Universidade Estadual de Campinas
Faculdade de Engenharia Elétrica e de Computação

Escalonamento Memético e Neuro-Memético de Tarefas

Autor: Tatiane Regina Bonfim
Orientador: Prof. Dr. Akebo Yamakami

Tese de Doutorado apresentada à Faculdade de Engenharia Elétrica e de Computação como parte dos requisitos para obtenção do título de Doutor em Engenharia Elétrica. Área de concentração: **Telecomunicações e Telemática**.

Banca Examinadora

Isamara Carvalho Alves, Dra. ..... IM/UFBA
Marcius Fabius Henriques de Carvalho, Dr. ..... CENPRA
Mauricio Fernandes Figueiredo, Dr. ..... DIN/UEM
Takaaki Ohishi, Dr. ..... DENSIS/FEEC/UNICAMP

Campinas, SP
Fevereiro/2006

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

Bonfim, Tatiane Regina
B641e Escalonamento memético e neuro-memético de tarefas / Tatiane Regina Bonfim. – Campinas, SP: [s.n.], 2006.

Orientador: Akebo Yamakami.
Tese (doutorado) - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Números difusos. 2. Redes neurais (Computação). 3. Algoritmos genéticos. 4. Possibilidade. 5. Escalonamento de produção. I. Yamakami, Akebo. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título

Título em Inglês: Memetic and neuro-memetic scheduling of tasks
Palavras-chave em Inglês: Tasks scheduling, Fuzzy numbers, Neural network, Memetic algorithm, Concepts of possibility
Área de concentração: Telecomunicações e Telemática
Titulação: Doutora em Engenharia Elétrica
Banca Examinadora: Isamara Carvalho Alves, Marcius Fabius Henriques de Carvalho, Mauricio Fernandes Figueiredo e Takaaki Ohishi
Data da defesa: 17/02/2006

Este exemplar da tese de doutorado de Tatiane Regina Bonfim, intitulada *Escalonamento Memético e Neuro-Memético de Tarefas* contempla as correções e sugestões formuladas pelos membros da Comissão Julgadora:

Prof. Dr. Akebo Yamakami (Presidente): _____

Prof. Dr. Maurício Fernandes Figueiredo: _____

Profa. Dra. Isamara Carvalho Alves: _____

Prof. Dr. Marcius Fabius Henrique de Carvalho: _____

Prof. Dr. Takaaki Ohishi: _____

Resumo

Este trabalho apresenta uma nova abordagem de resolução, por algoritmo memético e pela coevolução de algoritmo memético com redes neurais, para o problema de escalonamento de tarefas em máquinas paralelas idênticas e para o problema de *job shop* com parâmetros precisos. Para os problemas de escalonamento com parâmetros com incertezas, onde os parâmetros não são precisamente conhecidos, torna-se difícil classificar um determinado escalonamento ótimo. A noção de **ótimo** também torna-se imprecisa e o grau de otimalidade de um dado escalonamento ("o quanto um escalonamento é ótimo") pode ser caracterizada por um número *fuzzy*. Foi aplicado também o conceito de otimalidade possível para medir a possibilidade de um determinado escalonamento ser ótimo. O algoritmo memético foi aplicado para encontrar soluções para o problema, a rede neural foi aplicada para encontrar a função de *fitness* das soluções encontradas pelo algoritmo memético, e o conceito de possibilidade foi aplicado para avaliar as melhores soluções. Foram utilizadas as redes neurais *backpropagation* e com aprendizado por reforço para encontrar o valor da função de *fitness*. As simulações mostraram que as redes neurais apresentaram uma boa performance na coevolução com o algoritmo memético e na resolução dos problemas, e mostraram que o conceito de possibilidade teve uma boa perfomance na avaliação da otimalidade das soluções.

Palavras-chave: escalonamento de tarefas, números *fuzzy*, redes neurais, algoritmo memético, conceitos de possibilidade.

Abstract

This work presents a new approach for the resolution of the problem of identical parallel machine scheduling and job shop scheduling with precise parameters, with memetic algorithm and memetic algorithm coevolving with neural networks. For problems with parameters with uncertainties, where the parameters of the problem are not precisely known, it is difficult to say in prior which schedule will be optimal. The notion of **optimal** also becomes imprecise and the degree of optimality of a given schedule ("how much a schedule is optimal") can be characterized by a *fuzzy* number. We was used also the concepts of possibility to measure the possibility of a given schedule be optimal. Memetic algorithm has been used to find the solutions of the problem, the neural network has been used to find the *fitness* function of these solutions, and the concept of possibility has been used to evaluate the best solutions. We was used neural networks with backpropagation and reinforcement learning to find the fitness function. Simulations showed that the neural networks presents a good performance in the coevolution of the memetic algorithm and in the resolution of the problems, and showed that the concept of possibility present a good performance in the evaluation of solutions optimality.

Keywords: tasks scheduling, fuzzy numbers, neural network, memetic algorithm, concepts of possibility.

Aos meus pais e irmãs

Agradecimentos

A Deus, agradeço pela vida e oportunidade de concretizar o sonho de concluir o doutorado.

Ao meu orientador, Prof. Dr. Akebo Yamakami, sou grata pela orientação, aprendizado e pela amizade.

À minha mãe, agradeço pelo exemplo de mulher guerreira e batalhadora, e por ser a mãe presente em todos os momentos.

Às minhas irmãs, agradeço pelo incentivo, companheirismo e amizade.

Aos Professores, membros da banca, agradeço pela ajuda na revisão desta tese.

Ao Eduardo, agradeço pelos momentos felizes proporcionados. Agradeço pela compreensão, incentivo e carinho sentidos mesmo à distância.

À Cristina, agradeço pelo carinho e grande amizade.

Às minhas amigas Andréia e Regina, agradeço pela amizade.

Aos amigos e irmãos de orientação, agradeço pela amizade e pelo convívio em um clima de harmonia que muito contribuiu para a concretização deste sonho.

Aos amigos de pós-graduação, agradeço pelo companheirismo e por momentos de muita alegria. Agradeço às belas amizades que fiz neste período.

Aos amigos do grupo Tati Sports, agradeço pelos alegres sábados de futebol.

A toda minha família agradeço pelo apoio durante esta jornada.

Ao CNPq, pelo apoio financeiro.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Lista de Símbolos

Capítulo 2 - Conceitos sobre conjuntos e números *fuzzy*

$\mu_A(x)$	-	Função de pertinência de x no conjunto *fuzzy* A
$supp(A)$	-	Suporte do conjunto *fuzzy* A
$hgt(A)$	-	Altura do conjunto *fuzzy* A
A_α	-	α_{corte} do conjunto *fuzzy* A
$\widetilde{T}$	-	Número *fuzzy* T
$\mu_{\widetilde{T}(x)}$	-	Função de pertinência de x no número *fuzzy* T
$\widetilde{t} = [\underline{t}, t^0, \overline{t}]$	-	Representação de um número *fuzzy*
t^0	-	Valor modal do número *fuzzy* T
$t^0 - \underline{t}$	-	Espalhamento à esquerda do número *fuzzy* T
$\overline{t} - t^0$	-	Espalhamento à direita do número *fuzzy* T
$\widetilde{t}_\alpha = [t_1^\alpha, t_2^\alpha]$	-	Representação de um número *fuzzy* na forma de intervalos de confiança
$\widetilde{a}_\alpha \oplus \widetilde{b}_\alpha$	-	Operação de adição de dois números *fuzzy* $\widetilde{a_\alpha}$ e $\widetilde{b_\alpha}$
$\widetilde{a}_\alpha \oplus k$	-	Operação de adição de um número *fuzzy* $\widetilde{a_\alpha}$ e um número ordinário k
$\widetilde{a}_\alpha \ominus \widetilde{b}_\alpha$	-	Operação de subtração de dois números *fuzzy* $\widetilde{a_\alpha}$ e $\widetilde{b_\alpha}$
$\widetilde{a}_\alpha \ominus k$	-	Operação de subtração de um número *fuzzy* $\widetilde{a_\alpha}$ e um número ordinário k
$\widetilde{a}_\alpha \otimes \widetilde{b}_\alpha$	-	Operação de multiplicação de dois números *fuzzy* $\widetilde{a_\alpha}$ e $\widetilde{b_\alpha}$
$\widetilde{a}_\alpha \otimes k$	-	Operação de multiplicação de um número *fuzzy* $\widetilde{a_\alpha}$ e um número ordinário k
$\widetilde{a}_\alpha \oslash \widetilde{b}_\alpha$	-	Operação de divisão de dois números *fuzzy* $\widetilde{a_\alpha}$ e $\widetilde{b_\alpha}$
$k \oslash \widetilde{a}_\alpha$	-	Operação de divisão de um número ordinário k por um número *fuzzy* $\widetilde{a_\alpha}$
$\exp(-\widetilde{a}_\alpha)$	-	Operação de exponencial de um número *fuzzy* $\widetilde{a_\alpha}$
Poss$(\widetilde{a}, \widetilde{b})$	-	Conceito de possibilidade do número *fuzzy* a sobrepor o número *fuzzy* b
$Nec(\widetilde{a}, \widetilde{b})$	-	Conceito de necessidade do número *fuzzy* b estar incluído no número *fuzzy* a
$Poss(\widetilde{a} \geq \widetilde{b})$	-	Possibilidade do número *fuzzy* a ser maior ou igual ao número *fuzzy* b
z^0	-	Representa o valor defuzzificado (valor *crisp* de um número *fuzzy*)
sup	-	Ponto superior
inf	-	Ponto inferior

