				
311	4	21	67	85	6	1	0	8						
312	4	21	67	86	2	1	0	8						
313	5	22	68	87	5	5	0	9	17	1	27	1		
314	5	22	68	87	9	4	0	6	12	1				
315	5	22	68	87	2	3	0	5						
316	5	22	68	87	8	1	0	9	17	1				
317	5	22	69	88	4	4	0	0	16	1	20	1	27	2
318	5	22	69	88	7	4	0	4						
319	5	22	69	88	6	3	0	10	20	1	25	1		
320	5	22	69	88	1	1	0	4						
321	5	22	70	89	7	3	0	8						
322	5	22	70	89	6	2	0	7						
323	5	22	70	89	5	3	0	3	25	1				
324	5	22	70	90	2	4	0	7	22	1				
325	5	22	70	90	5	3	0	10						
326	5	22	70	90	5	3	0	8	21	1				
327	5	22	70	90	9	3	0	4	17	1				
328	5	22	70	90	6	4	0	3	16	1	22	1		
329	5	22	71	91	6	1	0	10	20	1	22	1		
330	5	22	72	92	0	3	0	4	17	1				
331	5	22	72	92	6	4	0	9						
332	5	22	72	92	5	1	0	10	16	1	20	1	25	1
333	5	22	72	92	3	3	0	2	16	1	20	2		
334	5	22	72	92	9	2	0	4						
335	5	22	72	92	4	1	0	9						
336	6	23	73	93	8	1	0	3						
337	6	23	73	93	6	2	0	5						
338	6	23	73	93	4	4	0	3	25	1				
339	6	24	74	94	6	4	0	10	14	1	17	1	20	1
340	6	24	74	94	6	3	0	6						
341	6	24	74	94	4	6	0	3						
342	6	24	74	94	4	2	0	8	17	1				
343	6	24	74	94	6	5	0	4	21	1				
344	6	25	75	95	6	1	0	1	29	1				

345	6	25	75	96	4	3	0	0	16	1				
346	6	25	75	96	5	4	0	7						
347	6	25	75	96	8	1	0	5	11	1	13	1		
348	6	25	75	96	7	5	0	3	20	1				
349	6	25	76	97	4	2	0	4	27	1				
350	6	25	76	97	4	4	0	3						
351	6	25	77	98	6	5	0	6						
352	6	25	77	98	5	3	0	7						
353	6	25	77	98	6	1	0	2	22	1				
354	6	26	78	99	3	3	0	8	23	1				
355	6	27	79	100	3	1	0	4	11	1	19	1	27	1
356	6	27	79	100	6	2	0	1	27	1				
357	6	27	79	100	6	3	0	7	16	1				
358	6	27	79	100	7	2	0	8	16	1				
359	6	28	80	101	3	1	0	1						
360	6	28	80	101	7	3	0	8						
361	6	28	80	101	8	3	0	4						
362	6	28	80	101	1	4	0	0	20	2	22	1		
363	6	28	80	101	1	3	0	5						
364	7	29	81	102	7	3	0	8	16	1	27	1		
365	7	29	81	102	8	3	0	7						
366	7	29	81	102	7	3	0	8	21	1	25	1		
367	7	29	81	102	8	2	0	6	22	1				
368	7	29	82	103	4	2	0	10						
369	7	29	82	103	6	4	0	9	12	1	22	1		
370	7	29	82	103	6	1	0	3						
371	7	29	82	103	6	2	0	8						
372	7	29	83	104	1	3	0	9	16	1	22	1		
373	7	29	83	104	2	1	0	4	20	1				
374	7	29	83	104	6	1	0	5	15	1	17	1		
375	7	29	83	104	5	2	0	8	11	1	22	2		
376	8	30	84	105	6	3	0	6	19	1	21	1	28	1
377	8	30	84	105	7	1	0	4	21	1	29	1		
378	8	30	84	105	3	1	0	5	20	1	22	1		
379	8	30	84	106	5	3	0	1	13	1	17	1	20	1
380	8	30	84	106	4	3	0	7						
381	8	30	84	106	5	4	0	5	20	1				
382	8	30	84	106	3	3	0	9	16	1				
383	8	30	84	106	10	2	0	10						
384	8	30	84	106	9	3	0	6	13	1	16	1	21	1
385	8	30	85	107	4	3	0	7						
386	8	30	85	107	6	2	0	1						
387	8	30	85	107	6	2	0	3						
388	8	30	85	107	2	4	0	0						
389	8	30	85	108	4	1	0	8						
390	8	30	85	108	5	2	0	3	20	1				
391	8	30	85	108	5	4	0	10						
392	8	30	86	109	8	4	0	4						
393	8	30	86	109	2	2	0	6	11	1	30	1		
394	8	30	86	109	2	4	0	4	16	1				

395	8	30	86	109	5	1	0	8	20	2				
396	8	30	86	109	4	3	0	3	22	2				
397	8	30	86	110	5	1	0	6						
398	8	30	86	110	4	5	0	9	16	1	18	1		
399	8	30	86	110	6	3	0	7	12	1	17	1	23	1
400	8	30	86	110	6	4	0	6	22	1				
401	8	30	87	111	4	3	0	4	20	1				
402	8	30	87	111	6	2	0	5	22	1				
403	8	30	87	111	3	5	0	8	22	1	25	1		
404	8	30	87	112	1	4	0	4	20	1				
405	8	30	87	112	4	1	0	8	21	1				
406	8	30	87	112	4	3	0	3	21	1				
407	8	30	87	112	5	1	0	1	17	1	21	1		
408	8	31	88	113	6	1	0	1						
409	8	31	88	113	4	1	0	7	23	1				
410	8	31	88	113	5	1	0	3	22	1				
411	8	31	89	114	5	4	0	7	13	1				
412	8	31	89	114	4	2	0	1						
413	8	31	89	114	5	3	0	4	25	1				
414	8	31	90	115	0	4	0	9						
415	8	31	90	115	4	2	0	6	22	1				
416	8	31	90	115	4	1	0	5	28	1				
417	8	31	90	115	6	5	0	0						
418	8	31	90	115	5	2	0	2	22	2				
419	8	31	90	115	4	2	0	10	11	2	16	1		
420	8	31	91	116	9	3	0	3	19	1				
421	8	31	91	116	8	3	0	7	19	1				
422	8	31	91	116	9	1	0	5						
423	8	31	91	116	4	3	0	4	16	1				
424	8	31	91	116	3	3	0	6	16	1	22	1	25	1
425	8	31	91	117	4	5	0	3						
426	8	31	91	117	9	4	0	9						
427	8	31	91	117	7	2	0	8	20	1	22	2	25	1
428	8	31	92	118	7	1	0	7						
429	8	31	92	118	6	1	0	5						
430	8	31	92	118	5	2	0	4	16	1				
431	8	31	92	118	7	4	0	8	29	1				
432	8	31	92	118	7	1	0	6						
433	8	31	92	119	6	4	0	10	17	1				
434	8	31	92	119	6	2	0	7						
435	8	31	92	119	8	4	0	10	11	1				
436	8	31	92	119	8	4	0	5	23	1				
437	8	32	93	120	6	1	0	7						
438	8	32	93	120	6	1	0	9	13	1	20	1		
439	8	32	93	120	7	3	0	6	22	1	25	1		
440	8	32	94	121	4	2	0	3	20	1	30	1		
441	8	32	94	121	7	1	0	9						
442	8	32	94	122	8	4	0	3	13	1	20	2	23	1
443	8	32	94	122	4	4	0	4						
444	8	32	95	123	6	4	0	3	16	1				

445	8	32	95	123	7	2	0	5						
446	8	32	95	123	6	3	0	2	25	1				
447	8	32	95	124	6	1	0	6	16	1	20	1	21	1
448	8	32	96	125	7	4	0	10						
449	8	32	96	125	4	3	0	7						
450	8	33	97	126	5	2	0	3						
451	8	33	97	126	6	3	0	8	13	1	18	1	22	1
452	8	33	98	127	5	4	0	7	16	1				
453	8	33	98	127	8	3	0	3						
454	8	33	98	127	4	1	0	4						
455	8	33	98	127	3	3	0	5	20	1	28	1		
456	8	33	99	128	1	3	0	10	12	1				
457	8	33	99	128	7	3	0	8	16	1	20	1		
458	8	33	99	128	6	3	0	5						
459	8	33	99	128	5	1	29	10	12	1	16	1		
460	8	33	99	128	8	1	0	8						
461	8	33	99	129	3	1	0	4	20	2	25	2		
462	8	33	99	129	7	3	0	9	13	1	19	1	20	1
463	8	33	99	129	5	4	0	3						
464	8	33	99	129	10	4	0	8	20	1	25	1		
465	8	33	99	129	7	4	0	7	13	1				
466	8	33	99	129	9	1	0	4	12	1	16	1		
467	8	33	100	130	4	5	0	6	22	2				
468	9	34	101	131	4	2	0	3						
469	9	35	102	132	7	3	0	7	20	1	22	1		
470	9	35	102	132	4	3	0	10						
471	9	35	102	132	0	4	0	3	16	1	21	1	25	1
472	9	35	103	133	8	2	0	7						
473	9	35	103	133	2	3	0	6	20	1	30	1		
474	9	35	103	133	5	5	0	3						
475	9	35	103	133	6	3	0	8						
476	9	35	103	133	8	3	0	0	25	1	28	1		
477	9	35	104	134	4	6	0	6						
478	9	35	104	134	9	4	0	9						
479	9	35	104	134	7	4	0	4	17	1	29	1		
480	9	35	104	134	4	3	0	1						
481	9	35	104	134	1	3	0	8						
482	9	35	105	135	6	5	0	5						
483	9	35	106	136	8	4	0	1	22	2				
484	9	35	106	136	5	4	0	6	19	1				
485	9	35	106	136	6	5	0	4	16	1				
486	9	36	107	137	4	4	0	10	20	1				
487	9	36	107	137	1	3	0	6						
488	9	36	107	137	3	3	0	3						
489	9	36	107	138	7	2	0	6						
490	9	36	108	139	4	1	0	9	22	1				
491	9	36	108	139	6	4	0	8						
492	9	36	108	139	8	3	0	10	21	1				
493	9	36	108	139	3	2	0	5	14	1				
494	9	36	108	139	8	1	0	10	12	1				

495	9	36	108	139	2	2	0	8	22	2	25	1		
496	9	36	109	140	1	2	0	7	15	1	20	1		
497	9	36	109	140	5	1	0	4	22	1				
498	9	36	109	140	3	1	0	5	16	1	25	1		
499	9	36	109	140	5	5	0	0	20	1				
500	9	37	110	141	4	4	0	9						
501	9	37	110	141	9	4	0	3	19	1				
502	9	37	110	141	4	3	0	8	17	1				
503	9	37	111	142	8	3	0	3	16	1	23	1	27	1
504	9	37	111	142	3	3	0	0						
505	9	37	111	142	6	2	0	6						
506	9	37	111	142	3	3	0	3						
507	9	37	111	142	6	2	0	6	17	1				
508	9	37	111	142	6	1	0	7						

