

Escalonamento Genético FJSP com tempo de configuração dependente de sequência

Kairon Freitas Guimarães

Programa de Pós-graduação em Ciência da Computação
Faculdade de Computação
Universidade Federal de Uberlândia
Minas Gerais – Brasil

Fevereiro de 2007

Kairon Freitas Guimarães

**Escalonamento Genético FJSP com tempo de configuração
dependente de seqüência**

Dissertação apresentada ao programa de Pós-graduação em Ciência da Computação da Universidade Federal de Uberlândia, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Inteligência Artificial.

Orientadora: Profa. Dra. Márcia Aparecida Fernandes

UBERLÂNDIA – MINAS GERAIS – BRASIL
Universidade Federal de Uberlândia - UFU
Faculdade de Computação
Programa de Pós-graduação em Ciência da Computação
Fevereiro 2007

Dados Internacionais de Catalogação na Publicação (CIP)

G963e Guimarães, Kairon Freitas, 1975-

Escalonamento Genético FJSP com tempo de configuração dependente
de seqüência / Kairon Freitas Guimarães. - 2007.

118 f. : il.

Orientadora: Márcia Aparecida Fernandes.

Dissertação (mestrado) – Universidade Federal de Uberlândia, Progra-
ma de Pós-Graduação em Ciência da Computação.

Inclui Bibliografia.

1. Inteligência Artificial – Teses. 2. Algoritmos genéticos – Teses. I.
Fernandes, Márcia Aparecida. II. Universidade Federal de Uberlândia.
Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDU:681.3:007.52

Elaborado pelo Sistema de Bibliotecas da UFU / Setor de Catalogação e Classificação

Kairon Freitas Guimarães

**Escalonamento Genético FJSP com tempo de configuração
dependente de sequência**

Dissertação apresentada ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de Uberlândia, para obtenção do título de Mestre.

Área de concentração: Inteligência Artificial.

Banca Examinadora:

Uberlândia, 08 de Fevereiro de 2007.

Profa. Márcia Aparecida Fernandes, Dra. – UFU (presidente)

Prof. Geraldo Robson Mateus, Ph.D. – UFMG

Prof. Keiji Yamanaka, Ph.D. – UFU

*Dedico este trabalho à minhas filhas e esposa e
também a minha mãe e a meu irmão Artelho.
Que Deus possa protegê-los sempre.*

Escalonamento Genético FJSP com tempo de configuração dependente de seqüência

Kairon Freitas Guimarães

Resumo

Job Shop Problem é considerado um difícil problema de otimização combinatória. Apesar da diversidade de métodos de solução e da evolução da tecnologia dos processadores, problemas de escalonamento ainda são considerados difíceis, devido à natureza combinatorial, o que o caracteriza como sendo um problema NP-Completo. Este trabalho apresenta uma proposta baseada em algoritmos genéticos para problemas Flexible Job Shop Problem, uma extensão de JSP, tendo como principais aspectos o tempo de configuração dependente da seqüência e otimização multi-critério. São vários os ambientes onde surgem a necessidade de realização de atividade de escalonamento ou sequenciamento. Ambientes de *workflow* freqüentemente apresentam a necessidade de sequenciamento das atividades a serem executadas por cada recurso. Neste sentido, este trabalho também apresenta a integração de escalonamento FJSP a *workflow* através de uma arquitetura para modelagem de processos denominada *Workflow* Genético, que serve como uma ferramenta para auxiliar na definição ou otimização de modelos de processos.

Palavras-chave: *FJSP, tempo de configuração dependente de seqüência, Algoritmos Genéticos, Workflow.*

Escalonamento Genético FJSP com tempo de configuração dependente de seqüência

Kairon Freitas Guimarães

Abstract

A Job Shop Problem (JSP) may be considered a hard combinatorial optimization problem. Although the various problem solution methods and processors technological evolutions, scheduling problems still figure out as hard to solve problem, mainly due its combinatorial nature which characterize them as NP-Complete Problems. This work present a genetic algorithm method to solve Flexible Job Shop Problem, an extension of JSP, which major aspects involves the separable sequence-dependent setup time and multi-objective optimization. There are several environments where the need of the best scheduling or sequencing of activities exists. Workflow environments frequently present this need of optimal performance for activity sequencing for each resource. In that way, this work also presents the integration of scheduling FJSP to a workflow through a process modeling architecture know as Genetic Workflow, a tool that may assist the definition or optimization of process modeling.

Keywords: Flexible Job-Shop Scheduling, Genetic Algorithms, Separable Sequence-Dependent Setup Time, Workflow.

Copyright

© Kairon Freitas Guimarães, 2006

Todos os direitos reservados.

Agradecimentos

Obrigado à Deus, pela vida, força e providências diárias.

Agradeço à minha família, pelo amor e pela paciência presente em tantos momentos em que estive ausente realizando este trabalho.

À meus pais, por tudo o que fizeram e ainda fazem por mim. Sei que não conseguirei retribuir todo o amor, dedicação e auxílio.

À Profa. Márcia, pela orientação, auxílio e presença nos momentos mais difíceis de nosso trabalho.

À Universidade Federal de Uberlândia, pela oportunidade e estrutura para realização deste trabalho.

”Devemos sempre fazer o melhor que pudermos. Esta é nossa responsabilidade sagrada como seres humanos”.

Albert Einstein.

”Não nos cansemos de aprender, entendendo que o progresso da alma é infinito, no espaço e no tempo”.

Emmanuel

”O universo é completamente balanceado e em perfeita ordem. Você sempre será compensado, na exata medida, por tudo o que fizer”.

Brian Track.

”Não é a posição que exalta o trabalhador, mas sim o comportamento moral com que se conduz dentro dela”.

André Luiz.

Sumário

Resumo	vi
Abstract	vii
Agradecimentos	ix
Lista de Abreviaturas	xiii
1 Introdução	1
1.1 Objetivos	2
1.2 Organização	3
2 Conceitos de Escalonamento	5
2.1 Classificação dos problemas de Escalonamento	6
2.1.1 Notação de Graham	7
2.1.2 Quanto ao Ambiente e à Natureza do escalonamento	10
2.2 Job Shop e Flexible Job Shop	11
2.2.1 Tempo de configuração	13
2.3 Formas de representação do Escalonamento	16
3 Abordagens para solução JSP	19
3.1 Baseados em AGs	20
3.1.1 Elementos de Algoritmos Genéticos	22
3.1.2 Aspectos de AG para JSP	28
3.2 Busca Tabu	35
3.3 Recozimento Simulado	37
3.4 Branch and Bound	39

4	Escalonador Genético para FJSP	42
4.1	Formulação do Problema	43
4.2	Elementos do Escalonador	46
4.2.1	Codificação da Solução	47
4.2.2	Geração da População Inicial	48
4.2.3	Operadores Genéticos	53
4.2.4	Sequenciamento das operações	60
4.3	Resultados e Discussões	61
5	Aplicação de Escalonamento a ambientes de <i>Workflow</i>	66
5.1	Conceitos de <i>Workflow</i>	67
5.2	Workflow e Escalonamento	72
5.3	Workflow Genético	73
5.3.1	Simulador Genético	76
5.4	Resultados e Discussões	79
6	Conclusão e Trabalhos Futuros	83
6.1	Trabalhos Futuros	84
	Apêndice	91
A	Problemas FJSP	91
A.1	Problemas	91
	Apêndice	97
B	Algoritmo de Associação	97
B.1	<i>AssignmentProcedure</i>	97

Lista siglas, acrônimos e abreviaturas

AG	<i>Algoritmo Genético</i>
AL	<i>Approach by Localization</i>
BT	<i>Busca Tabu</i>
BTSL	<i>Balancing Task Sequencing List</i>
BWMC	<i>Balancing Workload Machine Crossover</i>
BWMM	<i>Balancing Workload Machine Mutation</i>
CDR	<i>Composite Dispatching Rules</i>
CE	<i>Computação Evolutiva</i>
FIFO	<i>First In First Out</i>
FJSP	<i>Flexible Job Shop Problem</i>
JSP	<i>Job Shop Problem</i>
LB	<i>Lower Bound</i>
LIFO	<i>Last In First Out</i>
LPT	<i>Longes Processing Time</i>
OGM	<i>Organismos Geneticamente Modificados</i>
PDDL	<i>Planning Domain Definition Language</i>
PFJSP	<i>Parcial Flexible Job Shop Problem</i>
POX	<i>Precedence Preserving Order Based Crossover</i>
PPS	<i>Precedence Preserving Shift Mutation</i>
SDST	<i>Sequence-Dependent Setup Time</i>
SGWf	<i>Sistema de Gerência de Workflow</i>
SPT	<i>Shortest Processing Time</i>

TFJSP	<i>Total Flexible Job Shop Problem</i>
TSL	<i>Task Sequencing List</i>
TSP	<i>Travelling Salesman Problem</i>
TST	<i>Tempo de Setup Total</i>
UB	<i>Upper Bound</i>
XPDL	<i>XML Process Definition Language</i>
WES	<i>Workflow Enactment Service</i>
WfMC	<i>Workflow Management Coalition</i>
WPDL	<i>Workflow Process Definition Language</i>

Lista de Figuras

2.1	Exemplo de tempo de configuração	14
2.2	Representação por Gráfico de Gantt	16
2.3	Relacionamento entre as propriedades dos Schedules	17
3.1	Fluxo geral de AG	22
3.2	Exemplo de roleta de Seleção	24
3.3	Operador Cruzamento	25
3.4	Exemplo de mutação em representação binária	26
4.1	Representação da Solução	48
4.2	Representação por TSL	48
4.3	Pais selecionados para cruzamento POX	54
4.4	Início da geração dos descendentes pelo operador POX	54
4.5	Descendentes gerados pelo POX	54
4.6	Indivíduo selecionado para mutação PPS	55
4.7	Descendente gerado pela mutação PPS	55
4.8	Indivíduos selecionados para o Cruzamento por Associação	56
4.9	Descendentes gerados pelo Operador de Cruzamento por Associação	56
4.10	Indivíduo selecionado para Mutação por Associação	56
4.11	Descendente gerado pela Mutação por Associação	56
4.12	Operador BWMM	59
4.13	Indivíduos selecionados para o cruzamento BWMC	60
4.14	Descendentes gerados por BWMC	60
5.1	Processos de negócio automatizados por sistemas de workflow	68
5.2	Relação entre as principais funções de um SGWf	69

5.3	Tipo de rotas em um fluxo de trabalho	70
5.4	Relação entre os termos tarefa, caso, item de trabalho e atividade . .	71
5.5	Arquitetura do <i>Workflow Genético</i>	75
5.6	Modelo preliminar gerado pelo Modulo Genetico	80
5.7	Modelos finais extraídos pelo Analisador	81
5.8	Casos obtidos pelo Gerador Casos a partir de M1	81

Lista de Tabelas

2.1	Job Shop tradicional	12
2.2	T-FJSP	13
2.3	Representação por tabela	17
4.1	Tabela T com tempos de processamento	48
4.2	Comparação de resultados da otimização multi objetivo	62
4.3	C_{max} sem tempo de configuração	62
4.4	P-FJSP Valores para makespan sem tempo de configuração	63
4.5	Resultados com tempo de configuração	64
4.6	Parâmetros Genéticos	65
5.1	Mapeamento entre os conceitos básicos de <i>workflow</i> e escalonamento	72
5.2	Especificação das atividades do repositório	79
5.3	Tempos de processamento	81
5.4	Análise Resultado	82
A.1	Problema 4x5	91
A.2	Problema 10x7	92
A.3	Problema 10x10	93
A.4	Problema 15x10	94
A.5	Problema 15x10	95
A.6	Setup 4x5	96
B.1	T' original	99
B.2	T' após primeira iteração	99
B.3	T' após segunda iteração	99
B.4	T' após terceira iteração	100

B.5	T' após quarta iteração	100
B.6	T' após quinta iteração	100
B.7	T' após sexta iteração	100
B.8	T' após sétima iteração	100

Capítulo 1

Introdução

Os problemas de escalonamento estão amplamente difundidos em uma grande diversidade de atividades industriais ou tecnológicas e têm adquirido um caráter notório, mediante as exigências de aumento da produção de bens. Nesse aspecto, pequenas falhas podem desencadear desde pequenos prejuízos de ordem cronológica até a paralisação completa de todo um setor.

Escalonamento é um processo de tomada de decisão que se preocupa com a alocação de recursos limitados para tarefas ao longo do tempo, e possui como meta a otimização de uma ou mais funções objetivo, conforme apresentado por Pinedo [45]. É um processo presente na maioria dos sistemas de produção e manufatura, bem como na maioria dos sistemas de processamento de informações tendo aplicação em diversas áreas tais como, sistemas de manufatura flexíveis, planejamento de produção, projeto de computadores, processamento de informação, logística, comunicações, entre outros.

Os recursos e tarefas podem apresentar uma grande variedade de formas. Os recursos podem ser máquinas em um chão de fábrica, pistas em um aeroporto, processadores em um sistema de computação, entre outros. As tarefas podem ser operações em um processo de produção, pousos e decolagens em pistas de um aeroporto, execução de software em um sistema de computação, entre outros. As funções objetivo também podem ser diversas, como a minimização do tempo médio gasto por operações de montagem de peças em máquinas de uma linha de produção ou a minimização do número de pousos e decolagens atrasadas em um aeroporto. Em geral, uma tarefa é chamada de *job* e possui uma sequência de operações, e um

recurso é chamado de máquina.

Apesar da diversidade de métodos de solução e da evolução da tecnologia dos processadores, problemas de escalonamento ainda são considerados difíceis, devido à natureza combinatorial. Além disso, podem apresentar restrições, aumentando assim esta dificuldade. A sua ampla utilização em problemas do mundo real, justifica o grande número de pesquisas já realizadas e mantém o interesse em desenvolver técnicas de resolução que consigam boas soluções e com tempo computacional adequado.

Ao longo do tempo, os ambientes para aplicação de escalonamento evoluíram e introduziram novas características que se por um lado aumentam a eficiência e flexibilidade, por outro tornam o problema ainda mais difícil. Dentre estas características destaca-se a possibilidade de uma mesma operação ser realizada por mais de um recurso, ou a necessidade de considerar tempos de configuração entre as execuções de operações consecutivas em uma mesma máquina.

O tempo de configuração é outra característica importante que pode ser introduzida em JSP (Job Shop Problem) e FJSP (Flexible Job Shop Problem) e torná-los problemas mais gerais. Allahverdi et al. [2] argumentam que o tempo de configuração tem sido negligenciado ou considerado parte do tempo de processamento de cada operação, mas que em muitos problemas do mundo real é necessário considerar a sua existência separadamente do tempo de processamento.

1.1 Objetivos

São várias as classes de problemas de escalonamento, tais como Máquina Unica, Máquinas em Paralelo, Flow-Shop, JSP e FJSP. Há também uma diversidade de propostas no sentido de obter soluções aproximadas, baseadas nas mais diferentes técnicas e teorias, tais como, Programação Inteira, Busca Tabu, Recozimento Simulado, Algoritmos Branch and Bound, Algoritmos Genéticos, entre outras.

Este trabalho apresenta uma proposta para FJSP com tempo de configuração dependente de sequência através de AGs híbridos. FJSP é uma extensão de JSP, desde que permite cada operação ser processada em mais de uma máquina. Esta flexibilização introduz duas novas dificuldades para o JSP clássico: associar cada operação a uma máquina apropriada e, determinar a sequência de operações em

cada máquina [28].

As pesquisas [11, 12, 15, 19, 25, 27, 32–34, 37, 41, 49, 50, 57, 59] mostram que AGs podem ser aplicados a problemas de escalonamento, em particular a problemas JSP e FJSP, por apresentar bons resultados em problemas de otimização combinatorial. Assim, o objetivo deste trabalho é apresentar uma proposta baseada em AG para FJSP com tempo de configuração dependente de seqüência. O AG proposto combina operadores genéticos e heurísticas para introduzir conhecimento específico do problema e, também técnicas de geração da população inicial, que visam melhorar a convergência e a qualidade das soluções geradas.

Recentemente, técnicas de escalonamento têm sido aplicadas a problemas de *workflow* como, por exemplo, Tramontina [51] que apresenta pontos de sinergia entre as duas áreas, que se explorados podem melhorar a eficiência dos ambientes de *workflow*, e em Tramontina et al. [6] têm-se uma proposta para a integração de JSP a *workflow*. Entretanto, neste trabalho é considerada a utilização de FJSP, pelo fato de uma mesma operação ser executada por mais de um recurso, uma característica importante e muito freqüente em ambientes de *workflow* e que não é considerada em um JSP.

Assim, este trabalho apresenta também a aplicação e a integração do escalonamento proposto ao Workflow Genético, Alves et al. [3] e Guimarães et al. [27], uma arquitetura para geração automática de modelos de processos em *workflow*, que pode ser utilizada como uma ferramenta para auxiliar na definição ou melhoria de modelos de processos a serem executados em sistemas SGWf (Sistema de Gerência de Workflow).

1.2 Organização

Este trabalho está organizado da seguinte forma. O Capítulo 2 caracteriza os problemas JSP e FJSP e apresenta os conceitos fundamentais da teoria de escalonamento. No Capítulo 3, são apresentadas algumas técnicas para solução de problemas JSP e FJSP, enfatizando a utilização de AGs. Assim, uma visão geral desta técnica também será apresentada. O Capítulo 4 descreve o AG proposto para FJSP com tempo de configuração dependente de seqüência, apresentando detalhes do desenvolvimento, análise e discussão dos resultados.

O Capítulo 5 explora a utilização de técnicas de escalonamento aplicadas a ambientes de *workflow* utilizando Workflow Genético, uma arquitetura para geração automática de modelos de processo. Para isso, é realizada uma contextualização de ambientes de *workflow* apresentando os principais conceitos e dificuldades envolvidas. Ao final deste capítulo são apresentados os resultados desta integração. Finalmente, no Capítulo 6, apresenta-se a conclusão, identificando aspectos a serem explorados em trabalhos futuros.

Capítulo 2

Conceitos de Escalonamento

Problemas de escalonamento estão presentes na maioria dos sistemas de produção e manufatura, bem como na maioria dos sistemas de processamento de informações, tendo aplicação em diversas áreas tais como, sistemas de manufatura flexíveis, planejamento de produção, projeto de computadores, processamento de informação, logística, comunicações, entre outros.

Problemas de escalonamento, em geral, consideram um conjunto de tarefas que devem ser executadas em um conjunto de máquinas, sendo seu principal objetivo determinar a atribuição destas tarefas às máquinas de forma a otimizar alguma medida de desempenho. Uma tarefa é chamada de *job* e um recurso de máquina, sendo que em geral, um *job* representa um processo com um conjunto de operações distintas, identificáveis, com pontos de início e término bem definidos e que juntas atingem um objetivo.

A solução de um problema de escalonamento consiste na associação das tarefas as máquinas, distribuídas ao longo do tempo, de forma que otimize a função objetivo desejada. Identificar uma atribuição ótima é uma tarefa difícil, pois o número de atribuições possíveis cresce rapidamente à medida que se aumenta o número de tarefas e máquinas, sendo que para n tarefas há $n!$ ordens diferentes de se executar as tarefas e, considerando ainda m máquinas ($m > 1$), o número de atribuições possíveis é dado por $(n!)^m$.

Este capítulo apresenta os conceitos fundamentais da teoria de escalonamento e estabelece os aspectos técnicos para o restante deste trabalho. Assim, a seção 2.1 apresenta classificações para problemas de escalonamento, em especial a classificação

que utiliza a notação de Graham. A seção 2.2 descreve problemas de escalonamento do tipo JSP e FJSP, delineando também considerações sobre tempos de configuração e, finalmente, na seção 2.3 tem-se as formas de apresentação de uma solução do problema de escalonamento.

2.1 Classificação dos problemas de Escalonamento

Há diversas classificações para caracterizar problemas de escalonamento, devido ao grande número de aspectos a serem considerados. Por isso, uma classificação deve ser tão abrangente quanto possível, a fim de absorver senão todos, mas pelo menos a maioria destes aspectos. A partir de uma tal classificação é possível antever direções e perspectivas das pesquisas conduzidas na área, assim como oferecer uma visão geral das principais abordagens e paradigmas propostos.

Por exemplo, Vaca [52] apresenta uma revisão das classificações utilizadas para caracterizar os problemas de escalonamento, baseada em aspectos gerais dos problemas e em hipóteses sobre os *jobs*, máquinas e tempos de processamento. Além disso, considera um esquema que classifica os problemas de escalonamento de acordo com restrições tecnológicas dos *jobs*, critério de programação, ambiente de escalonamento, natureza do escalonamento e complexidade de processamento.

Maccarthy et. al [39] apresentam um esquema de classificação, criado por Conway, baseado em quatro descritores que consideram aspectos como o número de jobs, número de máquinas, tipo de fluxo e de restrições tecnológicas e de administração, além do critério a ser otimizado. Por exemplo, $n/m/J/C_{max}$ refere-se ao problema *job shop* com n jobs e m máquinas, cujo objetivo é minimizar o tempo total de processamento (*makespan*).

Lenstra [31] introduziu um parâmetro λ ao terceiro descritor desta classificação para representar algumas variações relevantes relativas ao tipo de fluxo. Uma classificação mais abrangente é apresentada por Graham et. al [26]. Ainda que similar às classificações anteriores e utilizando o esquema de descritores, permite considerar a variedade de aspectos presentes em problemas de escalonamento. Assim, a partir desta notação é possível identificar as diferentes classes de escalonamento de uma forma simples e, por isso, será vista com mais detalhes a seguir. Além disso, são apresentadas as classificações quanto ao ambiente e à natureza do escalonamento

apresentados por Vaca [52] e que serão utilizadas no decorrer deste trabalho.

2.1.1 Notação de Graham

Esta notação baseada em três termos $\alpha/\beta/\gamma$, é descrita em Pinedo [45], onde os valores para cada um dos descritores são definidos em função dos conceitos abaixo.

- **máquinas:** número de máquinas disponíveis para executar os *jobs* e o índice k representa uma máquina deste conjunto.
- **jobs:** quantidade de *jobs* a ser processada, considerada finita e representada por n , onde o índice j indica um elemento do conjunto de *jobs*.
- **alocação de máquina:** o par (j,k) refere-se ao processamento do job j na máquina k .
- **tempo de processamento (processing time)($p_{j,k}$):** tempo para executar o job j na máquina k ;
- **data de entrega (due date)(d_j):** tempo em que o job deve ser concluído;

Termo α

O descritor α descreve o número de máquinas e o tipo de fluxo a ser utilizado para processamento dos jobs, podendo conter um único valor. Abaixo é apresentada uma descrição dos valores permitidos para este descritor, conjuntamente com a descrição dos problemas associados ao valor.

- **máquina única (single machine)(1):** ambiente que possui uma única máquina, é um caso especial de todos os outros ambientes mais complexos;
- **máquinas idênticas paralelas (identical machines in parallel)(P_m):** denota que existem m máquinas idênticas em paralelo, o job j requer uma operação apenas, e pode ser processado em qualquer uma das máquinas;
- **máquinas paralelas com diferentes velocidades (machines in parallel with different speeds)(Q_m):** existem m máquinas em paralelo, mas estas diferem na velocidade que podem executar os jobs;

- **máquinas não relacionadas em paralelo (unrelated machines in parallel) (Rm)**: uma generalização do ambiente anterior, onde existem m máquinas diferentes em paralelo;
- **flow shop(Fm)**: existem m máquinas em série e cada *job* deve ser processado em todas elas tendo a mesma “rota”, ou seja, todos devem ser processados primeiro na máquina 1, depois na 2 e assim por diante;
- **flexible flow shop(FFs)**: é uma generalização do flow shop e do ambiente com máquinas paralelas, agora considerando s estágios em série e em cada um há um número de máquinas em paralelo. Cada job deve ser processado primeiro no estágio 1, depois no estágio 2 e assim por diante. Geralmente, cada job requer apenas uma máquina e pode ser processado por qualquer uma delas, dentro de um estágio;
- **open shop(Om)**: cada job deve ser processado novamente em cada uma das m máquinas, no entanto, alguns desses tempos de execução podem ser zero e não há restrições quanto ao percurso dos jobs no ambiente das máquinas, sendo que o escalonador pode tomar decisões quanto a esse aspecto;
- **job shop(J_m)**: as operações de cada *job* são executadas em uma seqüência específica de máquinas, ou seja, há um fluxo próprio ou rota através das máquinas para cada *job*.
- **flexible job shop(FJm)**: uma extensão do *job shop* por permitir que as operações de um job possam ser executadas em mais de uma máquina. Desta forma, os *jobs* não apresentam uma seqüência determinada de máquinas.

Termo β

O descritor β indica as características de processamento dos *jobs* ou alguma restrição sobre os *jobs*, podendo conter nenhum, apenas um, ou múltiplos valores. A seguir são apresentados alguns dos possíveis valores para o termo β .