Capítulo 3 - Conceitos sobre escalonamento de tarefas em máquinas paralelas idênticas com parâmetros precisos

m	-	Número de máquinas
n	-	Número de tarefas
$x_{i,j}$	-	Variável que tem o valor 1 se a tarefa i é processada pela máquina j, caso contrário, tem o valor 0
M	-	*Makespan*, que é o tempo total de processamento na máquina mais ocupada
t_i	-	Tempo de processamento *crisp* da tarefa i no problema de escalonamento de tarefas em máquinas paralelas idênticas
$T(m_j)$	-	Tempo total de processamento das tarefas alocadas à máquina j
x	-	Solução factível para o problema de escalocamento de tarefas em máquinas paralelas idênticas
X	-	Conjunto de todas as soluções possíveis
$N(x)$	-	Vizinhança de soluções da solução x
x'	-	Nova solução para o problema de escalonamento de tarefas em máquinas paralelas idênticas
p	-	Máquina mais ocupada
$num_{tarefas}(p)$	-	Número de tarefas alocadas à máquina p
$V(t)$	-	Vetor de temperaturas do *simulated annealing*
T_{pop}	-	Tamanho da população
T_{subpop}	-	Tamanho da subpopulação
N_{gera}	-	Número de gerações
$V(ind)$	-	Vetor de indivíduos (cromossomos) da população
$V2(ind)$	-	Vetor de indivíduos (cromossomos) da subpopulação
$roulettewheel$	-	Método de seleção da roleta ponderada
$M(ind)$	-	Vetor com o *makespan* de cada indivíduo
$F(ind)$	-	Vetor com o valor da função *fitness* de cada indivíduo
ind	-	Indivíduo específico ($0 \leq ind \leq T_{pop}$)

Capítulo 3 - Conceitos sobre escalonamento *jobshop* com parâmetros precisos

m	-	Número de máquinas
n	-	Número de tarefas
$O = 0, ..., n-1$	-	Conjunto de operações, onde 0 é considerada a operação inicial de todas tarefas e $n-1$ é considerada a operação final de todas as tarefas
A	-	Conjunto de pares ordenados de operações
E_k	-	Conjunto de todos os pares de operações a serem executadas na máquina k
p_i	-	Tempo de processamento da operação i
ti_{il}	-	Tempo possível para início do processamento da operação i na máquina l
tf_{il}	-	Tempo final de processamento da operação i na máquina l
M	-	*Makespan*, que é o tempo total gasto entre o início da primeira operação e o término da última operação
O_{nxm}	-	Matriz com a seqüência de operações para cada tarefa
$o_{ij} = (m_{ij}, p_{ij})$	-	Par ordenado de cada operação composto pela máquina que irá processá-la e o tempo de processamento da operação, onde $0 \leq i \leq n-1$ e $0 \leq j \leq m-1$
T_{pop}	-	Tamanho da população
N_{gera}	-	Número de gerações
V	-	Vetor com representação do cromossomo do indivíduo
A_{mxn}	-	Matriz com representação do cromossomo do indivíduo
JT_{mxn}	-	Matriz de janela de tempo do indivíduo
F	-	*Fitness* do indivíduo
job_{troca}	-	Tarefa escolhida para a troca
ind	-	Indivíduo específico ($0 \leq ind \leq T_{pop}$)
num_t	-	Número de uma tarefa específica ($0 \leq num_t \leq n-1$)
$MAX\{tf_j\}$	-	Tempo final máximo na janela de tempo do indivíduo ($0 \leq j \leq m-1$)
p_{cruz}	-	Taxa de pontos de cruzamento ($0 \leq p_{cruz} \leq ((nxm)-1)/2)$), que representa a quantidade de pares de operações que serão permutadas na operação de *crossover*
op	-	Operação específica ($0 \leq op \leq m$) escolhida para a operação de *crossover*

Capítulo 4 - Conceitos sobre escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas

m	-	Número de máquinas
n	-	Número de tarefas
$x_{i,j}$	-	Variável que tem o valor 1 se a tarefa i é processada pela máquina j, caso contrário, tem o valor 0
$\widetilde{M}$	-	*Makespan fuzzy*, que é o tempo total de processamento na máquina mais ocupada
$\widetilde{t_i}$	-	Tempo de processamento *fuzzy* da tarefa i
$\widetilde{T_j}$	-	Tempo total *fuzzy* de processamento na máquina j
p	-	Máquina mais ocupada
$num_{tarefas}(p)$	-	Número de tarefas alocadas à máquina p
T_{pop}	-	Tamanho da população
N_{gera}	-	Número de gerações
$V(ind)$	-	Vetor de indivíduos (cromossomos) da população
ind	-	Indivíduo específico ($0 \leq ind \leq T_{pop}$)
$\widetilde{T}_{j_max}$	-	Valor *fuzzy* do *Makespan*
$\widetilde{F}$	-	Vetor de *fitness fuzzy* da população de indivíduos
$\widetilde{F}(k_{min})$	-	Valor do fitness *fuzzy* do melhor indivíduo
$\widetilde{M}(ind)$	-	Vetor de *makespan fuzzy* do indivíduo

Capítulo 4 - Conceitos sobre escalonamento *jobshop* com parâmetros com incertezas

m	-	Número de máquinas
n	-	Número de tarefas
$O = 0, ..., n-1$	-	Conjunto de operações, onde 0 é considerada a operação inicial de todas as tarefas e $n-1$ é considerada a operação final de todas as tarefas
A	-	Conjunto de pares ordenados de operações
E_k	-	Conjunto de todos os pares de operações a serem executadas na máquina k
$\widetilde{p_i}$	-	Tempo de processamento *fuzzy* da operação i
$\widetilde{ti}_{il}$	-	Tempo possível *fuzzy* para início do processamento da operação i na máquina l
$\widetilde{tf}_{il}$	-	Tempo final *fuzzy* de processamento da operação i na máquina l
$\widetilde{M}$	-	*Makespan fuzzy*, que é o tempo total gasto entre o início da primeira operação e o término da última operação
O_{nxm}	-	Matriz com a seqüência de operações para cada tarefa
$o_{ij} = (m_{ij}, \widetilde{p}_{ij})$	-	Par ordenado de cada operação composto pela máquina que irá processá-la e o tempo de processamento da operação, onde $0 \leq i \leq n-1$ e $0 \leq j \leq m-1$
T_{pop}	-	Tamanho da população
N_{gera}	-	Número de gerações
V	-	Vetor com representação do cromossomo do indivíduo
A_{mxn}	-	Matriz com representação do cromossomo do indivíduo
$\widetilde{JT}_{mxn}$	-	Matriz de janela de tempo *fuzzy* do indivíduo
$\widetilde{F}$	-	*Fitness fuzzy* do indivíduo
ind	-	Indivíduo específico ($0 \leq ind \leq T_{pop}$)
num_t	-	Número de uma tarefa específica ($0 \leq num_t \leq n-1$)
$MAX\{tf_j\}$	-	Tempo final máximo na janela de tempo do indivíduo ($0 \leq j \leq m-1$)
$\widetilde{tf}_{j_{max}}$	-	*Makespan* até um determinado momento
$\widetilde{F}(k_{min})$	-	*Fitness* do melhor indivíduo até um determinado momento
p_{cruz}	-	Taxa de pontos de cruzamento ($0 \leq p_{cruz} \leq ((nxm)-1)/2)$), que representa a quantidade de pares de operações que serão permutadas na operação de *crossover*
op	-	Operação específica ($0 \leq op \leq m$) escolhida para a operação de *crossover*