Tabela Anexo A .9 - Definição das Operações - Problema 2

Operação	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup	Operação Anterior	Setup
7	43	4	78	4	140	2	192	6	201	4	212	5	222	2	265	4	302	4	333	1	353	6	418	6								
21	62	3	65	5	88	5	163	3	172	3	193	6	199	5	301	2	304	5	317	3	362	5	388	8	417	2	476	5	499	5		
43	48	5	78	3	99	3	140	8	192	6	201	7	222	3	265	0	302	9	333	3	418	6	446	6								
48	7	5	78	5	140	7	192	7	201	7	212	4	222	5	302	4	333	7	353	5	418	5	446	6								
57	71	3	75	7	162	7	227	9	233	6	261	10	273	5	303	5	356	9	379	7	407	7	412	7								
62	21	2	65	4	88	4	152	6	163	1	172	6	199	6	301	6	304	4	317	8	362	7	388	7	417	6	476	5	499	5		
65	21	5	62	3	88	4	152	8	163	8	193	4	199	5	304	7	317	4	362	6	388	6	417	8	476	6	499	6				
71	57	4	75	3	162	4	227	2	261	8	273	3	300	3	344	4	356	2	359	4	407	4	408	6	412	6						
75	57	4	71	6	162	2	167	6	221	8	227	5	233	6	273	2	344	5	356	5	359	6	379	4	407	6	408	6	412	8		
78	7	4	43	5	48	4	99	4	212	7	222	5	265	5	302	3	333	3	418	4												
88	67	2	65	3	163	5	172	5	193	8	199	3	304	7	317	5	362	5	388	7	417	6	476	2	499	6						
99	7	3	43	7	48	3	78	2	140	8	201	5	212	7	222	5	265	7	302	7	333	6	353	8	418	5	446	6				
140	43	6	48	6	78	6	99	3	192	2	201	5	302	6	333	2	353	4	418	8	446	4										
152	21	8	62	5	65	3	163	2	172	2	193	3	199	1	301	5	304	4	317	7	362	5	388	5	476	4	499	2				
162	57	8	71	5	75	5	221	3	227	5	233	4	261	4	273	3	300	5	303	5	344	3	356	2	379	5	412	5				
163	21	10	62	4	65	4	88	5	152	5	172	4	193	5	199	9	301	4	304	6	317	5	362	3	388	6	476	7	499	8		
167	71	9	75	5	162	2	221	5	233	1	300	8	303	3	356	3	359	3	379	5	407	1	408	4	412	4						
172	21	2	62	5	88	2	152	1	163	2	193	2	199	8	301	5	304	1	317	7	362	7	476	7	499	7						
192	7	7	43	6	78	3	99	7	140	5	201	3	222	8	265	3	302	6	333	0	353	3	418	3	446	3						
193	21	4	62	6	163	4	172	6	199	1	301	5	317	2	362	5	388	4	417	9	476	2	499	4								
199	21	4	62	5	65	4	88	4	152	4	163	6	172	4	193	5	301	4	304	3	317	4	362	6	417	5	476	5	499	4		
201	7	4	78	1	99	6	140	1	212	7	222	5	265	8	302	6	353	4	418	5	446	2										
212	7	3	43	5	48	3	78	7	99	2	140	8	192	5	201	1	222	5	302	8	333	4	418	4								
221	57	4	71	8	75	2	162	3	167	0	227	4	233	3	261	5	300	5	344	6	356	8	379	5	407	6	408	4	412	4		
222	7	6	43	6	48	6	78	3	99	5	140	3	192	5	201	9	212	6	265	5	302	3	353	6	446	3						
227	57	3	71	4	75	1	162	7	167	5	233	7	261	2	300	8	303	4	344	0	356	2	359	4	379	5	408	7				
233	57	2	71	5	162	4	167	7	221	0	227	8	261	4	300	5	303	8	344	5	407	7	408	3	412	7						
261	57	8	71	7	75	7	167	3	221	4	227	4	233	6	273	1	300	6	303	6	359	2	379	4	408	6	412	7				
265	7	4	43	4	48	7	78	7	99	4	140	6	192	1	201	4	222	4	333	6	353	7	418	9	446	6						
273	57	6	71	3	75	6	162	5	221	2	227	3	261	4	300	6	303	6	356	5	359	4	407	4	408	5	412	5				
300	57	7	71	5	75	2	162	8	221	9	227	5	261	2	303	8	356	5	359	5	379	5	407	5	412	5						
301	21	0	62	5	88	4	152	5	172	4	193	3	199	9	304	10	317	3	362	7	388	7	417	7	499	4						
302	7	8	43	4	48	6	78	3	99	8	140	4	212	8	265	3	333	5	353	5	418	4	446	5								
303	57	3	71	6	75	4	162	2	167	3	227	6	233	6	273	6	300	7	344	4	356	10	359	6	407	8	408	7	412	7		
304	62	10	65	3	88	4	152	5	163	4	172	5	193	7	199	5	301	4	317	8	362	7	388	5	499	5						
317	21	5	62	9	65	3	88	8	152	9	163	6	199	7	301	7	304	6	362	7	388	6	417	5	476	5	499	9				
333	7	6	43	7	48	5	140	6	192	4	201	5	212	0	222	5	302	6	353	6	418	2	446	1								
344	71	7	75	8	167	4	221	5	227	3	233	5	261	4	273	7	300	3	303	8	356	4	359	2	379	7	407	2	408	3	412	8
353	43	9	48	2	78	4	99	4	140	7	192	4	201	10	212	5	265	3	302	7	418	5	446	2								
356	57	7	71	7	75	5	162	8	221	4	227	5	233	5	261	7	273	8	300	5	303	3	359	8	379	4	407	5	408	5	412	5
359	71	4	75	4	162	4	227	5	233	2	261	1	273	5	300	2	356	2	407	5	408	2	412	1								
362	21	3	62	5	65	4	88	5	152	5	163	2	193	6	199	3	301	6	304	4	388	4	417	4	476	4	499	3				
379	57	7	71	2	162	4	167	7	221	3	227	3	233	3	261	8	273	6	300	7	303	3	344	8	359	8	407	6	412	3		
388	21	5	65	7	88	7	152	4	163	4	172	6	193	3	304	6	317	8	476	6	499	5										
407	57	7	71	4	162	3	221	7	227	8	261	3	300	2	303	3	344	6	356	8	359	2	379	4								
408	57	5	71	4	75	5	162	7	167	0	227	4	233	9	261	9	300	3	303	9	344	3	356	10	359	4	379	7	407	4		
412	57	7	71	7	75	0	162	6	221	8	233	3	261	0	273	6	300	8	303	3	356	5	359	8	379	0						
417	21	3	62	8	65	0	88	5	152	4	163	5	172	7	193	1	199	7	304	7	317	4	362	7	388	5	476	9	499	7		
418	7	4	48	6	78	4	99	4	140	4	192	6	201	6	212	7	222	6	265	6	302	8	333	3	353	7	446	4				
446	7	5	43	4	48	1	78	4	99	8	140	7	192	4	201	5	212	4	222	10	265	8	302	5	333	3	353	3	418	4		
476	62	4	65	6	88	4	163	5	172	4	193	7	301	7	304	6	317	6	362	2	499	4										
499	21	6	62	9	65	3	88	6	152	4	199	2	304	3	317	4	362	6	417	4	476	7										

Tabela Anexo A .10 - Definição dos Tempos de Preparo Dependentes da Seqüência - Problema 1

Anexo B - Diagramas de Fluxo do Sistema

Para todos os arquivos .m criados serão apresentados os diagramas de fluxo. Serão listados, inicialmente, aqueles arquivos que são utilizados tanto no tratamento dos casos clássicos quanto no tratamento dos problemas realistas estudados. Depois, aqueles aplicáveis apenas nos problemas clássicos, seguidos daqueles arquivos que somente são utilizados no tratamento dos casos realistas abordados. Os diagramas de fluxo estão listados na ordem em que são chamados durante a execução do algoritmo genético. Nos fluxogramas, todos os processos com sombra cinza são arquivos da *toolbox*, utilizados sem mudanças, e, portanto, não serão listados. Apenas aqueles que foram especialmente criados, ou modificados serão apresentados.