- **data de liberação(r_j)**: quando os jobs tem uma data mínima para que sua execução seja iniciada, isso indica que o job j não pode começar seu processamento antes de data de liberação r_j ;

- **tempo de configuração dependente de seqüência (sequence dependent setup times)(s_{ij})**: o símbolo s_{ij} indica que o tempo de configuração da máquina é dependente da seqüência de execução das operações i e j , ou seja, haverá diferentes tempos de configuração para a máquina k dependendo da ordem na qual as operações sejam executadas.
- **interrupção(preemptions)($prmp$)**: diz respeito à possibilidade de se interromper a execução de um job antes do término. O tempo já processado do job não é perdido e, quando retorna a execução, o *job* permanece na máquina apenas o tempo restante necessário para conclusão.
- **restrições de precedência (precedence constraints)($prec$)**: indica que existe uma relação de precedência entre os jobs, de forma que um ou mais jobs devem ser executados antes que outros;
- **quebra das máquinas (breakdowns)($brkdown$)**: implica que as máquinas não estão continuamente disponíveis, sendo que o tempo de indisponibilidade de uma máquina pode ser considerado fixo ou ser modelado através de distribuições de probabilidade;

Termo γ

O descritor γ define o critério de otimização e se refere à função objetivo. Geralmente, está relacionado com a otimização de critérios como tempo de término ou prazos dos *jobs*. O tempo de término de um *job* j é denotado por C_j e corresponde ao instante de finalização do processamento do *job* j . O tempo de entrega de um *job* j é denotado por d_j e equivale ao tempo no qual o *job* deve ser concluído.

Assim, *lateness* de um *job* é dada pela Equação 2.1, que representa a diferença entre o tempo de término do *job* e o seu tempo de entrega, podendo o valor ser positivo quando o *job* está atrasado e negativo quando está adiantado.

$$L_j = C_j - d_j \quad (2.1)$$

O *tardiness* é definido conforme a Equação 2.2 e pode ser interpretado como o tempo de atraso do *job*. E, assim, não considera *jobs* que estejam adiantados.

$$T_j = \max(C_j - d_j, 0) = \max(L_j, 0) \quad (2.2)$$

A seguir, tem-se alguns exemplos de funções objetivo.

- **makespan**(C_{max}): equivale ao tempo de término do último *job* a deixar o sistema, sendo definido como: $C_{max} = \max(C_1, C_2, \dots, C_n)$, onde C_j é o tempo de término do *job* j .
- **lateness máximo**(L_{max}): mede a pior violação de tempo de entrega do sistema, sendo definida por: $L_{max} = \max(L_1, L_2, \dots, L_n)$
- **tardiness total ponderado**: soma ponderada da função de *tardiness* de cada *job*, dado pela Equação 2.3.

$$\sum_{j=1}^n w_j T_j \quad (2.3)$$

- **% tardiness**: percentual de *jobs* atrasados.

2.1.2 Quanto ao Ambiente e à Natureza do escalonamento

Entre os aspectos considerados na classificação apresentada por Vaca [52], destaca-se aqueles relativos ao ambiente e à natureza do escalonamento. O primeiro diz respeito à possibilidade de considerar eventos previstos, mas que não se sabe, a priori, se ocorrerão e em que tempo do processo de escalonamento poderão ocorrer. O segundo aspecto identifica o dinamismo do sistema.

Em relação ao ambiente de escalonamento, os problemas podem ser determinístico ou estocásticos. Um problema é determinístico, quando os valores dos parâmetros, tais como tempo de execução dos jobs em cada máquina, data de chegada, data de entrega entre outros são previamente conhecidos e exatos. Problemas estocásticos são aqueles que consideram eventos e distúrbios aleatórios, tais como, quebras de máquinas, tempos variáveis de processamento, de liberação e de entrega.

Quanto à natureza do escalonamento, os problemas de escalonamento podem ser estáticos ou dinâmicos. No primeiro caso, todos os *jobs* são conhecidos antecipadamente e o objetivo é escalonar estes jobs apenas uma vez. E, no segundo caso, o sistema não possui um conjunto de *jobs* fixos, pois estes estão constantemente

entrando e saindo, de acordo com algum processo estocástico. A cada entrada ou saída de um *job* no/do sistema, altera-se o conjunto de jobs conhecidos.

Em ambientes dinâmicos, a simples inserção de um novo *job* no sistema pode invalidar a ordem de execução obtida até aquele instante. Assim, este dinamismo pode implicar na necessidade de refazer escalonamento de forma a contemplar o novo *job*. Bierwirth et al. [8] apresenta uma proposta para problemas dinâmicos que necessitem de re-escalonamento.

2.2 Job Shop e Flexible Job Shop

JSP tem motivado o interesse de um significativo número de pesquisadores em diferentes áreas nos últimos anos, à medida que surgiram diversas aplicações importantes em problemas do mundo real, como por exemplo sistemas de manufatura, planejamento de produção, projeto de computadores e comunicações como apresentado por Cheng et. al [13].

Lenstra et al. [38] classifica JSP como um problema NP-Completo além de considerá-lo um dos problemas combinatoriais mais difíceis de se resolver. Muitas propostas distintas e baseadas em métodos como Branch and Bound, programação dinâmica, Programação Inteira, Algoritmos Genéticos e técnicas híbridas têm sido apresentadas no sentido de obter soluções aproximadas para este problema.

O problema clássico de escalonamento JSP pode ser descrito da seguinte maneira: Sejam m máquinas diferentes e n jobs diferentes para serem escalonados. Cada *job* é composto de um conjunto de operações e para cada *job* a ordem de execução destas operações em cada máquina é pré-especificada. Cada operação requer para execução uma máquina específica em um tempo de processamento fixo. As restrições presentes em um JSP clássico descritas em Bagchi [5] são enumeradas a seguir.

- Todas as operações de um *job* ocorrem sequencialmente.
- As operações são não-preemptivas, ou seja, não é permitida a interrupção de execuções.
- Nenhum *job* é processado duas vezes na mesma máquina.
- Cada *job* é executado até a conclusão, embora possa haver esperas e atrasos

durante a execução das operações.

- *Jobs* podem ser iniciados a qualquer momento, desde que não existam no problema requisitos de data de liberação.
- *Jobs* devem esperar que a próxima máquina esteja disponível para sua execução.
- Nenhuma máquina pode executar mais de uma operação ao mesmo tempo.
- Tempos de configuração para as máquinas são independentes de seqüência e inclusos no tempo de processamento.
- Existe sempre apenas uma máquina de cada tipo.
- As restrições de precedência (restrições tecnológicas) são conhecidas e imutáveis.

A tabela 2.1 apresenta um exemplo de problema JSP composto por três *jobs* e três máquinas, onde cada linha representa a seqüência das máquinas para cada operação do *job* com o respectivo tempo de processamento entre parênteses.

Tabela 2.1: Job Shop tradicional

Job	Máquina(tempo)		
1	1(3)	2(3)	3(3)
2	1(2)	3(3)	2(4)
3	2(3)	1(2)	3(1)

FJSP é uma extensão de JSP, pois permite que cada operação seja processada em mais de uma máquina. FJSP especifica uma série de *jobs* a serem processados por uma lista de máquinas. Cada operação constituinte dos *jobs* pode ser processada em um conjunto de máquinas pré-definido e com diferentes tempos de processamento.

Esta flexibilização introduz uma nova dificuldade ao problema JSP clássico, a associação de cada operação a uma máquina apropriada. Somente após esta fase é possível determinar a seqüência de execução em cada uma das máquinas, os respectivos tempos de início, como apresentado por Ho et. al [28]. Conseqüentemente, FJSP é mais apropriado que JSP para encontrar soluções aproximadas em aplicações do mundo real, nas quais é fácil observar características como permitir mais de um

recurso executar uma operação. Jansen et al. [30] argumentam que problemas FJSP são mais complexos do que problemas JSP, devido à necessidade de associar as operações às máquinas.

A atribuição dos tempos de início à cada operação em sua respectiva máquina é denominada escalonamento, ou *schedule* no termo original. O termo “escalonamento” pode gerar alguma confusão quando utilizado na língua portuguesa, já que na literatura o termo *scheduling*, quer dizer a operação de gerar um *schedule*, que poderia ser também traduzido como escalonamento. Portanto, neste trabalho, o termo original *schedule* é utilizado para se referir ao resultado do processo de escalonamento.

O problema FJSP pode ser classificado quanto ao grau de flexibilidade permitida.

- FJSP Total: qualquer operação pode ser processada por qualquer máquina do conjunto de máquinas
- FJSP Parcial: cada operação pode ser processada por um subconjunto do conjunto de máquinas

A tabela 2.2 apresenta um problema FJSP 3x4, 3 *jobs* e 4 máquinas, de ordem total, onde cada valor apresentado na tabela corresponde ao tempo de processamento de cada operação em cada máquina.

Tabela 2.2: T-FJSP		M_1	M_2	M_3	M_4
J_1	$O_{1,1}$	1	4	5	8
	$O_{2,1}$	7	5	6	5
J_2	$O_{1,2}$	2	5	6	2
J_3	$O_{1,3}$	12	5	4	7
	$O_{2,3}$	5	6	3	5
	$O_{3,3}$	2	4	12	5

2.2.1 Tempo de configuração

O tempo de configuração da máquina é o tempo necessário para preparo das máquinas, do processo ou da oficina para a fabricação de produtos. Inclui o tempo de obtenção

de ferramentas, posicionamento dos materiais a serem utilizados no trabalho, processos de limpeza, preparação e ajustes das ferramentas e inspeção de materiais, conforme apresentado por Allahverdi et. al [2], que apresenta uma revisão da literatura de problemas de escalonamento que envolvem tempos de configuração.

A maioria das pesquisas em escalonamento negligenciam ou consideram os tempos de configuração como parte do tempo de processamento das operações. Esta consideração simplifica a análise e os reflexos em certas aplicações, mas por outro lado afeta a qualidade das soluções para muitas aplicações que exigem considerações explícitas dos tempos de configuração. Tais aplicações, juntamente com conceitos de produção recentes, tais como competição baseada em tempo e tecnologia de grupo, têm motivado o interesse crescente na inclusão de considerações de tempo de configuração em problemas de escalonamento.

Considerando as aplicações que exigem um tratamento de configuração separado existem dois tipos de problema, no primeiro o tempo de configuração depende apenas da tarefa a ser processada e é conhecido como independente de seqüência (sequence-independent). O segundo é quando o tempo de configuração depende tanto da tarefa a ser processada quanto daquela que foi processada imediatamente antes, problema este conhecido como dependente da seqüência (sequence-dependent). A Figura 2.1 apresenta um exemplo da ocorrência de tempo de configuração dependente de seqüência para a execução das operações O_{11} e O_{12} na máquina M_1 . Nesta figura pode-se notar que a quantidade de tempo de configuração, varia de acordo com a ordem de execução das operações na máquina.

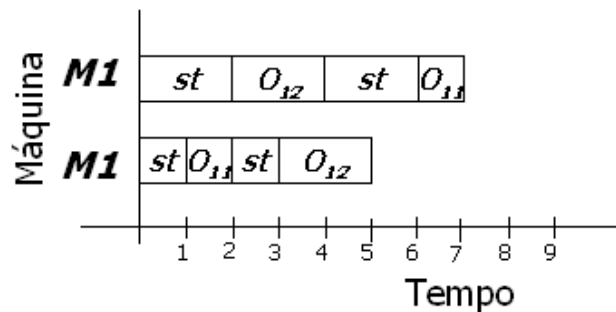


Figura 2.1: Exemplo de tempo de configuração

Diversas aplicações podem ser encontradas em vários tipos de ambientes (*shops*)

de escalonamento, principalmente em linhas de produção, serviços e processamento de informações. Um exemplo claro para esta abordagem, pode ser visto em sistemas de produção como indústrias químicas, na produção de tintas por exemplo, onde o processo de limpeza é diferenciado, dependendo da cor que estava sendo produzida e daquela que será produzida em seguida.

A importância do tempo de configuração tem sido investigado em muitos estudos. Wilbrecht et. al [55] indicam que tempos de configuração dependente de sequência são significantes quando um JSP opera próximo a capacidade total. Wortman [56] delinea a importância de considerar tempos de configuração dependentes de sequência para o efetivo gerenciamento da capacidade de fabricação.

Allahverdi et. al [2] consideraram também os diversos ambientes de fabricação como máquina única, máquinas paralelas, *flow shops* e *job shops*. Um ponto importante deste trabalho foi a constatação de que poucos trabalhos que consideram o ambiente *job shop* com tempo de configuração dependente de sequência foram desenvolvidos.

Durante o tempo de configuração as máquinas não executam operações específicas dos *jobs*, entretanto deve ser levado em consideração para o processamento dos mesmos. Para isso, é importante realizar pesquisas no sentido de identificar técnicas de sequenciamento que reduzam o tempo de configuração em cada máquina, e conseqüentemente o tempo total de configuração

Tempo de configuração foi considerado em Choi et al. [17] [16], onde um problema JSP com tempo de configuração dependente de sequência é formulado através programação inteira. O primeiro trabalho usa uma heurística de tempo polinomial e no segundo uma sequência de operações alternativas é considerada, além de um algoritmo de busca local que encontra a solução através da resolução dos sub-problemas.

Como mencionado anteriormente, considerações sobre tempo de configuração são normalmente negligenciadas, entretanto considerações sobre tempo de configuração para JSP podem ser vistas em [17], [15], [16] e [47]. Kacem et. al [47] apresentam uma alternativa para tratar problemas de uma única máquina (*single machine*) com tempo de configuração.

Vale ressaltar que as pesquisas para incluir o tempo de configuração dependente de sequência se encontram nos primeiros estágios, como foi observado por Allahverdi

et. al [2], principalmente em ambientes generalizados. Um Trabalho recente com FJSP e tempo de configuração dependente de seqüência pode ser encontrado em Philip [44], uma proposta para FJSP reentrantes.

2.3 Formas de representação do Escalonamento

O resultado de um processo de escalonamento, como dito anteriormente, é denominado *schedule*, que contém a atribuição das operações às máquinas e os respectivos tempos de início de processamento. Isto pode ser representado de algumas formas. A Figura 2.2 mostra um diagrama Gantt, uma ferramenta criada por H. L. Gantt em 1917 e que representa graficamente a ocupação das máquinas pelos jobs ao longo do tempo, indicando os tempos de início e fim das operações. Uma vantagem deste diagrama é proporcionar uma representação visual simples do que ocorre com cada operação. Neste exemplo, observa-se as atribuições das operações de um problema 3x3, 3 *jobs* e 3 máquinas.

Outra forma de representar um *schedule* é descrita na Tabela 2.3. As linhas representam as máquinas, as colunas os *jobs*, e a tabela contém o tempo de início do processamento de cada operação do job na máquina.

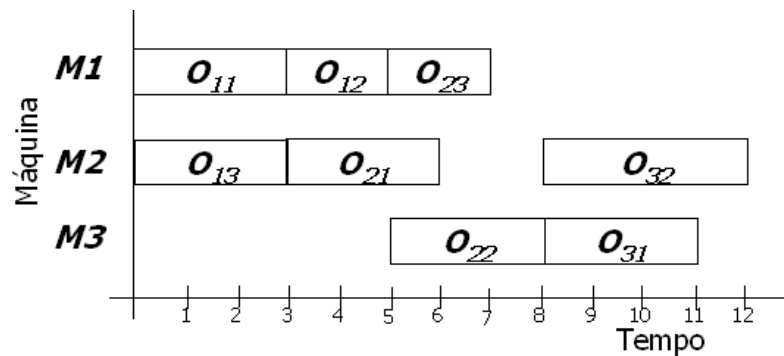


Figura 2.2: Representação por Gráfico de Gantt

Os *schedules* gerados para um problema de escalonamento podem ser de diferentes tipos dependendo de algumas características. Abaixo são apresentado os três tipos mais relevantes e que foram definidos por Pinedo [45].

- **Sem-Atraso (Non-delay):** um *schedule* onde nenhuma máquina é mantida parada enquanto existem operações disponíveis para processamento.

Tabela 2.3: Representação por tabela

	J1	J2	J3
M1	0	3	5
M2	3	8	0
M3	8	5	11

- **Semi-Ativo (Semi-active):** um *schedule* onde nenhuma operação pode ser completada mais cedo sem alterar a ordem de processamento nas máquinas
- **Ativo (Active):** nenhuma operação pode ser completada mais cedo através da alteração da ordem de processamento nas máquinas, sem que essa alteração atrase outra operação.

Schedules ativos são também semi-ativo e *schedules* sem atraso são necessariamente ativos. Estas propriedades são mostradas na Figura 2.3. Bierwirth et al. [8] argumentam que, em relação a minimização do *makespan*, sabe-se que entre os *schedules* ótimos, pelo menos um é *schedule ativo*, mas infelizmente não se pode garantir que exista pelo menos um *schedule* ótimo no conjunto dos Sem-Atraso. Entretanto, existem evidências empíricas fortes de que os *schedules* Sem-Atraso possuam uma qualidade média de solução melhor do que os Ativos. Com isso, algoritmos de escalonamento normalmente varrem o espaço de busca dos *schedules* ativos visando garantir que uma solução ótima possa ser encontrada.

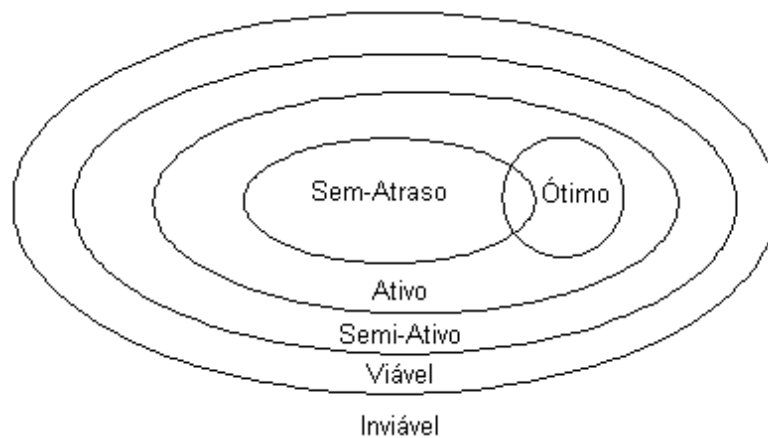


Figura 2.3: Relacionamento entre as propriedades dos Schedules

Além disso, é possível observar na Figura 2.3 que o universo de soluções contém, as soluções descritas anteriormente, assim como as soluções viáveis e inviáveis. As primeiras não violam nenhuma das restrições do problema, mas também não se adequam às classificações anteriores. As inviáveis violam uma ou mais restrições do problema.

Capítulo 3

Abordagens para solução JSP

Apesar da diversidade de métodos de solução e da evolução da tecnologia dos processadores, problemas de escalonamento continuam difíceis. Na tentativa de superar esta barreira métodos heurísticos e meta-heurísticos podem ser utilizados para encontrar boas soluções aproximadas.

Método heurístico é um processo de solução de problema apoiado em critérios racionais ou computacionais para escolher um caminho entre vários possíveis, evitando considerar todas as possibilidades para atingir a melhor opção. Esta busca por um determinado objetivo visa encontrar uma solução viável, pelo menos próxima da ótima utilizando um tempo computacional aceitável.

Meta-heurísticas são heurísticas de busca no espaço de soluções e podem ser divididas em duas classes. A primeira compreende os métodos que exploram uma vizinhança a cada iteração, alterando tanto a vizinhança quanto a forma de explorá-la de acordo com estratégias e escolhendo apenas um elemento dessa vizinhança nessa iteração. Esse tipo de varredura do espaço de soluções gera um caminho ou trajetória de soluções, obtida pela transição de uma solução para outra, de acordo com os movimentos permitidos pela meta-heurística. Dentro dessa classe de métodos pode-se citar a Busca Tabu proposta por Glover [23] e o Recozimento Simulado (*Simulated Annealing*) proposto por Kirkpatrick et. al [35] e utilizado por Najib et. al [40].

Em contraposição a esses métodos, existem aqueles que exploram uma população de soluções a cada iteração. Esses métodos constituem a segunda classe de meta-heurísticas, cujas estratégias de busca são capazes de explorar várias regiões do

espaço de soluções simultaneamente. Dessa forma, ao longo das iterações não se constrói uma trajetória única de busca pois novas soluções são obtidas através de combinação de soluções anteriores. Nessa classe estão os Algoritmos Genéticos propostos por Holland [29] e utilizados por Goldberg [25].

A literatura mostra que AGs podem ser aplicados a problemas de escalonamento por apresentar bons resultados em problemas de otimização combinatorial. Para isso deve explorar de forma inteligente o espaço de busca, evitando caminhos inúteis e explorando áreas mais promissoras.

Este capítulo apresenta algumas abordagens utilizadas na resolução de problemas JSP e FJSP, descrevendo as principais características relacionadas à representação das soluções, mas concentrando principalmente em AGs. Nas seções seguintes deste capítulo serão apresentadas as características gerais e os elementos fundamentais de AGs, e aplicações destes a problemas de escalonamento, abordando as técnicas utilizadas na representação dos indivíduos, as estratégias genéticas aplicadas a problemas de escalonamento, algoritmos genéticos híbridos que combinam heurísticas para auxiliar na exploração eficiente do espaço de busca. Além disso, algumas técnicas são mencionadas nas seções finais do capítulo.

3.1 Baseados em AGs

AG é o ramo mais conhecido da Computação Evolutiva (CE) e foram criados e desenvolvidos por John Holland na Universidade de Michigan na década de 70. O objetivo original de Holland não era projetar algoritmos para resolver problemas específicos, mas principalmente estudar formalmente o fenômeno da adaptação, conforme ocorre na natureza e desenvolver formas nas quais este mecanismo de adaptação natural pudesse ser importado para os sistemas de computador. Holland [29] apresenta AGs como uma abstração da evolução biológica e faz um esboço teórico para a adaptação através de AGs.

O AG de Holland é um método para evoluir populações de indivíduos usando um tipo de “seleção natural”, juntamente com os operadores geneticamente inspirados, cruzamento (*crossover*), mutação (*mutation*) e inversão (*inversion*). Cada indivíduo é formado por “gens” e cada gen é uma instância de um alelo particular.

Por que utilizar a evolução como uma inspiração para resolver problemas com-

putacionais? Para os pesquisadores de computação evolutiva, o mecanismo de evolução mostra-se bem adequado para muitos problemas computacionais importantes em muitos campos de pesquisa. Muitos problemas computacionais necessitam de busca através de um grande número de possibilidades para solução e também de programas de computação adaptativos para continuarem a execução, mesmo em face a mudanças no ambiente. Para outros problemas, os programas de computador devem ser inovadores para construir algo verdadeiramente novo e original, tal como um novo algoritmo para completar uma tarefa computacional, ou mesmo uma nova descoberta científica.

Fuchigami [22] classifica AG como um método meta-heurístico, por ser um procedimento que busca no espaço de soluções suportado por estratégias que exploram apropriadamente a topologia de tal espaço. Além disso, atribui o sucesso a fatores como a alusão aos mecanismos de otimização da natureza, a aplicabilidade geral da abordagem, a facilidade de implementação e, finalmente, à qualidade gerada na solução aliada a um esforço computacional relativamente baixo.

A idéia de busca em uma coleção de soluções candidatas por uma solução desejada é tão comum em Ciência da Computação que foi criado um termo específico: buscando em um “espaço de busca”.

Não há uma definição rigorosa para AG que o diferencia de outros métodos de computação evolutiva. Entretanto, pode-se dizer que a maioria dos métodos chamados de AGs têm em comum as características propostas por Holland: população de indivíduos, seleção de acordo com a aptidão (*fitness*), cruzamento e mutação aleatória para produzirem novos descendentes. *Inversion*, o terceiro operador definido por Holland é raramente utilizado nas implementações atuais. Os AGs mais comuns freqüentemente exigem uma função de aptidão que associa uma pontuação (aptidão) para cada indivíduo na população corrente. A aptidão do indivíduo depende de quão bem este cromossomo soluciona o problema.

AGs são algoritmos de otimização global, que empregam uma estratégia de busca paralela e estruturada, mas aleatória, no sentido de reforçar a busca em pontos de “alta aptidão”, ou seja, pontos nos quais a função a ser minimizada (ou maximizada) tem valores relativamente baixos (ou altos).

Apesar de aleatórios, não são buscas aleatórias não direcionadas, pois exploram

informações históricas para encontrar novos pontos de busca onde são esperados melhores desempenhos. Isto é feito através de processos iterativos, onde cada iteração é chamada de geração. Embora sejam computacionalmente simples, são bastante poderosos. Além disso, não são limitados por suposições sobre o espaço de busca relativas à continuidade e existência de derivadas, sendo assim, podem tratar problemas de domínios abrangentes, onde descontinuidade e ruídos ocorrem com frequência. A figura 3.1 apresenta a estrutura geral de um AG.