Capítulo 5 - Rede Neural *Backpropagation*

- p - Número de neurônios da camada de entrada
- q - Número de neurônios da camada intermediária
- u - Número de neurônios da camada de saída
- X_i - Neurônio da camada de entrada da rede neural, onde $1 \leq i \leq p$
- Z_j - Neurônio da camada intermediária da rede neural, onde $1 \leq j \leq q$
- Y_k - Neurônio da camada de saída da rede neural, onde $1 \leq k \leq u$
- v_{ij} - Peso entre a camada de entrada i e a camada intermediária j
- w_{jk} - Peso entre a camada intermediária j e a camada de saída k
- x_i - Sinal de entrada da camada de entrada
- z_in_j - Sinal de entrada da camada intermediária j
- z_j - Sinal de saída da camada intermediária j
- y_in_k - Sinal de entrada da camada de saída
- y_k - Sinal de saída da camada de saída
- t_k - Valor de saída desejado
- δ_k - Fator de erro da camada de saída k
- Δw_{jk} - Termo de correção entre as camadas de saída k e intermediária j
- α - Taxa de aprendizagem
- δ_j - Fator de erro da camada intermediária j
- δ_in_j - Fator de erro da camada de entrada
- Δv_{ij} - Termo de correção entre as camadas de entrada i e intermediária j

Capítulo 5 - Rede Neural com Aprendizado por Reforço

p	-	Número de neurônios da camada de entrada
q	-	Número de neurônios da camada intermediária
u	-	Número de neurônios da camada de saída
X_i	-	Neurônio da camada de entrada da rede neural, onde $1 \leq i \leq p$
Z_j	-	Neurônio da camada intermediária da rede neural, onde $1 \leq j \leq q$
Y_k	-	Neurônio da camada de saída da rede neural, onde $1 \leq k \leq u$
v_{ij}	-	Peso entre a camada de entrada i e a camada intermediária j
w_{jk}	-	Peso entre a camada intermediária j e a camada de saída k
x_i	-	Sinal de entrada da camada de entrada
z_in_j	-	Sinal de entrada da camada intermediária j
z_j	-	Sinal de saída da camada intermediária j
y_in_k	-	Sinal de entrada da camada de saída
y_k	-	Sinal de saída da camada de saída
t_k	-	Valor de saída desejado
$f(x)$	-	Função bipolar sigmóide
$f'(x)$	-	Derivada da função bipolar sigmóide
num_ep	-	Número de épocas de realização do algoritmo
α	-	Taxa de aprendizagem
γ	-	Taxa de desconto
l	-	Número da iteração atual
r	-	Valor de recompensa
V_l	-	Valor atual da recompensa do agente
ϵ	-	Valor utilizado como critério de parada

Capítulo 6 - Rede Neural-*Fuzzy Backpropagation*

- p - Número de neurônios da camada de entrada
- q - Número de neurônios da camada intermediária
- u - Número de neurônios da camada de saída
- $\widetilde{X_i}$ - Neurônio da camada de entrada da rede neural, onde $1 \leq i \leq p$
- $\widetilde{Z_j}$ - Neurônio da camada intermediária da rede neural, onde $1 \leq j \leq q$
- $\widetilde{Y_k}$ - Neurônio da camada de saída da rede neural, onde $1 \leq k \leq u$
- v_{ij} - Peso entre a camada de entrada i e a camada intermediária j
- w_{jk} - Peso entre a camada intermediária j e a camada de saída k
- $\widetilde{x_i}$ - Sinal de entrada da camada de entrada
- $\widetilde{z\_in_j}$ - Sinal de entrada da camada intermediária j
- $\widetilde{z_j}$ - Sinal de saída da camada intermediária j
- $\widetilde{y\_in_k}$ - Sinal de entrada da camada de saída
- $\widetilde{y_k}$ - Sinal de saída da camada de saída
- $\widetilde{t_k}$ - Valor de saída desejado
- $f(\widetilde{x})$ - Função bipolar sigmóide *fuzzy*
- $f'(\widetilde{x})$ - Derivada da função bipolar sigmóide *fuzzy*
- $\widetilde{\delta_k}$ - Fator de erro da camada de saída k
- $\widetilde{\Delta w_{jk}}$ - Termo de correção entre as camadas de saída k e intermediária j
- α - Taxa de aprendizagem
- $\widetilde{\delta_j}$ - Fator de erro da camada intermediária j
- $\widetilde{\delta\_in_j}$ - Fator de erro da camada de entrada
- $\widetilde{\Delta v_{ij}}$ - Termo de correção entre as camadas de entrada i e intermediária j

Capítulo 6 - Rede Neural-*Fuzzy* com Aprendizado por Reforço

p - Número de neurônios da camada de entrada
q - Número de neurônios da camada intermediária
u - Número de neurônios da camada de saída
$\widetilde{X_i}$ - Neurônio da camada de entrada da rede neural, onde $1 \leq i \leq p$
$\widetilde{Z_j}$ - Neurônio da camada intermediária da rede neural, onde $1 \leq j \leq q$
$\widetilde{Y_k}$ - Neurônio da camada de saída da rede neural, onde $1 \leq k \leq u$
v_{ij} - Peso entre a camada de entrada i e a camada intermediária j
w_{jk} - Peso entre a camada intermediária j e a camada de saída k
$\widetilde{x_i}$ - Sinal de entrada da camada de entrada
$\widetilde{z\_in_j}$ - Sinal de entrada da camada intermediária j
$\widetilde{z_j}$ - Sinal de saída da camada intermediária j
$\widetilde{y\_in_k}$ - Sinal de entrada da camada de saída
$\widetilde{y_k}$ - Sinal de saída da camada de saída
$\widetilde{t_k}$ - Valor de saída desejado
num_ep - Número de épocas de realização do algoritmo
α - Taxa de aprendizagem
γ - Taxa de desconto
r - Valor de recompensa *crisp*
$\widetilde{r}$ - Valor de recompensa *fuzzy*
l - Número da iteração atual
V_k - Valor atual da recompensa *crisp* do agente
$\widetilde{V_k}$ - Valor atual da recompensa *fuzzy* do agente
ϵ - Valor utilizado como critério de parada

Capítulo 7 - Resultados Numéricos

$6.6.mu$	-	Instância do problema *job shop* com 6 tarefas e 6 máquinas
$15.10la21$	-	Instância do problema *job shop* com 15 tarefas e 10 máquinas
$15.10la24$	-	Instância do problema *job shop* com 15 tarefas e 10 máquinas
$15.10la25$	-	Instância do problema *job shop* com 15 tarefas e 10 máquinas
$20.10la27$	-	Instância do problema *job shop* com 20 tarefas e 10 máquinas

Capítulo 1

Introdução Geral

Um problema de escalonamento trata da alocação de recursos, levando-se em consideração a execução de um conjunto de operações, com o intuito de alcançar um determinado objetivo, sujeito a um conjunto de restrições.

Neste trabalho são abordados dois problemas de escalonamento de tarefas considerando-os primeiramente com parâmetros precisos, e depois com parâmetros com incertezas.