Script EXECVRGA

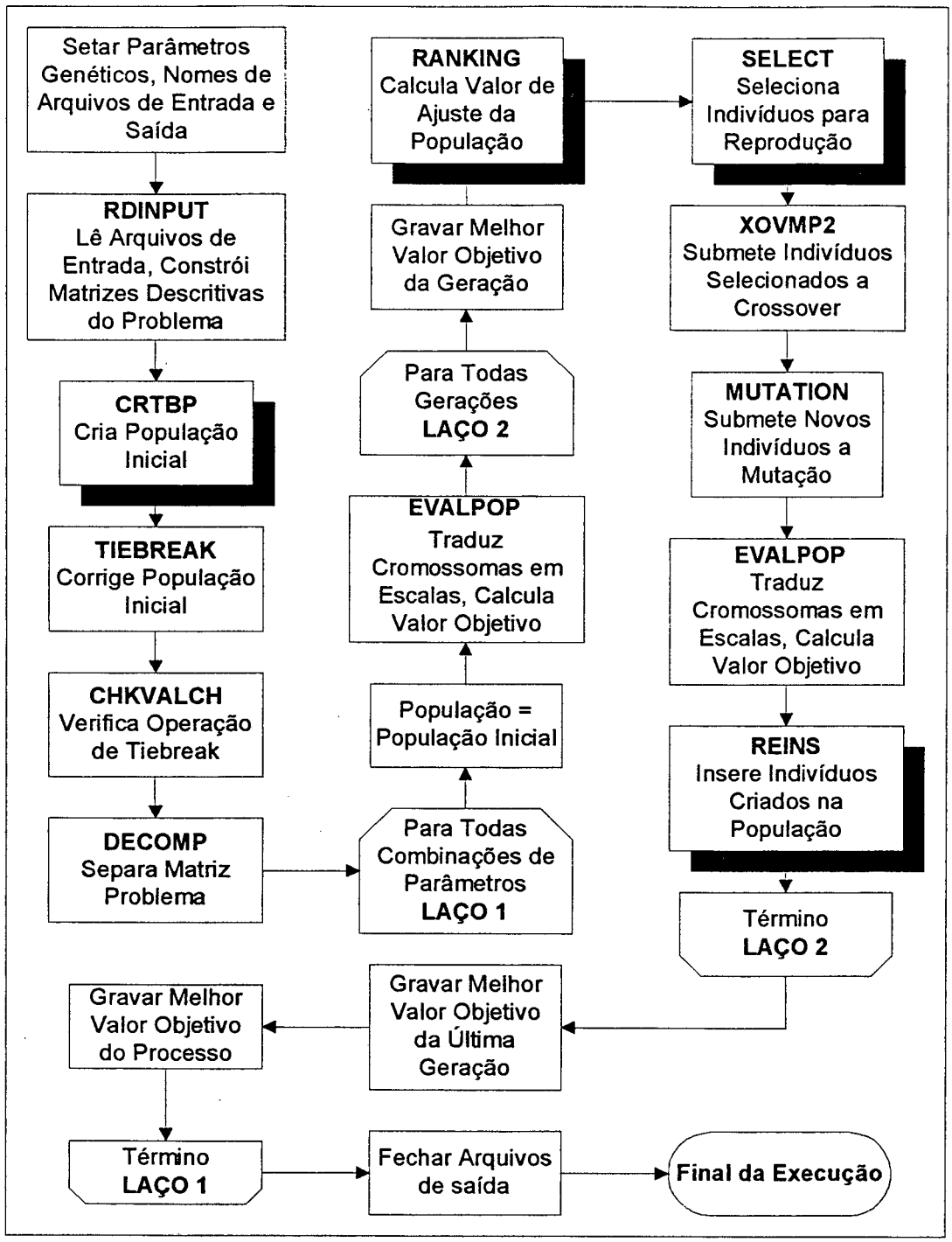

Figura Anexo B .1 - Diagrama de Fluxo - Script EXECVRGA.m

Função RDINPUT

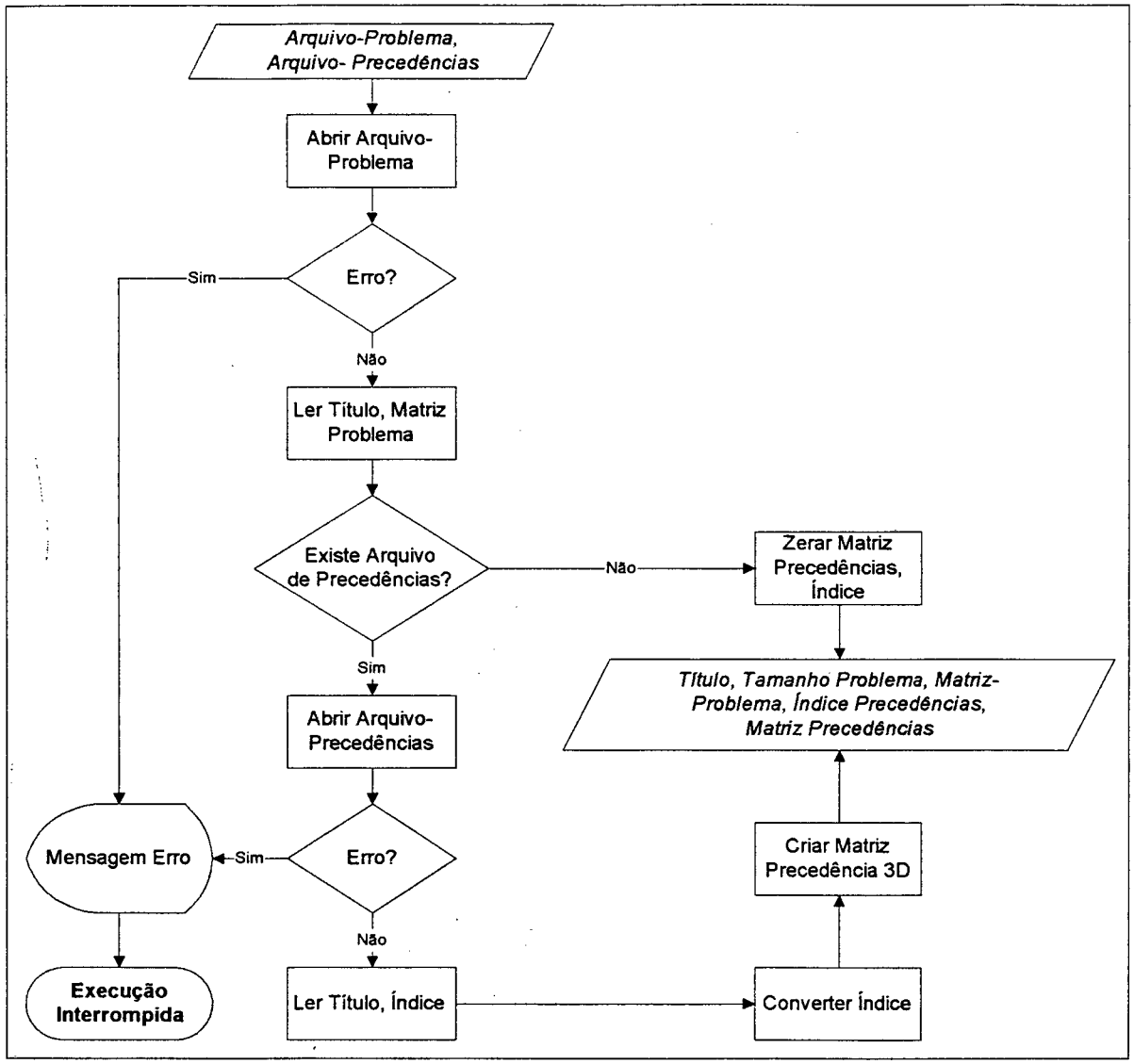

Figura Anexo B .2 - Diagrama de Fluxo - Função RDINPUT.m

Função TIEBREAK

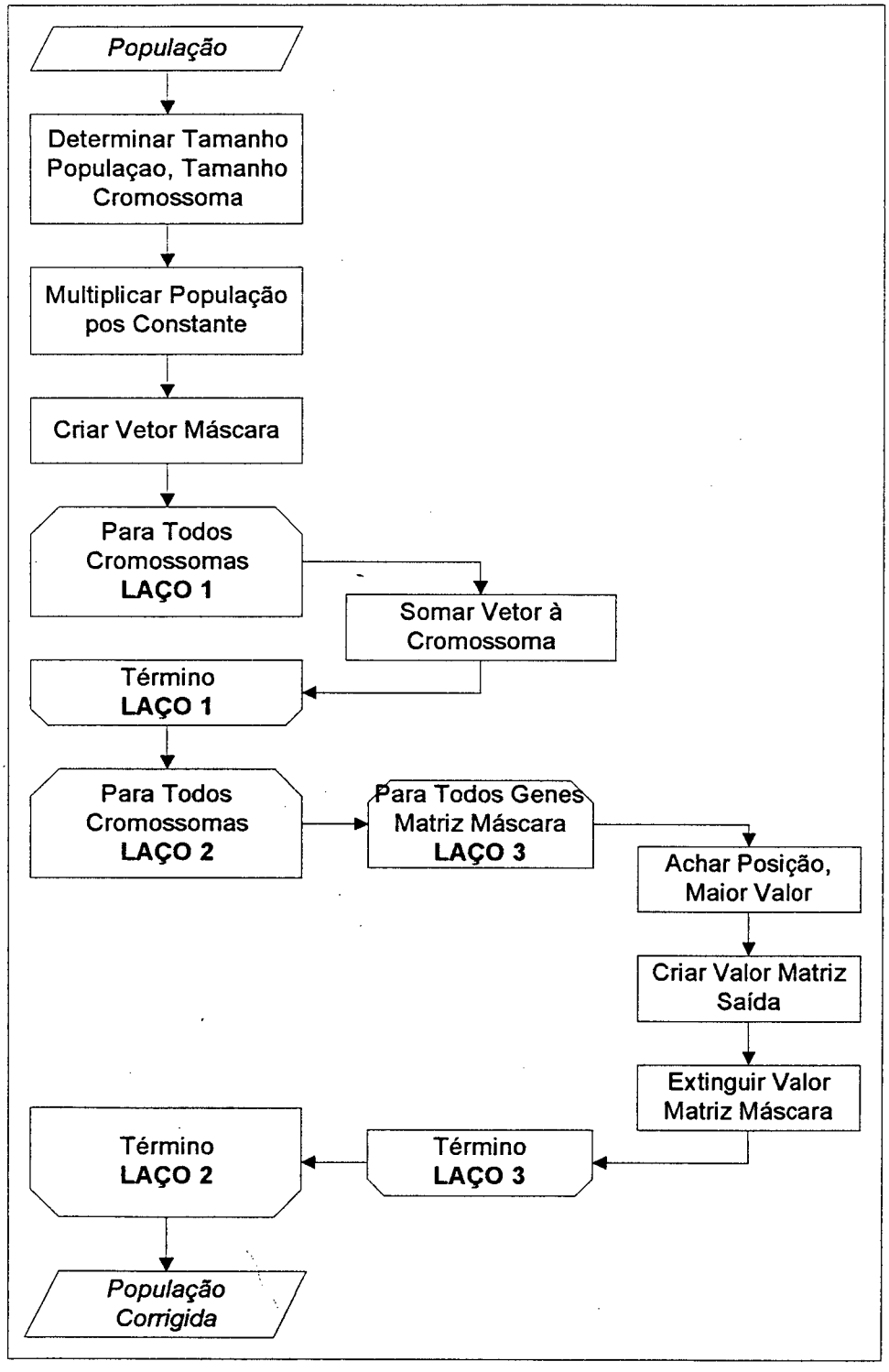

Figura Anexo B .3 - Diagrama de Fluxo - Função TIEBREAK.m

Função CHKVALCH

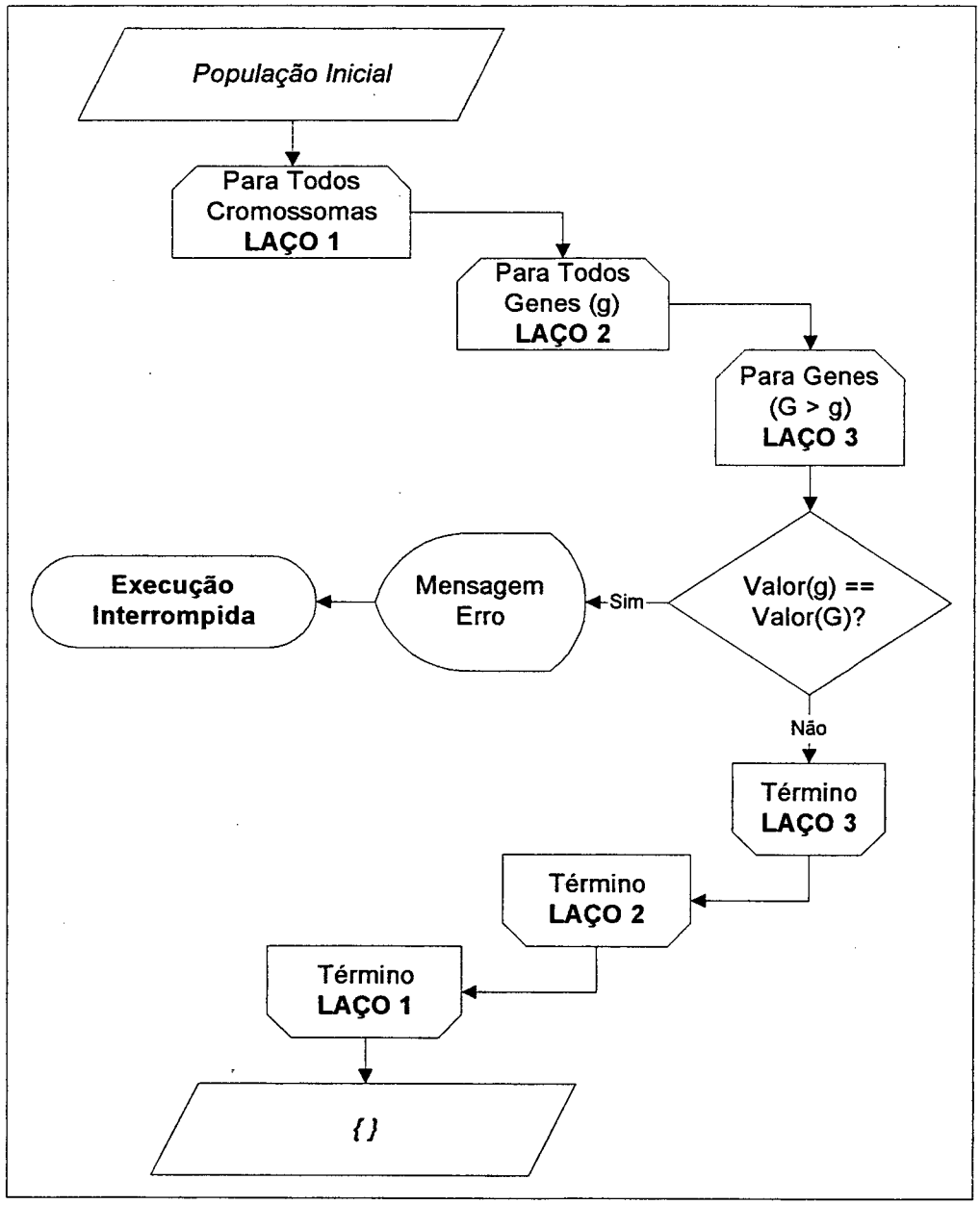

Figura Anexo B .4 - Diagrama de Fluxo - Função CHKVALCH.m

Função DECOMP

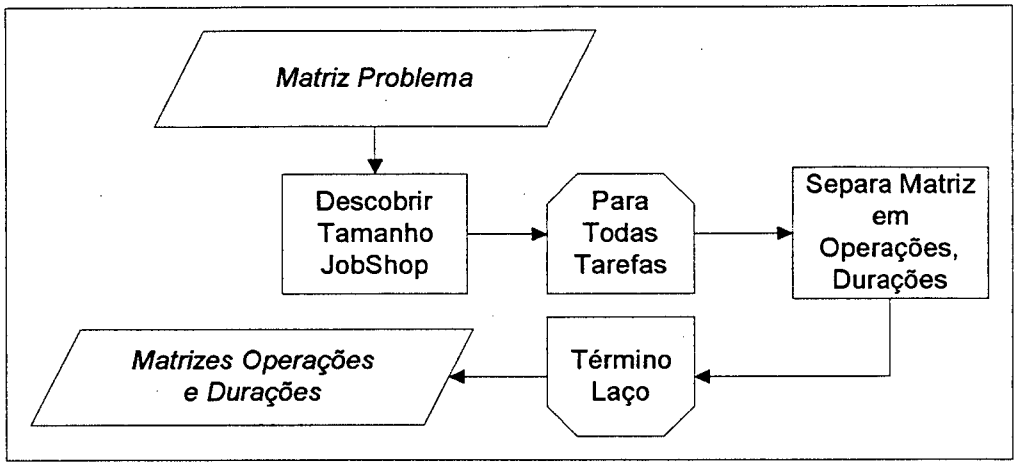

Figura Anexo B .5 - Diagrama de Fluxo - Função DECOMP.m

Função EVALPOP

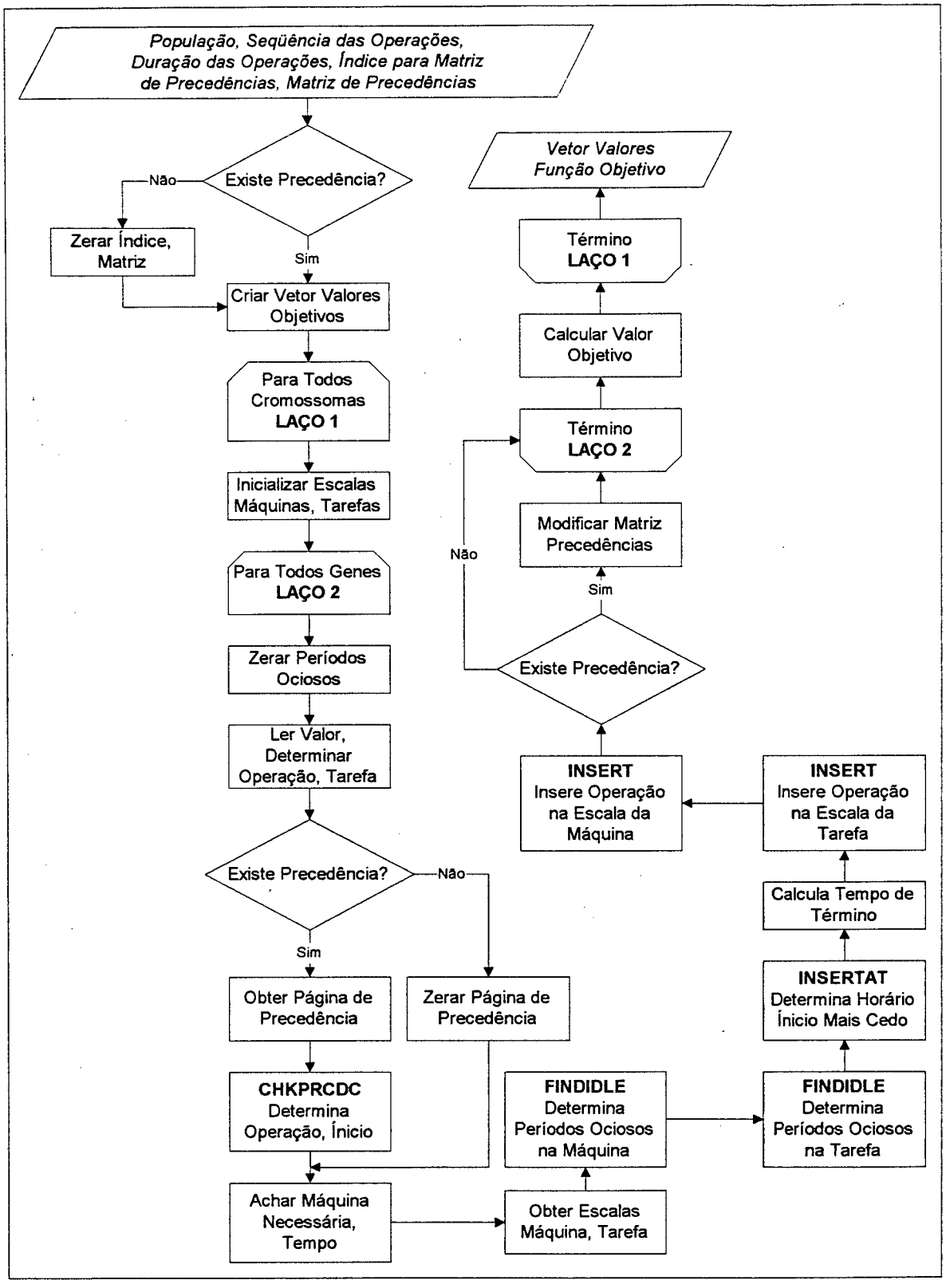