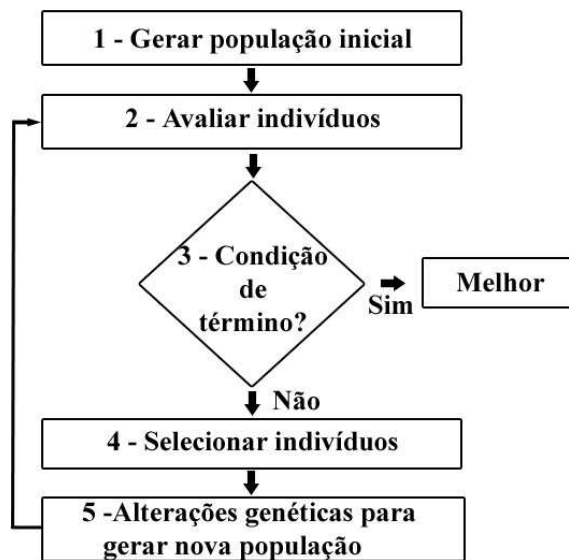


Figura 3.1: Fluxo geral de AG

3.1.1 Elementos de Algoritmos Genéticos

Em geral, um AG é especificado como descrito no Algoritmo 1. Inicialmente é gerada uma população inicial formada por um conjunto aleatório de indivíduos que podem ser vistos como possíveis soluções do problema. Durante o processo evolutivo, esta população é avaliada e cada indivíduo recebe um índice de aptidão, refletindo sua habilidade de adaptação a determinado ambiente. Um algoritmo genético mantém uma população de indivíduos $P(t) = \{x_1, \dots, x_n\}$ na iteração (geração) t . Cada indivíduo representa um candidato à solução do problema e, em qualquer implementação computacional, assume a forma de alguma estrutura de dados. Cada solução x_i é avaliada e produz alguma medida de adaptação.

Alguns indivíduos da população são submetidos a um processo de alteração por

Algoritmo 1 Procedimento Algoritmo Genético

```
1:  $t \leftarrow 0$ 
2: Inicia População  $P(t)$ 
3: Avaliação  $P(t)$ 
4: while não condição parada faça
5:    $t \leftarrow t + 1$ 
6:   Seleciona os pais  $P(t)$ 
7:   Cruzamento  $P(t)$ 
8:   Mutação  $P(t)$ 
9:   Avaliação  $P(t)$ 
10:  Sobrevivem  $P(t)$ 
11: fim while
```

meio de operadores genéticos para formar novas soluções. Então, uma nova população é formada na iteração $t + 1$ pela seleção dos indivíduos mais adaptados. Após um número de gerações, a condição de parada deve ser atendida, indicando a existência de um ou mais indivíduos de alta aptidão na população, ou quando o número máximo de gerações for atingido.

Mecanismos de Seleção

O princípio básico de AGs é que a partir de um critério de seleção, após muitas gerações, o conjunto inicial de indivíduos evoluirá para indivíduos mais aptos. A maioria dos métodos de seleção são projetados para escolher preferencialmente indivíduos com maiores valores de aptidão, embora não exclusivamente, a fim de manter a diversidade da população.

Há vários métodos de seleção utilizados para escolher indivíduos da população entre eles Truncamento, Ranking, Amostragem Universal Estocástica e Torneio. O método mais utilizado entretanto é o Método da Roleta, onde indivíduos de uma geração são escolhidos para fazer parte da próxima geração, através de um sorteio de roleta.

Neste método, cada indivíduo da população é representado na roleta proporcionalmente ao seu índice de aptidão. Assim, aos indivíduos com alta aptidão são dadas porções maiores da roleta, enquanto aos de aptidão mais baixa são dadas

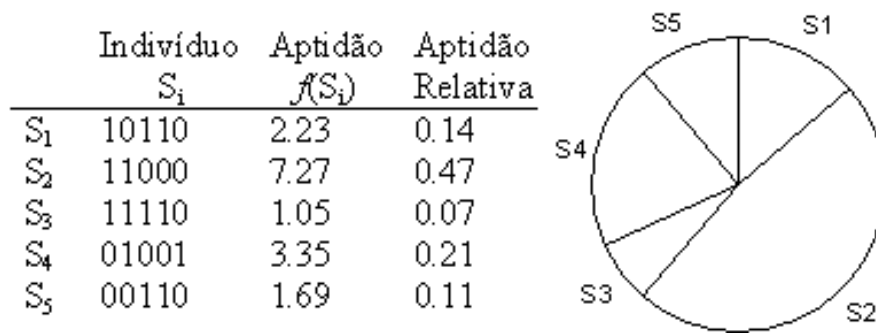


Figura 3.2: Exemplo de roleta de Seleção

porções menores da roleta. Finalmente, a roleta é girada um determinado número de vezes, dependendo do tamanho da população, e são escolhidos, como indivíduos que participarão da próxima geração, aqueles sorteados na roleta. A figura 3.2 mostra uma representação deste método.

Operadores Genéticos

O objetivo dos operadores genéticos é transformar a população através de sucessivas gerações, extendendo a busca até obter um resultado satisfatório. Os operadores genéticos são necessários para que a população se diversifique e mantenha características de adaptação adquiridas pelas gerações anteriores.

Existem transformações unárias m_i (mutação) que criam novos indivíduos através de pequenas modificações de atributos em um indivíduo ($m_i : S \rightarrow S$), e transformações de ordem superior c_j (cruzamento), que criam novos indivíduos através da combinação de dois ou mais indivíduos ($c_j : S \times \dots \times S \rightarrow S$). A recombinação atua como busca local, enquanto a mutação realiza uma busca global do espaço de busca.

Após a geração e avaliação da população inicial de indivíduos, uma porcentagem dos mais adaptados é mantida, enquanto os outros são descartados. Os indivíduos mantidos pela seleção podem sofrer modificações em características fundamentais através de mutação e cruzamento, gerando descendentes para a próxima geração.

O cruzamento é o operador responsável pela recombinação de características dos pais, permitindo que as próximas gerações herdem essas características. Ele é considerado o operador genético predominante, por isso é aplicado com probabili-

dade dada pela taxa de crossover P_c , que geralmente deve ser maior que a taxa de mutação. O operador de cruzamento pode ser projetado de algumas maneiras.

- um-ponto: um ponto de cruzamento é escolhido e a partir deste ponto as informações genéticas dos pais serão trocadas. As informações anteriores a este ponto em um dos pais são ligadas às informações posteriores a este ponto no outro pai.
- multi-pontos: é uma generalização desta idéia de troca de material genético através de pontos, onde muitos pontos de cruzamento podem ser utilizados.
- uniforme: não utiliza pontos de cruzamento, mas determina, através de um parâmetro global, qual a probabilidade de cada variável ser trocada entre os pais.

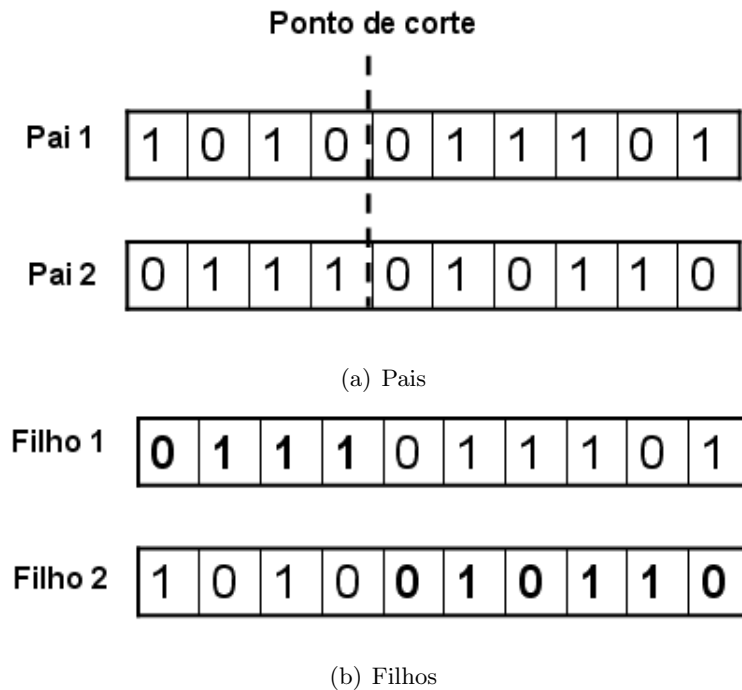


Figura 3.3: Operador Cruzamento

A Figura 3.3 apresenta o operador de cruzamento, com um ponto de corte aplicado sobre uma representação por vetor que utiliza uma codificação binária. Para a aplicação deste operador, são selecionados dois indivíduos (pais) e a partir destes são gerados dois novos indivíduos (filhos). Para tanto, seleciona-se um mesmo ponto

de cruzamento aleatoriamente nos cromossomos dos pais, e os segmentos de cromossomo até este ponto de cruzamento são trocados. Considere, por exemplo, os indivíduos *Pai 1* e *Pai 2* e suponha o ponto de cruzamento escolhido (aleatoriamente) entre as posições 4 e 5, após o cruzamento, os indivíduos *Filho 1* e *Filho 2* são gerados, conforme Figura 3.3.

A mutação é um operador genético que tem a função de introduzir características novas ao indivíduo ou mesmo restaurar características que se perderam em operações, como, por exemplo, de cruzamento. Este operador permite a introdução e manutenção da diversidade genética da população, alterando arbitrariamente um ou mais gens do indivíduo, como é ilustrado na Figura 3.4, fornecendo assim, meios para introdução de novos indivíduos na população.

Desta forma, a mutação assegura que a probabilidade de se atingir qualquer ponto do espaço de busca nunca será zero, além de contornar o problema de mínimos locais, pois com este mecanismo, altera-se levemente a direção da busca. O operador de mutação é aplicado aos indivíduos com uma probabilidade dada pela taxa de mutação P_m , geralmente se utiliza uma taxa de mutação pequena.

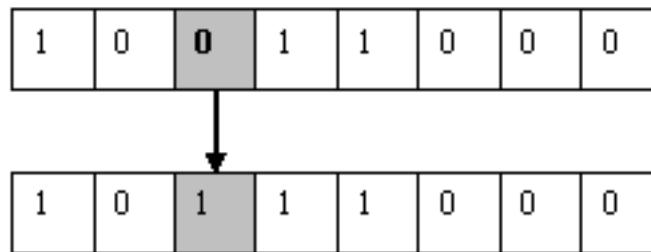


Figura 3.4: Exemplo de mutação em representação binária

Parâmetros de um AG

Os operadores de cruzamento e mutação são aplicados sobre a população de indivíduos segundo taxas de probabilidade, pois na natureza nem todos os seres cruzam e, quando o fazem, há geralmente o envolvimento de um parceiro. Portanto, mantendo a analogia com sistemas naturais, deve ser feita a seleção de dois indivíduos para o cruzamento. Escolhido o par, o cruzamento só se efetivará com uma probabilidade P_c . Em relação a mutação ocorre algo semelhante, há uma probabilidade

de ocorrência sobre cada um dos gens de cada indivíduo.

Com a evolução das gerações, indivíduos de alta aptidão começam a compartilhar partes comuns em seus cromossomos. Estas partes foram chamadas de esquemas, e o teorema fundamental dos AG's diz que esquemas que tiverem maior aptidão do que a média da população tendem a crescer exponencialmente nas próximas gerações, e esquemas com aptidões menores do que a média tendem a diminuir, também exponencialmente, isto é, as soluções convergirão para um ponto de maior aptidão.

É importante também, analisar de que maneira alguns parâmetros influem no comportamento dos AGs, para que se possa estabelecê-los conforme as necessidades do problema e dos recursos disponíveis.

- **Tamanho da População:** O tamanho da população afeta o desempenho global e a eficiência. Com uma população pequena o desempenho pode cair, pois deste modo a população fornece uma pequena cobertura do espaço de busca do problema. Uma grande população geralmente fornece uma cobertura representativa do domínio do problema, além de prevenir convergências prematuras para soluções locais ao invés de globais. No entanto, grandes populações, exigem mais recursos computacionais ou mais tempo de processamento.
- **Taxa de Cruzamento:** É responsável pela inserção de novos indivíduos na população. Quanto maior for a taxa de cruzamento, mais rapidamente serão inserido novos indivíduos na população. No entanto, se a taxa for muito elevada, a maior parte da população será substituída, e indivíduos de boas aptidões poderão ser retirados da população. No caso de uma taxa muito baixa, o algoritmo pode ficar muito lento.
- **Taxa de Mutação:** Esta taxa previne que a população fique estagnada em certas regiões do espaço de busca, além de possibilitar a descoberta ou a recuperação de bons indivíduos na população. Em geral, esta taxa é pequena. Com uma taxa muito alta a busca se torna essencialmente aleatória.

A seguir são apresentadas duas situações comuns e indesejáveis que geralmente surgem na aplicação de AGs. A primeira, chamada de “Super Indivíduo”, ocorre quando um indivíduo tem uma avaliação muito maior do que os demais. Assim, este

indivíduo poderá monopolizar as seleções e rapidamente ter-se-ão soluções praticamente idênticas (até porque descendem do mesmo ascendente). Este fenômeno também é chamado convergência precoce. Para evitar este problema, uma técnica possível é a chamada normalização linear, que coloca os indivíduos em ordem decrescente de avaliação e atribui um valor de aptidão diminuído em taxa constante para todos. Neste caso, o valor da função de avaliação serviu apenas para determinar a ordem dos indivíduos, e o valor de aptidão difere da avaliação original.

Outra situação é o surgimento de um grupo de indivíduos com avaliações muito baixas. Tais indivíduos terão pequena probabilidade de serem escolhidos para gerar descendentes. No entanto, é importante que produzam descendentes, para manter a diversidade na população. Se eliminados do processo nas primeiras gerações, a população também rapidamente tenderá para uma convergência precoce.

Se a função apresentar um único máximo (mínimo) global, a convergência precoce em si não é prejudicial. Mas no mundo real, poucas funções apresentam esta característica, pois, em geral são multimodais e, neste caso, a convergência precoce pode-se dar em um máximo (mínimo) local. Utilizando-se populações com a maior diversidade possível, aumenta-se a possibilidade de, quando se chegar a um ótimo, este ser global. A correção para este segundo problema pode ser a chamada técnica de janelamento, isto é, atribuição de um valor de aptidão mínimo aos indivíduos cujos valores de aptidão estejam abaixo de um certo valor pré definido.

Finalmente, como os AG's são estocásticos por excelência, boas soluções podem não ser escolhidas para gerar descendência. Uma maneira de evitar este fato é usar o conceito de elitismo. Por esta técnica, uma certa quantidade dos melhores indivíduos de uma geração passarão automaticamente para a próxima.

3.1.2 Aspectos de AG para JSP

Uma característica interessante de AGs é mapeamento entre o espaço de codificação e o espaço de solução. A evolução opera sobre o espaço de codificação (cromossomo) enquanto que a avaliação trabalha sobre o espaço de solução. No trabalho de Holland a codificação foi realizada usando cadeias binárias, entretanto nos últimos anos várias técnicas de codificação que não utilizam esta representação têm sido criadas para problemas específicos.

Como apresentado anteriormente JSP é um dos mais difíceis problemas de otimização da classe de problemas NP-Completo. Por ser uma otimização combinatorial, a utilização AGs é uma alternativa viável para resolução deste tipo de problema, embora AGs não garantam encontrar uma solução globalmente ótima, são um poderoso procedimento de busca, uma vez que varrem grandes espaços de busca em diferentes pontos e de forma eficiente.

Um ponto importante ao se construir um AG é planejar uma codificação apropriada do indivíduo para a solução do problema considerando aspectos específicos do problema e os operadores genéticos. Esta é uma fase crucial e que afeta todos os passos subsequentes.

Codificação

Cheng et. al [13] apresenta algumas propostas de codificações para problemas de escalonamento classificando-as dentro de dois grupos básicos, codificação diretas e indiretas.

Nas representações diretas, a solução do problema de escalonamento é codificada diretamente dentro do indivíduo, sendo que AGs são utilizados para evoluir este indivíduo para encontrar uma solução melhor. Já a representação indireta o indivíduo não representa uma solução direta para o problema, introduzindo a necessidade de um procedimento para traduzir o indivíduo em uma solução. Assim, de acordo com Cheng et. al [13], as representações podem ser conforme descritas abaixo.

- **Baseado em operação (*Operation-based*):** Esta representação codifica a solução (schedule) como uma sequência de operações e cada gene representa uma operação. Uma forma natural de identificar cada operação é utilizando um número natural, como nas representações por permutação utilizadas em problemas do Caixeiro Viajante. Entretanto, devido às restrições de precedência nem todas as permutações possíveis representam um *schedule* viável;
- **Baseado em *Job* (*Job-based*):** Consiste de uma lista de n jobs e uma schedule é construído de acordo a sequência dos jobs. Para uma dada sequência de jobs, todas as operações do primeiro job da lista são associadas, seguidas por todas as operações do segundo job e, assim por diante, de acordo com

as restrições tecnológicas de cada job. Neste tipo de representação qualquer permutação possível gera um schedule viável;

- **Lista de preferência (*Preference list-based*):** Para uma lista de n jobs e m máquinas, o cromossomo é formado de m sub-cromossomos, sendo cada um para uma máquina. Cada sub-cromossomo é uma cadeia de símbolos com tamanho n , onde cada símbolo identifica uma operação a ser processada na respectiva máquina. Os sub-cromossomos não descrevem a seqüência das operações nas máquinas, sendo apenas uma lista de preferência de cada máquina;
- **Baseado na relação de pares de *Jobs* (*Job pair relation-based*):** Nakano et. al [41] utilizaram uma matriz para codificar o schedule e a matriz é determinada de acordo com a relação de precedência de um par de jobs na máquina correspondente.
- **Regra de Prioridades (*Priority rule-based*):** Dorndorf et al. [21] propuseram um algoritmo genético baseado em regras de prioridades, onde o cromossomo é codificado como uma seqüência de regras de despacho para associação
- **Grafo Disjuntivo (*Disjunctive graph-based*):** Tamaki et. al [49] apresentaram uma codificação baseada em grafos disjuntos, que pode ser vista como um tipo de representação baseada na relação de pares de *jobs*.
- **Tempo de Término(*Completion time-based*):** Yamada et. al [57] propuseram uma representação baseada no tempo de término, onde o cromossomo é uma lista de operações ordenada pelos tempos de término.
- **Baseado em Máquinas (*Machine-based*):** Foi apresentada por Dorndorf et al. [21] e, nesta codificação, o cromossomo é uma seqüência de máquinas e o escalonamento é construído utilizando-se um procedimento baseado em heurística.
- **Chave Aleatória (*Random key*):** A solução é codificada através de números aleatórios que são utilizados como chaves de ordenação para decodificar a solução. Cada gen do cromossomo é um número aleatório, onde a parte inteira,

representando a máquina, se encontra no intervalo $[1,m]$ e a parte fracionária é gerada aleatoriamente no intervalo $(0,1)$. O procedimento de decodificação ordena ascendentemente a parte fracionária e determina a ordem das operações em cada máquina. Esta representação foi proposta por Bean [7].

Este trabalho utiliza uma representação indireta baseada em operação, codificada na forma de uma permutação estendida de forma a gerar apenas indivíduos viáveis e considerando um procedimento de decodificação simples. Detalhes do método proposto neste trabalho estão detalhados nos capítulos subsequentes.

Estratégia Genética

Muito esforço tem sido feito afim de desenvolver uma implementação eficiente de AG aplicado ao problema JSP e uma das principais estratégias adotadas é incrementar o desempenho da busca genética através da incorporação de métodos heurísticos, uma vez que algoritmos genéticos não são adequados para realizar uma busca local refinada em regiões próximas à solução ótima. Vários métodos de AGs híbridos têm sido sugeridos para compensar esta limitação.

Uma característica comum em problemas de otimização combinatória é que se a permutação e/ou combinação pode ser determinada, a solução pode então ser derivada através de um procedimento de decodificação, conseqüentemente pode-se utilizar AGs para evoluir uma permutação e/ou combinação apropriada para então se aplicar um método heurístico para construir a solução. Esta alternativa tem sido aplicada com sucesso pela maioria dos pesquisadores em AGs aplicados a problemas de escalonamento, como descrito por Cheng et. al [14].

Existem duas principais restrições de ordem em JSP, a seqüência das operações em cada máquina e a restrição de precedência entre as operações do mesmo *job*. A primeira deve ser determinada pelo método de solução, enquanto a segunda deve ser mantida no escalonamento. Algumas propostas para manipular estas restrições são descritas a seguir.

1. Manter as informações de ambas as restrições codificadas simultaneamente no cromossomo, dispensando o tratamento de inviabilidades e ilegalidades dos indivíduos no desenvolvimento de novos operadores de cruzamento.

2. Somente informações relativas à seqüência das operações são incluídas na codificação e um procedimento de decodificação ou de construção do *schedule* é utilizado para resolver as restrições de precedência.
3. Nenhuma das informações são mantidas na codificação apenas algumas informações associadas são codificadas no cromossomo. Esta opção geralmente é utilizada pelas codificações de permutação pura de cadeias literais.

Cheng et. al [14] apresentam vários operadores que foram propostos para codificações por permutação simples, sendo os principais: PMX - cruzamento por mapeamento Parcial (*Partial-Mapped Crossover*), OX - cruzamento por Ordem (*Order Crossover*), CX - Cruzamento por ciclo (*Cycle Crossover*), cruzamento baseado em posição (*Position-Based*), cruzamento baseado em ordem (*Order-Based*), LOX - cruzamento de ordem linear (*Linear Order Crossover*), cruzamento por troca de subsequência (*Subsequence exchange*), cruzamento baseado em ordem de *job* (*Job-Based order Crossover*), cruzamento de troca parcial de schedule (*Partial Schedule exchange*) e cruzamento por troca de substring (*Substring exchange*).

Cheng et. al [14] também apresenta uma lista de operadores de mutação para codificações baseadas em permutação, tais como a mutação por inversão (*Inversion*), mutação por inserção (*Insertion*), mutação por deslocamento (*Displacement*).

Operadores genéticos que utilizam métodos heurísticos de sucesso também têm sido propostos para JSP. Estes procedimentos podem ser classificados como operadores de cruzamento, pois combinam indivíduos para produzir descendentes. Quando utilizados como operadores de mutação, estes métodos heurísticos alteram uma parte dos gens do indivíduo pai para produzir um novo descendente. Exemplos destes são o algoritmo de cruzamento por Yamada et. al [57] e a mutação baseada em busca na vizinhança apresentada por Cheng [12].

Algoritmos Genéticos Híbridos

Devido à dificuldade de busca em regiões próximas a pontos ótimos, Davis [19] sugere a hibridização, que consiste em introduzir heurísticas ou algoritmos para refinar a busca em pontos próximos do ótimo. Os métodos para hibridização em JSP podem ser classificados em três categorias, como apresentado por Cheng et. al [14].

- Operadores Genéticos Adaptados - consiste em adaptar ou criar operadores genéticos de acordo com as características da representação adotada.
- Operadores Genéticos Baseados em Heurísticas - criar novos operadores baseados em heurísticas convencionais.
- Algoritmos Genéticos Híbridos - considera a inclusão de heurísticas convencionais dentro do laço principal de AG, quando possível.

A técnica de hibridização resulta na integração de uma boa maneira convencional de resolver um problema aos conceitos de AG's. O resultado costuma ser melhor que o obtido com qualquer uma das duas técnicas isoladamente como apresentado por Davis [19]. Isto permite a incorporação de heurísticas otimizadoras ao conjunto de operadores genéticos (cruzamento e mutação) que se tornam, portanto, dependentes do domínio. E assim AG, pode ser visto mais como uma filosofia de otimização do que um método.

Incorporar a capacidade de realização de buscas locais tem sido uma estratégia adotada recentemente, devido, principalmente à complementariedade das duas técnicas e à qualidade dos resultados obtidos, pois estes são melhores quando comparados aos resultados alcançados com a aplicação isolada de cada técnica. A hibridização pode ser feita de algumas maneiras, conforme apresentado por Cheng et. al [14].

- Incluir heurísticas na criação da população inicial, gerando populações iniciais bem adaptadas que, ao serem combinadas com elitismo, podem garantir resultados melhores em relação a heurísticas tradicionais.
- Introduzir heurísticas na função de avaliação, na decodificação do indivíduo e na geração do *schedule*.
- Realizar busca local com heurística através de um procedimento adicional que, em conjunto com os operadores de cruzamento e mutação, possibilitam uma otimização rápida e localizada e melhoram os descendentes antes de serem avaliados.

Uma das formas mais comuns de AGs híbridos é a incorporação de técnicas de busca local como um procedimento adicional no operador cruzamento e no mecanismo de seleção, para converter cada filho em um *schedule* ativo. Nos modelos

híbridos, de acordo com Cheng et. al [14], AGs são utilizados na execução de uma busca global entre as populações, enquanto os métodos heurísticos executam uma exploração local nos cromossomos.

Trabalhos Relacionados

Tamaki et al. [50] considera FJSP aplicado a um ambiente de máquinas paralelas. A formulação do problema é dada através de programação inteira mista, sendo que um algoritmo genético é utilizado para obter *schedules* próximos do ótimo para problemas de grande escala. Em trabalho recente, Ong et al. [43] propõem um método de solução para FJSP baseado na Seleção dos Clones que ocorre no sistema imunológico dos seres humanos.

Ho et. al [28] apresentam o GENACE como uma metodologia para resolver FJSP com recirculação utilizando uma arquitetura evolutiva cultural e regras de prioridade compostas para obter soluções aproximadas para o problema.

Kacem et. al [32,33] apresentam uma proposta para FJSP, que utiliza AGs para solucionar simultaneamente os problemas de associação das operações às máquinas e de sequenciamento destas operações em cada máquina. Neste trabalho, um método para otimização multi-critério para FJSP é proposto, utilizando o algoritmo *Approach by Localization (AL)* para geração da população inicial.