O primeiro problema abordado foi o de escalonamento de tarefas em máquinas paralelas idênticas. Este problema é considerado *NP-completo* e consiste de um conjunto de n tarefas independentes que devem ser escalonadas em m máquinas paralelas idênticas. O objetivo do problema é encontrar uma forma de escalonamento que minimize o *makespan*. *Makespan*, neste caso, é o tempo total de processamento na máquina mais ocupada, que é a última a terminar a execução das tarefas a ela alocadas. Foi considerado que as tarefas devem ser sequenciadas de forma não-preemptiva.

No problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas, a incerteza foi aplicada aos tempos de processamento das tarefas, na forma de números *fuzzy*. Cada tempo de processamento das tarefas foi representado por um número triangular *fuzzy*.

O segundo problema abordado foi o *job shop*, que é um problema de considerável importância industrial. O problema de escalonamento *job shop* é considerado um problema *NP-completo* e consiste de um conjunto de n tarefas que precisam ser processadas em m máquinas. Neste problema, cada tarefa consiste de uma seqüência de operações que especificam a ordem das máquinas pelas quais a tarefa deverá ser processada. Cada tarefa é composta por uma lista ordenada de operações contendo a máquina que deve processá-la e em qual tempo de processamento. Cada operação precisa ser processada por uma dada máquina durante um período de tempo, sem preempção. O problema tem como objetivo encontrar uma forma de escalonar as tarefas pelas máquinas que minimize o *makespan*.

A grande dificuldade do problema do *job shop* está no gerenciamento de um alto número de restrições. No problema não existe restrição de precedência entre operações de tarefas distintas, e

existe restrição de precedência entre operações dentro de uma tarefa, indicando qual máquina a tarefa passará em cada tempo. Neste problema, as operações, quando estiverem sendo processadas, não poderão ser interrompidas (sem preempção). Cada máquina poderá processar apenas uma tarefa por vez, e cada tarefa deverá ser executada por uma máquina por vez.

No problema de escalonamento *job shop* com parâmetros com incertezas, foi considerado que os tempos de processamento das operações e os tempos possíveis para o início do processamento das operações não são precisamente conhecidos, e são representados por números triangulares *fuzzy*.

Vários estudos com relação ao problema clássico de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros precisos podem ser encontrados em (Nichols et al. (1978) e Ho and Wong (1995)).

Na resolução do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros precisos, foi utilizado o algoritmo memético para evoluir boas formas de escalonamento. O algoritmo memético aplicado neste caso foi o algoritmo genético acoplado com uma busca local do tipo *first improving*. Os resultados obtidos foram comparados com a heurística construtiva (Blocher and Chand (1991)), a heurística de melhoria (Jr. and Phillips (1981), Anderson (1996)), com o *simulated annealing* (Eglese (1990)) e com o algoritmo genético puro (Holland (1973)). A resolução deste problema, utilizando algoritmo memético, foi publicada em (Bonfim et al. (2002a)). O algoritmo memético (Moscato (1989), Moscato and Normam (1992)) é uma versão híbrida de algoritmos genéticos com busca local, e que tem como objetivo aumentar a performance do algoritmo genético. O algoritmo genético tem provado ser uma aproximação versátil e efetiva para a resolução de problemas de otimização. Contudo, existem muitas situações onde o algoritmo genético puro não oferece bons resultados e, sendo assim, vários métodos de hibridização têm sido propostos. A estratégia do *first improving* é uma heurística de melhoria que tem como objetivo melhorar localmente a solução de um problema.

Na resolução do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros precisos, para evoluir boas formas de escalonamento, foi aplicado também o algoritmo memético coevoluindo com redes neurais. Neste caso, o algoritmo memético foi aplicado para encontrar soluções para o problema, e as redes neurais foram alicadas para encontrar a função de *fitness* destas soluções. O algoritmo memético utilizado foi o algoritmo genético acoplado com uma busca local do tipo *first improving*. O uso de redes neurais deve-se ao fato de que nem sempre se conhece a função de *fitness* de um problema e, desta forma, pretende-se com este trabalho apresentar que, com o uso de redes neurais, coevoluindo com o algoritmo memético, é possível estimar a função de *fitness* de um problema. Neste trabalho foram utilizadas as redes neurais do tipo *backpropagation* (Bertsekas and Tsitsiklis (1996)) e com aprendizado por reforço (Barto and Sutton (1998)). A resolução deste problema foi publicada em (Bonfim et al. (2002b), Bonfim and Yamakami (2002b) e Bonfim and

Yamakami (2002a)).

A tomada de decisão *fuzzy* originou da idéia proposta por (Bellmann and Zadeh (1970)), que introduziu os conceitos de restrições *fuzzy*, de objetivo e decisão *fuzzy*. O problema de escalonamento com parâmetros com incertezas foi formulado por (Dubois et al. (1995)) como um problema de satisfação de restrições *fuzzy*, onde as restrições são mais ou menos relaxáveis ou sujeitas a preferências. Em casos onde os parâmetros do problema não são precisamente conhecidos, torna-se difícil estabelecer qual é o escalonamento ótimo das tarefas. Neste caso, a noção de ótimo também é considerada imprecisa, e o grau de otimalidade de um escalonamento (o quanto determinado escalonamento é **ótimo**) passa a ser analisado através de um número *fuzzy* no intervalo [0,1]. Este grau de otimalidade é importante e pode ser utilizado em situações onde se tem vários escalonamentos e se quer definir qual é o melhor entre eles, segundo o critério do decisor.

Na resolução do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas, foi utilizado o algoritmo memético com população estruturada (Moscato and Berretta (1999)) para evoluir boas formas de escalonamento. O algoritmo memético com população estruturada organiza a população de indivíduos em uma estrutura de árvore ternária com 3 níveis, com um número fixo de 13 agentes. Com o intuito de se adotar uma estrutura de dados eficiente e de rápida manipulação, foi escolhido o uso da população estruturada em árvores. Por se tratar de um problema de escalonamento com parâmetros incertos, onde torna-se difícil classificar um determinado escalonamento como ótimo, pois o problema agora possui restrições *fuzzy*, foi aplicado, neste caso, o conceito de otimalidade possível para medir a possibilidade de um determinado escalonamento ser ótimo. Sendo assim, o algoritmo memético foi aplicado para encontrar soluções para o problema, e o conceito de possibilidade foi aplicado para avaliar as melhores soluções. Esta abordagem do problema foi publicada em (Bonfim and Yamakami (2004a)).

Na resolução do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas, para evoluir boas formas de escalonamento, foi aplicado também o algoritmo memético com população estruturada coevoluindo com redes neurais. Neste caso, o algoritmo memético foi aplicado para encontrar soluções para o problema, as redes neurais para encontrar a função de *fitness* destas soluções, e os conceitos de possibilidade para avaliar as melhores soluções. As redes neurais utilizadas foram a *backpropagation* (Bertsekas and Tsitsiklis (1996)) e a com aprendizado por reforço (Barto and Sutton (1998)). Esta abordagem do problema foi publicada em (Bonfim and Yamakami (2004b)).

Em (Chanas and Kasperski (2004)), foram aplicados os conceitos de necessidade e possibilidade para encontrar soluções de um escalonamento ótimo para o problema de uma máquina simples. O grau de otimalidade possível e necessária mede a possibilidade e a necessidade de um escalonamento ser ótimo.

Vários estudos podem ser encontrados com relação ao problema clássico de escalonamento *job shop* com parâmetros precisos em (Blazewicz et al. (1996) e Fisher and Thompson (1963)). Na resolução do problema de escalonamento *job shop* com parâmetros precisos, foi utilizado o algoritmo memético com população estruturada para evoluir boas formas de escalonamento. Foi aplicado também, na resolução deste problema, o algoritmo memético com população estruturada coevoluindo com redes neurais, da mesma forma aplicada ao problema de escalonamento de tarefas em máquinas paralelas idênticas.