Figura Anexo B .6 - Diagrama de Fluxo - Função EVALPOP.m

Função CHKPRCDC

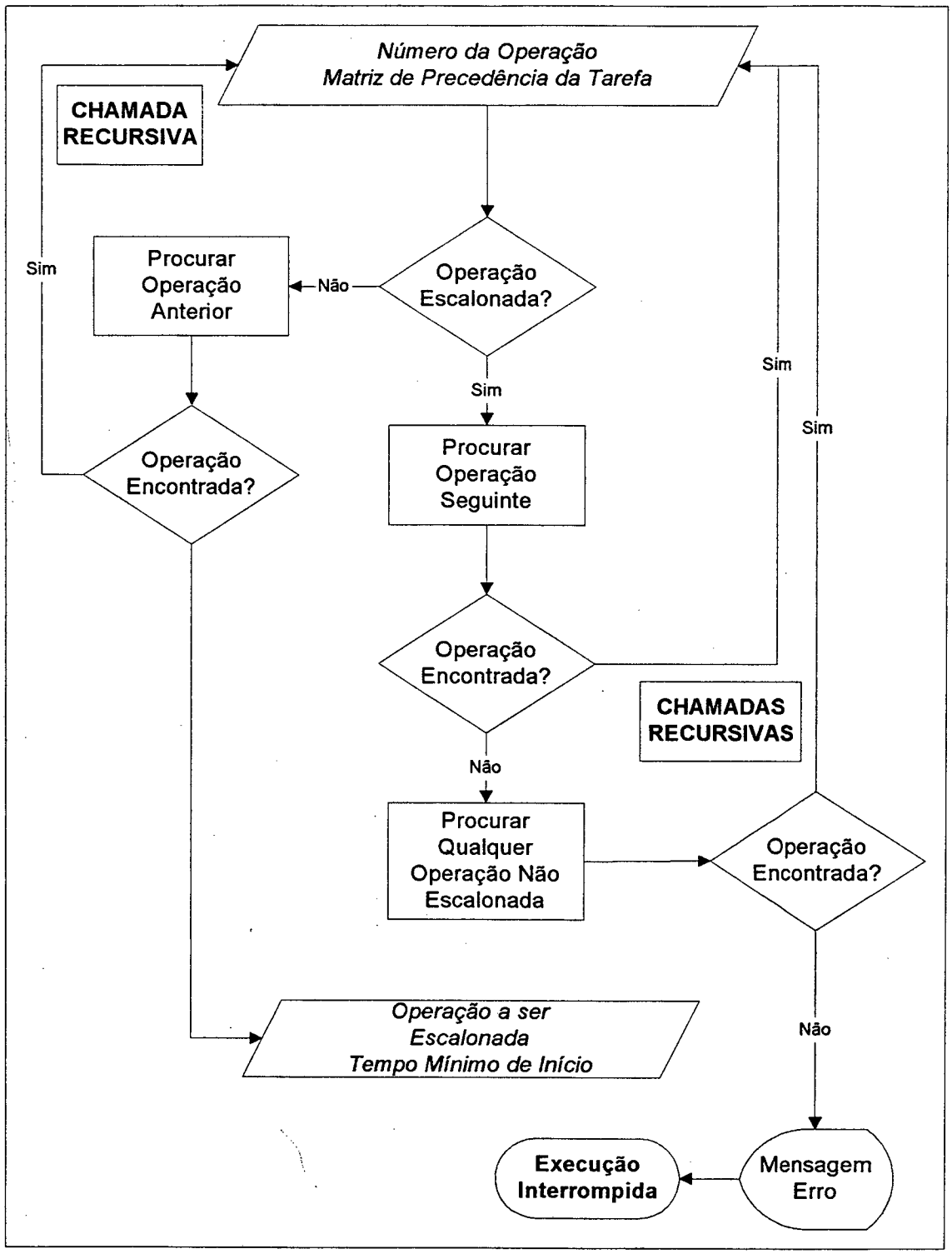

Figura Anexo B .7 - Diagrama de Fluxo - Função CHKPRCDC.m

Função FINDIDLE

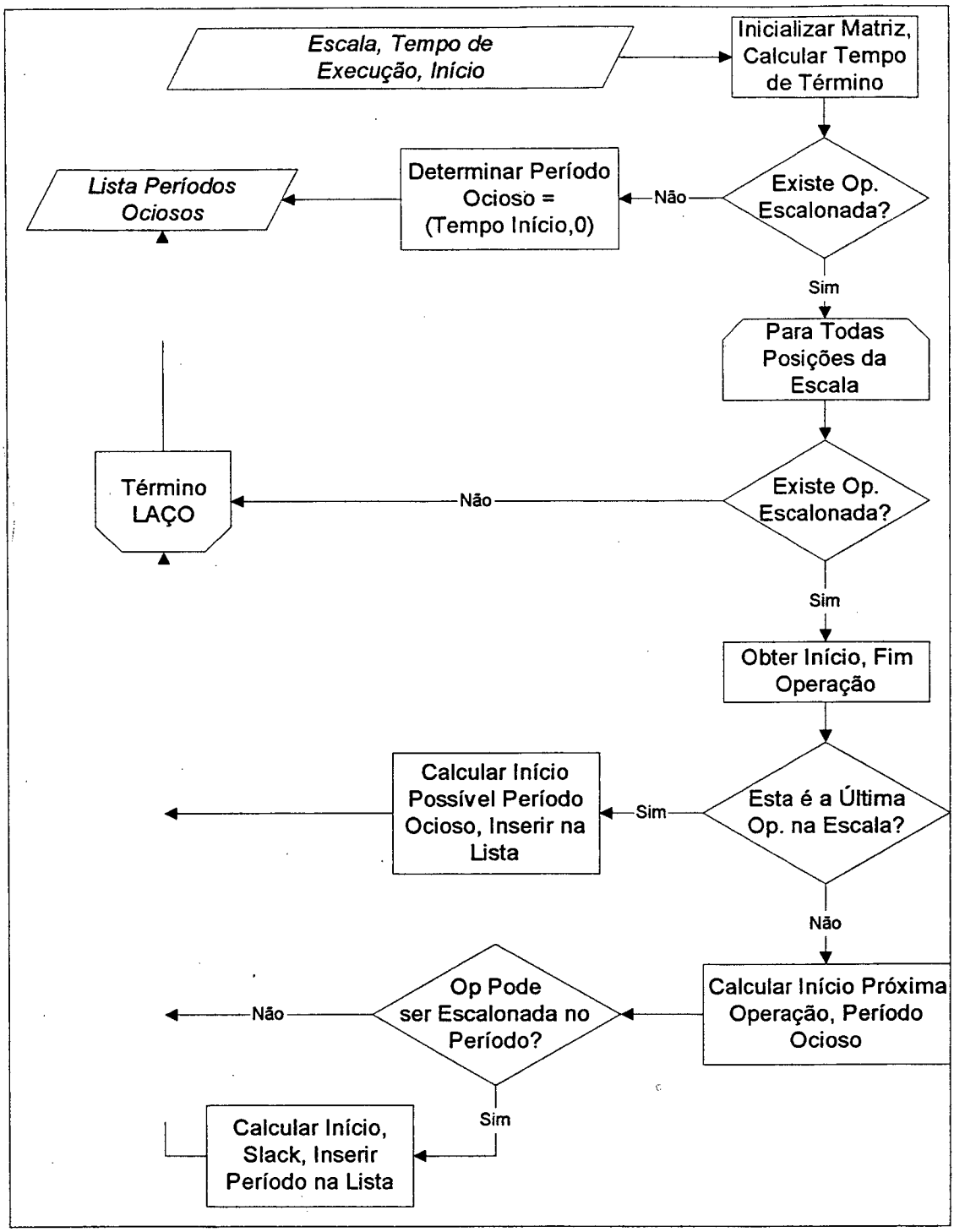

Figura Anexo B .8 - Diagrama de Fluxo - Função FINDIDLE.m

Função INSERTAT

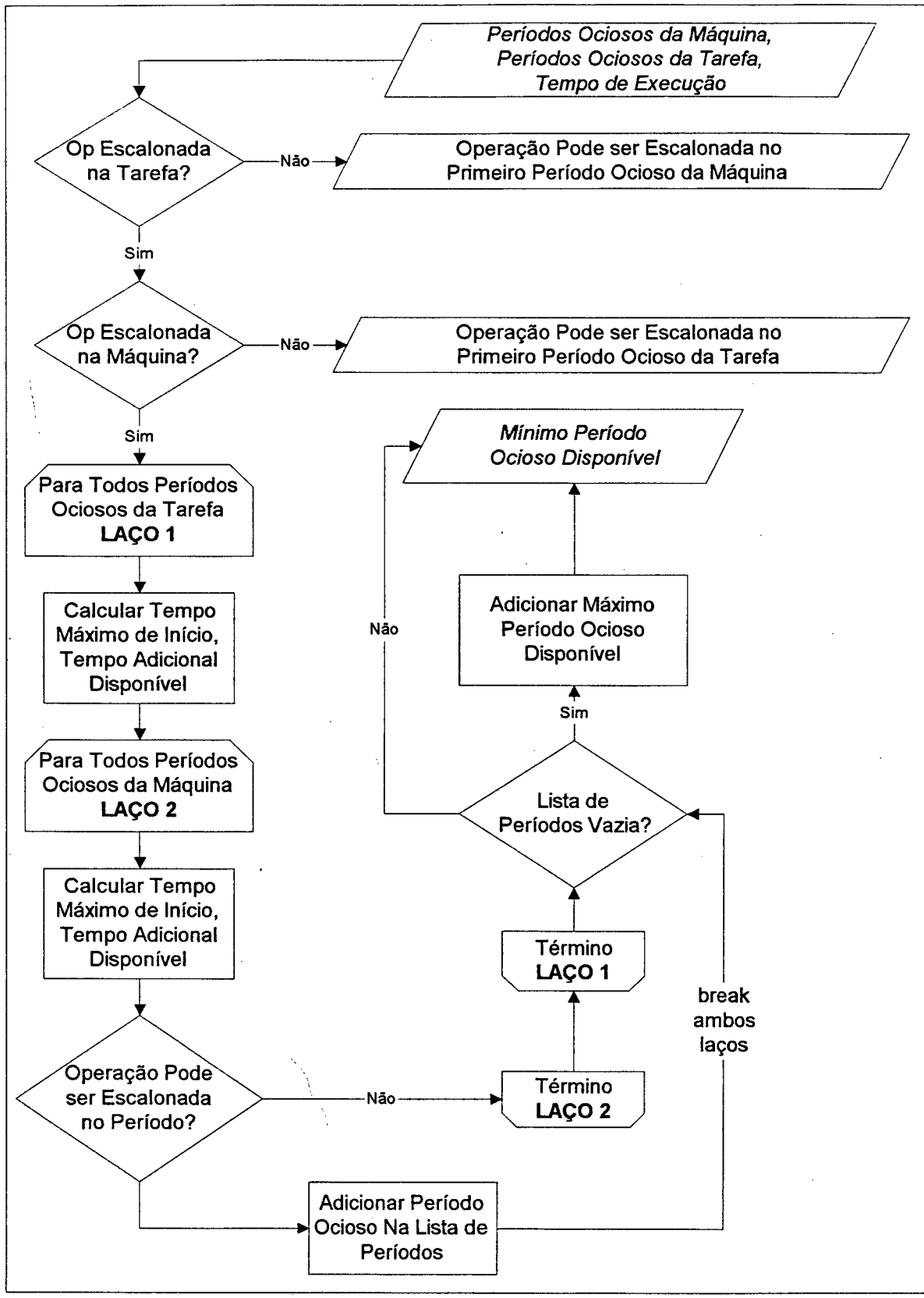

Figura Anexo B .9 - Diagrama de Fluxo - Função INSERTAT.m

Função INSERT

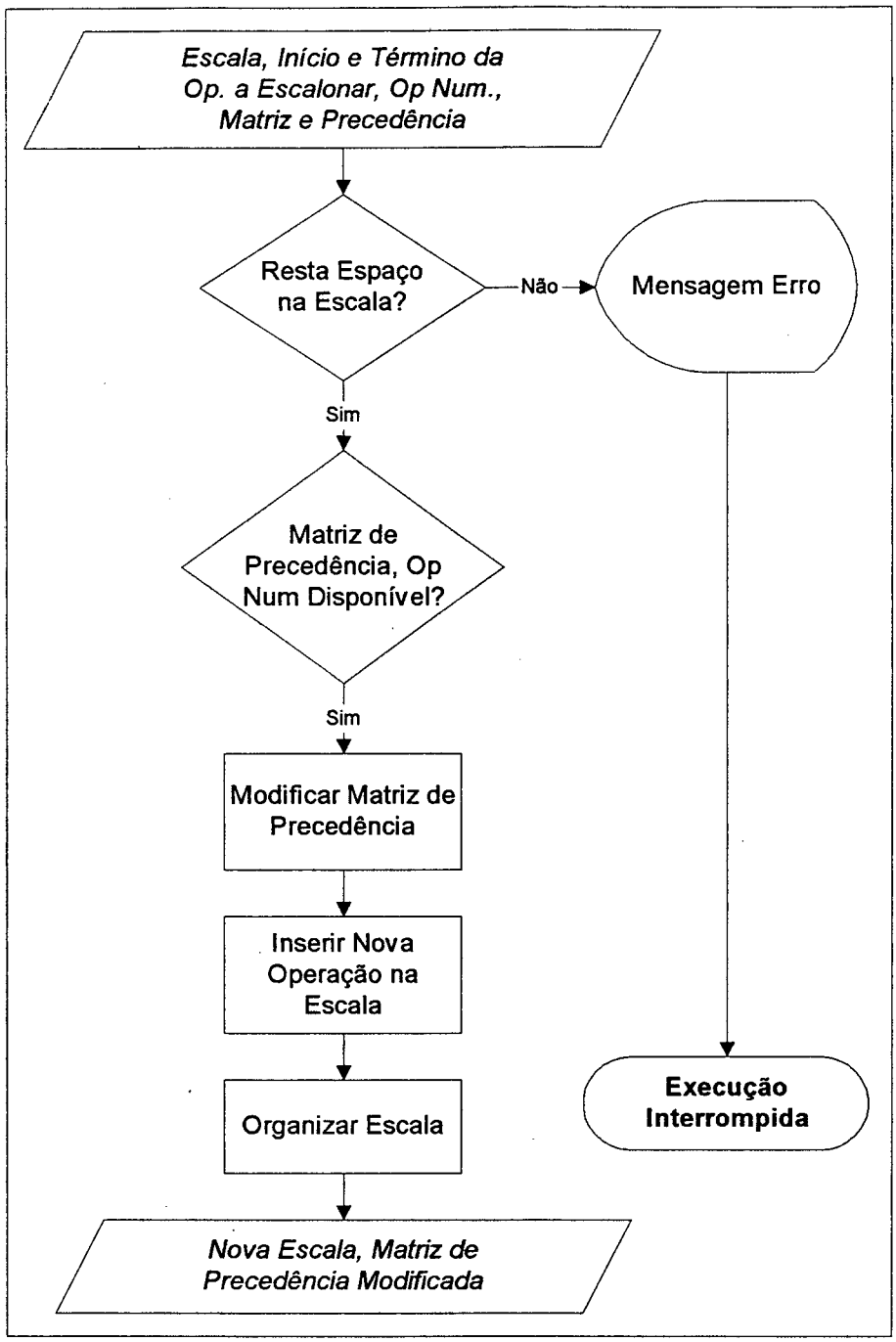

Figura Anexo B .10 - Diagrama de Fluxo - Função INSERT.m

Função XOVMP2

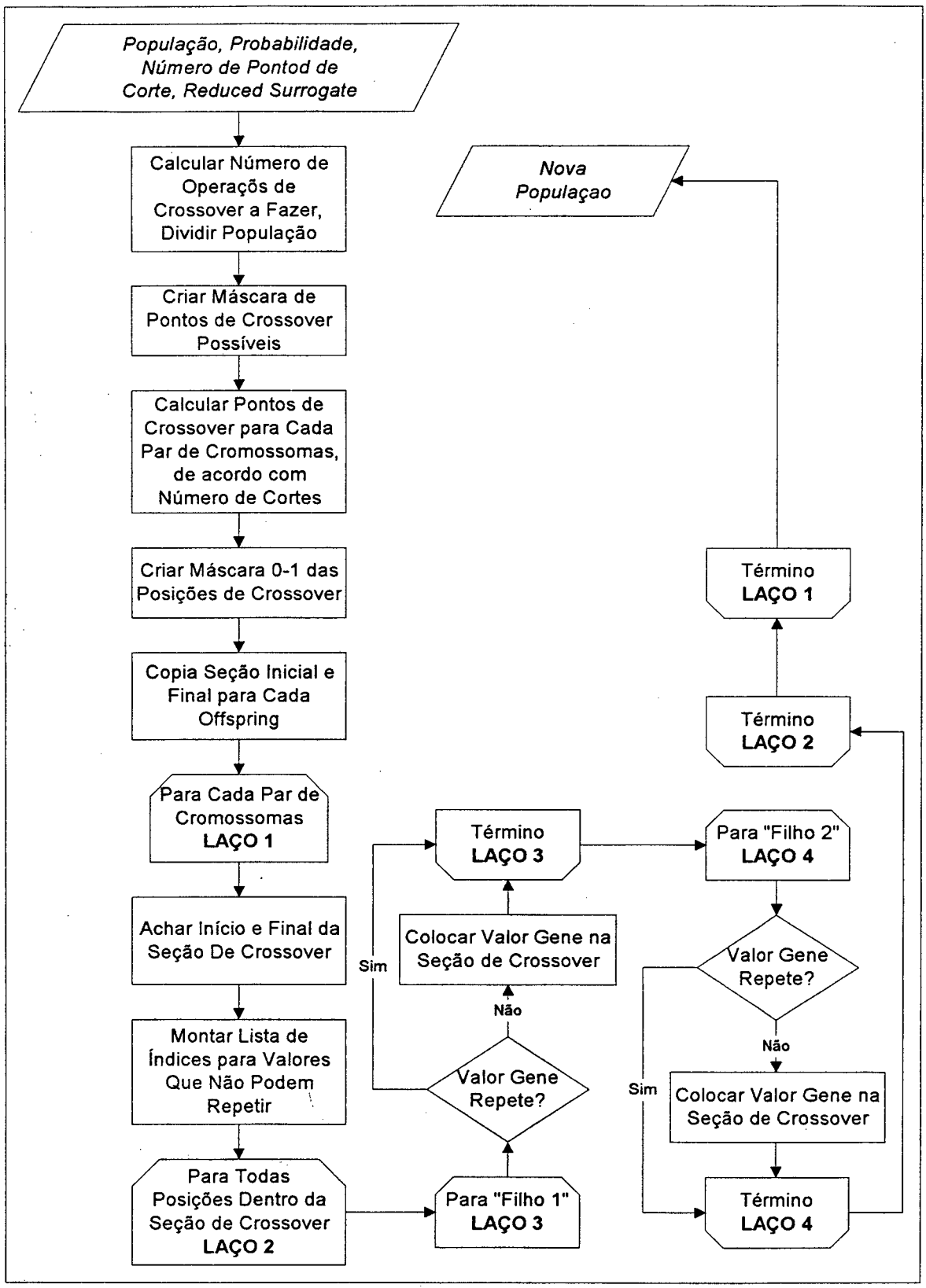

Figura Anexo B .11 - Diagrama de Fluxo - Função XOVMP2.m