AL gera apenas indivíduos viáveis, pois utiliza uma heurística para associação das operações às máquinas que considera o tempo de processamento e a carga de trabalho das máquinas com operações já associadas. Os operadores genéticos de cruzamento e mutação também mantêm a viabilidade dos indivíduos gerados e introduzem o conceito de manipulações genéticas, com o objetivo de melhorar a qualidade das soluções. A representação dos indivíduos é baseada em operação, codificada através de uma permutação e portanto, é utilizado um procedimento de decodificação para transformar cada indivíduo em uma solução.

O objetivo nas seções seguintes não é o de esgotar cada um dos assuntos, mas apresentar outros métodos utilizados, freqüentemente, para a resolução de problemas de escalonamento.

3.2 Busca Tabu

A meta-heurística Busca Tabu (BT), proposta por Glover [23] teve origem a partir de uma solução para problemas de programação inteira e, posteriormente, foi descrita como método para uso geral em problemas de otimização combinatória realizada por Adams et al. [1].

A BT apresentada por Glover [23] com o objetivo de encontrar boas aproximações para a solução ótima global de qualquer problema de otimização, considera três princípios fundamentais: (i) uso de uma estrutura de dados (lista) para guardar o histórico da evolução do processo de busca, (ii) uso de um mecanismo de controle para fazer um balanceamento entre a aceitação ou não de uma nova configuração, com base nas informações registradas na lista tabu referentes às restrições e aspirações desejadas e (iii) incorporação de procedimentos que alternam as estratégias de diversificação e intensificação conforme apresentado por Glover et. al [24]. A utilização deste tipo de algoritmo supõe uma função objetivo. Após esta definição, é gerada uma solução inicial viável independente (Regra de Despacho). Sendo que, para a geração da solução inicial, é obrigatório que esta faça parte do conjunto de soluções possíveis do espaço amostral.

Buscaylet et. al [10] propuseram um método utilizando busca tabu para resolver um problema de *job shop* considerando tempos de setup dependentes de sequência. Este método se baseou na proposta de Nowicki et. al [42] e utiliza uma representação de grafos disjuntos onde um nó é definido para cada operação e são incluídos dois nós adicionais, 0 e *, representando o início e o fim do *schedule*, respectivamente. Dois tipos de arestas são definidas, arestas de *jobs* conjuntivos e arestas de máquinas disjuntivas. Existe uma aresta *job* conjuntiva entre duas operações consecutivas de um mesmo *job* e uma aresta de máquina disjuntiva entre duas operações associadas a mesma máquina. É assumido que uma aresta de *job* conjuntiva é avaliada pelo tempo de processamento da operação de origem. Após a definição do sentido das arestas de máquina disjuntiva, estas são avaliadas pelo tempo de processamento da operação de origem mais o tempo de configuração necessário para executar a operação de origem seguida pela operação destino.

Por definição, uma seleção é um estado de um grafo disjuntivo G , onde uma direção é escolhida para algumas arestas disjuntivas. A seleção é dita completa

quando todas as arestas têm direções. Uma seleção completa corresponde a um *schedule* semi-ativo se o grafo resultante é acíclico. Neste contexto, minimizar o *makespan* é equivalente a encontrar uma seleção com o menor caminho máximo. Um caminho crítico de um grafo G orientado pode ser decomposto dentro de blocos $B_1, \dots, B_r$, tal que, cada bloco é a máxima seqüência de tamanho ao menos 1 de operações adjacentes processadas na mesma máquina e pertencentes ao caminho crítico.

A partir de uma solução inicial, o método de Buscaylet et. al [10] executa uma série de movimentos visando reduzir o tamanho do maior caminho, que representa o *makespan*. A solução inicial é gerada através de um algoritmo de ordem restrita que gera *schedules* ativos. A base deste algoritmo é selecionar a cada passo um operação não escalonada e atribuir esta operação na máquina que a completar primeiro.

Nowicki et. al [42] propuseram uma função de vizinhança *INTB* que obtém uma solução vizinha trocando-se duas operações adjacentes em um bloco. Além disso, esta função considera que uma vizinhança é vazia quando existem blocos de apenas uma operação ou quando existe um único bloco, sendo que em ambos os casos corresponde a encontrar um escalonamento ótimo. A inclusão de tempo de configuração dependente de seqüência invalida esta propriedade, por isso Buscaylet et. al [10] propuseram *STN* uma função de vizinhança adaptada para considerar o impacto da inclusão do tempo de configuração entre as operações executadas em uma mesma máquina. Quando *INTB* não é vazia $STN = INTB$, caso contrário *STN* troca para outro vizinho, como utilizado no problema clássico de *JSP*. Esta vizinhança considera troca de duas operações adjacentes em um bloco, mesmo se ambas as operações forem as internas e mesmo se houver somente um bloco. Por último o gerenciamento da lista tabu é realizado conforme proposto por Nowicki et. al [42].

Foi apresentado pelo autores que este método apresenta uma instabilidade na obtenção dos resultados, além de não ser capaz de escapar de regiões de ótimos locais do espaço de busca. Para melhorar a estabilidade, o método foi embarcado em uma heurística multi-passo.

3.3 Recozimento Simulado

Como o próprio nome diz, o algoritmo Recozimento Simulado (*Simulated Annealing*) refere-se à forma como que os metais líquidos são resfriados vagarosamente de forma a garantir uma baixa energia e formatos de estrutura altamente sólidos. O recozimento simulado pode ser encarado como um algoritmo que “resfria” vagarosamente a solução para garantir que ela possua o menor valor possível para a função objetivo. Por garantir um alto nível de movimentação através do espaço de busca, o recozimento simulado busca varrer todo esse espaço, de forma a buscar uma solução global. Com o decorrer do processo, o recozimento permitirá apenas pequenos movimentos no espaço de soluções, e o processo converge finalmente para uma solução final.

No método de Recozimento Simulado, em cada iteração, busca-se o próximo candidato a ponto de mínimo na vizinhança do candidato corrente, de acordo com a diferença entre os valores da função objetivo (chamada, nesse contexto, de função de energia ou potencial). A maior vantagem de Recozimento Simulado sobre outros métodos é a possibilidade de evitar mínimos locais. Para isso, o algoritmo emprega uma busca aleatória que, por vezes, aceita vizinhos cuja energia seja mais elevada. Ou seja, em algumas iterações, o Simulated Annealing tende a maximizar a função-objetivo em vez de minimizá-la.

Entretanto, uma característica importante desse algoritmo é que a probabilidade de aceitar um vizinho de maior energia decresce com o tempo, devido ao uso do parâmetro “temperatura”, que decresce a cada iteração. Por fim, em qualquer temperatura, dados dois vizinhos de maior energia que o candidato a mínimo corrente, A e B, se $\text{energia}(A) > \text{energia}(B)$, a probabilidade de aceitação de A será menor que a de B. O algoritmo começa com uma solução inicial e as soluções vizinhas são visitadas aleatoriamente. Em cada iteração, as soluções vizinhas que melhoram a solução corrente são sempre aceitas. Já aquelas que pioram a solução corrente são aceitas com uma certa probabilidade $p(p < 1)$. Nas iterações iniciais a maioria das soluções são aceitas, entretanto com o progresso do algoritmo, processo de recozimento, a probabilidade de aceitar soluções ruins decresce. Na prática, para se aplicar Recozimento Simulado, três pontos básicos devem ser observados: uma representação concisa para o problema, uma função de vizinhança e uma escala de

recozimento.

Najib et. al [40] apresenta uma proposta baseada em um algoritmo de Recozimento Simulado alterado para resolver problema de escalonamento FJSP. Este trabalho se preocupa em minimizar o *makespan* e utiliza uma representação que utiliza grafos disjuntos adaptados para permitir a representação do problema.

Esta representação utiliza um conjunto de vértices V formado pelas operações que compõem os *jobs*, além de dois vértices adicionais 0 e *, representando o início e o fim do *schedule* respectivamente. Um conjunto de arestas $C \cup D$, onde C é um conjunto de arestas conjuntivas entre todos os pares de operações consecutivas de cada *job*, acrescido de arestas criadas entre o vértice inicial 0 e a primeira operação de cada *job* e a última operação de cada *job* e a operação final *. D é um conjunto de arestas disjuntas (não orientadas), tal que existe uma aresta entre pares de operações que podem ser associadas a mesma máquina. A associação das operações às máquinas é realizada fixando-se a direção das arestas disjuntas, como no caso de representação de grafos disjuntos para *JSP*. O maior caminho entre a operação inicial e final é chamado de caminho crítico e seu tamanho corresponde ao tempo necessário para completá-lo, conhecido como *makespan*.

A função de vizinhança utilizada foi desenvolvida por Dauzère-Pérès et. al [18] e se baseia em pequenos deslocamentos, sendo sua principal contribuição a redução do conjunto de vizinhos da solução atual enquanto garante que o conjunto resultante contém o melhor vizinho. Um vizinho da solução S é obtido deslocando uma operação. Primeiro, uma operação é escolhida e removida da seqüência de sua máquina atual e, em seguida, é associada à seqüência de uma máquina candidata, que pode ser a máquina a qual já estava associada, finalmente é definida a seqüência na máquina candidata, desde que a solução obtida seja viável. Uma inserção é dita favorável se gera uma solução com um valor de *makespan* menor.

Geralmente, os valores de temperatura inicial utilizados são valores altos para permitir a busca aceitar um grande número de soluções. O ajuste da temperatura T é calculado por um procedimento executado a cada l iterações de acordo com o desempenho da busca na última temperatura. O valor de l é definido a partir do número de iterações que não melhoraram a função objetivo.

No sentido de explorar o máximo número de vizinhanças o diagrama de recozi-

mento é realizado passo a passo, isto é $T_{k+1} = \alpha T_k$, onde α é uma constante positiva menor que 1 e k representa o passo atual. O tamanho do passo é determinado por um trade-off entre o tempo de execução e a qualidade da solução.

Geralmente a solução inicial é escolhida a partir de um boa regra heurística ou de sequenciamento. A escolha da solução inicial afetará o tipo de vizinhança utilizada e se as melhorias nas soluções serão efetivas. Najib et. al [40] utiliza na obtenção da solução inicial uma heurística específica baseada na regra de prioridade SPT. As operações são associadas de acordo com seu tempo de início, levando-se em consideração a carga das máquinas à medida que o processo de construção da solução avança. Em cada iteração, primeiro determina-se qual máquina será carregada, em seguida, obtém-se a operação com menor tempo de início, do conjunto de operações disponíveis. Uma operação é dita disponível se todas as suas operações predecessoras já foram associadas.

3.4 Branch and Bound

O algoritmo branch-and-bound é um algoritmo enumerativo, cuja estrutura de resolução baseia-se na construção de uma árvore onde os nós representam os problemas candidatos e os ramos representam as novas restrições que devem ser consideradas. Por intermédio dessa árvore, todas as soluções inteiras da região viável do problema são enumeradas de modo implícito ou explícito o que garante que todas as soluções ótimas serão encontradas. A estrutura geral apresenta três elementos fundamentais, que estão detalhados a seguir: separação, relaxação e sondagem.

Na etapa de separação, o problema original P é separado em q subproblemas $P^1, P^2, \dots, P^q$ sujeitos às seguintes condições. Toda solução viável de (P) é uma solução de somente um dos subproblemas P^i , $i = 1, 2, \dots, q$ e uma solução viável de qualquer um dos subproblemas P^i , $i = 1, 2, \dots, q$, também é uma solução viável de P . Estas condições asseguram que o conjunto das soluções viáveis de cada um dos subproblemas P^i , $i = 1, 2, \dots, q$ é uma partição do conjunto das soluções viáveis de P . Os subproblemas P^i , $i = 1, 2, \dots, q$ são denominados descendentes de P e podem, sucessivamente, gerar seus próprios descendentes.

O interesse na separação (*branching*) é utilizar a estratégia de “dividir para conquistar” para resolver o problema P . Pode-se descrever sumariamente esta estratégia

como, enquanto a solução de P não é possível, separa-se P em dois ou mais sub-problemas descendentes, gerando uma lista de problemas candidatos PC . A seguir seleciona-se um dos candidatos desta lista e tenta-se resolvê-lo. Se a solução não é possível o problema é, novamente, separado e seus descendentes são adicionados à lista dos candidatos; caso contrário o problema é resolvido e uma nova solução é obtida. O valor da função objetivo dessa nova solução é, então, comparado com o valor da melhor solução viável conhecida até o momento. Caso a nova solução seja melhor ela se torna a nova melhor solução. A seguir, retorna-se a lista e seleciona-se o próximo candidato. Isto é repetido até que a lista esteja vazia, quando se pode afirmar que a solução do problema é dada pela melhor solução final.

A relaxação consiste em, temporariamente, ignorar algumas restrições do problema P visando torná-lo mais fácil de resolver. A condição que deve ser satisfeita é que o conjunto de soluções viáveis do problema original P esteja contido no conjunto de soluções viáveis do problema relaxado P_R . Isto implica que:

- Se P_R não tem solução viável, então o mesmo é verdadeiro para P .
- O valor mínimo de P não é menor que o valor máximo de P_R .
- Se uma solução ótima de P_R é viável em P , então ela é uma solução ótima de P .

Dentre as forma possíveis de relaxação, destaca-se a eliminação das restrições de integralidade das variáveis, o que transforma o problema inteiro misto em um problema linear padrão.

Na análise dos problemas candidatos, é necessário determinar quais são promissores e, quais podem ser sumariamente descartados. Isto é realizado na etapa de sondagem onde o problema candidato PC é eliminado, juntamente com todos os seus descendentes, se pelo menos um dos seguintes critérios for satisfeito.

- O problema candidato relaxado PC_R não tem solução viável.
- A solução ótima do problema candidato relaxado PC_R é pior (*bounding*) do que a melhor solução atualmente conhecida P .
- Uma solução ótima do problema relaxado PC_R é viável, também em PC .

Artigue et. al [4] propõem um método para minimização do *Makespan* em problema JSP com tempo de configuração dependente de seqüência. Este método encontra uma solução ótima para o problema através da resolução iterativa de uma versão decisional do algoritmo Branch-and-Bound. Nesta versão, cada nó da árvore de branch-and-bound algoritmos de propagação de restrições adaptados para tempo de configuração são executados para verificarem a viabilidade e realizarem uma filtragem de domínio.

As relaxações são baseadas em um problema TSP com janela de tempo e são resolvidos para executarem podas adicionais. O TSP é formulado como um problema de caminho mais curto elementar, sendo resolvido através de programação dinâmica.

A versão decisional do problema (FP) é definida considerando que $T \geq 0$, uma *upper bound* do makespan gerado por tentativa. Considera-se o problema $FP(T)$ como o problema de encontrar um *schedule* $\mathcal{T}$, tal que $makespan \leq T$ e as restrições de cálculo do makespan mais a restrição de que o tempo de início de toda operação ocorre após o tempo de configuração estejam satisfeitas. Então, seja UB o *makespan* de qualquer *schedule* viável e LB o *lower bound* da solução ótima de (P), o C_{max}^* denota a solução ótima de P , é calculado pela Equação 3.1.

$$C_{max}^* = \min_{LB \leq T \leq UB} \{T | FP(T) \text{ tenha uma solucao} \} \quad (3.1)$$

A representação utilizada por este método é a de grafo disjunto. O cálculo do *makespan* é realizado como apresentado no método de recozimento simulado descrito na Seção 3.3.

Capítulo 4

Escalonador Genético para FJSP

O Problema de escalonamento JSP tem atraído a atenção de muitos pesquisadores devido a sua larga aplicabilidade e dificuldade inerente de resolução, sendo classificado por Lenstra et. al [38] um problema NP-Completo, como dito anteriormente. O JSP clássico envolve n *jobs* e m máquinas, onde cada *job* é processado em uma sequência específica de máquinas

Em problemas práticos do mundo real a configuração dos ambientes onde JSP são aplicados, frequentemente apresentam múltiplas instâncias das máquinas para reduzir os gargalos nas execuções das operações ou para minimizar o tempo em que as máquinas ficam ocupadas. Desta forma, permitir que uma operação seja executada em mais de uma máquina, além de tornar o problema mais geral, também aumenta a complexidade para obtenção de soluções ótimas ou aproximadas.

Embora o JSP tenha sido bem estudado, aplicações do mundo real são limitadas pela restrição de realizar um mapeamento um-para-um entre operações e máquinas. Por isso, FJSP surge como uma extensão do JSP tradicional por permitir que cada operação ser processada em mais de uma máquina.

AGs têm sido uma técnica bastante utilizada na resolução de problemas FJSP, como pode ser visto em [8, 11, 32, 33, 48, 59]. Entretanto, AGs tradicionais quando aplicados a problemas de escalonamento, frequentemente criam um número muito grande de indivíduos inviáveis, devido à aleatoriedade presente nos operadores genéticos e também na geração da população inicial. Então podem ocorrer violações das restrições do problema, exigindo procedimentos para reparação dos indivíduos, e assim, garantir que apenas indivíduos viáveis estejam presentes em toda população.

Conforme descrito no capítulo anterior, a construção de AGs híbridos tem sido também uma alternativa para FJSP. Assim, neste trabalho propõe-se um AG híbrido, que utiliza operadores genéticos com conhecimento específico do problema. Os indivíduos são representados através de permutações de todas as operações. A geração da população inicial e os operadores genéticos garantem a manutenção de apenas indivíduos viáveis na população.

Os operadores genéticos convencionais são utilizados para manter a diversidade na população de indivíduos, enquanto os operadores com conhecimento específico do problema visam a inclusão de indivíduos geneticamente melhores. Esta combinação pretende incrementar a qualidade das soluções geradas. A proposta de solução apresentada neste trabalho foi denominada BTSL (*Balancing Task Sequencing List*), uma vez que os operadores genéticos têm como principal objetivo balancear a distribuição das operações entre às máquinas.

Assim, neste capítulo é apresentada BTSL, considerando FJSP com tempo de configuração dependente de sequência e tendo como critério de otimização uma função multi-objetivo. Na seção 4.1, a formulação do problema é apresentada. Na seção 4.2, descreve-se BTSL, detalhando a representação utilizada, a geração da população inicial, os operadores genéticos e o procedimento de decodificação da solução. Na seção 4.3 são discutidos os resultados obtidos.

4.1 Formulação do Problema

Um problema de escalonamento FJSP com tempo de configuração dependente de sequência SDST-FJSP (Sequence-Dependent Setup Time - Flexible Job Shop Problem) consiste em associar cada operação a uma máquina e sequenciar a execução das operações em cada máquina, determinando os tempos de início e de término de cada operação, levando em consideração os tempos de configuração entre a execução de operações consecutivas na mesma máquina.

A formulação do problema considera as seguintes variáveis e características.

- n jobs a serem processados, onde $n \geq 1$.
- m máquinas, onde $m \geq 1$.
- Cada *job* é uma sequência de n_j operações.

- i denota o índice das operações do *job* j , onde $1 \leq j \leq n$ e $1 \leq i \leq n_j$.
- $\mathcal{O}$ é o conjunto de todas as operações.
- $N = \sum_{j=1}^n n_j$ é o número total de operações de todos os jobs.
- $O_{i,j}$ representa a operação i do job j .
- $O_{i,j,k}$ representa a operação i do job j executada na máquina k
- $p_{i,j,k}$ o tempo de processamento da operação i do *job* j na máquina k , onde $p_{i,j,k} > 0$.
- Cada operação $O_{i,j}$ deve ser processada por uma máquina do conjunto $M_{i,j}$ que representa às máquinas onde $O_{i,j}$ pode ser processada.
- st_{ik} representa o tempo de configuração entre as operações consecutivas $O_{i,j}$ e $O_{x,y}$ na máquina k , para quaisquer *jobs* j e y .
- st_k representa a soma do tempo de configuração na máquina k .
- st_0 é o tempo de configuração inicial da máquina.
- t_{ij} é o tempo de início de processamento da operação $O_{i,j}$
- W_k é a carga de trabalho da máquina k , e equivale ao tempo de processamento total da máquina k .
- tf_{ij} é o tempo de término de processamento da operação $O_{i,j}$.

O problema consiste então em determinar as associações das operações às máquinas, e a seqüência de execução das operações em cada máquina, de forma a minimizar a função objetivo. Associar operações a máquinas significa estabelecer um mapeamento μ de $\mathcal{O}$ no conjunto das máquinas, tal que $\mu(O_{ij}) = k$, onde k é a máquina na qual a operação O_{ij} será executada.

São consideradas as seguintes condições na resolução do problema:

- Todos os *jobs* e máquinas estão disponíveis no tempo zero.
- A ordem das operações nos jobs é fixa e não pode ser alterada (restrição de precedência).

- Uma máquina pode executar somente uma operação por vez.
- Uma vez iniciado o processamento de uma operação, este não pode ser interrompido, ou seja as operações são não preemptivas.
- Dados relativos aos *jobs* e máquinas são conhecidos antecipadamente, ou seja, é um problema de escalonamento determinístico.
- O tempo de configuração não verifica a desigualdade triangular.

Formalmente o problema consiste em minimizar a função multi-objetivo f que considera como critérios, o tempo máximo de término de todos os jobs (*Makespan*), a carga total das máquinas (W), a máquina mais carregada (W_k) e o tempo total de configuração (TST). Então, sejam $q = 1, \dots, N_q$, onde N_q é o número de critérios a serem otimizados, e $\delta_q \in [0, 1]$, o valor do peso de cada critério Cr_q , com $\sum_{q=1}^{N_q} \delta_q = 1$, a função f é dada pela Equação 4.1.

$$f(Cr_1, \dots, Cr_q) = \sum_{q=1}^{N_q} \delta_q Cr_q \quad (4.1)$$

Desde que $N_q = 4$, os quatro critérios são dados pelas Equações 4.2, 4.4, 4.3 e 4.5.

$$C_{max} = \max_{1 \leq j \leq n} \{t f_{n_j, j}\} \quad (4.2)$$

$$W = \sum_{k=1}^m W_k \quad (4.3)$$

$$W_k = \max_{1 \leq k \leq m} \{W_k\} \quad (4.4)$$

$$TST = \sum_{k=1}^m st_k \quad (4.5)$$

A Equação 4.2 refere-se ao *Makespan*, que é o tempo de término da última operação executada no sistema. A Equação 4.4 é o critério relativo à carga da máquina mais carregada, a Equação 4.3 indica a carga total das máquinas. Os critérios 4.4 e 4.3 visam manter o balanceamento de carga entre todas as máquinas. A Equação 4.5 representa a soma total do tempo de configuração, obtida a partir

da soma dos tempos de configuração em cada máquina. O tempo de configuração necessário entre a execução de operações na mesma máquina depende da ordem em que estas operações são executadas na máquina. Além disso, o problema está sujeito as restrições 4.6, 4.7 e 4.8

$$t_{ij} + p_{ijk} \leq t_{i+1,j} \quad 1 \leq i \leq n_j, 1 \leq j \leq n \text{ e } 1 \leq k \leq m \quad (4.6)$$

$$\begin{aligned} t_{ij} + p_{ijk} + st_{ixk} &\leq t_{xy} \quad \text{ou} \\ t_{xy} + p_{xyk} + st_{xik} &\leq t_{ij} \quad \forall O_{ij} \text{ e } O_{xy} \in \mathcal{O}, O_{ij} \neq O_{xy} \text{ e } 1 \leq k \leq m \end{aligned} \quad (4.7)$$

$$t_{ij} \geq st_0 \quad \forall O_{ij} \in \mathcal{O} \text{ e } 1 \leq k \leq m \quad (4.8)$$

A restrição 4.6 representa a restrição de precedência, onde uma operação $O_{i+1,j}$ não pode iniciar o processamento antes do tempo de término da operação precedente O_{ij} no job j . A restrição 4.7 determina que duas operações distintas O_{ijk} e O_{xyk} , que compartilham a mesma máquina k , não podem ser escalonadas simultaneamente e a máquina deve ser preparada entre operações consecutivas. Conseqüentemente nem O_{ij} pode iniciar antes do início de O_{xy} mais o tempo de configuração necessário st_{xik} , nem O_{xy} pode iniciar antes do final de O_{ij} mais o tempo de configuração st_{ixk} . A restrição 4.8 reforça que o tempo de início de qualquer operação ocorre após o tempo inicial de configuração da respectiva máquina. Estas restrições foram obtidas em Artigues et al. [4].

Este problema particular é denotado $FJm|st|Ms, W, W_k, TST$ na notação de Graham et. al [26]. Na literatura não foi encontrada uma denominação utilizada para definição de problemas de escalonamentos FJSP, com isso, ao valor Jm do termo α incluímos a letra F , pois Jm normalmente é utilizado para designar problemas JSP, formando FJm .

4.2 Elementos do Escalonador

Esta seção apresenta a proposta BTSL, baseada em AGs para resolução de FJSP, com tempo de configuração dependente de seqüência. Assim, o AG proposto considera as duas dificuldades associadas ao problema: a associação de cada operação

$O_{i,j}$ a uma máquina k , e o sequenciamento das operações em cada máquina, além dos tempos de configuração, de forma a minimizar a Função 4.1.

Najib et. al [40] classifica as propostas de solução de problema de FJSP em propostas simultâneas e hierárquicas. Na primeira, tanto o problema de associação das operações quanto o problema de sequenciamento destas operações nas máquinas são tratadas simultaneamente. Enquanto na segunda proposta, estes dois problemas são tratados de forma independente. O método proposto neste trabalho é uma proposta simultânea, pois a representação dos indivíduos contém informações sobre as associações e os operadores genéticos atuam na ordem das associações nas máquina, como também na troca da máquina associada a uma operação.