Na resolução do problema de escalonamento *job shop* com parâmetros com incertezas, foi utilizado o algoritmo memético com população estruturada (Moscato and Berretta (1999)) para evoluir boas formas de escalonamento. Por se tratar de um problema de escalonamento *fuzzy*, o conceito de otimalidade possível foi aplicado para medir a possibilidade de um determinado escalonamento ser ótimo. Sendo assim, foi aplicado o algoritmo memético para encontrar soluções para o problema, e os conceitos de possibilidade para avaliar as melhores soluções. Foi aplicado também, na resolução deste problema, o algoritmo memético com população estruturada coevoluindo com redes neurais, da mesma forma aplicada ao problema de escalonamento de tarefas em máquinas paralelas idênticas.

Esta tese está dividida em sete outros capítulos. No Capítulo 2 são apresentados alguns conceitos de conjuntos e números *fuzzy*. No Capítulo 3 é apresentada uma explanação a respeito do problema de escalonamento de tarefas com parâmetros precisos e, neste caso, é abordado o problema de escalonamento de tarefas em máquinas paralelas idênticas e o escalonamento *job shop*. No Capítulo 4 é apresentada uma explanação a respeito do problema de escalonamento de tarefas com parâmetros com incertezas e, neste caso, são abordados os dois problemas de escalonamento definidos neste trabalho. No Capítulo 5 é apresentada a arquitetura do escalonador neuro-memético, abordando seus componentes e sua aplicação aos problemas de escalonamento de tarefas com parâmetros precisos e, neste caso, são abordados os dois problemas de escalonamento definidos neste trabalho. No Capítulo 6 é apresentada a arquitetura do escalonador neuro-memético, abordando seus componentes e sua aplicação aos problemas de escalonamento de tarefas com parâmetros com incertezas e, neste caso, são abordados os dois problemas de escalonamento definidos neste trabalho. No Capítulo 7 são apresentadas as simulações e os resultados obtidos para todos os problemas abordados. Por último as conclusões e propostas de trabalhos futuros são apresentadas no Capítulo 8.

Capítulo 2

Considerações sobre conjuntos e números *fuzzy*

Neste capítulo são descritos alguns conceitos que são aplicados durante o trabalho como: números *fuzzy*, operações com números *fuzzy*, conceitos de possibilidade e comparação de números *fuzzy*. Para a abordagem destes itens são introduzidas algumas definições sobre o conceito *fuzzy*, conjuntos *fuzzy* e características de conjuntos *fuzzy*.

2.1 O conceito *fuzzy*

O conceito de *fuzzy* é muito utilizado no cotidiano para considerar certas situações como: o dia está "parcialmente" nublado, preciso perder "alguns" kilos para ficar "bem". E é também utilizado para classificar objetos como "largo", "comprido"; pessoas como: "velho", "jovem", "alto", "gordo", "magro". Estes termos entre aspas são considerados *fuzzy* porque envolvem imprecisão.

O conceito *fuzzy* pode ser entendido como uma situação onde não podemos responder simplesmente "sim" ou "não". Mesmo conhecendo as informações necessárias sobre a situação, dizer algo entre "sim" ou não", como por exemplo "talvez", "quase", se torna mais apropriado. Considere, por exemplo, informações como "homens baixos" ou "dias frios". Nada existe que determine exatamente qual a "altura" ou "temperatura" que podemos considerar como limites para tais informações. Se considerarmos como baixos todos os homens com menos de $1,50m$, então um homem com $1,52m$ não seria "baixo" e sim "quase baixo".

2.2 Conjuntos *fuzzy*

Na teoria clássica, os conjuntos são denominados *crisp* e um dado elemento do universo em discurso (domínio) pertence ou não pertence ao referido conjunto.

A teoria dos conjuntos *fuzzy*, que foi introduzida por (Zadeh (1965)), é uma generalização da teoria clássica. A extensão sugerida por Zadeh está na possibilidade de um determinado elemento poder pertencer a um conjunto com um valor chamado *grau de pertinência*. Assim, um elemento não simplesmente pertence ou não a um conjunto, como na teoria clássica, mas poderá pertencer a um conjunto com um grau de pertinência que varia no intervalo $[0, 1]$, onde o valor 0 indica uma completa exclusão, o valor 1 representa completa pertinência e os valores deste intervalo representam graus intermediários de pertinência do elemento com relação ao conjunto. A função que define os graus de pertinência dos elementos é chamada função de pertinência e é uma generalização da função característica da teoria clássica, uma vez que associa para todo elemento do universo de discurso um valor do intervalo $[0, 1]$ ao invés do conjunto de apenas dois elementos $\{0, 1\}$.

Definição 2.2.1. *Um conjunto fuzzy A é caracterizado por uma função de pertinência $\mu_A(x)$, representada pela Equação 2.1, mapeando os elementos x em um domínio X no intervalo* $[0, 1]$.

$$\mu_A(x) : X \rightarrow [0, 1]. \tag{2.1}$$

De acordo com a Equação 2.1, seja A um conjunto *fuzzy* definido no universo de discurso X, cada elemento $x \in X$ receberá um grau de pertinência com relação ao conjunto A através da função de pertinência $\mu_A(x)$. Quanto mais perto de 1 for o valor retornado por $\mu_A(x)$, com maior certeza o elemento x pertencerá ao conjunto A. Sendo assim, a função de pertinência vai dizer quanto é possível um elemento $x \in X$ pertencer ao conjunto A.

Pelo exemplo abaixo, seja o conjunto universo $X = \{5, 10, 20, 30, 40, 50, 60, 70, 80, 90\}$, e consideremos os seguintes conjuntos *fuzzy*: $A =$ jovens, $B =$ adultos, $C =$ velhos, para os quais atribuímos os graus de pertinência dos elementos dos conjuntos X na Tabela 2.1.

De acordo com a Tabela 2.1, as funções de pertinência podem ser representadas graficamente pela Figura 2.1.

O suporte de um conjunto *fuzzy* A, chamado de *supp*(A), no conjunto universo X é o conjunto clássico que contém todos os elementos de X que têm grau de pertinência maior do que zero, conforme apresenta a (Definição 2.2.2).

Definição 2.2.2. *supp*$(A) = x \in X | \mu_A(x) > 0$.

Tab. 2.1: Exemplo de pertinência para os conjuntos *fuzzy* A, B e C.

Idade	*Jovem*	*Adulto*	*Velho*
5	0	0	0
10	1	0	0
20	0.8	0.2	0
30	0.2	0.8	0
40	0	1	0
50	0	0.8	0.2
60	0	0.6	0.4
70	0	0.4	0.6
80	0	0.2	0.8
90	0	0	1

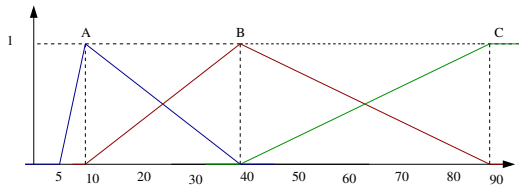

Fig. 2.1: Representação gráfica das funções de pertinência dos conjuntos *fuzzy* A, B e C.

Para o exemplo acima, o suporte do conjunto *fuzzy* A de jovens seria: $supp(jovem) = 10, 20, 30$.

Com possibilidade de pertinência parcial dos elementos, a teoria de Zadeh possui um espaço de aplicabilidade muito maior do que a teoria clássica na solução de vários tipos de problemas reais. Existem diversas classes de objetos no mundo real que não possuem um critério bem definido de pertinência. Por exemplo, o conjunto das pessoas jovens, das pessoas altas, dos carros velhos, entre muito outros conjuntos que não possuem uma fronteira bem definida para podermos delimitar com certeza os elementos pertencentes daqueles não pertencentes ao conjunto. Em tais casos, fica muito difícil utilizar a teoria clássica.

A lógica *fuzzy* foi inicialmente aplicada entre 1970 e 1980 em indústrias européias. O japão incluiu seu uso com aplicações na indústria após 1980, e nos Estados Unidos o interesse pelo seu uso foi despertado por volta de 1990. Devido ao seu desenvolvimento e às inúmeras possibilidades práticas dos sistemas *fuzzy*, e o grande sucesso comercial de suas aplicações, a lógica *fuzzy* é considerada uma técnica padrão e tem grande aceitação na área de controle de processos industriais.