Função MUTATION

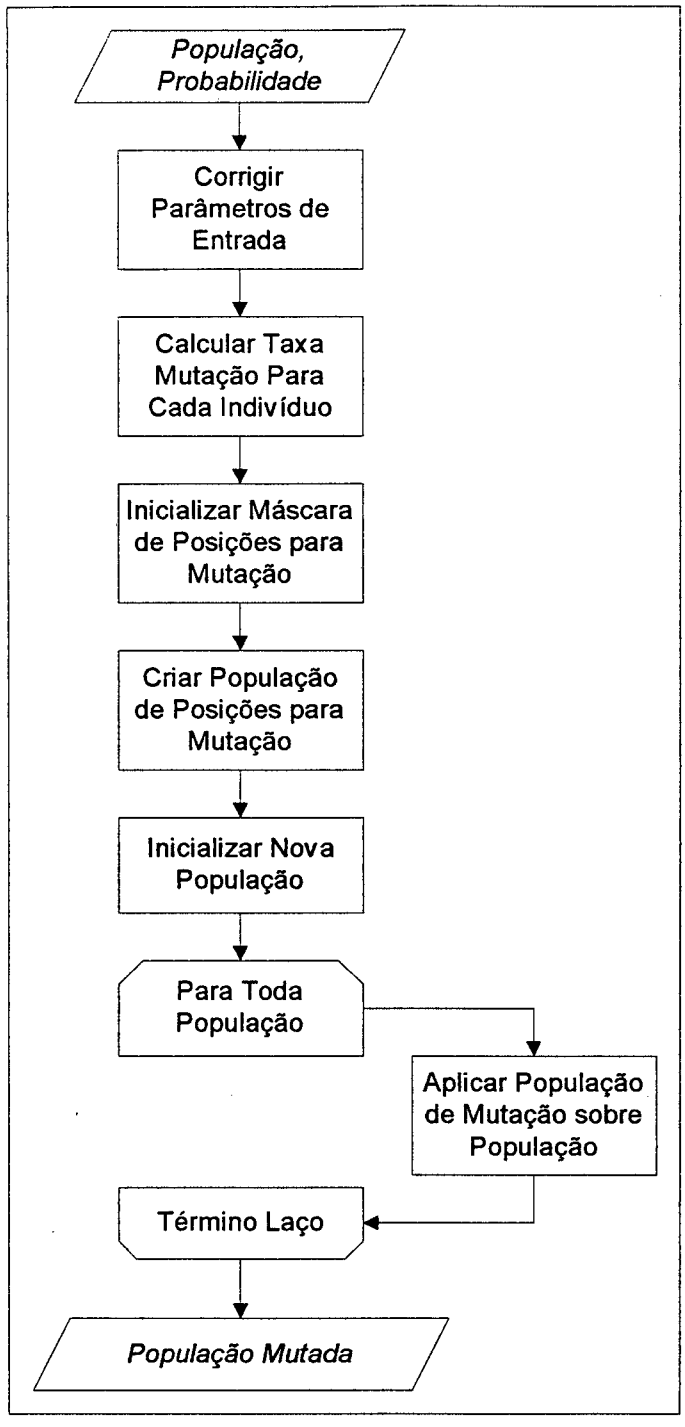

Figura Anexo B .12 - Diagrama de Fluxo - Função MUTATION.m

Script EXECVRGA2

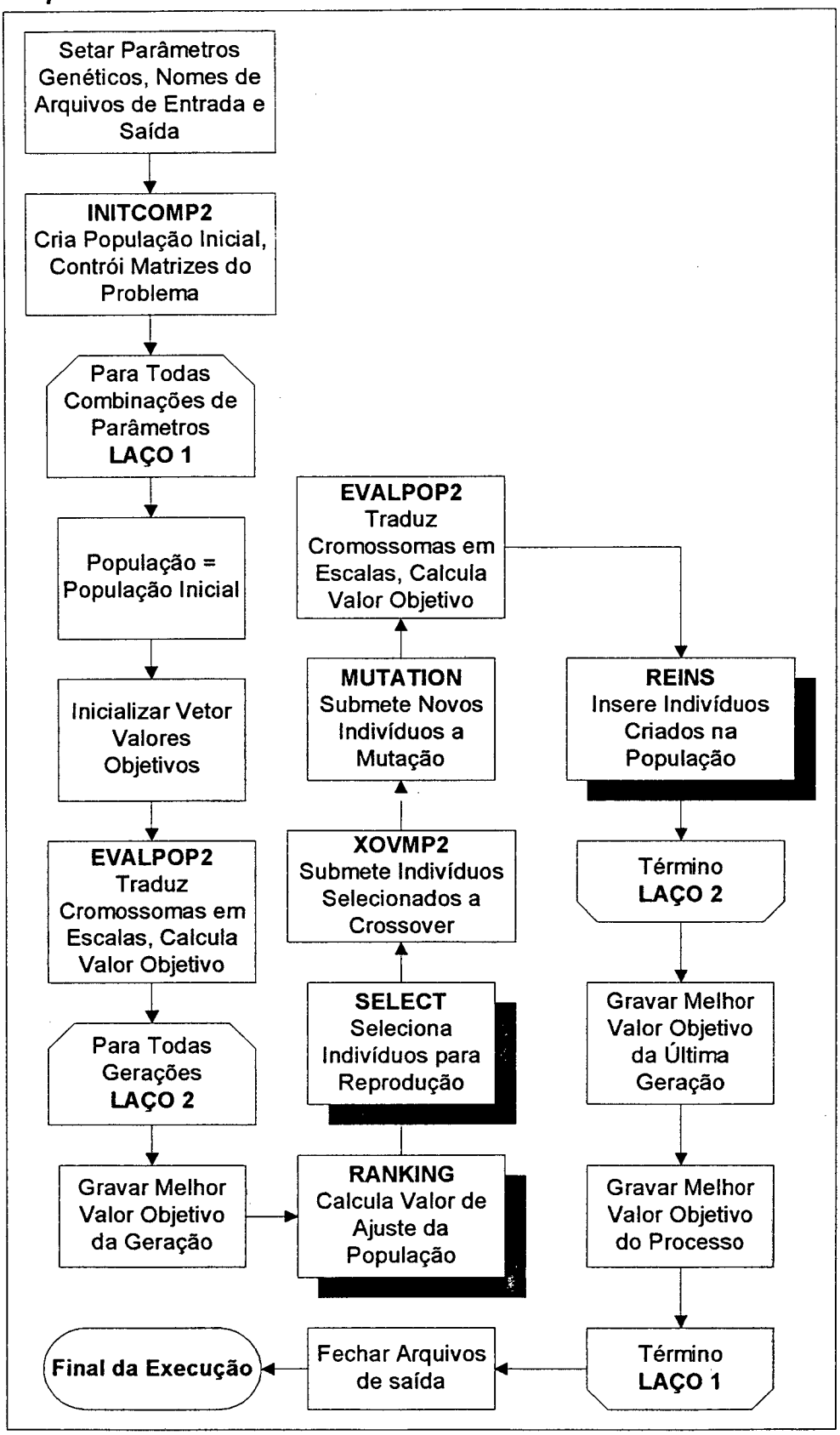

Figura Anexo B .13 - Diagrama de Fluxo - Script EXECVRGA2.m

Função INITCOMP2

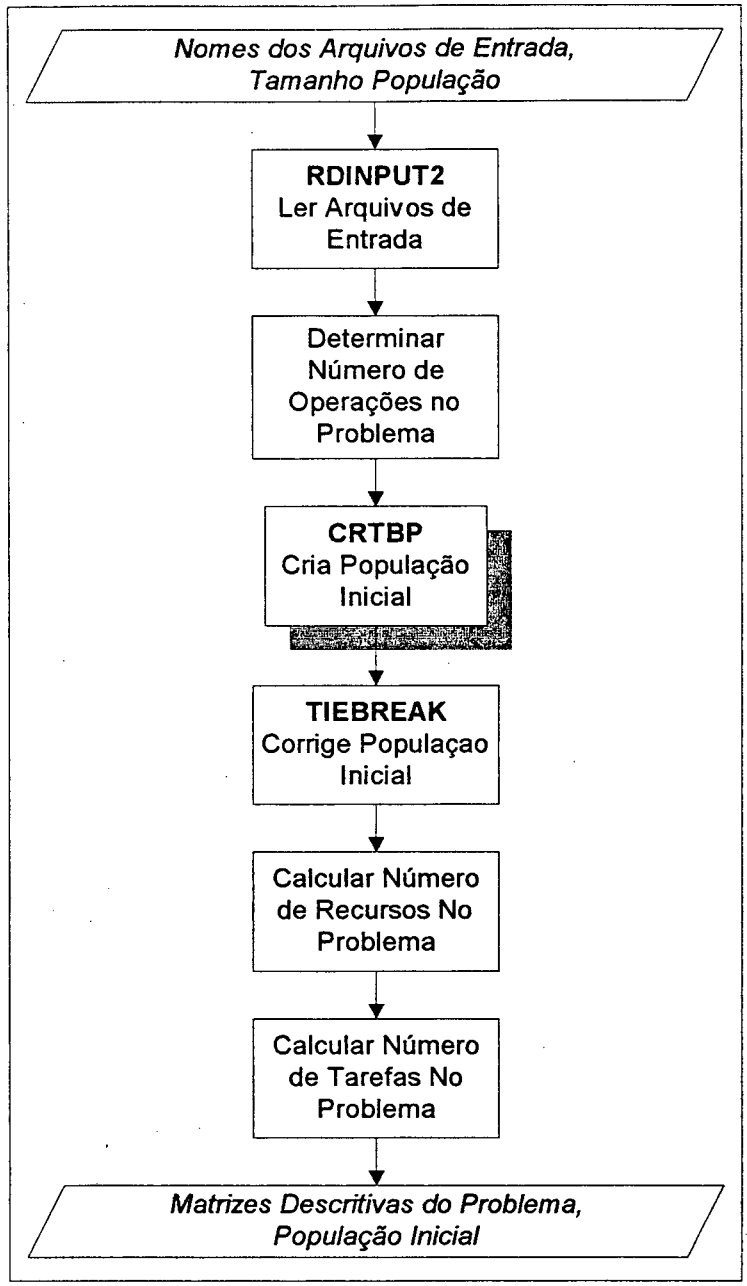

Figura Anexo B .14 - Diagrama de Fluxo - Função INITCOMP2.m

Função EVALPOP2

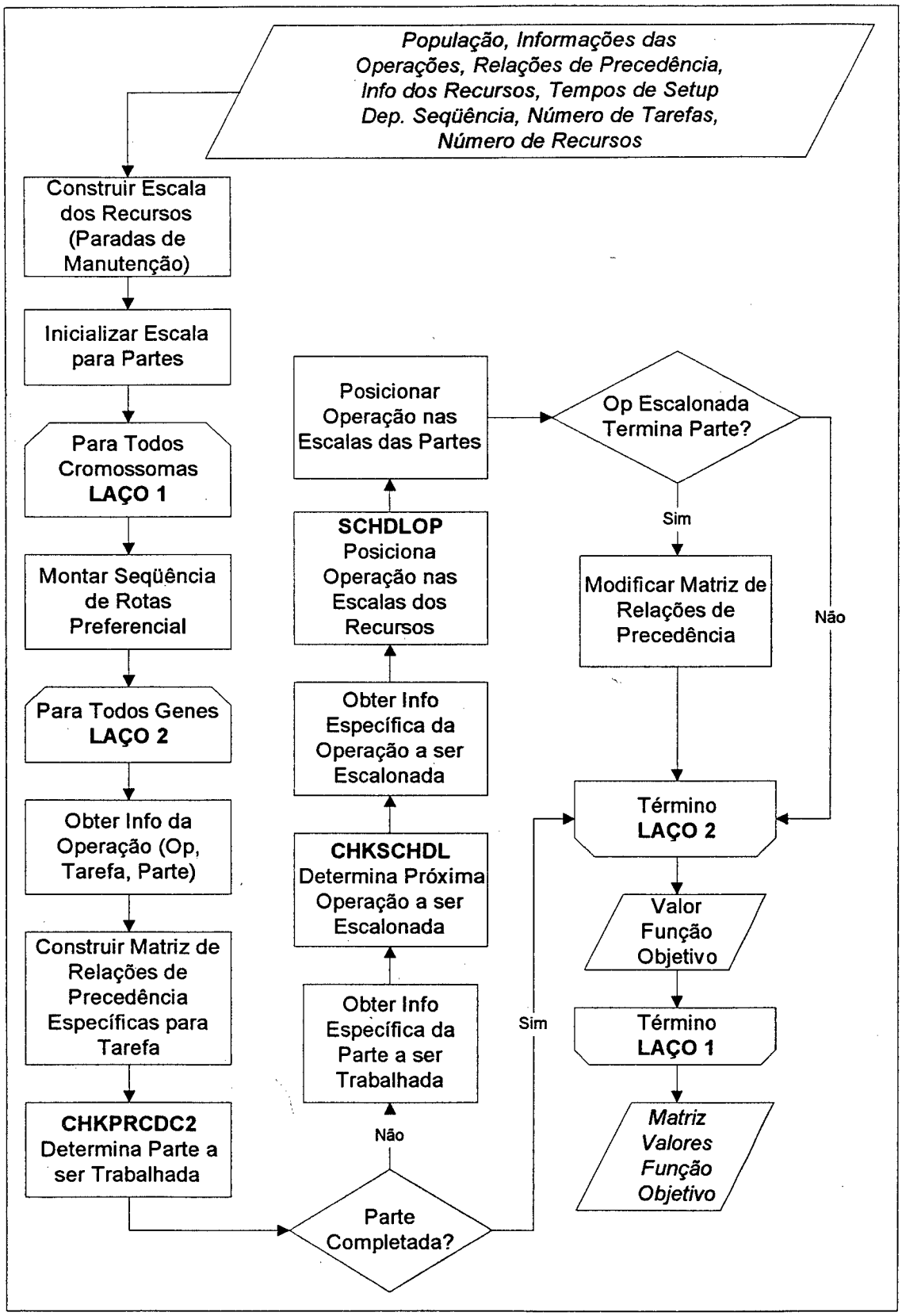

Figura Anexo B .15 - Diagrama de Fluxo - Função EVALPOP2.m

Função CHKPRCDC2

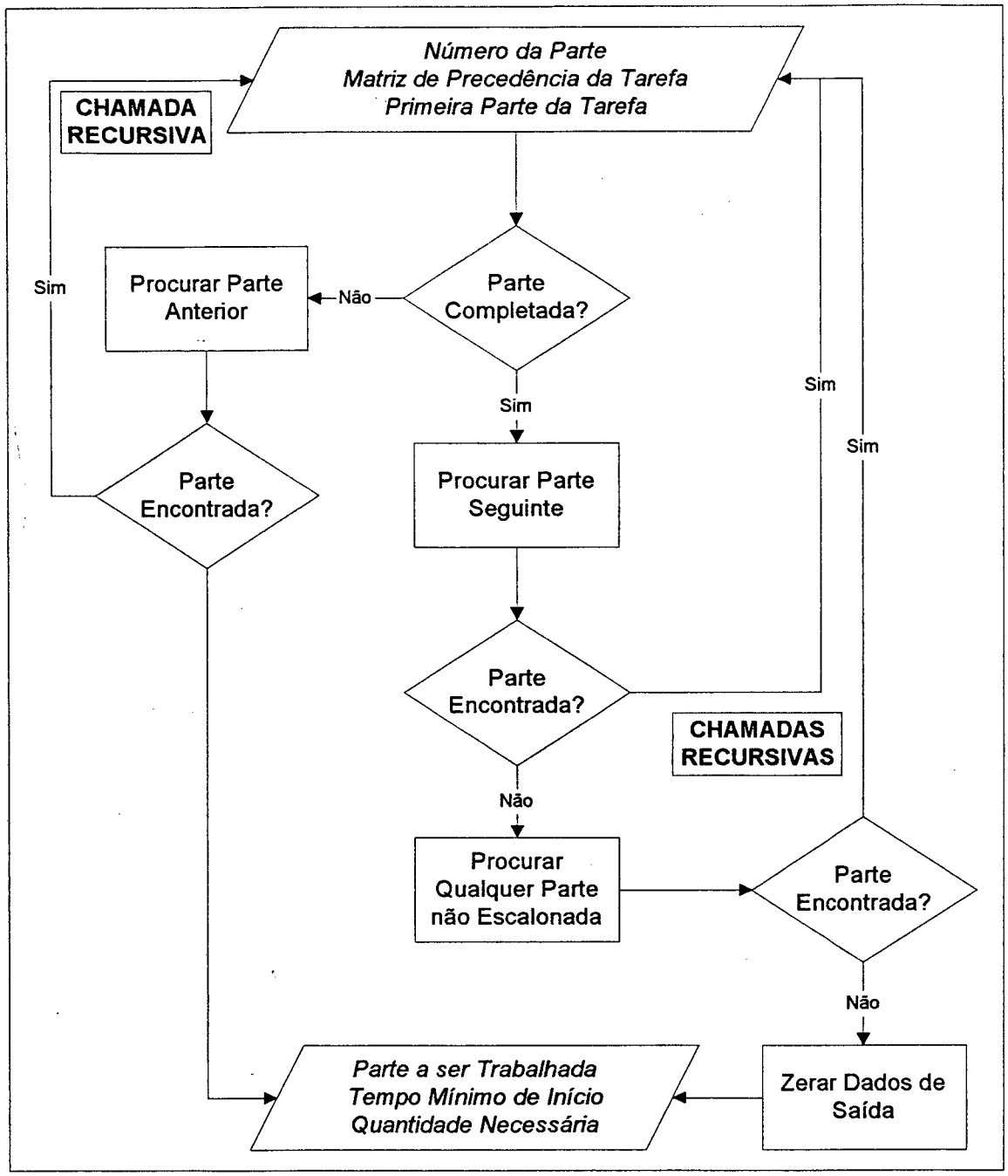

Figura Anexo B .16 - Diagrama de Fluxo - Função CHKPRCDC2.m