Assim, a seguir são apresentados a codificação dos indivíduos, os algoritmos para a geração da população inicial e os operadores genéticos, destacando dois novos operadores propostos. Além disso, também são apresentados o procedimento de decodificação e o cálculo dos valores de aptidão de cada indivíduo.

4.2.1 Codificação da Solução

A codificação utilizada neste trabalho estende a representação TSL (Task Sequence List), adotada por Kacem [32] pela introdução do tempo de processamento em um quarto campo em cada gen. Um esboço desta representação pode ser visto na Figura 4.1. Um gen genérico do indivíduo é dado por (i, j, k, p_{ijk}) , onde i , j e k representam, respectivamente, operação, job e máquina, e p_{ijk} o tempo de processamento de $O_{i,j,k}$. A Figura 4.1 mostra a codificação genérica para um indivíduo que contém N gens.

A fim de gerar apenas indivíduos viáveis, a restrição de precedência dos jobs é suportada observando-se que, em TSL, o índice z da operação $O_{i+1,j}$, isto é, o índice z do gen T_z , não pode ser menor que o índice de $O_{i,j}$. Esta restrição deve ser mantida na geração da população inicial e também na aplicação de cada um dos operadores genéticos. O procedimento de decodificação utiliza esta ordem para transformar um indivíduo da população em um *schedule*.

A Tabela 4.1, representa um T-FJSP 3x4, 3 *jobs* e 4 máquinas, e contém os tempos de processamento para cada operação em cada máquina. A Figura 4.2 apresenta uma codificação possível para um indivíduo, gerado a partir do problema apresentado na Tabela 4.1. A inclusão do tempo de processamento em cada gen

acelera o procedimento de decodificação, desde que o tempo de processamento de cada operação na máquina em que está associada já está disponível no gen do próprio indivíduo a ser decodificado.

Tabela 4.1: Tabela T com tempos de processamento

		M_1	M_2	M_3	M_4
J_1	O_{11}	1	3	4	1
	O_{21}	3	8	2	1
	O_{31}	3	5	4	7
J_2	O_{12}	4	1	1	4
	O_{22}	2	3	9	3
	O_{32}	9	1	2	2
J_3	O_{13}	8	6	3	5
	O_{23}	4	5	8	1

T_1				$\dots$				T_N			
Op_1	J_1	M_1	p_1					Op_N	J_N	M_N	p_N

Figura 4.1: Representação da Solução

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
(1,1,4,1)	(1,2,2,1)	(2,2,1,2)	(2,1,4,1)	(3,2,2,2)	(3,1,1,3)	(1,3,3,3)	(2,3,4,1)

Figura 4.2: Representação por TSL

4.2.2 Geração da População Inicial

A população inicial é gerada por algoritmos baseados no método *Approach by Localization* (AL), proposto por Kacem [32], e no algoritmo *CDR-Bootstrap*, proposto por Ho et. al [28]. O primeiro é uma heurística para associar operações às máquinas, levando em consideração o tempo de processamento da operação e a carga de trabalho das máquinas para as quais já foram associadas operações e, o segundo utiliza regras de prioridade compostas. Os Algoritmos 2 e 3 apresentam a implementação dos algoritmos criados *AssignmentProcedure_BTSL()* e *EFO()* respectivamente.

O algoritmo *AssignmentProcedure_BTSL()*, baseado em AL, gera a partir da tabela T , uma tabela T' que contém os tempos de processamento de cada operação em cada máquina. Após a criação de T' troca-se as posições dos *jobs* em T' aleatoriamente, a fim de evitar que se gere indivíduos idênticos a cada iteração do algoritmo. Resumidamente, o algoritmo consiste em associar uma operação $O_{i,j}$ à máquina k , pertencente ao conjunto das máquinas M , somente se o tempo de processamento da operação $O_{i,j}$ na máquina k , $p_{i,j,k}$, representar o menor valor da tabela T' . A atualização da tabela T' é realizada para que as próximas associações considerem a carga de trabalho já associada a cada máquina.

A cada iteração do algoritmo, uma associação é gerada, eliminando então uma linha de T' correspondente à operação associada e atualizando os tempos de processamento das operações vinculadas à máquina, ou seja, a coluna de T' relativa à máquina é incrementada com o tempo de processamento da operação associada. Considerando a tabela T , apresentada na Tabela 4.1, as iterações geradas pelo primeiro laço do algoritmo são apresentadas no Apêndice B.

Após definidas as associações de todas as operações às máquinas, a representação TSL é gerada aplicando-se a regra de prioridade SPT (Shortest Processing Time) aos conjuntos de operações E . Cada conjunto E é formado pelas operações com índice i em todos os *jobs*. A geração da TSL desta maneira garante que nenhuma regra de precedência é violada, pois a operação $i + 1$ de um *job* estará sempre à direita da operação i do respectivo *job*.

Pode-se notar que o Algoritmo 2 privilegia a associação de operações com o menor tempo de processamento e de configuração, evitando associar operações a máquinas com tempo de processamento muito grande. Baseado neste fato, é possível obter um FJSP parcial a partir de um FJSP total através da atribuição de um valor muito grande (infinito) para o tempo de processamento das operações que não possam ser executadas em um certo subconjunto de máquinas. Desta forma, a aplicação do próprio algoritmo evitará naturalmente que uma associação proibida seja realizada.

O Algoritmo 3 simplifica *CDR-Bootstrap*, devido à duas diferenças. A primeira diferença é a eliminação da lista M_{prox} que armazena as operações associadas às máquinas que ainda não estão disponíveis. Uma vez identificada a máquina, a associação é realizada e os tempos de disponibilidade da máquina e do *job* são acresci-

Algoritmo 2 AssignmentProcedure_BTSL()

```

1: Cria uma tabela  $T'$  de tamanho igual a  $T$ ;
2: Troque a posição de 2 jobs de  $T'$ , aleatoriamente;
3: while  $T'$  não vazia faça
4:   Ordena  $T'$  considerando o menor  $p_{ijk}$  associado a cada  $O_{ij}$ 
5:    $O_{ijk} = T'[0]$ , onde  $k$  é a máquina com menor SPT que executa  $O_{ij}$ 
6:   Adiciona  $O_{ijk}$  em  $J[O_{ij}]$ 
7:   Remove  $O_{ij}$  de  $T'$ 
8:   Soma  $p_{ijk}$  à coluna  $k$  de  $T'$ 
9: fim while
10: para  $i = 1$  até  $\max(n_j)$  faça
11:   para  $j = 1$  até  $n$  faça
12:      $E[j] = J[O_{ij}]$ 
13:     se  $st_k = 0$  então
14:       Soma  $st_0$  e  $p_{i,j,k}$  de  $E[j]$ 
15:     senão
16:       Soma  $st_{xik}$  e  $p_{i,j,k}$  de  $E[j]$ , onde  $O_{xy}$  antecede  $O_{ij}$  em  $k$ 
17:     fim se
18:     Remove  $O_{ij}$  de  $J$ 
19:   fim para
20:   Ordena  $E$  pelo novo tempo de processamento  $p_{ijk}$ 
21:   while  $E$  não vazia faça
22:     Adiciona  $E[l]$  na primeira posição disponível de TSL
23:     Remove  $E[l]$  de  $E$ 
24:   fim while
25: fim para

```

dos do tempo de configuração e do tempo de processamento. Na versão original, a remoção das operações de M_{prox} é realizada através de uma heurística baseada em regras de prioridade, tais como SPT, FIFO ou LPT. Sendo assim, no Algoritmo 3 estas regras não são utilizadas, evitando tempo de processamento excessivo.

Pode ser observado que, de uma forma implícita, a regra de prioridade FIFO ainda é utilizada, uma vez que as operações são associadas assim que identificadas para associação, entretanto não se utiliza processamento adicional para decidir entre outras regras possíveis. Desta forma, o algoritmo apresenta a vantagem de diminuir o tempo de processamento enquanto produz boas soluções.

A segunda diferença é a determinação da melhor máquina, que utiliza o tempo de configuração st_k , o tempo de processamento da operação $p_{i,j,k}$ em cada máquina k ($k = 1, \dots, m$) e o máximo tempo entre o tempo de disponibilidade do *job*, $DJ[j]$, e o tempo de disponibilidade das máquinas, $DM[k]$, considerando as associações realizadas até o momento.

O Algoritmo 3, *EFO()*, descreve o procedimento responsável por identificar a associação da operação $O_{i,j}$ ao conjunto de máquinas, onde se utiliza o Algoritmo 4, *Assingment_EFO()* para identificar a melhor máquina para a associação. Assim, inicialmente, cria-se T' conforme o Algoritmo 2 e gera-se aleatoriamente a ordem dos *jobs* em um vetor, que define a ordem de associação da primeira operação de cada *job*. Para cada associação realizada, um gen é adicionado ao indivíduo, a operação correspondente removida de T' e os tempos de término do respectivo *job* e de disponibilidade da máquina atualizados.

A ordem de associação das demais operações é definida de acordo com o tempo de término de cada *job*. A próxima operação do *job* com menor tempo de término é obtida e associada à máquina. É considerada a melhor máquina para a operação aquela que apresentar a menor soma do tempo de processamento, o tempo de configuração e do máximo tempo de disponibilidade entre o *job* e a máquina.

De forma geral, o Algoritmo 4 associa cada operação à máquina, reduzindo a carga de trabalho, o tempo de configuração e o tempo de conclusão de todos os *jobs*. São gerados indivíduos distintos dependendo da ordem em que os *jobs* são considerados no vetor de ordem, conseqüentemente há diversidade na população inicial.

Algoritmo 3 EFO()

```

1: para ( $k = 1$  até  $m$ ) faça
2:    $DM[k] = 0$ 
3: fim para
4: para ( $j = 1$  até  $n$ ) faça
5:    $DJ[j] = 0$ 
6: fim para
7: Cria uma tabela  $T'$  de tamanho igual a  $T$ ;
8: para ( $j = 1$  até  $n$ ) faça
9:    $Ass = \text{Assignment\_EFO}(O_{1,j})$ 
10:  Insere  $Ass$  em  $TSL$ 
11:  Remove  $O_{1,j}$  de  $T'$ 
12: fim para
13: while  $T'$  não vazia faça
14:    $j = \text{job}$  com o menor valor de  $DJ[j]$ 
15:    $O_{i,j} =$  próxima operação do  $job$   $j$  ainda não associada
16:    $Ass = \text{Assignment\_EFO}(O_{i,j})$ 
17:   Insere  $Ass$  em  $TSL$ 
18:   Remove  $O_{i,j}$  de  $T'$ 
19: fim while

```

Algoritmo 4 Assingment_EFO($O_{i,j}$)

Entrada: O_{ij} operação i do job j a ser associada

```

1:  $menor = -\infty$ 
2: para ( $k = 1$  até  $m$ ) faça
3:   Obtém  $st_{ixk}$ 
4:   se  $menor < (st_{ixk} + p_{i,j,k} + \max(DJ[k], DM[k]))$  então
5:      $menor = st_{ixk} + p_{i,j,k} + \max(DJ[k], DM[k])$ 
6:      $melhor = k$ 
7:   fim se
8: fim para
9: Associe  $O_{ij}$  a melhor
10:  $DJ[j] = DJ[j] + st_{i,x,melhor} + p_{i,j,melhor} + \max(DJ[j], DM[melhor])$ 
11:  $DM[melhor] = DM[melhor] + st_{i,x,melhor} + p_{i,j,melhor}$ 

```

Os indivíduos gerados pelos Algoritmos 2 e 3 são todas soluções viáveis, já que a atribuição de uma operação à TSL é feita de acordo com a restrição de precedência. No Algoritmo 2 esta restrição é garantida pela geração da TSL a partir do conjunto E e, no Algoritmo 3, a linha 15 identifica a próxima operação do *job* que pode ser associada.

Ambos os algoritmos apresentados possuem uma vantagem adicional que é capacidade de considerar problemas que necessitam de tempo de chegada (*release dates*), tornando a aplicação da proposta mais geral. Isto pode ser realizado através da inicialização dos vetores de disponibilidade dos *jobs* e das máquinas ($DJ[j]$ e $DM[k]$), com as respectivos datas de chegada.

4.2.3 Operadores Genéticos

BTSL utiliza seis operadores genéticos. Os operadores POX e PPS, propostos por Lee et al. [37] essencialmente alteram a ordem de execução das operações nas máquinas, observando as restrições de precedência. Os operadores Cruzamento por associação e Mutação por associação, propostos por Kacem [32], possuem conhecimento específico do problema e têm como principal objetivo a alteração das associações das operações às máquinas.

Além destes quatro operadores, BTSL introduz dois novos operadores a fim de gerar soluções finais de alta qualidade em relação ao *Makespan*, atuando na distribuição de carga de trabalho entre as máquinas. Estes operadores, BWMM e BWMC são, respectivamente, um operador de mutação e de cruzamento. O objetivo destes operadores é balancear a carga de trabalho das máquinas e evitar a concentração da execução de operações em uma mesma máquina. Assim com uma distribuição balanceada das operações entre as máquinas, é possível obter soluções que também sejam boas em relação a outros critérios considerados na otimização.

Pode-se dizer que a carga de trabalho de duas máquinas está balanceada quando o tempo de utilização de ambas apresentam valores próximos. O cálculo da carga de trabalho de cada máquina é realizado no Algoritmo 7 (*Scheduling_Procedure_TSL*) e os respectivos valores são armazenados no vetor *Vet_workload*. Ambos os operadores propostos atuam simultaneamente tanto na troca da máquina associada à operação, quanto na ordem de execução das operações na máquina, sempre con-

siderando a carga de trabalho.

Operador POX

A partir de dois indivíduos pais, I_1 e I_2 seleciona-se aleatoriamente uma operação $O_{i,j}$ de I_1 , em seguida, copia-se para o descendente D_1 todas as operações do *job* j , mantendo as posições em que aparecem em I_1 . As operações do *job* j em I_2 são eliminadas e as posições vazias de D_1 são preenchidas com as operações restantes de I_2 , na ordem em que aparecem. O procedimento para formação do descendente D_2 é o mesmo, mas considerando outra operação aleatoriamente selecionada e a ordem das operações em I_2 .

$$\begin{array}{rcccccccc} I_1 & = & T_1 & T_2 & T_3 & T_4 & T_5 & \overline{T_6} & T_7 & T_8 \\ I_2 & = & T_2 & T_1 & T_3 & T_4 & T_6 & T_7 & \overline{T_5} & T_8 \end{array}$$

Figura 4.3: Pais selecionados para cruzamento POX

Considerando o indivíduo da Figura 4.2 e de acordo com as restrições de precedência tem-se que os *jobs* são $T_1T_4T_6$, $T_2T_3T_5$ e T_7T_8 . A ordem das operações nos *jobs* deve ser respeitada nos indivíduos gerados pelos operadores. A Figura 4.3 mostra um exemplo de dois indivíduos selecionados para o cruzamento, onde para o indivíduo I_1 foi selecionada a operação $\overline{T_6}$ e para o indivíduo I_2 foi selecionado a operação $\overline{T_5}$. Na figura 4.4 pode-se observar que as operações do *job* ao qual as operações pertencem são mantidas nas mesmas posições de D_1 e D_2 respectivamente.

$$\begin{array}{rcccccccc} D_1 & = & T_1 & \bullet & \bullet & T_4 & \bullet & \overline{T_6} & \bullet & \bullet \\ D_2 & = & T_2 & \bullet & T_3 & \bullet & \bullet & \bullet & \overline{T_5} & \bullet \end{array}$$

Figura 4.4: Início da geração dos descendentes pelo operador POX

A Figura 4.5 apresenta a geração final de D_1 e D_2 , onde observa-se a preservação das precedências.

$$\begin{array}{rcccccccc} D_1 & = & T_1 & T_2 & T_3 & T_4 & T_7 & \overline{T_6} & T_5 & T_8 \\ D_2 & = & T_2 & T_1 & T_3 & T_4 & T_6 & T_7 & \overline{T_5} & T_8 \end{array}$$

Figura 4.5: Descendentes gerados pelo POX

Operador PPS

A Figura 4.6 mostra o exemplo de um indivíduo selecionado para a mutação PPS, que consiste em selecionar uma operação O_{ij} , no caso T_4 . Em seguida, obtém-se as operações $O_{i-1,j}$ e $O_{i+1,j}$, que considerando o exemplo da Figura 4.2 correspondem a T_1 e T_6 . Determina-se as posições destas operações em TSL, então a nova posição de T_4 é uma posição aleatória entre a posição de T_1 e de T_6 .

$$I_1 = T_1 \ T_2 \ T_3 \ \overline{T_4} \ T_5 \ T_6 \ T_7 \ T_8$$

Figura 4.6: Indivíduo selecionado para mutação PPS

O resultado da aplicação do operador de mutação PPS no indivíduo I_1 da Figura 4.6 é apresentado na figura 4.7. Neste caso, T_1 ocupa no indivíduo I_1 a posição 1 e T_6 a posição 6. A posição escolhida aleatoriamente para T_4 foi a posição 2, então a operação T_4 é removida de I_1 e todas as operações a partir da posição 2 são deslocadas em uma posição a esquerda. Em seguida, a operação T_4 é incluída na posição 2.

$$D_1 = T_1 \ \overline{T_4} \ T_2 \ T_3 \ T_5 \ T_6 \ T_7 \ T_8$$

Figura 4.7: Descendente gerado pela mutação PPS

Como o indivíduo é sempre decodificado partindo das operações mais à esquerda para as operações mais à direita, os operadores genéticos propostos por Lee et al. [37] apenas alteram a ordem de execução das operações nas máquinas, mas não alteram as associações.

Operador Cruzamento por Associação

Considere os indivíduos selecionados para cruzamento I_1 e I_2 apresentados na Figura 4.8. A partir de I_1 escolhe-se aleatoriamente um conjunto de operações. Por exemplo, considere a seleção das operações T_1 , T_3 , T_6 e T_8 . O cruzamento, consiste então em identificar em cada indivíduo selecionado as operações escolhidas e trocar as associações entre os dois indivíduos. O resultado da aplicação deste operador são os descendentes mostrados na Figura 4.9.

I_1							
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
(1,1,4,1)	(1,2,2,1)	(2,2,1,2)	(2,1,4,4)	(3,2,2,1)	(3,1,1,3)	(1,3,3,3)	(2,3,4,1)

I_2							
T_2	T_1	T_3	T_4	T_5	T_7	T_6	T_8
(1,2,1,4)	(1,1,3,4)	(2,2,2,3)	(2,1,1,3)	(3,2,4,2)	(1,3,4,5)	(3,1,2,5)	(2,3,1,4)

Figura 4.8: Indivíduos selecionados para o Cruzamento por Associação

D_1							
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
(1,1,3,4)	(1,2,2,1)	(2,2,2,3)	(2,1,4,4)	(3,2,2,1)	(3,1,2,5)	(1,3,3,3)	(2,3,1,4)

D_2							
T_2	T_1	T_3	T_4	T_5	T_7	T_6	T_8
(1,2,1,4)	(1,1,4,1)	(2,2,1,2)	(2,1,1,3)	(3,2,4,2)	(1,3,4,5)	(3,1,1,3)	(2,3,4,1)

Figura 4.9: Descendentes gerados pelo Operador de Cruzamento por Associação

Operador Mutação por Associação

Considere o indivíduo selecionado para mutação conforme descrito na Figura 4.10. Após selecionar duas operações aleatórias no indivíduo, por exemplo, T_2 e T_4 . O descendente será construído trocando-se as associações correspondentes entre as duas operações, conforme mostrado na Figura 4.11

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
(1,1,4,1)	(1,2,2,1)	(2,2,1,2)	(2,1,4,4)	(3,2,2,1)	(3,1,1,3)	(1,3,3,3)	(2,3,4,1)

Figura 4.10: Indivíduo selecionado para Mutação por Associação

T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
(1,1,4,1)	(1,2,4,4)	(2,2,1,2)	(2,1,2,8)	(3,2,2,1)	(3,1,1,3)	(1,3,3,3)	(2,3,4,1)

Figura 4.11: Descendente gerado pela Mutação por Associação

Operador BWMM

O objetivo deste operador de mutação é associar operações a máquinas que estejam com baixa carga de trabalho. Assim, tendo identificado a máquina (mmc) que apresenta a menor carga de trabalho, o objetivo deste operador é atribuir a operação *melhor* à mmc , considerando operações à direita ou à esquerda da última operação, pos_{ultima} , associada à mmc .

A direção da busca de *melhor* depende do índice de pos_{ultima} no indivíduo. Se o índice é maior que $N/2$, *melhor* será obtida considerando as operações à esquerda de pos_{ultima} . Caso contrário, índice menor que $N/2$, *melhor* será obtida considerando somente as operações à direita de pos_{ultima} . Em ambos os casos, somente operações não associadas à mmc são consideradas.

Esta regra leva em consideração que uma máquina que apresenta baixa carga de trabalho, na qual a última operação atribuída está no início do indivíduo, é uma máquina que rapidamente torna-se disponível e não executa nenhuma outra operação. Por outro lado, máquinas com baixa carga de trabalho, nas quais a última operação atribuída encontra-se no final do indivíduo, são máquinas muito disponíveis no início do tempo de processamento do FJSP.

A operação *melhor* é aquela com o menor tempo de processamento somado ao tempo de configuração e que pode ser atribuída a mmc . A máquina mmc é determinada a partir de uma posição aleatória do vetor $Vet_workload$, para evitar que sempre sejam selecionadas as máquinas com menor índice neste vetor. O Algoritmo 5 descreve este operador.

O resultado da aplicação deste algoritmo é apresentado na Figura 4.12, onde I_1 é o indivíduo selecionado para mutação e D_1 o descendente gerado. Neste exemplo, a máquina 3 apresenta a menor carga, após a aplicação do operador, a operação O_{21} foi associada à máquina 3, em D_1 .

Operador BWMC

A partir de dois indivíduos I_1 e I_2 , os descendentes D_1 e D_2 são gerados através de comparações das cargas das máquinas entre I_1 e I_2 . Para cada operação O_{ijk} em I_1 , compara-se a carga de trabalho de k entre I_1 e I_2 . Caso W_k seja menor em I_1 , adiciona-se a D_1 a associação O_{ijk} de I_1 . Caso contrário, W_k é menor em I_2 , então

Algoritmo 5 Operador_BWMM()

```

1:  $I$  = Um indivíduo TSL selecionado na população para mutação
2:  $mmc$  = Máquina com a menor carga de trabalho em  $I$ 
3:  $z = N$ 
4: while ( $\mu(I[z]) \neq mmc$ ) faça
5:    $z = z - 1$ 
6:    $pos_{ultima} = z$ 
7: fim while
8:  $menortempo = +\infty$ 
9: se  $pos_{ultima} < N/2$  então
10:  para ( $z = 0$  até  $pos_{ultima}$ ) faça
11:    se  $p_{i,j,mmc}[z] + st_{xy} < menortempo$ , onde  $O_{xy}$  antecede  $O_{ij}$  em  $mmc$  então
12:       $menortempo = p_{i,j,mmc}[z] + st_{xy}$ 
13:       $melhor = I[z]$ 
14:    fim se
15:  fim para
16: senão
17:  para ( $z = pos_{ultima}$  até  $N$ ) faça
18:    se  $p_{i,j,mmc}[z] + st_{xy} < menortempo$ , onde  $O_{xy}$  antecede  $O_{ij}$  em  $mmc$  então
19:       $menortempo = p_{i,j,mmc}[z] + st_{xy}$ 
20:       $melhor = I[z]$ 
21:    fim se
22:  fim para
23: fim se
24: Associe  $melhor$  a  $mmc$ 

```

I_1							
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
(1,2,1,4)	(1,1,3,4)	(2,2,2,3)	(2,1,1,3)	(3,2,4,2)	(1,3,4,5)	(3,1,2,5)	(2,3,1,4)

D_1							
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
(1,2,1,4)	(1,1,3,4)	(2,2,2,3)	(2,1,3,2)	(3,2,4,2)	(1,3,4,5)	(3,1,2,5)	(2,3,1,4)

Figura 4.12: Operador BWMM

$O_{i,j,l}$ é associada a D_1 , onde l é a máquina associada a O_{ij} em I_2 . Para a geração de D_2 , o procedimento é repetido baseado no indivíduo I_2 . Ao final da execução

Algoritmo 6 Operador_BWMC()

- 1: I_1 = Um indivíduo TSL aleatório na população
 - 2: I_2 = Outro indivíduo TSL aleatório na população
 - 3: $Vet_workload_1$ = Vetor com a carga de trabalho de cada máquina do I_1
 - 4: $Vet_workload_2$ = Vetor com a carga de trabalho de cada máquina do I_2
 - 5: **para** ($i = 1$ até N) **faça**
 - 6: **se** $Vet_workload_1[\mu(I_1[i])] \leq Vet_workload_2[\mu(I_1[i])]$ **então**
 - 7: Inclua $I_1[i]$ em D_1
 - 8: **senão**
 - 9: Encontre a operação $I_1[i]$ em I_2 e inclua em D_1
 - 10: **fim se**
 - 11: **se** $Vet_workload_2[\mu(I_2[i])] \leq Vet_workload_1[\mu(I_2[i])]$ **então**
 - 12: Inclua a operação $I_2[i]$ em D_2
 - 13: **senão**
 - 14: Encontre a operação $I_2[i]$ em I_1 e inclua em D_2
 - 15: **fim se**
 - 16: **fim para**
-

Ao final da execução do algoritmo são gerados dois novos indivíduos que mantêm boas associações dos pais. O algoritmo 6 descreve este operador. Considerando os indivíduos selecionados para o cruzamento da Figura 4.13, os descendentes gerados pela aplicação deste operador são apresentados na Figura 4.14.