2.2.1 Princípio de extensão

O princípio de extensão, que foi introduzido por Zadeh (Zadeh (1975a), Zadeh (1975b) e Zadeh (1975c)), permite mapear conjuntos *fuzzy* de um domínio em conjuntos *fuzzy* de outro domínio a partir de uma função clássica. Este princípio surge da necessidade de aplicarmos uma função clássica a argumentos imprecisos. Neste sentido, se temos uma função f e precisa-se aplicar esta função aos argumentos *fuzzy*, então, é preciso recorrer ao princípio de extensão.

Um argumento *fuzzy* descreve a distribuição de possibilidade do argumento da função f. Assim, para cada possível valor que a variável da função pode assumir, o funcional produz sua possível imagem fornecendo também a distribuição de possibilidade dessa imagem.

Dado um conjunto X e a função $y = f(x), x \in X$, o princípio de extensão é definido pela Equação 2.2.

$$\mu_B(y) = \sup_{x \in X, y = f(x)} \{\mu_A(x)\} \tag{2.2}$$

onde A é o conjunto *fuzzy* em X e B é o conjunto imagem de A em Y.

Dependendo da função pode ocorrer que diferentes valores de entrada sejam mapeados no mesmo valor de saída. Neste caso, é preciso determinar a possibilidade de cada um dos valores de saída, através da combinação dos graus de pertinência de todos os valores de entrada que mapeiam o mesmo valor de saída, o que pode ser feito usando o operador **sup**.

Pode ocorrer também de, ao fim do processo, obtermos diferentes distribuições de possibilidades para cada argumento $(x_1, x_2, ..., x_n)$ da função f. Neste caso, é necessário aplicar um operador de conjunção *fuzzy* para decidir qual será o grau de possibilidade da imagem. Mais precisamente, considere uma função clássica como a definida pela Equação 2.3, e $A_1, A_2, ..., A_n$, subconjuntos *fuzzy* de $X_1, X2, ..., X_n$, respectivamente.

$$f : X_1 x X_2 x ... x X_n \rightarrow Y \tag{2.3}$$

O princípio de extensão produz um conjunto $B = f(A_1, A_2, ..., A_n)$ que é um subconjunto *fuzzy* de Y, cuja função de pertinência é dada pela Equação 2.4.

$$\mu_B(y) = \sup_{\substack{x_1, x_2, ..., x_n \\ y = f(x_1, x_2, ..., x_n)}} \{\min[\mu_{A_1}(x), \mu_{A_2}(x), ..., \mu_{A_n}(x)]\} \tag{2.4}$$

Portanto, o princípio de extensão nos permite obter a distribuição de possibilidade da imagem de uma função cujos argumentos são *fuzzy*, consistindo em uma técnica extremamente útil na investigação sobre a ação de uma função clássica sobre argumentos imprecisos.

2.3 Números *fuzzy*

Um número *fuzzy* expressa a incerteza relacionada ao parâmetro modelado por este número.

Definição 2.3.1. *Um número fuzzy $\widetilde{T}$ é um conjunto fuzzy no espaço dos números reais* $\mathbf{R}$ *com função de pertinência* $\mu_{\widetilde{T}} : \mathbf{R} \to [0, 1]$.

De acordo com (Dubois et al. (1995)), quando quiser exibir um elemento $x \in X$ que tipicamente pertença ao conjunto *fuzzy* A, pode-se representá-lo com seu valor de função de pertinência sendo maior que um $\alpha \in [0, 1]$. O conjunto de tais elementos é chamado de α_{corte} (A_α) de A.

Definição 2.3.2. *O α_{corte}, $\alpha \in [0, 1]$, de um número fuzzy $\widetilde{T}$ é o conjunto descrito pela Equação 2.5.*

$$\widetilde{T}_\alpha = \begin{cases} \text{Supp}(\widetilde{T}) & \text{para} \quad \alpha = 0. \\ x : \{\mu_{\widetilde{T}}(x) \geq \alpha\} & \text{para} \quad \alpha \in (0, 1]. \end{cases} \tag{2.5}$$

2.3.1 Número *fuzzy* triangular

Um número *fuzzy* triangular, que é um caso especial de número *fuzzy*, é denotado por:

$$\widetilde{t} = [\underline{t}, t^0, \overline{t}].$$

Um número *fuzzy* triangular tem um valor modal t^0 (valor máximo da função de pertinência associada), um espalhamento à esquerda $t^0 - \underline{t}$ e um espalhamento à direita $\overline{t} - t^0$. A função de pertinência de um número *fuzzy* triangular, com esta representação, é definida de acordo com a Equação 2.6.

$$\mu_{\widetilde{t}}(x) = \begin{cases} \frac{x-\underline{t}}{t^0-\underline{t}} & \text{se} \quad x \in [\underline{t}, t^0] \\ \frac{\overline{t}-x}{\overline{t}-t^0} & \text{se} \quad x \in [t^0, \overline{t}] \\ 0 & \text{caso contrário} \end{cases} \tag{2.6}$$

A representação gráfica de um número *fuzzy* triangular do tipo $\widetilde{t} = [\underline{t}, t^0, \overline{t}]$ é apresentada pela Figura 2.2.

Uma outra forma de representar um número *fuzzy* é usando Intervalos de Confiança, como apresentado em (Kaufmann and Gupta (1991)). Neste caso, se escrevermos a função inversa da Equação 2.6, na forma de intervalos e usando α-cortes, um número *fuzzy* triangular pode ser denotado por:

$$\widetilde{t}_\alpha = [t_1^\alpha, t_2^\alpha] \in \mathbb{F}(\mathbb{R}), \text{onde} t_1^\alpha = (t^0 - \underline{t})\alpha + \underline{t} \text{ e } t_2^\alpha = -(\overline{t} - t^0)\alpha + \overline{t}.$$

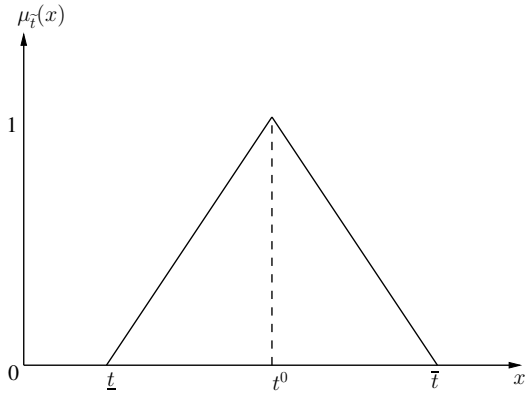

Fig. 2.2: Número *fuzzy* triangular

Na notação acima, t_1^α e t_2^α representam, respectivamente, a parte crescente e decrescente da função de pertinência, $\forall \alpha \in [0,1]$. $\mathbb{F}(\mathbb{R})$ é o conjunto *fuzzy* sobre $\mathbb{R}$.

O uso de α-cortes, de acordo com (Cantão (2003)), nos permite operar com números *fuzzy* ponto-a-ponto, ou seja, discretizarmos os números *fuzzy* em n α-cortes, e efetuarmos operações aritméticas em cada α-corte, usando as operações algébricas usuais. Outro motivo para o uso desta representação é devido à facilidade em trabalhar, por exemplo, com números *fuzzy* trigonométricos e exponenciais.

Com o uso desta representação com intervalos de confiança, a implementação torna-se mais simples e evita distorções das características *fuzzy*, como ocorrem no uso de operações *fuzzy* apresentadas por (Dubois and Prade (1980)). A construção algorítmica de um número *fuzzy* como intervalo de confiança pode ser verificada através do algoritmo Algoritmo 1 e maiores detalhes podem ser observados em (Cantão (2003)).