Função CHKSCHDL

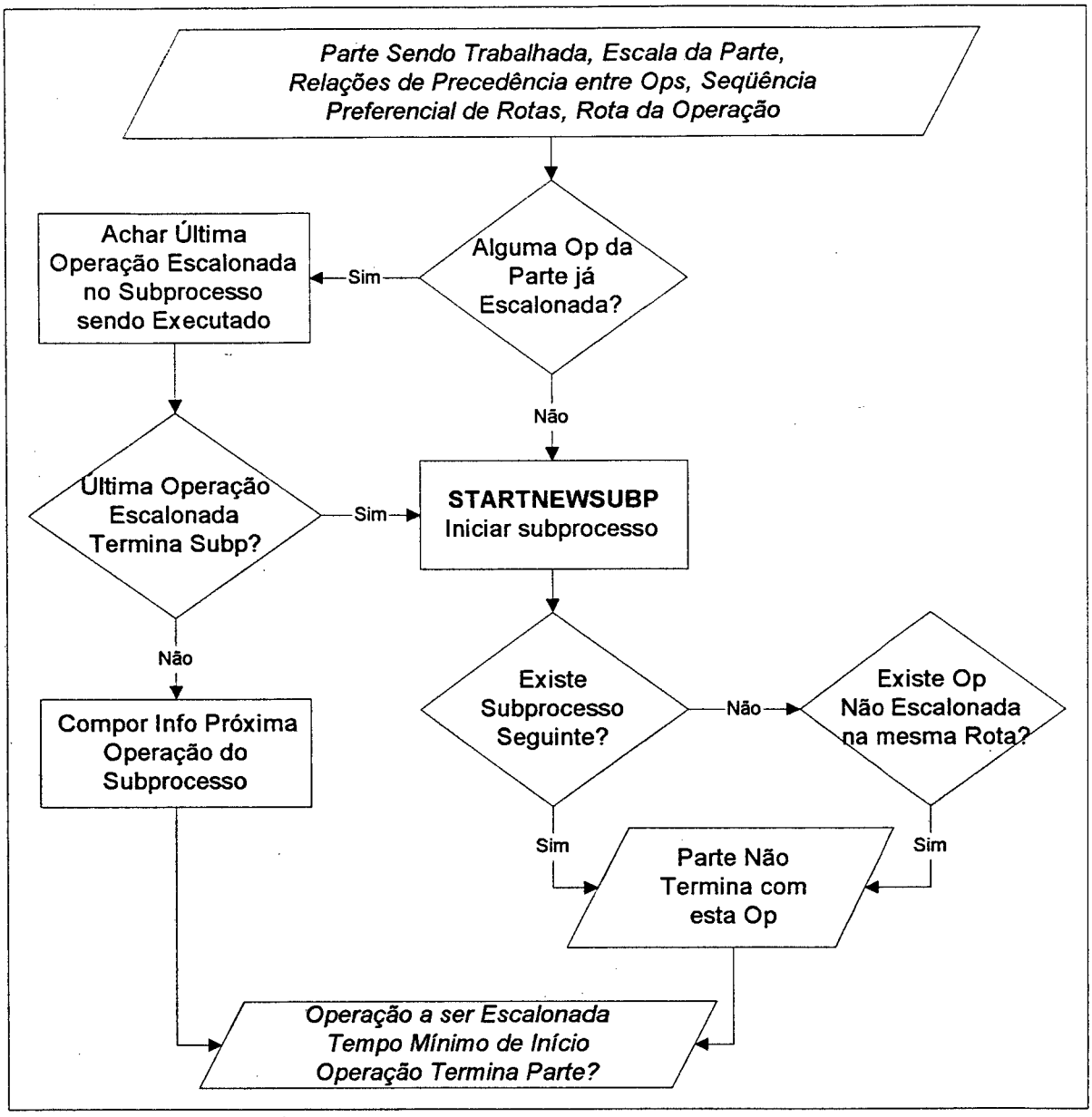

Figura Anexo B .17 - Diagrama de Fluxo - Função CHKSCHDL.m

Função STARTNEWSUBP

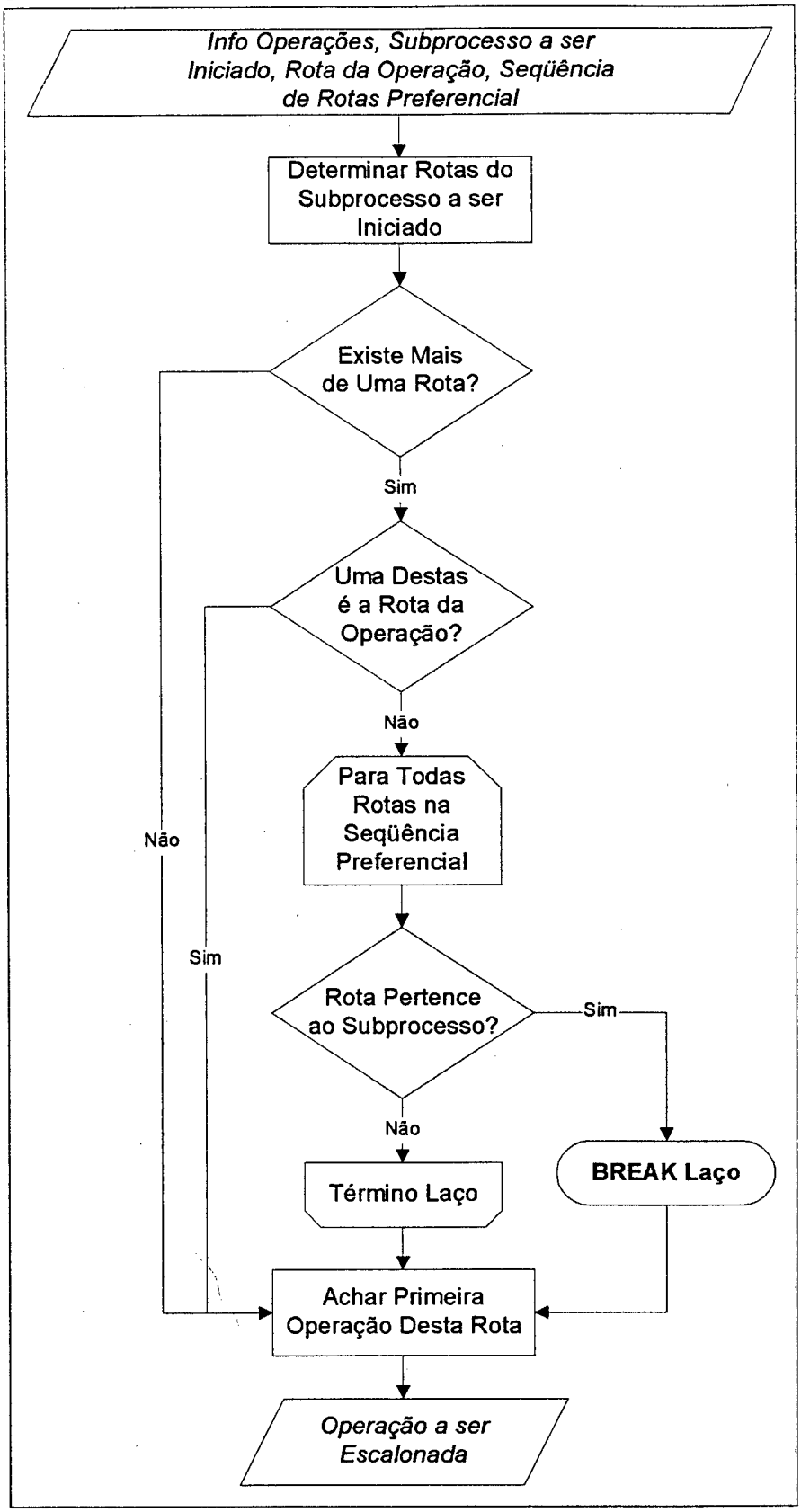

Figura Anexo B .18 - Diagrama de Fluxo - Função STARTNEWSUBP.m

Função SCHDLOP

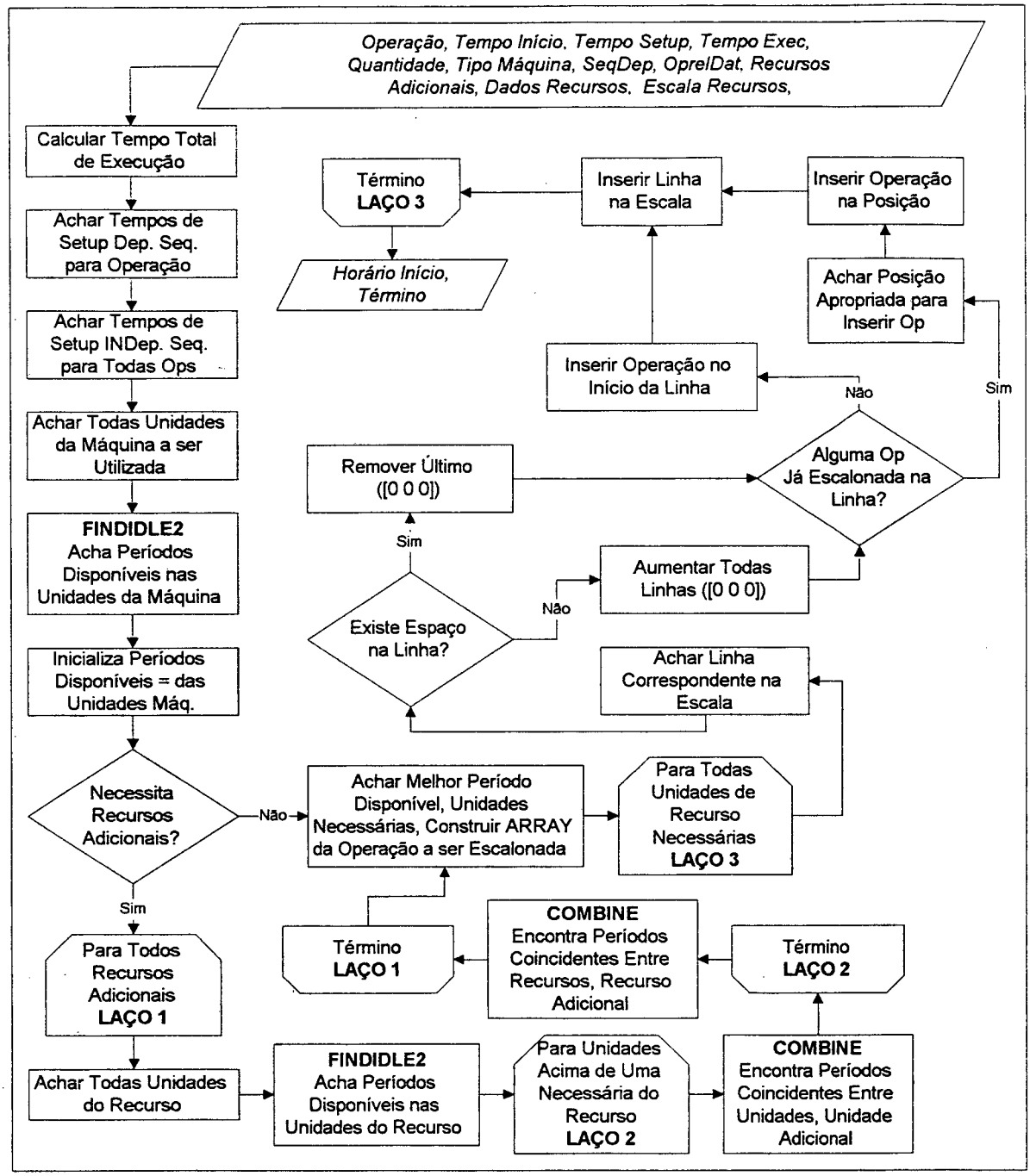

Figura Anexo B .19 - Diagrama de Fluxo - Função SCHDLOP.m

Função FINDIDLE2

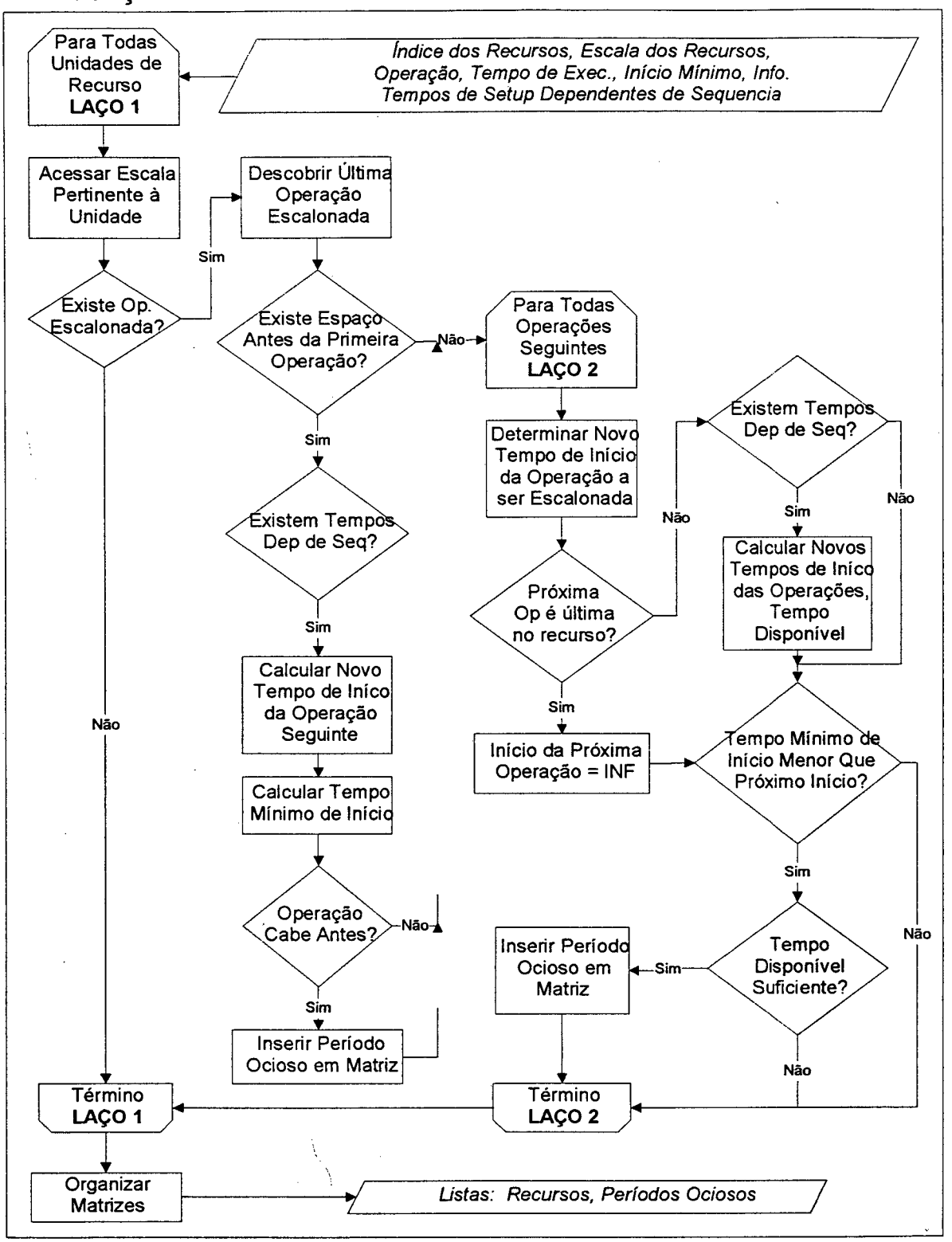

Figura Anexo B .20 - Diagrama de Fluxo - Função FINDIDLE2.m

Função COMBINE

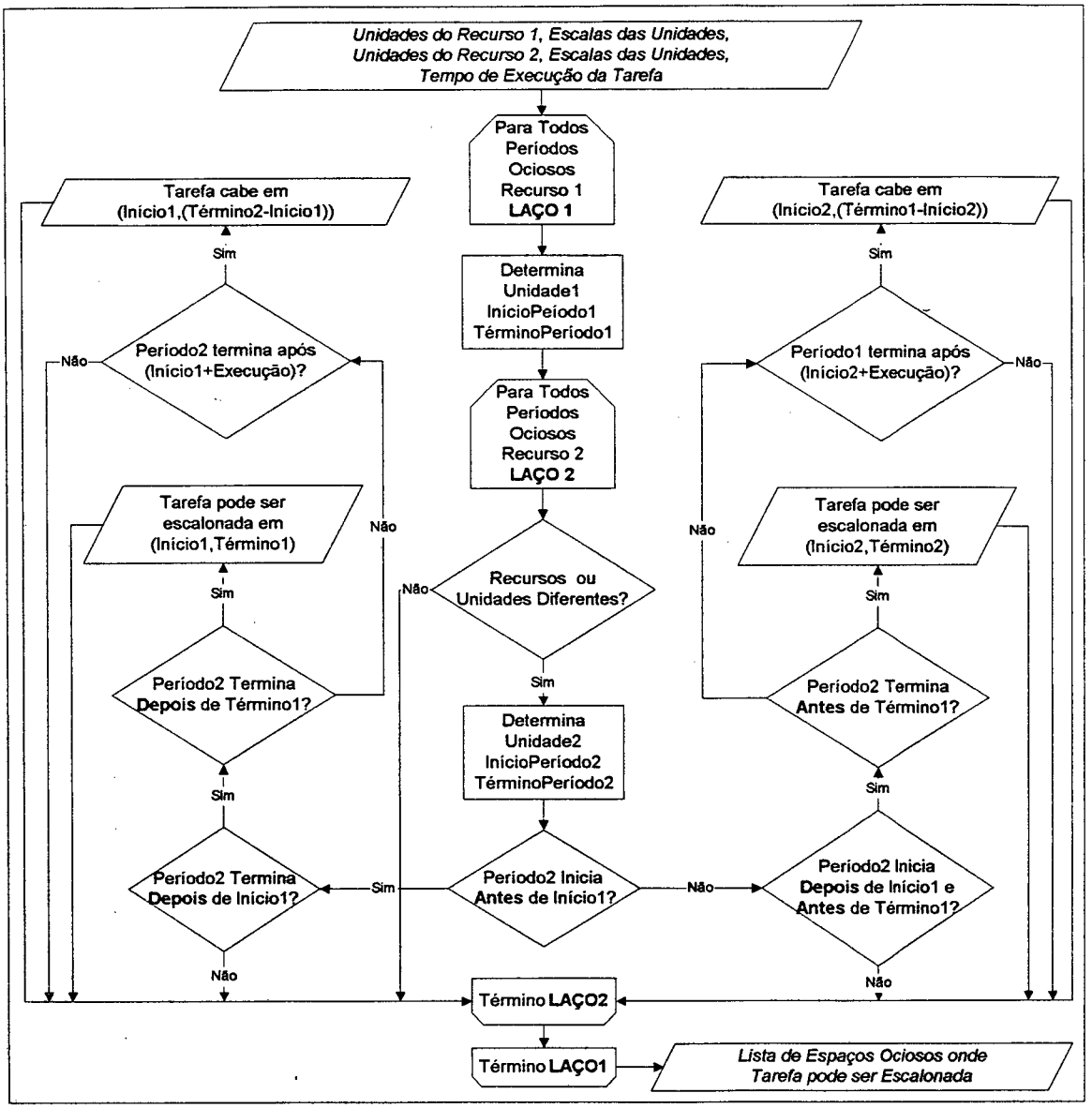

Figura Anexo B .21 - Diagrama de Fluxo - Função COMBINE.m

Referências Bibliográficas

[1] WALL, M. B. *A Genetic Algorithm for Resource-Constrained Scheduling.* Boston, 1996. Tese (Doutorado em Engenharia Mecânica) - Massachusetts Institute of Technology.

[2] CÂNDIDO, M. A. B. *A Hybrid Genetic Algorithm to Solve Real Make-to-Order Job-Shop Scheduling Problems.* Florianópolis, 1997. Tese (Doutorado em Engenharia da Produção) - Centro Técnológico, Universidade Federal de Santa Catarina.

[3] FRENCH, S. *Sequencing and Scheduling.* Chichester: Ellis Horwood Limited, 1982.

[4] UCKUN, S.; BAGCHI, S.; KAWAMURA, K. et al. Managing Genetic Search in Job-Shop Scheduling. *IEEE Expert.* p. 15-24, Out. 1993.