I_1							
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
(1,1,4,1)	(1,2,2,1)	(2,2,1,2)	(2,1,4,1)	(3,2,2,1)	(3,1,1,3)	(1,3,3,3)	(2,3,4,1)

I_2							
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
(1,2,1,4)	(1,1,3,4)	(2,2,2,3)	(2,1,1,3)	(3,2,4,2)	(1,3,4,5)	(3,1,2,5)	(2,3,1,4)

Figura 4.13: Indivíduos selecionados para o cruzamento BWMC

D_1							
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
(1,1,4,1)	(1,2,2,1)	(2,2,1,2)	(2,1,4,1)	(3,2,2,1)	(3,1,1,3)	(1,3,3,3)	(2,3,4,1)

D_2							
T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
(1,2,2,1)	(1,1,4,1)	(2,2,1,2)	(2,1,4,1)	(3,2,2,1)	(1,3,3,3)	(3,1,1,3)	(2,3,4,1)

Figura 4.14: Descendentes gerados por BWMC

4.2.4 Seqüenciamento das operações

Após as associações das operações, a definição da seqüência das operações nas máquinas é gerada através do Algoritmo 7, utilizando os vetores $DM[k]$ e $DJ[j]$. Estes vetores armazenam, respectivamente, os tempos de disponibilidade de cada máquina e de cada *job*, após a associação de uma operação.

O *schedule* é construído percorrendo o indivíduo da esquerda para a direita. A cada associação em TSL, define-se o sequenciamento das operações nas máquinas e o tempo de início do processamento é calculado considerando o máximo valor entre $DM[k]$ e $DJ[j]$. O tempo de término é definido adicionado-se a este valor o tempo de processamento da respectiva operação. Em BTSL o algoritmo de decodificação é muito simples, pois a ordem dos gens em TSL determina também o sequenciamento das operações nas máquinas. As regras de precedência entre as operações de um mesmo *job* são garantidas na geração da população inicial e preservadas na aplicação dos operadores genéticos.

Algoritmo 7 SchedulingProcedureTSL()**Entrada:** I ; um indivíduo da população

```

1: para ( $k = 1$  até  $m$ ) faça
2:    $Disp\_Maquina[k] = 0$ 
3: fim para
4: para ( $j = 1$  até  $n$ ) faça
5:    $Disp\_Job[j] = 0$ 
6: fim para
7: para ( $z = 1$  até  $N$ ) faça
8:    $k = \mu(I[z])$ 
9:    $t_{i,j} = \text{Max}(Disp\_Maquina[k], Disp\_Job[j]) + st_{xy}$ , onde  $O_{xy}$  antecede  $O_{ij}$  em  $k$ 
10:   $Disp\_Maquina[k] = t_{i,j} + p_{i,j,k} + st_{xyk}$ 
11:   $Disp\_Job[j] = t_{i,j} + p_{i,j,k} + st_{xyk}$ 
12: fim para

```

4.3 Resultados e Discussões

Para verificar o desempenho de BTSL, primeiro realizou-se uma comparação dos resultados com outros métodos, sem considerar tempos de configuração dependente de seqüência. Em seguida, apresenta-se os resultados considerando tempo de configuração dependente de seqüência. Essa divisão foi realizada, pois durante a pesquisa não foram encontrados trabalhos que consideraram FJSP com tempo de configuração dependente de seqüência.

O objetivo desta comparação é verificar os valores de cada um dos critérios otimizados. Para os testes considerou-se um conjunto¹ de 4 problemas T-FJSP, utilizados e obtidos em Kacem et al. [34] que também considera como critérios de otimização o *makespan*, a carga total das máquinas e a máquina mais carregada. A Tabela 4.2 apresenta o resultado da comparação. Portanto, pode-se notar que os valores apresentados por BTSL referentes ao *makespan* (C_{max}), superam em muito os valores obtidos por Kacem et al. [34]. Quanto aos valores obtidos para a carga total das máquinas (W) e para a carga da máquina mais carregada (W_k), os valores obtidos por BTSL igualam ou se aproximam muito dos valores conseguidos por

¹Os problemas estão descritos no Apêndice A, nas Tabelas A.1, A.2, A.3 e A.5, respectivamente.

Kacem et al. [34].

Vale lembrar que os valores obtidos por Kacem et al. [34] são limites inferiores, isto é, não correspondem a uma única solução, mas aos menores valores obtidos para cada critério, que podem ter sido obtidos em soluções diferentes. Em BTSL, considerou-se apenas soluções que minimizavam os critérios simultaneamente.

Tabela 4.2: Comparação de resultados da otimização multi objetivo

	Kacem [34]			BTSL		
Instância	C_{max}	W	W_k	C_{max}	W	W_k
T-FJSP(4x5)	16	32	7	11	32	10
T-FJSP(10x7)	15	60	9	11	61	11
T-FJSP(10x10)	7	41	5	7	42	5
T-FJSP(15x10)	23	91	10	12	91	11

Ainda considerando problemas T-FJSP, a proposta BTSL foi comparada com trabalhos mais recentes, realizados por Ho et al. [28] e Ong et al. [43]. Entretanto, estes trabalhos consideram apenas a otimização do critério *makespan*. A tabela 4.3 mostra os resultados desta comparação, onde pode-se notar que BTSL iguala ou supera os valores alcançados por Ho et al. [28] e Ong et al. [43], com a vantagem de executar uma otimização multiobjetivo. Estes resultados corroboram o ganho de qualidade obtido com a aplicação de BTSL.

Tabela 4.3: C_{max} sem tempo de configuração

Instância	Kacem [34]	Genace [28]	Clonal [43]	BTSL
T-FJSP(4x5)	16	11	11	11
T-FJSP(10x7)	15	12	11	11
T-FJSP(10x10)	7	7	7	7
T-FJSP(15x10)	23	12	12	12

Outra análise importante realizada é o comportamento de BTSL na resolução de problemas P-FJSP. Para isso, foi realizada uma comparação dos valores para o *makespan* de outros métodos que otimizam apenas este critério. Os problemas foram extraídos de Ho et al. [28] e consideram instâncias de problemas elaboradas

por Brandimarte [9]. A comparação dos resultados com os métodos de Ho et al. [28] e Ong et al. [43] está descrita na Tabela 4.4. A partir desta tabela, é possível verificar que o desempenho de BTSL foi inferior aos resultados apresentados pelos trabalhos acima, entretanto em quase todas as instâncias de problema superou os resultados apresentados por Brandimarte [9].

Tabela 4.4: P-FJSP Valores para makespan sem tempo de configuração

Instância	Brandimarte [9]	Genace [28]	Clonal [43]	BTSL
Mk01(10x6)	42	41	39	45
Mk02(10x6)	32	29	27	30
Mk04(15x8)	81	67	70	76
Mk05(15x4)	186	176	173	180
Mk06(10x15)	86	68	65	77
Mk07(20x5)	157	148	145	153
Mk08(20x10)	523	523	523	531
Mk09(20x10)	369	328	311	355
Mk10(20x15)	296	231	-	255

Como dito anteriormente, para os resultados apresentados considerando tempo de configuração, não foi possível realizar uma comparação com outros métodos, uma vez que nenhum trabalho considerando FJSP com tempo de configuração dependente de sequência foi encontrado na literatura. Então, somente uma análise dos valores obtidos para cada critério otimizado foi realizada, além de uma comparação com o mesmo problema sem a ocorrência de tempo de configuração. A Tabela 4.5 mostra o resultado da aplicação do algoritmo para os quatro problemas T-FJSP já mencionados, considerando tempo de configuração dependente de sequência.

A introdução de tempo de configuração incrementa diretamente o *makespan* em função do atraso gerado na disponibilidade das máquinas. Em relação à carga de trabalho total das máquinas e da máquina mais carregada um impacto proporcionalmente menor foi verificado, o que prova que o algoritmo encontra boas associações mesmo sob a influência do tempo de configuração.

Os tempos de configuração são apresentados em uma matriz $N \times N$, onde cada linha identifica a operação a ser executada na máquina imediatamente antes da

Tabela 4.5: Resultados com tempo de configuração

Instância	C_{max}	W	W_k	TST
T-FJSP(4x5)	15	32	10	18
T-FJSP(10x7)	16	63	11	38
T-FJSP(10x10)	14	44	6	45
T-FJSP(15x10)	21	98	12	73

operação correspondente a da coluna. Os tempos de configuração são valores inteiros aleatoriamente selecionados no intervalo $[1, 3]$. A diagonal principal representa o tempo de configuração necessário quando a operação da linha e coluna é a primeira máquina a ser executada. Um exemplo da matriz de tempo de configuração é apresentado na Tabela A.6 do Apêndice A, , considerando o problema da Tabela A.1.

Observe que o tempo de configuração é definido entre quaisquer duas operações consecutivas em qualquer máquina. Foi assumido que o tempo de configuração entre duas operações, em todas as máquinas, é o mesmo. Isto pode ser modificado facilmente criando-se uma matriz para cada máquina.

Para introduzir tempo de configuração em FJSP poucas alterações foram realizadas, restringindo-se à função de adaptação, aos operadores genéticos BWMM e BWMC e, à geração da população inicial. Estas modificações se relacionam à seleção da melhor máquina para associar cada operação, que deve considerar o valor do respectivo tempo de configuração.

Outras operadores com conhecimento específico do problema foram criados, como por exemplo operadores de mutação que atuam sobre a máquina com maior carga de trabalho e outro para atuar sobre o job com maior tempo de processamento efetivo, sempre no sentido diminuir estes valores. Um operador de cruzamento que também atua sobre os *jobs* com maior tempo de processamento efetivo também foi desenvolvido. Entretanto, nos testes de desempenho realizados estes operadores não contribuíram para a melhoria dos resultados e, por isso, não foram considerados neste trabalho.

A população inicial é composta de S soluções, onde cada metade foi gerada por um dos algoritmos discutidos na Seção 4.2.2. As probabilidades de mutação,

cruzamento e elitismo foram definidas da seguinte maneira: $P_{cruzamento} = 0.75$, $P_{mutacao} = 0.15$ e $P_{elitismo} = 0.10$. Os indivíduos de uma nova população foram gerados aplicando-se os vários operadores genéticos proporcionalmente, ou seja, a quantidade de indivíduos gerados por cada operador é igual e considera o tamanho da população e as respectivas taxas de cruzamento e mutação. A Tabela 4.6 apresenta os valores dos parâmetros genéticos utilizados para cada uma das instâncias do problema utilizado na comparação dos resultados. Com o intuito de permitir comparações futuras com relação ao tempo computacional os valores gastos na resolução de cada problema também são apresentados.

Tabela 4.6: Parâmetros Genéticos

Instância	População	Gerações	Tempo (seg.)	
			Sem Config.	Com Config.
T-FJSP(4x5)	100	200	4	5
T-FJSP(10x7)	100	200	9	9
T-FJSP(10x10)	100	200	9	10
T-FJSP(15x10)	100	200	15	18
Mk01(10x6)	100	300	22	-
Mk02(10x6)	100	300	23	-
Mk04(15x8)	100	300	37	-
Mk05(15x4)	100	300	43	-
Mk06(10x15)	150	500	144	-
Mk07(20x5)	150	500	93	-
Mk08(20x10)	150	500	228	-
Mk09(20x10)	150	500	240	-
Mk10(20x15)	150	500	244	-

Capítulo 5

Aplicação de Escalonamento a ambientes de *Workflow*

Como dito anteriormente, são vários os ambientes onde surgem a necessidade de realização de escalonamento. Este capítulo apresenta uma proposta para aplicação de técnicas de escalonamento a ambientes de *workflow*, promovendo a integração destas duas áreas do conhecimento. De acordo Tramontina [51], estas possuem pontos de sinergia que, se explorados, podem melhorar a eficiência dos ambientes de *workflow*.

Tramontina [51] desenvolve um estudo que considera a substituição de regras de prioridades, principalmente FIFO, como política de alocação das atividades, por técnicas de escalonamento. Também apresenta um mapeamento entre os conceitos da área de escalonamento e *workflow* como uma proposta que permite a integração das duas áreas.

Neste trabalho a integração de escalonamento e *workflow* não ocorre na fase de execução dos fluxos, como explorado por Tramontina [51], mas sim na fase de definição dos modelos de processos. Para mostrar a viabilidade desta integração, utiliza-se o Workflow Genético, uma arquitetura para a geração automática de modelos de processos que utiliza técnicas de planejamento combinadas com AGs.

A definição e a identificação de bons modelos de processos aumentam a eficiência dos SGWf e da organização como um todo. Como são vários os modelos de processos gerados, resta ainda a necessidade de identificar entre estes os mais adequados para implementação na organização.

Assim, este capítulo apresenta os componentes da arquitetura *Workflow Genético*, em especial, o Simulador Genético, uma ferramenta baseada no Escalonador Genético, descrito no Capítulo 4, aplicado a ambientes dinâmicos, para identificar modelos de processo mais adequados. Na seção 5.1 são apresentados os conceitos fundamentais e os elementos que compõem um *workflow*. Na seção 5.2 mostra-se o mapeamento entre os conceitos de *workflow* e escalonamento e a proposta de integração das duas áreas. Na seção 5.3 detalhes da arquitetura *Workflow Genético* são apresentados e, finalmente, na seção 5.4, os resultados obtidos.

5.1 Conceitos de *Workflow*

Este trabalho, considera a definição de *workflow* proposta pela WfMC [54]. De acordo com esta entidade, “*Workflow é a automação de processos de negócios, no todo ou em parte, no qual documentos, informações ou atividades são passadas de um participante para outro, de acordo com um conjunto de regras*”.

A execução de atividades complexas como, por exemplo, aquelas relacionadas aos sistemas de produção, a entrega de um serviço ou a geração de conhecimento, exige geralmente a participação de mais de um recurso para atingir o objetivo. Isto é um fato comum em empresas, onde há negócio ou processos de produção definidos por conjuntos de tarefas que cooperam para atingir o resultado planejado. Neste contexto, as ferramentas de *workflow* têm um papel importante, desde que estão associadas à automação e à gerência de processos.

O conjunto de atividades necessárias para a realização de um trabalho é denominado processo que, também, pode ser entendido como um procedimento em que documentos, informações e tarefas são passados entre os participantes para a realização de um objetivo de negócio. Tem-se, então, o que se denomina processo de negócio [20, 53]

Os processos de negócio são representados por fluxos de trabalho, ou seja, modelos que especificam as atividades que compõem o processo, a ordem e as condições que as atividades devem ser executadas, os executores de cada atividade, as ferramentas a serem utilizadas e as informações manipuladas durante a execução. Os fluxos de trabalho, que representam os processos de negócio, podem ser interpretados/automatizados por sistemas de *workflow* como mostrado na Figura 5.1. Um

sistema de gerência de *workflow* é conhecido por SGWf.

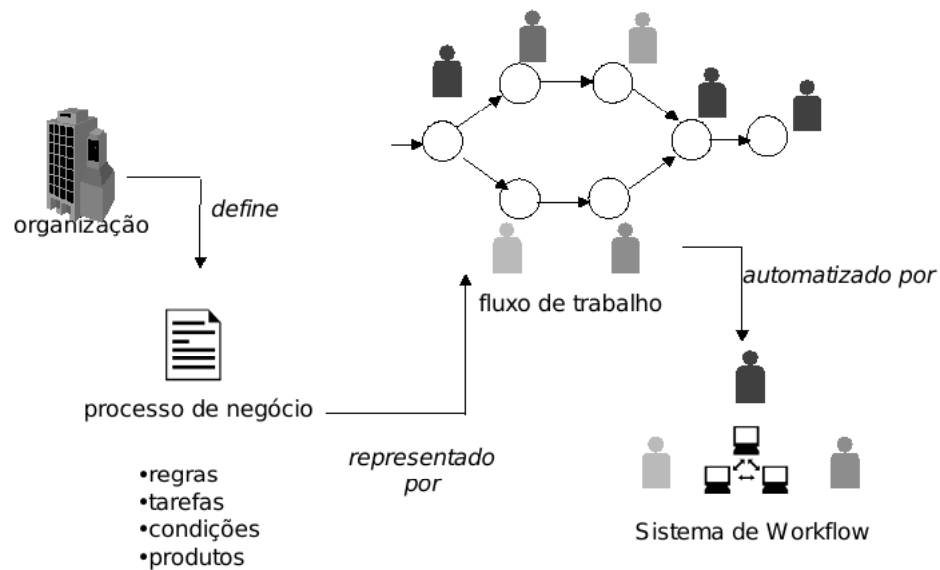


Figura 5.1: Processos de negócio automatizados por sistemas de workflow

Definir um processo de negócio significa traduzí-lo do mundo real para uma formalização computacional e processável através do uso de uma ou mais técnicas de análise e modelagem. O resultado da definição é um modelo ou a representação do processo a ser executado. Uma vez definido, um processo pode ser executado através da interpretação de sua definição por um sistema de *workflow*. Esta interpretação compreende a atribuição das atividades definidas entre os recursos designados para executá-las.

Existem no mercado vários produtos diferentes que se colocam como ferramentas de gerência de *workflow*, que possuem características comuns em relação às suas funções principais [51]. A WfMC identifica que, em um nível mais alto, todos os SGWfs têm as seguintes funcionalidades, que se interagem de acordo com o diagrama da Figura 5.2.

- **Modelagem** - funções de “tempo de construção” que lidam com a definição e modelagem dos processos de *workflow* e as tarefas constituintes. A modelagem de processo em *Workflow*, consiste na organização das atividades inerentes ao problema com os recursos responsáveis pela execução, de forma a conduzir o processo do estado inicial até o estado meta.

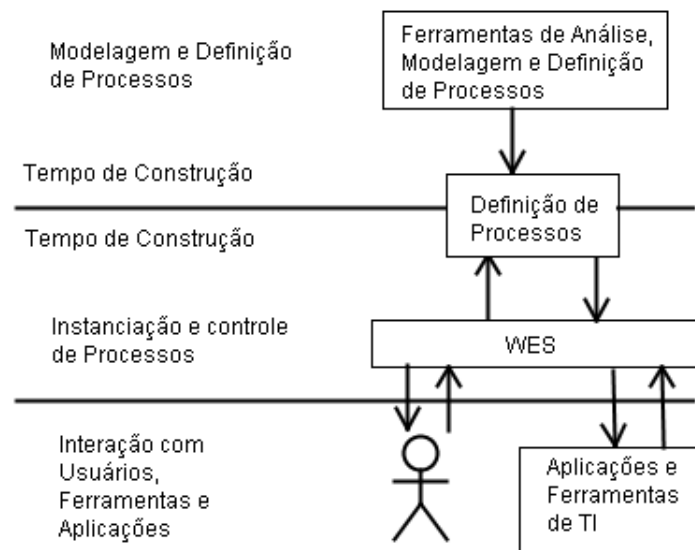


Figura 5.2: Relação entre as principais funções de um SGWf

- **Execução** - funções de “tempo de execução” que gerenciam e executam os processos de workflow em um ambiente operacional, além de sequenciar e distribuir as tarefas necessárias para a execução desses processos. Essas funções geram uma instância de processo para lidar com algum objetivo específico, distribuir as tarefas entre os usuários participantes e acompanhar sua execução.
- **Monitoramento** - funções de “interação”(em tempo de execução) com os usuários participantes e com ferramentas computacionais para o processamento dos vários passos de cada tarefa.

Em processos de negócio, o trabalho é direcionado aos participantes, também chamados de recursos [53, 54]. Portanto, recurso é o nome genérico dado a uma pessoa, máquina ou grupo de pessoas e/ou máquinas que podem realizar tarefas específicas.

A definição de um fluxo de trabalho compreende também a descrição do encadeamento de atividades do processo. Atividade em um fluxo de trabalho podem ser executadas sequencialmente (uma após a outra), em paralelo ou sob determinadas condições. Quando executadas sequencialmente, logo que uma atividade é terminada, a atividade subsequente é ativada e roteada para execução pelo recurso responsável, como mostrado na Figura 5.3(a). Quando definidas para a execução em paralelo, duas atividades são ativadas e colocadas em execução simultaneamente, um

exemplo deste caso é mostrado na Figura 5.3(b). Roteamentos condicionais também podem ser definidos em um fluxo de trabalho, como apresentado na Figura 5.3(c). A definição de uma condição implica na tomada de alguma decisão baseada nas informações da execução do processo disponíveis, decisão esta que também deve fazer parte da definição do processo.

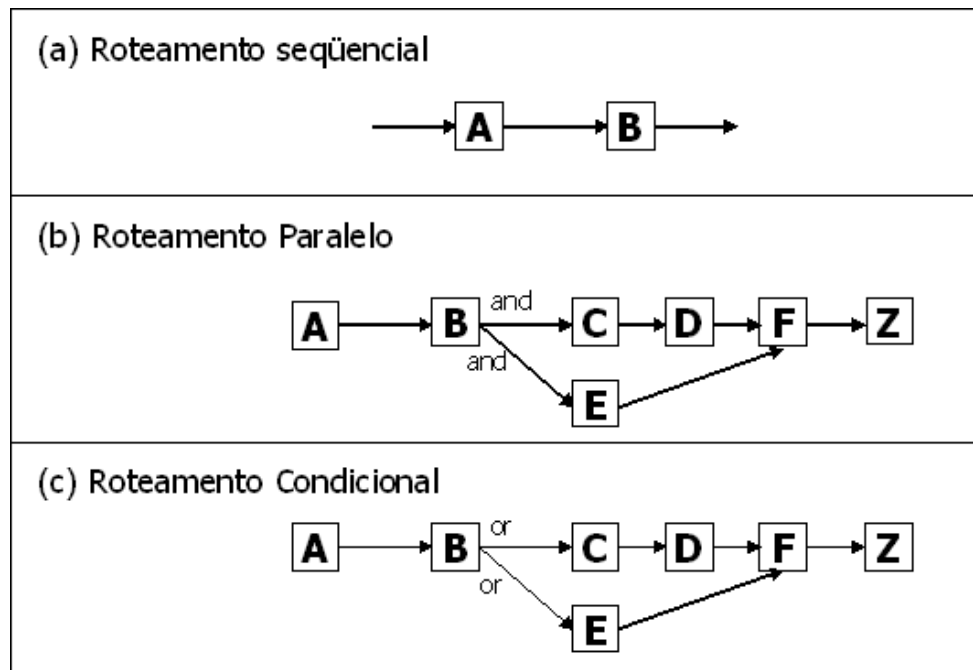


Figura 5.3: Tipo de rotas em um fluxo de trabalho

Fluxos de trabalho são executados através da ativação de instâncias. A partir de uma mesma definição de fluxo de trabalho, instâncias diferentes podem ser colocadas em execução. Várias instâncias de um mesmo processo ou de processos distintos podem estar em execução simultaneamente em um sistema de *workflow*. Este acompanha e coordena a execução de cada uma das instâncias ativas, seguindo o fluxo definido no modelo de processo e encaminhando cada atividade para os recursos correspondentes.

O encaminhamento de atividade aos seus respectivos executores provoca a inclusão de itens de trabalho nas listas de trabalhos destes recursos. Uma lista de trabalho contém as atividades a serem realizadas por um recurso específico e, geralmente, contém simultaneamente atividades de várias instâncias de diferentes processos em execução.

Uma instância é a representação de uma única ocorrência de um modelo de

processo em execução. Uma tarefa refere-se a uma especificação genérica de algo a ser feito de maneira manual, automática ou semi-automática. Aalst et. al [53] identificam a tarefa em dois tipos que, dependendo da instância de processo de *workflow*, podem ser um item de trabalho ou uma atividade.

O item de trabalho é a combinação de um caso (instância de processo) e uma tarefa que aguarda para ser executada, conforme a representação da Figura 5.4. Dessa forma, um item de trabalho é uma tarefa que ainda não está em execução, mas pode ser executada a qualquer momento. Em geral, itens de trabalho estão contidos em listas de trabalho (ou *work list*, em inglês) até que sejam selecionados para a execução. Uma atividade, ao contrário do item de trabalho, refere-se à tarefa em estado de realização na instância de processo. Portanto, a atividade constitui-se do item de trabalho que foi selecionado da lista de trabalho e está sendo executado. É importante notar que, diferente da tarefa, o item de trabalho e a atividade estão ligados a uma instância específica.