Algoritmo 1 Construção de um número *fuzzy*

Entrada: $\delta = \frac{1}{n-1}$ para $n \in \mathbb{N}$ tal que $n > 1$ (tamanho da discretização); $\alpha = 0$ e t^0, $\underline{t}$ e $\overline{t}$ constantes.
para $i = 0$ até $i = n$ **faça**
  $t_1^\alpha = (t^0 - \underline{t})\alpha + \underline{t}$
  $t_2^\alpha = -(\overline{t} - t^0)\alpha + \overline{t}$
  $\alpha = \alpha + \delta$
fim para
Saída: Número *fuzzy* $\widetilde{t}_\alpha$

2.3.2 Operações com números *fuzzy*

A operação de *adição* entre números *fuzzy* triangulares, de notação $\widetilde{a} = [\underline{a}, a^0, \overline{a}]$ e $\widetilde{b} = [\underline{b}, b^0, \overline{b}]$, é realizada de acordo com a Equação 2.7.

$$\widetilde{a} \oplus \widetilde{b} = [\underline{a} + \underline{b}, a^0 + b^0, \overline{a} + \overline{b}]. \tag{2.7}$$

A seguir são apresentadas algumas operações com números *fuzzy*, definidos como intervalos de confiança. Para maiores detalhes consultar (Kaufmann and Gupta (1991) e Cantão (2003)).

A operação de *adição* entre números *fuzzy*, de notação $\widetilde{a}_\alpha = [a_1^\alpha, a_2^\alpha]$ e $\widetilde{b}_\alpha = [b_1^\alpha, b_2^\alpha]$, é realizada de acordo com a Equação 2.8.

$$\widetilde{a}_\alpha \oplus \widetilde{b}_\alpha = [a_1^\alpha + b_1^\alpha, a_2^\alpha + b_2^\alpha]. \tag{2.8}$$

A operação de *adição* de um número *fuzzy* $\widetilde{a}_\alpha = [a_1^\alpha, a_2^\alpha]$ com um número ordinário k, é realizada de acordo com a Equação 2.9.

$$\widetilde{a}_\alpha \oplus k = [a_1^\alpha + k, a_2^\alpha + k]. \tag{2.9}$$

A operação de *subtração* entre números *fuzzy*, de notação $\widetilde{a}_\alpha = [a_1^\alpha, a_2^\alpha]$ e $\widetilde{b}_\alpha = [b_1^\alpha, b_2^\alpha]$, é realizada de acordo com a Equação 2.10.

$$\widetilde{a}_\alpha \ominus \widetilde{b}_\alpha = [a_1^\alpha - b_2^\alpha, a_2^\alpha - b_1^\alpha]. \tag{2.10}$$

A operação de *subtração* de um número *fuzzy* $\widetilde{a}_\alpha = [a_1^\alpha, a_2^\alpha]$ por um número ordinário k, é realizada de acordo com a Equação 2.11.

$$\widetilde{a}_\alpha \ominus k = [a_1^\alpha - k, a_2^\alpha - k]. \tag{2.11}$$

A operação de *multiplicação* entre números *fuzzy*, de notação $\widetilde{a}_\alpha = [a_1^\alpha, a_2^\alpha]$ e $\widetilde{b}_\alpha = [b_1^\alpha, b_2^\alpha]$, é realizada de acordo com a Equação 2.12.

$$\widetilde{a}_\alpha \otimes \widetilde{b}_\alpha = [min(a_1^\alpha b_1^\alpha, a_1^\alpha b_2^\alpha, a_2^\alpha b_1^\alpha, a_2^\alpha b_2^\alpha), max(a_1^\alpha b_1^\alpha, a_1^\alpha b_2^\alpha, a_2^\alpha b_1^\alpha, a_2^\alpha b_2^\alpha)]. \tag{2.12}$$

A operação de *multiplicação* entre um número *fuzzy* ($\widetilde{a}_\alpha = [a_1^\alpha, a_2^\alpha]$) e um número ordinário k, é realizada de acordo com a Equação 2.13.

$$\widetilde{a}_\alpha \otimes k = [min(a_1^\alpha k, a_2^\alpha k), max(a_1^\alpha k, a_2^\alpha k)]. \tag{2.13}$$

A operação de *divisão* de números *fuzzy*, de notação $\widetilde{a}_\alpha = [a_1^\alpha, a_2^\alpha]$ e $\widetilde{b}_\alpha = [b_1^\alpha, b_2^\alpha]$, é realizada de acordo com a Equação 2.14.

$$\widetilde{a}_\alpha \oslash \widetilde{b}_\alpha = [min(\frac{a_1^\alpha}{b_1^\alpha}, \frac{a_1^\alpha}{b_2^\alpha}, \frac{a_2^\alpha}{b_1^\alpha}, \frac{a_2^\alpha}{b_2^\alpha}), max(\frac{a_1^\alpha}{b_1^\alpha}, \frac{a_1^\alpha}{b_2^\alpha}, \frac{a_2^\alpha}{b_1^\alpha}, \frac{a_2^\alpha}{b_2^\alpha})]. \tag{2.14}$$

A operação de *divisão* de um número ordinário k por um número *fuzzy* ($\widetilde{a}_\alpha = [a_1^\alpha, a_2^\alpha]$), é realizada de acordo com a Equação 2.15.

$$k \oslash \widetilde{a}_\alpha = [min(\frac{k}{a_1^\alpha}, \frac{k}{a_2^\alpha}), max(\frac{k}{a_1^\alpha}, \frac{k}{a_2^\alpha})]. \tag{2.15}$$

A operação de *exponencial* de um número *fuzzy* negativo (-$\widetilde{a}_\alpha = [-a_1^\alpha, -a_2^\alpha]$), é realizada de acordo com a Equação 2.16.

$$\exp(-\widetilde{a}_\alpha) = [\exp(min(-a_1^\alpha, -a_2^\alpha)), \exp(max(-a_1^\alpha, -a_2^\alpha))]. \tag{2.16}$$

2.3.3 Conceitos de possibilidade e necessidade

A seguir são definidas duas medidas importantes para este trabalho, de *possibilidade* e *necessidade*.

Definição de possibilidade. Dados dois números *fuzzy* $\widetilde{a}$ e $\widetilde{b}$, a possibilidade $Poss(\widetilde{a}, \widetilde{b})$ é definida conforme a Equação 2.17.

$$\text{Poss}(\widetilde{a}, \widetilde{b}) = \sup_{x \in X}\{\min[\mu_{\widetilde{a}}(x), \mu_{\widetilde{b}}(x)]\}. \tag{2.17}$$

Definição de necessidade. Dados dois números *fuzzy* $\widetilde{a}$ e $\widetilde{b}$, a necessidade $Nec(\widetilde{a}, \widetilde{b})$ é definida conforme a Equação 2.18.

$$Nec(\widetilde{a}, \widetilde{b}) = \inf_{x \in X}\{\max[\mu_{\widetilde{a}}(x), 1 - \mu_{\widetilde{b}}(x)]\}. \tag{2.18}$$

A possibilidade fornece uma idéia de *sobreposição* e a necessidade fornece uma idéia de *inclusão* de B em A, se considerarmos os números como conjuntos *fuzzy*.

Considerando os números *fuzzy* a e b, com funções de pertinência $\mu_{\widetilde{a}}(x)$, $\mu_{\widetilde{b}}(x)$ e $\overline{\mu_{\widetilde{b}}(x)} = 1 - \mu_{\widetilde{b}}(x)$ apresentadas pela Figura 2.3, a possibilidade $Poss(\widetilde{a}, \widetilde{b})$ é apresentada pela Figura 2.4, e a necessidade $Nec(\widetilde{a}, \widetilde{b})$ é apresentada pela Figura 2.5.

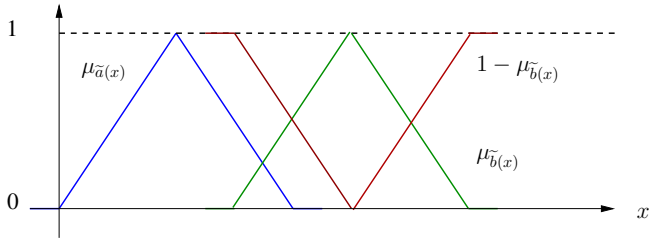

Fig. 2.3: Funções de pertinência $\mu_{\widetilde{a}}(x)$, $\mu_{\widetilde{b}}(x)$ e $\overline{\mu_{\widetilde{b}}(x)} = 1 - \mu_{\widetilde{b}}(x)$.