[5] DE JONG, K. A.; SPEARS, W. M. Using Genetic Algorithms to Solve NP-Complete Problems. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS (III.: Fairfax, Virginia). *Proceedings.* p. 124-132.

[6] DAVIS, L. Job-Shop Scheduling with Genetic Algorithms. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS (I. : 1985). *Proceedings.* 1985. p.136-140.

[7] MATURANA, F.; GU, P.; NAUMANN, A. et al. Object-Oriented Job-Shop Scheduling using Genetic Algorithms. *Computer in Industry.* v. 32, p. 281-294, 1997.

[8] HAMADA, K.; BABA, T.; SATO, K. et al. Hybridizing a Genetic Algorithm with Rule-Based Reasoning for Production Planning. *IEEE Expert.* p. 60-67, Out. 1996.

[9] NAGAR, A.; HERAGU, S.; HADDOCK, J. A Combined Branch-and-Bound and Genetic Algorithm Based Approach for a Flow Shop Scheduling Problem. *Annals of OR.* v. 63, n. 1, p. 397-414, 1996.

[10] DORNDORF, U.; PESCH, E. Evolution Based Learning in Job-Shop Scheduling Environment. *Computer Ops Res.* v. 22, n. 1, p. 25-40, 1995.

[11] CHENG, R.; GEN, M.; TSUJIMURA, Y. A Tutorial Survey of Job-Shop Scheduling Problems using Genetic Algorithms - I. Representation. *Computers in Industry and Engineering.* Great Britain, v. 30, n. 4, p. 983-997, 1996.

[12] FANG, H.; ROSS, P.; CORNE, D. A Promising Genetic Algorithm Approach to Job Shop Scheduling, Rescheduling, and Open Shop Scheduling Problems. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS (V.: 1993). *Proceedings.* San Mateo, California: Morgan Kaufmann, 1993. p. 375-382.

[13] HUSBANDS, P. Genetic Algorithms for Scheduling. *AISB Quarterly.* v. 89.

[14] MURATA, T.; ISHIBUCHI, H.; TANAKA, H. Genetic Algorithms for Flow-Shop Scheduling Problems. *Computer Ind Engng.* v. 30, n. 4, p. 1061-1071, 1996.

[15] SIKORA, R. A Genetic Algorithm for Integrating Lot-Sizing and Sequencing in Scheduling a Capacitated Flow Line. *Computer Ind Engng.* v. 30, n. 4, p. 969-981, 1996.

[16] LEE, C. Y.; CHOI, J. Y. A Genetic Algorithm for Job Sequencing Problems with Distinct Due Dates and General Early-Tardy Penalty Weights. *Computer Ops Res.* v. 22, n. 8, p. 857-869, 1995.

[17] CASKEY, K.; STORCH, R. L. Heterogenous Dispatching Rules in Job and Flow-Shop. *Production Planning and Control.* v.7, n. 4, p. 351-361, 1996.

[18] ADAMS, J.; BALAS, E.; ZAWACK, D. The Shifting Bottleneck Procedure for Job-Shop Scheduling. *Mgmt Sci.* v. 34, n. 3, p. 391-401, 1988.

[19] HOLTHAUS, O. Design of Efficient Job-Shop Scheduling Rules. *Computers Ind Engng.* v. 33, n. 1-2, p. 249-252, 1997.

[20] RUSSEL, S.; NORVIG, P. *Artificial Intelligence - A Modern Approach.* Englewood Cliffs, New Jersey: Prentice Hall, 1995.

[21] TADEI, R. DELLA CROCE, F.; MENCA, G. Advanced Search Techniques for the Job Shop Problem: a Comparison. *Operations Research.* v. 29, n. 2, p. 179-194, 1995.

[22] LOW, C. Y. Job Shop Scheduling Heuristics for Sequence-Dependent Setups. *Computers in Industry and Engineering.* v. 29, n. 1-4, p. 278-283, 1995.

[23] LOUIS, S. J.; XU, Z. Genetic Algorithms for Open Shop Scheduling and Re-Scheduling. In: ISCA CONFERENCE ON COMPUTERS AND THEIR APPLICATIONS. (XI. : 1996). *Proceedings.* p. 99-102.

[24] KUMAR, N. S. H.; SRINIVASAN, G. A Genetic Algorithm for Job-Shop Scheduling - A Case Study. *Computers in Industry.* Great Britain, v. 31, p. 155-160, 1996.

[25] SYSWERDA, G. Schedule Optimization using Genetic Algorithms. In: DAVIS, L. *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold, 1991. p. 332-349.

[26] WHITLEY, D.; STARKWEATHER, T.; SHANER, D. The Traveling Salesman and Sequence Scheduling: Quality Solutions using Genetic Edge Recombination. In: DAVIS, L. *Handbook of Genetic Algorithms.* New York: Van Nostrand Reinhold, 1991. p. 350-372.

[27] POON, P. W.; CARTER, J. N. Genetic Algorithm Crossover Operators for Ordering Applications. *Computer Ops Res.* v. 22, n. 1, p. 135-147, 1997.

[28] BEZERRA, C. D. *Evoluçâo Interativa e a Aplicação de Algoritmos Genéticos no Design de Produtos.* Florianópolis, 1996. Tese (Doutorado em Engenharia da Produção) - Centro Tecnológico, Universidade Federal de Santa Catarina.

[29] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning.* New York: Addison-Wesley Publishing Co., 1980.

[30] BAGCHI, S.; UCKUN, S.; MIYABE, Y. et al. Exploring Problem-Specific Recombination Operators for Job Shop Scheduling. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS (IV.:1991). *Proceedings.* 1991. p. 10-17.

[31] KOLEN, A.; PESCH, E. Genetic Local Search in Combinatorial Optimization. *Discrete Applied Mathematics.* v. 48, p. 273-284, 1994.

[32] BITTENCOURT, G. *Inteligência Artificial: Ferramentas e Teorias*. Florianópolis: Editora da Univerisdade Federal de Santa Catarina, 1998.

[33] PARK, L.; PARK, C. H. Application of Genetic Algorithms to Job-Shop Scheduling Problems with Active Schedule Constructive Crossover. In: IEEE INTERNATIONAL CONFERENCE ON SYSTEMS, MAN AND CYBERNETICS (1. :1995). *Proceedings*. p. 530-535.

[34] DELLA CROCE, F.; TADEI, R.; VOLTA, G. A Genetic Algorithm for the Job Shop Problem. *Computer Ops Res.* v. 22, n. 1, p.15-24, 1995.

[35] MURATA, T.; ISHIBUCHI, H.; TANAKA, H. Multi-Objective Genetic Algorithms and its Applications to Flow-Shop Scheduling. *Computer Ind Engng.* v. 30, n. 4, p. 957-968, 1996.

[36] SOARES, C. Evolutionary Computation for the Job-Shop Scheduling Problem. Disponível via FTP anônimo em ftp:\\archive.cs.ruu.nl (pub/ruu/cs/neuro/planning). Em torno de 42 páginas, 1994. Último acesso feito em 1999.

[37] LIN, S.; GOODMAN, E. D.; PUNCH, W. F. A Genetic Algorithm Approach to Dynamic Job Shop Scheduling Problems. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS (VII.: Julho 1997). *Proceedings*. San Francisco: Morgan Kaufmann, 1997. p. 481-488.

[38] LEE, C. Y.; KIM, S. J. Parallel Genetic Algorithms for the Earliness-Tardiness Job Scheduling Problem with General Penalty Weights. *Computers Ind Engng.* v. 28, p. 231-243, 1995.

[39] PARK, L.; PARK, C. H. Genetic Algorithms for Job Shop Scheduling Problems Based on Two Representational Schemes. *Electronics Letters*. v. 31, n. 23, p. 2051-2053, 1995.

[40] GOLDBERG, D. E.; LINGLE, R. Alleles, Loci, and the Travelling Salesman Problem. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS (I.: 1985). *Proceedings*. 1985. p. 154-159.

[41] BEASLEY, D.; BULL, D.; MARTIN, R. R. An Overview of Genetic Algorithms: Part 2, Research Topics. *University Computing.* v. 15, n. 4, p. 170-181, 1993.

[42] MICHALEWICZ, Z. The Significance of the Evaluation Function in Evolutionary Algorithms. In: WORKSHOP ON EVOLUTIONARY ALGORITHMS (Outubro 1996: Minneapolis, Minessota). *Proceedings*. Springer-Verlag, 1996. p. 151-166.

[43] REEVES, C. R. A Genetic Algorithm for Flow-Shop Sequencing. *Computer Ops Res.* v. 22, n. 1, p. 5-13, 1995.

[44] BEASLEY, D.; BULL, D.; MARTIN, R. R. An Overview of Genetic Algorithms: Part 1, Fundamentals. *University Computing.* v. 15, n. 2, p. 58-69, 1993.

[45] CHIPPERFIELD, A.; FLEMING, P. POHLHEIM, H. et al. *Genetic Algorithm Toolbox - For Use With Matlab*. Guia do Usuário, Versão 1.2. Departamento de Controle Automático e Engenharia de Sistemas, Universidade de Sheffield.

[46] BLICKLE, T.; THIELE, L. A Comparison of Selection Schemes Used in Genetic Algorithms. Institut für Technische Informatik und Kommunikationsnetze: *TIK-Report* n. 11, 1995.

[47] MILLER, B. L.; GOLDBERG, D. E. Genetic Algorithms, Selection Schemes, and the Varying Effects of Noise. Illinois Genetic Algorithms Laboratory: *IlliGAL Report* n. 95009, Nov. 1995.

[48] BÄCK, T. The Interaction of Mutation Rate, Selection, And Self- Adaptation Within a Genetic Algorithm. In: MÄNNER, R.; MANDERIK, B. *Parallel Problem Solving from Nature, 2*. Amsterdam: Elsevier Science Publishers, 1992. p. 85-94.

[49] MÜHLENBEIN, H.; SCHLIERKAMP-VOOSEN, D. Analysis of Selection, Mutation and Recombination Operators in Genetic Algorithms. In: *Evolution as a Computational Process, Lecture Notes in Computer Science*. Berlin: Springer-Verlag, 1995. p. 188-214.

[50] BÄCK, T.; HOFFMEISTER, F. Extended Selection Mechanisms in Genetic Algorithms. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS (IV.: 1991: San Mateo, California). *Proceedings*. San Diego, California: Morgan Kaufmann Publishers, 1991. p. 92-99.

[51] DE JONG, K. A.; SARMA, J. Generation Gaps Revisited. In: FOUNDATIONS OF GENETIC ALGORITHMS WORKSHOP (1992: Vail, Colorado). *Proceedings*. Morgan Kaufmann, 1992. p. 19-28.

[52] SPEARS, W. M. Crossover or Mutation? In: FOUNDATIONS OF GENETIC ALGORITHMS WORKSHOP (1992: Vail, Colorado). *Proceedings*. Morgan Kaufmann, 1992. p. 221-237.

[53] SPEARS, W. M. Recombination Parameters. In: BAECK, T.; FOGEL, D.; MICHALEWICZ, Z. *Handbook of Evolutionary Computation*. Oxford: Oxford University Press, 1997.

[54] DE JONG, K. A.; SPEARS, W. M. A Formal Analysis of the Role of Multi-Point Crossover in Genetic Algorithms. *Annals of Mathematics and Artificial Intelligence Journal*. v. 5, n. 1, p. 1-26.

[55] SPEARS, W. M.; DE JONG, K. A. An Analysis of Multi-Point Crossover. In: FOUNDATIONS OF GENETIC ALGORITHMS WORKSHOP (1990: Bloomington, Illinois). *Proceedings*.

[56] SPEARS, W. M.; DE JONG, K. A. On the Virtues of Parameterized Uniform Crossover. In: INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS (IV.: La Jolla, California). *Proceedings*. Morgan Kaufmann. p. 230-236.

[57] SPEARS, W. M. Adapting Crossover in Evolutionary Algorithms. In: ANNUAL CONFERENCE ON EVOLUTIONARY ALGORITHMS (IV.: San Diego). *Proceedings*.

[58] SRINIVAS, M.; PATNAIK, L. M. Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms. *IEEE Transactions on System, Man, and Cybernetics*. v. 24, n. 4, p. 656-666, 1994.

[59] SCHAFFER, J. D.; ESHELMAN, L. J. On Crossover as an Evolutionarily Viable Strategy. In: INTERNATION CONFERENCE ON GENETIC ALGORITHMS (IV.: 1991). *Proceedings*. 1991. p. 61-68.

[60] HANSELMAN, D.; LITTLEFIELD, B. *MATLAB5 - Versão do Estudante*. Guia do Usuário. São Paulo: Makron Books, 1999.

[61] CHEN, C.; NEPPALLI, R.; ALJABER, N. Genetic Algorithms Applied to the Continuous Flow-Shop Problem. *Computer Ind Engng*. v. 30, n. 4, p. 919-929, 1996.

[62] BÄCK, T. Selective Pressure in Evolutionary Algorithms: a Characterization of Selection Mechanisms. In: IEEE CONFERENCE ON EVOLUTIONARY COMPUTATION (I. : 1994). *Proceedings*. Piscataway, New Jersey: IEEE Press, 1994. p. 57-62.

[63] BIEGEL, J. E.; DAVERN, J. J. Genetic Algorithms and Job Shop Scheduling. *Computer Ind Engng. v. 19, n. 1-4, p.81-99, 1990.*

[64] GILKINSON, J. C.; RABELO, L. C.; BUSH, B. O. A Real-World Scheduling Problem Using Genetic Algorithms. *Computer Ind. Engng*. v. 29, n. 1-4, p. 177-181, 1995.

[65] BEASLEY, J. E. Operations Research Library. http://mscmga.ms.ic.uk/jeb/orlib/. Último acesso feito em Agosto, 2000.

[66] SHAW, K. J.; FLEMING, P. J. Which Line is it Anyway? Use of Rules and Preferences for Schedule Builders in Genetic Algorithms for Production Scheduling. Disponível em http://www.shef.ac.uk/uni/projects/gaipp/aisb97dld.html. Em torno de 6 páginas. Último acesso feito em Outubro, 2000.