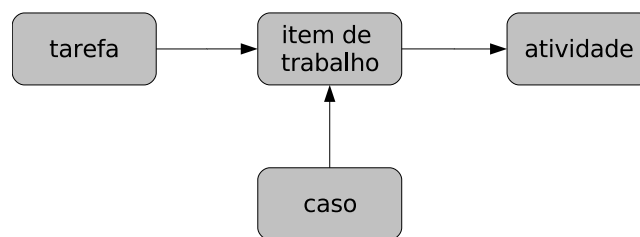


Figura 5.4: Relação entre os termos tarefa, caso, item de trabalho e atividade

Sistemas de workflow são geralmente baseados em uma arquitetura cliente-servidor, onde na porção cliente encontram-se as aplicações para definição de processos, interação, acompanhamento e administração do sistema. Na porção servidora, encontra-se o motor ou “máquina de workflow”, responsável pela interpretação e roteamento do fluxo de trabalho para os diversos recursos/executores do processo. O motor de *workflow* é o principal componente de um SGWf [53], pois fornece o ambiente de execução para os processos e funcionalidades, tais como a criação de instâncias de processos, coordenação e roteamento de atividades, geração de itens de trabalho e controle de recursos.

5.2 Workflow e Escalonamento

Tramontina [51] cita uma série de trabalhos que reconhecem a existência de aspectos de escalonamento em sistemas de workflow [46, 53, 58]. Especialmente Aalst et. al [53] que apresentam aspectos comuns existentes entre estas áreas.

“Se existe um excesso de trabalho em determinados instantes de tempo, não podemos transformar cada item de trabalho em uma atividade imediatamente. Podem existir mais itens de trabalho do que recursos disponíveis. Se este for o caso, então uma escolha deve ser feita quanto à ordem na qual esses itens de trabalho são selecionados.”

Desta maneira, Aalst et. al [53] colocam que o roteamento de um caso de *workflow* através de vários recursos exibe muitas similaridades com o roteamento de um produto através das máquinas de um departamento de produção, onde as técnicas de escalonamento são ferramentas presentes. Tramontina [51] introduz um mapeamento dos conceitos básicos de workflow em conceitos de escalonamento.

Tabela 5.1: Mapeamento entre os conceitos básicos de *workflow* e escalonamento

Workflow	Escalonamento
instância de processo (caso)	job
atividade	operação
aplicações/pessoas	máquinas

A Tabela 5.1 mostra este mapeamento, onde pode-se notar que uma instância de processo sendo executada pelo SGWf é considerada um *job* em um sistema de escalonamento. A execução de uma atividade é vista como um operação de um *job* em uma máquina. Um recurso de *workflow* (aplicações/pessoas e máquinas) é mapeado com uma máquina no ambiente de escalonamento. Esse mapeamento permite a aplicação de técnicas de escalonamento às partes corretas do sistema de *workflow*.

Tramontina [51] apresenta uma lista de problemas, descritos abaixo, que surgem quando se integram as áreas de escalonamento e *workflow*.

- Incerteza quanto ao tempo de execução das tarefas: sistemas de *workflows* lidam com casos de trabalho reais, geralmente envolvendo processos de nível

mais alto que processos de linhas de montagem ou chão de fábrica. Neste tipo de processo é mais difícil ter uma previsão dos tempos das tarefas, principalmente por envolver fatores não controláveis.

- Roteamento Paralelo: ocorre quando, em um mesmo caso, há duas ou mais atividades sendo executadas simultaneamente, caso este não encontrado na literatura de escalonamento.
- Incertezas quanto às rotas dos casos: em um modelo de processos podem ser definidos caminhos distintos que podem ser executados pelas instâncias de processo de acordo com pontos de decisão condicionais(*OR*).
- A escolha da Função Objetivo: em geral em ambientes de escalonamento é utilizado o *Makespan* como critério a ser otimizado. Embora essa também seja uma das funções de um sistema de *workflow*, objetivos relacionados com o atraso dos casos são igualmente relevantes, tais como a porcentagem de casos atrasados ou o atraso médio dos casos.
- Atividades Interrompíveis e Não Interrompíveis (*Preemptive e Non-Preemptive*): muitas das atividades realizadas em ambientes de *workflow* são inerentemente interrompíveis, outras são inerentemente não interrompíveis, sendo que os dois tipos de atividades podem estar presentes em um mesmo modelo de processo. Ambientes de escalonamento consideram geralmente atividades não interrompíveis.
- A Natureza dinâmica dos sistemas de *workflow*: *Workflows* são sistemas dinâmicos, enquanto existem várias instâncias de processo em andamento, novos casos entram e/ou saem o sistema, que deve estar ciente deste fato.
- O mapeamento $m : n$ entre recursos e atividades: Em *workflows* as atividades podem ser executadas por mais de um recurso, mas um recurso também pode executar mais de um tipo de atividade.

5.3 Workflow Genético

A arquitetura do *Workflow Genético* apresentada em Alves et al. [3] é esboçada na Figura 5.5. A primeira etapa *mapeamento atividade(serviço)/ação* (2) consiste em

receber as informações das atividades disponíveis (1) e que estão no formato XPDL e gerar o equivalente em PDDL (Planning Domain Definition Language).

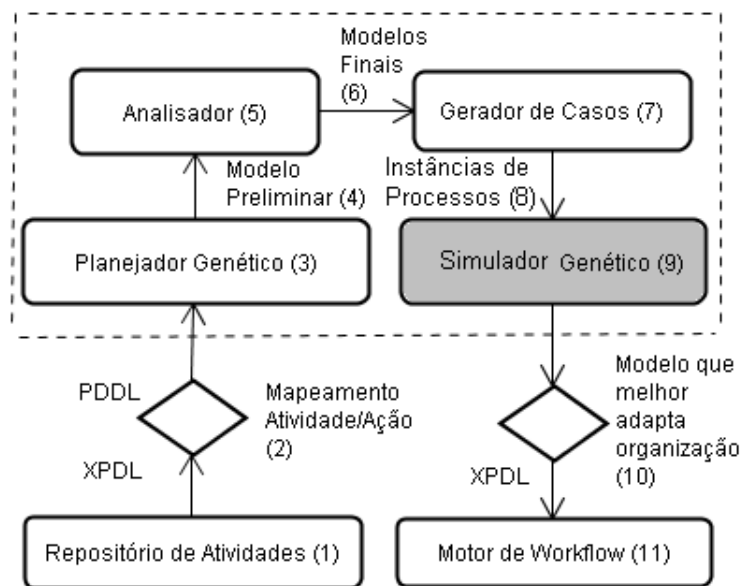
Baseado nas regras de produção das atividades publicadas, o módulo *Planejador Genético* (3) gera um modelo preliminar contendo uma representação unificada de possíveis combinações de modelos (4) de processos. Na geração destas combinações todas as atividades unificáveis, ou seja, todas atividades cujas pré-condições e pós-condições são satisfeitas em determinado estágio do processo são consideradas na representação. O *Planejador Genético* utiliza técnicas de planejamento combinados com AGs para obter modelos de processo.

O Planejador Genético permite evoluir representações de modelos de processos garantindo a corretude do modelo, considerando não somente roteamentos seqüenciais, mas o uso de roteamentos condicionais e paralelos. Todos estes roteamentos são tratados, devido ao uso do paradigma de algoritmo genético e um planejador de objetivo geral, possibilitando considerar problemas de vários domínios do mundo real.

Atividades ou blocos de atividades que executam o mesmo papel definem pontos de conflito e indicam que é possível ter mais de um modelo de processo para o problema. Assim, o módulo *Analizador* (5) tem a responsabilidade de extrair os modelos finais, ou seja, os possíveis modelos de processos válidos. O *Analizador* extrai os modelos finais considerando atividades condicionais (OR), paralelas (AND) e seqüenciais (SEQ).

Cada modelo de processo válido (6) juntamente com as informações dos recursos associados a cada atividade e seus respectivos tempos de execução podem ser encaminhados para simulação utilizando os módulos Gerador de Casos e Simulador Genético. O objetivo do Gerador de Casos (7) é gerar as possíveis instâncias de processos existentes, ou seja, as possíveis rotas do modelo de processo. Na situação onde o Gerador de Casos encontra um roteamento do tipo *OR*, instâncias relativas a cada uma das alternativas possíveis serão geradas. As atividades em roteamentos do tipo *AND*, são colocadas em forma seqüencial e instâncias que alternam a execução dessas atividades são geradas. Para os outros casos onde as atividades do modelo de processo são seqüências, o mapeamento é feito diretamente.

No passo (8), as instâncias de processos geradas são enviadas para o módulo

Figura 5.5: Arquitetura do *Workflow Genético*

Simulador Genético (9) que, juntamente com as informações obtidas dos recursos associados para cada atividade e os respectivos tempos de execução, analisa os modelos mais adequados. Esta análise é realizada através de simulações que utilizam o escalonador proposto neste trabalho. Problemas FJSP e *workflow* são similares em relação a aspectos relativos a restrições de precedência entre as atividades e também pelo fato de que para uma mesma atividade existe mais de um recurso disponível para sua execução. Os modelos gerados (10) são também convertidos para formato XDPL, visando a execução (11) em um motor de *Workflow* que contemple tal representação.

Observa-se que o módulo *Gerador de Casos* é responsável por fornecer uma alternativa às dificuldades “*Roteamento Paralelo*” e “*Incertezas quanto às rotas dos casos*”, uma vez que gera instância de processos distintos para cada uma destas situações. O “*Mapeamento $m : n$ entre recursos e atividades*” é verificado pela própria natureza dos problemas FJSP considerados neste trabalho. A “*Natureza dinâmica*” dos ambientes de *workflow* é atendida pelo Simulador Genético, responsável por identificar os casos que entram e saem do sistema. Mas, em relação à natureza das atividades, são consideradas apenas atividades não interrompíveis.

5.3.1 Simulador Genético

Um ambiente de simulação é um eficiente meio de se estudar e prever o desempenho de processos, como apresentado por Laguna et. al [36]. Os modelos de processos gerados podem apresentar características ausentes neste trabalho, como por exemplo, a ocorrência de fluxos condicionais e paralelos. Para superar esta dificuldade é utilizado um método que extrai as instâncias de processos sequências presentes no modelo de processo com operações paralelas. Para modelos de processos com fluxo condicional, são geradas instâncias de processo distintas para cada opção do fluxo condicional. Desta maneira, elimina-se estes aspectos que não foram abordados neste trabalho, e permite a utilização de técnicas de escalonamento visando identificar os modelos de processo com características mais adequadas para serem implementados nos sistemas SGWf.

Este módulo é responsável por identificar o modelo mais adequado entre os modelos gerados pelo Analisador. Esta análise considera a capacidade de cada recurso executar várias atividades do modelo de processo e aplica o escalonador FJSP proposto neste trabalho. O objetivo final do Escalonador Genético é identificar entre os modelos de processos gerados pelo Analisador os mais adequados para a organização.

Para tanto, simula-se a execução de cada instância de processo, atribuindo as atividades a um dos recursos aptos para executá-las. Esta atribuição e o sequenciamento de cada atividade aos respectivos recursos são realizados pelo escalonador FJSP, descrito no capítulo 4.

Ambientes de workflow são dinâmicos, ou seja, enquanto existem várias instâncias de processo em andamento, novas instâncias de processo entram e/ou saem do sistema. Portanto, é necessário a cada evento deste realizar reescalonamento, retirando atividades que já foram concluídas e incluindo atividades que compõem a nova instância de processo. Neste sentido, são utilizadas técnicas de escalonamento e reescalonamento propostas por Bierwirth et. al [8].

O módulo Simulador Genético foi dividido em dois sub-módulos. O primeiro é o Gerenciador de Casos, responsável por decompor o problema dinâmico gerado a cada instante em um conjunto de problemas estáticos distribuídos ao longo do tempo da simulação. Cada problema estático é formado pelo conjunto de atividades pertencentes ao ambiente de simulação, que ainda não iniciaram sua execução, e

pelas atividades pertencentes à nova instância de processo.

O segundo sub-módulo refere-se ao Escalonador, responsável por resolver cada problema de escalonamento estático gerado, determinando o tempo de início e a associação de cada atividade ao respectivo recurso para executá-la. Então, após a solução de cada problema estático, pode-se determinar quais atividades devem sair do ambiente de simulação quando um novo caso entra. Este processo é repetido até que todos os casos com as respectivas atividades tenham sido escalonados.

Gerenciador de Casos

Este módulo determina então o intervalo de tempo entre a chegada de duas instâncias de processo consecutivas. Este intervalo determina a carga de trabalho e o dinamismo do sistema. A carga de trabalho do sistema é definida como sendo o número de operações do sistema que estão aguardando para serem processadas [8], e tem um papel importante quando se analisa o desempenho de um dado sistema. Este intervalo é calculado usando um modelo proposto em Bierwirth et al. [8]. Se estes intervalos são pequenos existirão mais instâncias de processo para serem executadas, provocando um incremento da carga de trabalho. Neste modelo o tempo médio entre a chegada de dois casos consecutivos, λ , é calculado através do tempo de processamento esperado, $\bar{p}$, para um caso qualquer, o número de recursos m do sistema e a taxa da carga de trabalho U . Se $\bar{p}$ é dado, então λ é calculado de acordo com a Fórmula 5.1.

$$\lambda = \bar{p}/mU \quad (5.1)$$

Desta maneira, um novo problema estático FJSP é gerado a cada intervalo de tempo, considerando uma lista de recursos capazes de executar cada atividade com seus respectivos tempos, informações estas geradas pelo Gerador de Casos. Para fins de simulação, $\bar{p}$ é o tempo de processamento médio esperado para cada caso, é calculado através do somatório da média dos tempos de execução esperados para cada atividade do caso em cada recurso. Desta forma, o valor de λ depende apenas de U .

Uma vez determinado o intervalo de tempo, o próximo problema de escalonamento FJSP estático é obtido do problema anterior removendo-se as atividades com

o tempo inicial de execução menor que o tempo corrente e incluindo as atividades do novo caso em posições aleatórias de cada indivíduo. Na remoção das atividades já iniciadas, os recursos associados estarão indisponíveis para executarem atividades subsequentes até o tempo de término das atividades removidas. Esta restrição é implementada incrementando-se a data de liberação (*release date*) da instância de processo e do recurso.

Para as novas instâncias, a data de liberação é o tempo corrente e a data de entrega (*due date*) é o tempo no qual se espera que o caso seja concluído, calculado adicionando-se ao tempo corrente o tempo médio para execução do caso. Já o tempo de término, que é o tempo em que a última atividade do caso sai do sistema, é determinado pelo módulo Escalonador, de acordo com a decomposição do problema dinâmico e com os casos que saem do sistema.

As datas de entrega e de término de cada instância são importantes para se determinar o seu atraso, que é calculado de acordo com a Equação 2.2. O atraso médio de todas as instâncias de cada modelo de processo foi considerado como critério para se determinar a qualidade do modelo de processo.

O modelo de processo mais adequado é então determinado pelo Gerenciador de Casos, considerando variáveis relacionadas com a data de entrega como sugerido por Tramontina et al. [6]. Então a função objetivo que identifica o melhor modelo de processo utiliza o percentual de instâncias de processo atrasadas para cada modelo ao final da simulação, de acordo com a carga de trabalho.

Escalonador

Este módulo é responsável por resolver cada problema de escalonamento estático gerado pelo Gerenciador de Casos e que contém todas as instâncias de processo, em execução em um dado instante, gerando um problema FJSP.

Para o primeiro problema FJSP gerado, assume-se que todos os recursos estão disponíveis no tempo zero, cada recurso pode executar somente uma atividade ao mesmo tempo e a ordem fixa das atividades na instância de processo não pode ser alterada (restrição de precedência). A função multi objetivo utilizada em cada problema considera os três critérios Makespan (M_s), carga total das máquinas (W), a máquina mais carregada (W_k) e o atraso médio das instância de processo.

Os aspectos relacionados ao tempo de configuração não são considerados, uma vez que este conceito não se aplica a ambientes de *workflow*. Outra importante diferença está relacionada ao tempo no qual o problema FJSP é gerado. Para considerar este tempo é necessário incrementar a data de liberação de cada caso e também o tempo de disponibilidade dos recursos. Desta forma, o tempo de início do processamento é compatível com o tempo do simulador.

5.4 Resultados e Discussões

O exemplo utilizado para teste da arquitetura proposta considera um conjunto de atividades conforme apresentado na Tabela 5.2. Pode-se verificar que cada uma das atividades é definida através de pré e pós condições. Estas informações são utilizadas pelos módulos Planejador Genético e Analisador a fim de construir modelos de processo que atendam a estas restrições.

Tabela 5.2: Especificação das atividades do repositório

Atividades	Pre_Condições	Pos_Condições
(A)		a_1
(B)	a_1	b_1 e b_2
(C)	b_1	c_1
(D)	c_1	d_1
(E)	b_2	x_1
(G)	b_2	g_1
(H)	g_1	x_1
(F)	d_1 e x_1	f_1
(Z)	f_1	

O resultado do Planejador Genético é um modelo preliminar M contendo uma representação de processos válidos, como apresentado na Figura 5.6. Esta representação pode conter mais de um modelo e, o módulo Analisador é responsável por extrair todos os modelos de processos válidos de M . Considerando o modelo M da Figura 5.6, pode-se notar que as pré-condições da atividade E e G são similares, assim como as pós-condições de E e H também são similares. Logo, Analisador constata que o bloco de atividades G e H executa o mesmo papel que a atividade

E , caracterizando a existência de mais de um modelo de processo. O resultado da aplicação do módulo Analisador no modelo preliminar M , são os modelos de $M1$ e $M2$ apresentados na figura 5.7. Os modelos de processo gerados pelo analisador garantem que todas as pré e pós condições das atividades constituintes são atendidas e o resultado final desejado é alcançado, como apresentado por Alves [3].

Para cada um dos modelos de processo gerados, no caso $M1$ e $M2$, o módulo Gerador de Casos extrai as instâncias de processo possíveis geradas a partir destes modelos. Na Figura 5.8 são representados as instâncias de processos extraídas do modelo $M1$.

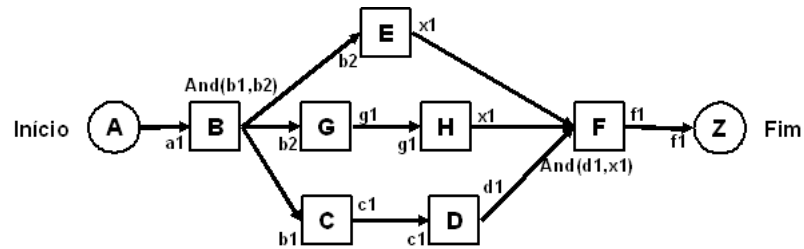


Figura 5.6: Modelo preliminar gerado pelo Módulo Genético

Concluída a extração das instâncias de processo de um modelo, a simulação para o modelo é realizada, incluindo-se um conjunto de cada instância de processo no Simulador Genético. Nos testes realizados foram gerados 10 fluxos para cada instância de processo obtida, os fluxos gerados para cada instância foram então intercalados para serem submetidos ao processo de simulação. Para a execução das atividades destes fluxos considerou-se a utilização de 5 recursos distintos. O tempo para execução das atividades em cada recurso, é um valor aleatório gerado no intervalo $[1, 10]$, e os tempos considerados estão apresentados na Tabela 5.3. Em problemas do mundo real estes tempos podem ser obtidos a partir do histórico de execução de cada modelo de processo, ou ainda, ser estimados a partir dos tempos de execução esperados para cada atividade em cada recurso. O desempenho de cada modelo de processo de processo foi verificado considerando-se três valores para a carga de trabalho U , 45, 75 e 90, simulando o desempenho de cada modelo em ambientes de baixa, média e alta carga de trabalho.

Entre os diferentes testes realizados, os resultados apresentados consideraram que o tempo médio de execução da atividade E é 50 % superior aos tempos das

Tabela 5.3: Tempos de processamento

Atividade	R_1	R_2	R_3	R_4	R_5
B	1	4	6	9	3
C	5	2	8	9	4
D	1	1	3	4	8
E	8	4	5	9	8
F	2	5	1	5	6
G	4	4	7	4	3
H	3	4	4	7	3

atividades equivalentes G e H . Observe que E é uma atividade equivalente a G e H e que, por isso, pode ser mais complexa e assim ter maior tempo de processamento.

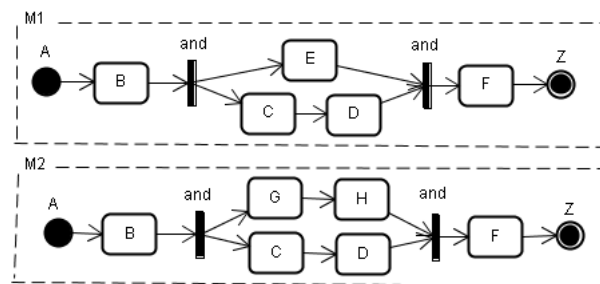


Figura 5.7: Modelos finais extraídos pelo Analisador

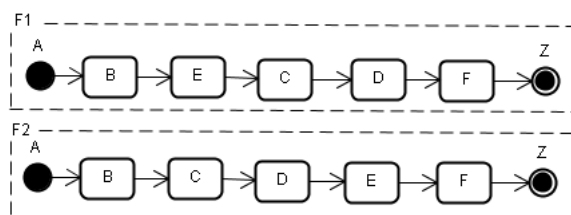


Figura 5.8: Casos obtidos pelo Gerador Casos a partir de M1

A Tabela 5.4 apresenta os resultados obtidos com a simulação para os dois modelos de processos, onde pode-se verificar que o modelo $M1$ é o mais adequado por apresentar menor percentual de atraso e menor quantidade de instâncias de processos atrasadas ao término da simulação. Observa-se que o aumento da carga de trabalho implica no aumento do percentual de atraso. Isto ocorre devido ao grande número de atividades aguardando para serem executadas. Este fato foi observado também

em resultados de trabalhos similares como os apresentados por Tramontina [6].

Tabela 5.4: Análise Resultado

	M1			M2		
Carga de Trabalho (U)	45	75	95	45	75	95
% Atraso	5	70	85	65	90	90

Capítulo 6

Conclusão e Trabalhos Futuros

O Escalonador Genético proposto considera problemas de escalonamento do tipo FJSP com tempo de configuração dependente de seqüência e otimização multi-objetivo. Propostas de resolução deste tipo de problema contribuem para aumentar a flexibilidade, pois permitem tratar um número maior de problemas do mundo real e também consideram problemas mais complexos. Na literatura não foram encontradas alternativas para resolução destes casos específicos do problema. Então, neste sentido, este trabalho pode ser considerado pioneiro, embora seja apenas um ponto de partida para novas pesquisas.

Os operadores genéticos com conhecimento específico do problema introduzidos permitiram a melhoria significativa de um dos critérios de otimização mais utilizados. Também as alterações propostas para a geração da população inicial contribuíram para esta melhoria, bem como para diminuir o tempo de execução dos algoritmos relativos a esta etapa do AG. A introdução destes operadores genéticos combinados com operadores tradicionais, e a realização de pequenas modificações nos métodos *CDR-Bootstrap* e *Approach by Localization* produziu diversidade na população, permitindo obter boas soluções para FJSP com tempo de configuração, utilizando pequenos tamanhos de população e números de iterações.

Os resultados numéricos obtidos pela proposta BTSL para problemas T-FJSP sem tempo de configuração podem ser considerados tão bons quanto os de outros métodos discutidos no trabalho. Entretanto, para problemas P-FJSP, BTSL não obteve o mesmo desempenho. A ausência de outros resultados considerando FJSP com tempo de configuração dependente de seqüência não permitiu uma melhor comparação da

proposta.

Outra contribuição importante deste trabalho está na aplicação de técnicas de escalonamento FJSP a problemas de *workflow*, melhorando a integração destas duas áreas do conhecimento e apresentando uma alternativa de solução para o problema do mapeamento $m : n$ entre recursos e atividades.

O Escalonador Genético também permite considerar datas de liberação para recursos e máquinas, situação comum em problemas de escalonamento e *workflow*.

6.1 Trabalhos Futuros

Considerar outros esquemas para representação dos indivíduos de forma a aumentar a eficiência dos operadores genéticos propostos e melhorar o desempenho do BTSL para problemas P-FJSP.

A definição de *benchmark* considerando problemas FJSP com tempo de configuração dependente de sequência, a fim de gerar uma base de comparação entre os métodos.

Melhorar a integração de técnicas de escalonamento e problemas de *workflow*, principalmente no que se refere à transformação de rotas paralelas e condicionais em rotas sequenciais, permitindo, por exemplo, a aplicação de um peso em cada rota definida. Em problemas do mundo real nem todos os caminhos são percorridos com a mesma frequência pelas instâncias de processo.

Referências Bibliográficas

- [1] ADAMS, J., BALAS, E., AND ZAWACK, D. The shifting bottleneck procedure for job shop scheduling. *Manage. Sci.* 34, 3 (1988), 391–401.
- [2] ALLAHVERDI, A., GUPTA, J., AND ALDOWAISAN, T. A review of scheduling research involving setup considerations. In *Omega The International Journal of Management Science* (1996), vol. 27, pp. 219–239.
- [3] ALVES, F. S., GUIMARAES, K. F., AND FERNANDES, M. A. Modeling workflow systems with genetic planner and scheduler. *ICTAI* (2006), 381–388.
- [4] ARTIGUES, C., AND FEILLET, D. A branch and bound method for the job-shop problem with sequence-dependent setup times.
- [5] BAGCHI, T. P. *Multiobjective Scheduling by Genetic Algorithms*. Kluwer Academic Publishers, 1999.
- [6] BAGGIO, G., WAINER, J., AND ELLIS, C. Applying scheduling techniques to minimize the number of late jobs in workflow systems. In *Proceedings of the 2004 ACM Symposium on Applied Computing* (New York, NY, USA, 2004), ACM Press, pp. 1396–1403.
- [7] BEAN, J. Genetics and random keys for sequencing and optimization. *ORSA Journal on Computing* 6, 1 (1994), 154–160.
- [8] BIERWIRTH, C., AND MATTFELD, D. C. Production Scheduling and Rescheduling with Genetic Algorithms. *Evolutionary Computation* 7, 1 (1999), 1–17.
- [9] BRANDIMARTE, P. Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research* 22, 0 (1993), 158–183.

- [10] BUSCAYLET, F., AND ARGIGUES, C. A fast tabu search method for the job-shop problem with sequence-dependent setup time. In *The Fifth Metaheuristics International Conference* (Agosto 2003).
- [11] CHEN, H., IHLOW, J., AND LEHMANN, C. A genetic algorithm for flexible job-shop scheduling. In *International Conference on Robotics & Automation* (1999).
- [12] CHENG, R. A study on genetic algorithms-based optimal scheduling techniques. Master's thesis, Tokyo Institute of Technology, 1997.
- [13] CHENG, R., GEN, M., AND TSUJIMURA, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms i: representation. *Comput. Ind. Eng.* 30, 4 (1996), 983–997.
- [14] CHENG, R., GEN, M., AND TSUJIMURA, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms, part ii: hybrid genetic search strategies. *Comput. Ind. Eng.* 36, 2 (1999), 343–364.
- [15] CHEUNG, W., AND ZHOU, H. Using genetic algorithms and heuristics for job-shop scheduling with sequence-dependent setup times. In *Annals of Operations Research* (2002), pp. 65–81.
- [16] CHOI, I.-C., AND CHOI, D.-S. A local search algorithm for job-shop scheduling problems with alternatives operations and sequence-dependent setups. In *Computers & Industrial Engineering* (2002), pp. 43–58.
- [17] CHOI, I.-C., AND KORKMAZ, O. Job-shop scheduling with separable sequence-dependent setups times. In *Annals of Operations Research* (1997), pp. 155–170.
- [18] DAUZÈRE-PÉRÈS, S., AND PAULLI, J. An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research* 70, 0 (1997), 281–306.
- [19] DAVIS, L. *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, 1991.
- [20] DE ARAUJO, R. M., AND DA SILVA BORGES, M. R. Sistemas de workflow. XX Jornada de Atualização em Informática, 2001. Congresso da Sociedade Brasileira de Computação, Fortaleza, Ceará, Brasil.

- [21] DORNDORF, U., AND PESCH, E. Evolution based learning in a job shop scheduling environment. *Comput. Oper. Res.* 22, 1 (1995), 25–40.
- [22] FUCHIGAMI, H. Y. Métodos heurísticos construtivos para o problema de programação da produção em sistemas flow shop híbridos com tempos de preparação das máquinas assimétricos e dependentes da sequência. Master's thesis, Escola de Engenharia de São Carlos, São Carlos, Brasil, 2005.
- [23] GLOVER, F. Future paths for integer programming and links with artificial intelligence. *Computers and Operations Research* 13 (1986), 533–549.
- [24] GLOVER, F., AND LAGUNA, F. Tabu search. *Kluwer Academic Publishers* (1997), 533–549.
- [25] GOLDBERG, D. E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley, 1989.
- [26] GRAHAM, R., LAWLER, E., LENSTRA, J., AND KAN, A. R. Optimization and approximation in deterministic sequencing and scheduling: A survey. In *Annals of Discrete Mathematics* (1979), vol. 5.
- [27] GUIMARÃES, K. F., AND FERNANDES, M. A. An approach for flexible job-shop scheduling with separable sequence-dependent setup time. In *IEEE International Conference on Systems, Man, and Cybernetics* (Taipei, Taiwan, 2006), pp. 3727–3731.
- [28] HO, N. B., AND TAY, J. C. Genace: An efficient cultural algorithm for solving the flexible job-shop problem. In *Congress on Evolutionary Computation* (2004), vol. 2.
- [29] HOLLAND, J. H. *Adaptation in Natural and Artificial System*. University of Michigan Press, 1975.
- [30] JANSEN, K., MASTROLILLI, M., AND SOLIS-OBA, R. Approximation algorithms for flexible job shop problems. In *Latin American Theoretical INformatics* (2000), pp. 68–77.
- [31] JK, L. *Sequencing by Enumerative Methods*. Mathematical Center Tract 69, 1977.

- [32] KACEM, I. Genetic algorithm for the flexible job-shop scheduling problem. In *IEEE International Conference on Systems, Man and Cybernetics*.
- [33] KACEM, I., HAMMADI, S., AND BORNE, P. Approach by localization and multiobjective evolutionary optimization for flexible job-shop scheduling problems. In *IEEE Transactions on Systems, Man and Cybernetics, Part C, IEEE Transactions on* (2002), pp. 1–13.
- [34] KACEM, I., HAMMADI, S., AND BORNE, P. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. In *Mathematics and Computers in Simulation* (2002), pp. 245–276.
- [35] KIRKPATRICK, S., GELLAT, D. C., AND VECCHI, M. P. Optimizations by simulated annealing. *Science* 220, 4598 (1983), 671–680.
- [36] LAGUNA, M., AND MARKLUND, J. *Business process modeling, simulation and design*. Prentice Hall, 2005.
- [37] LEE, K. M., AND YAMA KAWA, T. A genetic algorithm for general machine scheduling problems. *International Journal of Knowledge Based Electronics Systems* 2 (1996), 60–66.
- [38] LENSTRA, J. K., AND KAN, A. H. G. R. Computational complexity of discrete optimization problems.
- [39] MACCARTHY, B., AND LIU, J. Addressing the gap in scheduling research: A review of optimization and heuristic methods in production scheduling. *International Journal of Production Research* 31, 1 (1993), 59–79.
- [40] NAJID, N. M., DAUZÈRE-PÉRÈS, S., AND ZAIDAT, A. A modified simulated annealing method for flexible job shop scheduling problem. In *IEEE International Conference on Systems, Man and Cybernetics* (2002), vol. 5.
- [41] NAKANO, R., AND YAMADA, T. A conventional genetic algorithms for job-shop problems. In *Proceedings of the Fourth International Conference on Genetic Algorithms* (Morgan Kaufman, San Diego, Calif, 1991), vol. 2.

- [42] NOWICKI, E., AND SMUTNICKI, C. A fast tabu search algorithm for the job shop problem. In *Management Science* (1996), vol. 42.
- [43] ONG, Z. X., TAY, J. C., AND KWOH, C. K. Applying the clonal selection principle to find flexible job-shop schedules. In *In Proceedings of the International Conference for Artificial Immune Systems* (2005).
- [44] PHILIP, D. Scheduling reentrant flexible job shops with sequence-dependent setup times. Master's thesis, Montana State University, Montana, USA, 2005.
- [45] PINEDO, M. *Scheduling: Theory, Algorithms and Systems*, second ed. Prentice Hall.
- [46] REIJERS, H. *Design and Control of Workflow Processes: Business Process Management for the Service Industry*. Springer-Verlag, 2003.
- [47] SOUISSI, A., KACEM, I., AND CHU, C. Minimizing total tardiness on a single machine with sequence-dependent setup times. In *IEEE International Conference on Systems, Man and Cybernetics* (2004), pp. 1481–1485.
- [48] T., J. M. Generating robust and flexible job shop schedules using genetic algorithms. In *IEEE Transactions on Evolutionary Computation* (2003), IEEE Computational Intelligence Society, pp. 275–288.
- [49] TAMAKI, H., AND NISHIKAWA, Y. A paralleled genetic algorithm based on a neighborhood model and its application to the jobshop scheduling. In *Proc. of 2nd Conf. on Parallel Problem Solving from Nature (PPSN'92)* (North-Holland, 1992).
- [50] TAMAKI, H., ONO, T., MURAO, H., AND KITAMURA, S. Modeling and genetic solution of a class of flexible job-shop scheduling problems. In *IEEE International Conference on Automation* (2001).
- [51] TRAMONTINA, G. B. Análise de problemas de escalonamento de processos em workflow. Master's thesis, Instituto de Computação, Universidade de Campinas (UNICAMP), Campinas, SP, Brazil, 2004.

- [52] VACA, O. C. L. Um algoritmo evolutivo para a programação de projetos multi-modos com nivelamento de recursos limitados. Master's thesis, Universidade Federal de Santa Catarina, Florianópolis, Brasil, 1995.
- [53] VAN DER AALST, W., AND VAN HEE, K. *Workflow Management: Models, Methods and Systems*. The MIT Press, 2002.
- [54] WFMC. Terminology and Glossary. Tech. Rep. WfMC-TC-1011, Workflow Management Coalition, Hampshire, UK, 1999.
- [55] WILBRECHT, J. K., AND PRESCOTT, W. B. The influence of setup time on job shop performance. *Management Science* 16, 4 (1969), 274–280.
- [56] WORTMAN, D. Managing capacity: getting the most from your firms assets. *Industrial Engineering* 24 (1992), 47–49.
- [57] YAMADA, T., AND NAKANO, R. A genetic algorithm applicable to large-scale job-shop problems. In *Proceedings of the Second International Conference on Parallel Problem Solving from Nature* (1992), vol. 2.
- [58] ZHAO, J. L., AND STOHR, E. A. Temporal workflow management in a claim handling system. In *WACC '99: Proceedings of the international joint conference on Work activities coordination and collaboration* (New York, NY, USA, 1999), ACM Press, pp. 187–195.
- [59] ZRIBI, N., KACEM, I., EL KAMEL, A., AND BORNE, P. Optimization by phases for the flexible job-shop scheduling problem. In *Asian Control Conference* (2004), pp. 1889–1895.

Apêndice A

Problemas FJSP

Neste apêndice são apresentadas algumas das instâncias de problemas FJSP consideradas neste trabalho.

A.1 Problemas

Tabela A.1: Problema 4x5						
		M_1	M_2	M_3	M_4	M_5
J_1	$O_{1,1}$	2	5	4	1	2
	$O_{2,1}$	3	4	5	7	5
	$O_{3,1}$	5	5	5	4	5
J_2	$O_{1,2}$	2	5	4	7	8
	$O_{2,2}$	5	6	9	8	5
	$O_{3,2}$	4	5	4	54	5
J_3	$O_{1,3}$	9	8	6	7	9
	$O_{2,3}$	6	1	2	5	4
	$O_{3,3}$	2	5	4	2	4
	$O_{4,3}$	4	5	2	1	5
J_4	$O_{1,4}$	1	5	2	4	12
	$O_{2,4}$	5	1	2	1	2

Tabela A.2: Problema 10x7

		M_1	M_2	M_3	M_4	M_5	M_6	M_7
J_1	$O_{1,1}$	1	4	6	9	3	5	2
	$O_{2,1}$	8	9	5	4	1	1	3
	$O_{3,1}$	4	8	10	4	11	4	3
J_2	$O_{1,2}$	6	9	8	6	5	10	3
	$O_{2,2}$	2	10	4	5	9	8	4
J_3	$O_{1,3}$	15	4	8	4	8	7	1
	$O_{2,3}$	9	6	1	10	7	1	6
	$O_{3,3}$	11	2	7	5	2	3	14
J_4	$O_{1,4}$	2	8	5	8	9	4	3
	$O_{2,4}$	5	3	8	1	9	3	6
	$O_{3,4}$	1	2	6	4	1	7	2
J_5	$O_{1,5}$	7	1	8	5	4	3	9
	$O_{2,5}$	2	4	5	10	6	4	9
	$O_{3,5}$	5	1	7	1	6	6	2
J_6	$O_{1,6}$	8	7	4	56	9	8	4
	$O_{2,6}$	5	14	1	9	6	5	8
	$O_{3,6}$	3	5	2	5	4	5	7
J_7	$O_{1,7}$	5	6	3	6	5	15	2
	$O_{2,7}$	6	5	4	9	5	4	3
	$O_{3,7}$	9	8	2	8	6	1	7
J_8	$O_{1,8}$	6	1	4	1	10	4	3
	$O_{2,8}$	11	13	9	8	9	10	8
	$O_{3,8}$	4	2	7	8	3	10	7
J_9	$O_{1,9}$	12	5	4	5	4	5	5
	$O_{2,9}$	4	2	15	99	4	7	3
	$O_{3,9}$	9	5	11	2	5	4	2
J_{10}	$O_{1,10}$	9	4	13	10	7	6	8
	$O_{2,10}$	4	3	25	3	8	1	2
	$O_{3,10}$	1	2	6	11	13	3	5

Tabela A.3: Problema 10x10

		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J_1	$O_{1,1}$	1	4	6	9	3	5	2	8	9	5
	$O_{2,1}$	4	1	1	3	4	8	10	4	11	4
	$O_{3,1}$	3	2	5	1	5	6	9	6	10	3
J_2	$O_{1,2}$	2	10	4	5	9	8	4	15	8	4
	$O_{2,2}$	4	8	7	1	9	6	1	10	7	1
	$O_{3,2}$	6	11	2	7	5	3	5	14	9	2
J_3	$O_{1,3}$	8	5	8	9	4	3	5	3	8	1
	$O_{2,3}$	9	3	6	1	2	6	4	1	7	2
	$O_{3,3}$	7	1	8	5	4	9	1	2	3	4
J_4	$O_{1,4}$	5	10	6	4	9	5	1	7	1	6
	$O_{2,4}$	4	2	3	8	7	4	6	9	8	4
	$O_{3,4}$	7	3	12	1	6	5	8	3	5	2
J_5	$O_{1,5}$	7	10	4	5	6	3	5	15	2	6
	$O_{2,5}$	5	6	3	9	8	2	8	6	1	7
	$O_{3,5}$	6	1	4	1	10	4	3	11	13	9
J_6	$O_{1,6}$	8	9	10	8	4	2	7	8	3	10
	$O_{2,6}$	7	3	12	5	4	3	6	9	2	15
	$O_{3,6}$	4	7	3	6	3	4	1	5	1	11
J_7	$O_{1,7}$	1	7	8	3	4	9	4	13	10	7
	$O_{2,7}$	3	8	1	2	3	6	11	2	13	3
	$O_{3,7}$	5	4	2	1	2	1	8	14	5	7
J_8	$O_{1,8}$	5	7	11	3	2	9	8	5	12	8
	$O_{2,8}$	8	3	10	7	5	13	4	6	8	4
	$O_{3,8}$	6	2	13	5	4	3	5	7	9	5
J_9	$O_{1,9}$	3	9	1	3	8	1	6	7	5	4
	$O_{2,9}$	4	6	2	5	7	3	1	9	6	7
	$O_{3,9}$	8	5	4	8	6	1	2	3	10	10
J_{10}	$O_{1,10}$	4	3	1	6	7	1	2	6	20	6
	$O_{2,10}$	3	1	8	1	9	4	1	4	17	15
	$O_{3,10}$	9	2	4	2	3	5	2	4	10	23

Tabela A.4: Problema 15x10

		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J_1	$O_{1,1}$	1	4	6	9	3	5	2	8	9	4
	$O_{2,1}$	1	1	3	4	8	10	4	11	4	3
	$O_{3,1}$	2	5	1	5	6	9	5	10	3	2
	$O_{4,1}$	10	4	5	9	8	4	15	8	4	4
J_2	$O_{1,2}$	4	8	7	1	9	6	1	10	7	1
	$O_{2,2}$	6	11	2	7	5	3	5	14	9	2
	$O_{3,2}$	8	5	8	9	4	3	5	3	8	1
	$O_{4,2}$	9	3	6	1	2	6	4	1	7	2
J_3	$O_{1,3}$	7	1	8	5	4	9	1	2	3	4
	$O_{2,3}$	5	10	6	4	9	5	1	7	1	6
	$O_{3,3}$	4	2	3	8	7	4	6	9	8	4
	$O_{4,3}$	7	3	2	1	6	5	8	3	5	2
J_4	$O_{1,4}$	6	2	5	4	1	2	3	6	5	4
	$O_{2,4}$	8	5	7	4	1	2	36	5	8	5
	$O_{3,4}$	9	6	2	4	5	1	3	6	5	2
	$O_{4,4}$	11	4	5	6	2	7	5	4	2	1
J_5	$O_{1,5}$	6	9	2	3	5	8	7	4	1	2
	$O_{2,5}$	5	4	6	3	5	2	28	7	4	5
	$O_{3,5}$	6	2	4	3	6	5	2	4	7	9
	$O_{4,5}$	6	5	4	2	3	2	5	4	7	5
J_6	$O_{1,6}$	4	1	3	2	5	9	8	5	4	2
	$O_{2,6}$	1	3	6	5	4	7	5	4	6	5
J_7	$O_{1,7}$	1	4	2	5	3	6	9	8	5	4
	$O_{2,7}$	2	1	4	5	2	3	5	4	2	5
J_8	$O_{1,8}$	2	3	6	2	5	4	1	5	8	7
	$O_{2,8}$	4	5	6	2	3	5	4	1	2	5
	$O_{3,8}$	3	5	4	2	5	49	8	5	4	5
	$O_{4,8}$	1	2	36	5	2	3	6	4	11	2
J_9	$O_{1,9}$	6	3	2	22	44	11	10	23	5	1
	$O_{2,9}$	2	3	2	12	15	10	12	14	18	16
	$O_{3,9}$	20	17	12	5	9	6	4	7	5	6
	$O_{4,9}$	9	8	7	4	5	8	7	4	56	2

Tabela A.5: Problema 15x10

		M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J_{10}	$O_{1,10}$	5	8	7	4	56	3	2	5	4	1
	$O_{2,10}$	2	5	6	9	8	5	4	2	5	4
	$O_{3,10}$	6	3	2	5	4	7	4	5	2	1
	$O_{4,10}$	3	2	5	6	5	8	7	4	5	2
J_{11}	$O_{1,11}$	1	2	3	6	5	2	1	4	2	1
	$O_{2,11}$	2	3	6	3	2	1	4	10	12	1
	$O_{3,11}$	3	6	2	5	8	4	6	3	2	5
	$O_{4,11}$	4	1	45	6	2	4	1	25	2	4
J_{12}	$O_{1,12}$	9	8	5	6	3	6	5	2	4	2
	$O_{2,12}$	5	8	9	5	4	75	63	6	5	21
	$O_{3,12}$	12	5	4	6	3	2	5	4	2	5
	$O_{4,12}$	8	7	9	5	6	3	2	5	8	4
J_{13}	$O_{1,13}$	4	2	5	6	8	5	6	4	6	2
	$O_{2,13}$	3	5	4	7	5	8	6	6	3	2
	$O_{3,13}$	5	4	5	8	5	4	6	5	4	2
	$O_{4,13}$	3	2	5	6	5	4	8	5	6	4
J_{14}	$O_{1,14}$	2	3	5	4	6	5	4	85	4	5
	$O_{2,14}$	6	2	4	5	8	6	5	4	2	6
	$O_{3,14}$	3	25	4	8	5	6	3	2	5	4
	$O_{4,14}$	8	5	6	4	2	3	6	8	5	4
J_{15}	$O_{1,15}$	2	5	6	8	5	6	3	2	5	4
	$O_{2,15}$	5	6	2	5	4	2	5	3	2	5
	$O_{3,15}$	4	5	2	3	5	2	8	4	7	5
	$O_{4,15}$	6	2	11	14	2	3	6	5	4	8

Tabela A.6: Setup 4x5

	$O_{1,1}$	$O_{2,1}$	$O_{3,1}$	$O_{1,2}$	$O_{2,2}$	$O_{3,2}$	$O_{1,3}$	$O_{2,3}$	$O_{3,3}$	$O_{4,3}$	$O_{1,4}$	$O_{2,4}$
$O_{1,1}$	1	1	1	2	1	3	2	3	3	2	2	1
$O_{2,1}$	2	3	2	2	3	3	2	1	1	1	1	1
$O_{3,1}$	1	1	2	1	2	2	3	2	1	1	2	1
$O_{1,2}$	3	2	1	1	3	1	1	2	1	1	1	1
$O_{2,2}$	3	2	2	1	2	3	1	1	1	2	2	1
$O_{3,2}$	2	1	1	2	3	1	1	2	2	2	3	2
$O_{1,3}$	1	1	2	2	2	2	3	1	1	3	2	2
$O_{2,3}$	2	1	2	1	1	2	1	2	1	1	1	3
$O_{3,3}$	2	1	1	1	2	1	1	2	1	1	1	1
$O_{4,3}$	1	2	3	1	2	1	2	2	2	3	2	2
$O_{1,4}$	1	2	1	1	1	1	1	2	3	1	2	3
$O_{2,4}$	1	3	3	3	2	1	2	1	1	1	2	1

Apêndice B

Algoritmo de Associação

Este apêndice apresenta o detalhamento do algoritmo de *AssignmentProcedure_BTSL* utilizado na geração da população inicial no método AL.

B.1 *AssignmentProcedure*

A aplicação do algoritmo 2 passo a passo gera os seguintes resultados.

Primeira Iteração:

O mínimo global corresponde a $p'_{1,1,1}=1$;

$k = 1; i = 1; j = 1$

Gera a associação $O_{1,1,1}$, elimina-se a linha (1,1) e adiciona 1 para a coluna Col_1 (Veja Tabela B.2).

Segunda Iteração:

O mínimo global corresponde a $p'_{2,1,4}=1$;

$k = 4; i = 2; j = 1$

Gera a associação $O_{2,1,4}$, elimina-se a linha (2,1) e adiciona 1 para a coluna Col_4 (Veja Tabela B.3).

Terceira Iteração:

O mínimo global corresponde a $p'_{1,2,2}=1$;

$k = 2; i = 1; j = 2$

Gera a associação $O_{1,2,2}$, elimina-se a linha (1,2) e adiciona 1 para a coluna Col_2 (Veja Tabela B.4).

Quarta Iteração:

O mínimo global corresponde a $p'_{3,2,2}=2$;

$$k = 2; i = 3; j = 2$$

Gera a associação $O_{3,2,2}$, elimina-se a linha (3,2) e adiciona 2 para a coluna Col_2 (Veja Tabela B.5).

Quinta Iteração:

O mínimo global corresponde a $p'_{2,3,4}=2$;

$$k = 4; i = 2; j = 3$$

Gera a associação $O_{2,3,4}$, elimina-se a linha (2,3) e adiciona 2 para a coluna Col_4 (Veja Tabela B.6).

Sexta Iteração:

O mínimo global corresponde a $p'_{2,2,1}=3$;

$$k = 1; i = 2; j = 2$$

Gera a associação $O_{2,2,1}$, elimina-se a linha (2,2) e adiciona 3 para a coluna Col_1 (Veja Tabela B.7).

Sétima Iteração:

O mínimo global corresponde a $p'_{1,3,3}=3$;

$$k = 3; i = 1; j = 3$$

Gera a associação $O_{1,3,3}$, elimina-se a linha (1,3) e adiciona 3 para a coluna Col_3 (Veja Tabela B.6).

Oitava Iteração:

O mínimo global corresponde a $p'_{3,1,3}=7$;

$$k = 3; i = 1; j = 3$$

Gera a associação $O_{3,1,3}$, elimina-se a linha (3,1). T' torna-se vazia e a geração de todas as associações é concluída.

Tabela B.1: T' original

		M_1	M_2	M_3	M_4
J_1	O_{11}	1	3	4	1
	O_{21}	3	8	2	1
	O_{31}	3	5	4	7
J_2	O_{12}	4	1	1	4
	O_{22}	2	3	9	3
	O_{32}	9	1	2	2
J_3	O_{13}	8	6	3	5
	O_{23}	4	5	8	1

Tabela B.2: T' após primeira iteração

		M_1	M_2	M_3	M_4
J_1	O_{21}	4	8	2	1
	O_{31}	4	5	4	7
J_2	O_{12}	5	1	1	4
	O_{22}	3	3	9	3
	O_{32}	10	1	2	2
J_3	O_{13}	9	6	3	5
	O_{23}	5	5	8	1

Tabela B.3: T' após segunda iteração

		M_1	M_2	M_3	M_4
J_1	O_{31}	4	5	4	8
J_2	O_{12}	5	1	1	5
	O_{22}	3	3	9	4
	O_{32}	10	1	2	3
J_3	O_{13}	9	6	3	6
	O_{23}	5	5	8	2

Tabela B.4: T' após terceira iteração

		M_1	M_2	M_3	M_4
J_1	O_{31}	4	6	4	8
J_2	O_{22}	3	4	9	4
	O_{32}	10	2	2	3
J_3	O_{13}	9	7	3	6
	O_{23}	5	6	8	2

Tabela B.5: T' após quarta iteração

		M_1	M_2	M_3	M_4
J_1	O_{31}	4	8	4	8
J_2	O_{22}	3	6	9	4
J_3	O_{13}	9	9	3	6
	O_{23}	5	8	8	2

Tabela B.6: T' após quinta iteração

		M_1	M_2	M_3	M_4
J_1	O_{31}	4	8	4	10
J_2	O_{22}	3	6	9	6
J_3	O_{13}	9	9	3	8

Tabela B.7: T' após sexta iteração

		M_1	M_2	M_3	M_4
J_1	O_{31}	7	8	4	10
J_3	O_{13}	12	9	3	8

Tabela B.8: T' após sétima iteração

		M_1	M_2	M_3	M_4
J_1	O_{31}	7	8	7	10