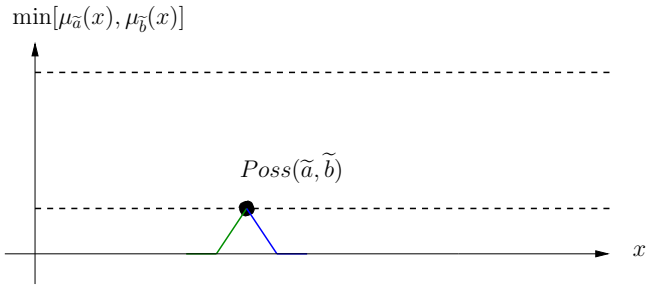

Fig. 2.4: Possibilidade $Poss(\widetilde{a}, \widetilde{b})$.

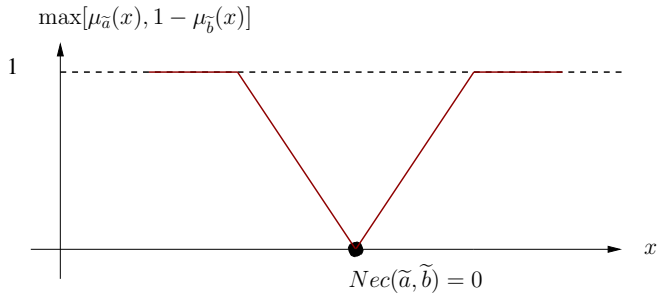

Fig. 2.5: Necessidade $Nec(\widetilde{a}, \widetilde{b})$.

2.3.4 Comparação de números *fuzzy*

Para respondermos a pergunta de qual é o maior ou menor número entre vários números *fuzzy* é preciso avaliar o grau de possibilidade do número *fuzzy* ser maior ou menor que outro número *fuzzy*.

Se quisermos verificar se $\widetilde{a} \geq \widetilde{b}$, devemos analisar a Equação 2.19.

$$Poss(\widetilde{a} \geq \widetilde{b}) = \sup_{x \geq y} \{\min[\mu_{\widetilde{a}}(x), \mu_{\widetilde{b}}(y)]\}. \tag{2.19}$$

Podemos concluir pela Equação 2.20 que, para $\widetilde{a} = [\underline{a}, a^0, \overline{a}]$ e $\widetilde{b} = [\underline{b}, b^0, \overline{b}]$:

$$\begin{cases} \text{Poss}(\widetilde{a} \geq \widetilde{b}) = 1; \text{ se e somente se } a^0 \geq b^0 \\ \text{Poss}(\widetilde{a} \geq \widetilde{b}) = 0; \text{ se } \overline{a} < \underline{b} \\ \text{Poss}(\widetilde{b} \geq \widetilde{a}) = hgt(\widetilde{a} \cap \widetilde{b}) = \mu_{\widetilde{a}}(d) \end{cases} \tag{2.20}$$

Pela Figura 2.6 podemos verificar a função de pertinência $\mu_{\widetilde{a}}(d)$.

De acordo com a Figura 2.6 e a Equação 2.20, temos as seguintes definições:

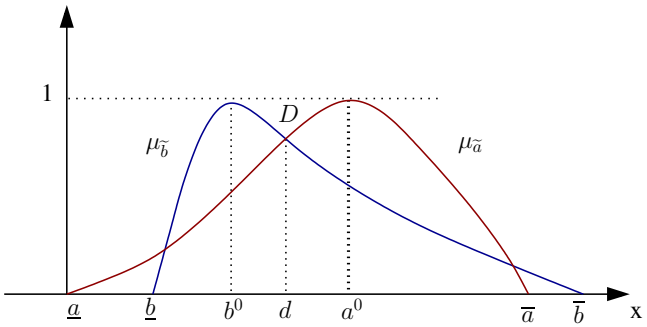

Fig. 2.6: Função de pertinência $\mu_{\widetilde{a}}(d)$.

Definição 2.3.3. *d é a distância do ponto D, que representa o máximo da intersecção entre $\mu_{\widetilde{a}}$ e $\mu_{\widetilde{b}}$.*

Definição 2.3.4. *$hgt(\widetilde{a} \cap \widetilde{b})$ representa a altura da intersecção dos números fuzzy $\widetilde{a}$ e $\widetilde{b}$.*

Para compararmos os números $\widetilde{a}$ e $\widetilde{b}$, é preciso calcular a $Poss(\widetilde{a} \geq \widetilde{b})$ e a $Poss(\widetilde{b} \geq \widetilde{a})$.

2.3.5 Defuzzificação

A defuzzificação é o processo inverso da fuzzificação e é utilizada quando se pretende converter um resultado *fuzzy* em um valor *crisp*. Este valor pode ser obtido através das áreas das funções de pertinência da saída de cada conjunto ou número *fuzzy* de entrada. A técnica de defuzzificação utilizada neste trabalho foi o Método da Centróide (Sugeno (1985)), que também é chamado de Centro de Gravidade. Neste método, o centro da área sob a curva de saída *fuzzy* é o valor *crisp*. O centro da área pode ser obtido pela Equação 2.21.

$$z^0 = \frac{\int_a^b \mu_{\widetilde{C(z)}} z dz}{\int_a^b \mu_{\widetilde{C(z)}} dz} \tag{2.21}$$

Na Equação 2.21, z^0 representa o centro de massa da função de pertinência $\mu_{\widetilde{C(z)}}$ do número *fuzzy* $\widetilde{C(z)}$ e, a e b são os intervalos de $\widetilde{C}$.

Capítulo 3

Escalonamento de tarefas com parâmetros precisos

Os problemas de manufatura constituem grandes desafios nos processos produtivos nas indústrias e fábricas, considerando que assistimos hoje, rápidas e constantes mudanças nos perfis de demandas.

Um dos problemas de manufatura comumente encontrado é o de determinação da seqüência de processamento de tarefas em máquinas, de tal forma a otimizar um determinado objetivo. Estes problemas são denominados de problemas de escalonamento de tarefas ou seqüenciamento de peças.

Um problema de escalonamento trata da alocação de recursos, levando-se em consideração a execução de um conjunto de operações, e com o intuito de alcançar um determinado objetivo, sujeito a um conjunto de restrições. Este problema é aplicado em várias áreas como, por exemplo, o escalonamento de programas em computadores, o escalonamento de carros que devem sofrer reparos em uma oficina, o escalonamento de professores em uma escola.

Neste capítulo são abordados dois tipos de problemas de escalonamento de tarefas com parâmetros precisos - o escalonamento de tarefas em máquinas paralelas idênticas e o problema do *job shop*.

3.1 Escalonamento de tarefas em máquinas paralelas idênticas com parâmetros precisos

O problema de escalonamento de tarefas em máquinas paralelas idênticas consiste de um conjunto de n tarefas independentes que devem ser escalonadas em m máquinas paralelas idênticas. Neste problema as tarefas são independentes e cada uma precisa ser alocada a uma das máquinas. Neste caso não existe preempção, as tarefas não são divididas entre as máquinas, e não existem restrições

de precedência. O objetivo do problema é encontrar uma forma de escalonamento que minimize o *makespan*. *Makespan* é o tempo total de processamento na máquina mais ocupada, que é a última a terminar a execução das tarefas a ela alocadas.

Muitas aplicações deste problema podem ser encontradas em sistemas de manufatura (Blazewicz et al. (1988) e Cheng and Sin (1990)). Um exemplo simples de tal problema poderia ser a atribuição de tarefas a um conjunto de datilógrafos. No início de cada dia, um número de tarefas deverão ser atribuídas aos datilógrafos. Assume-se que todos os datilógrafos têm as mesmas habilidades. O supervisor precisa decidir quais tarefas serão atribuídas para quais datilógrafos, de forma que a última tarefa seja completada no menor tempo possível. O problema parece simples se as tarefas pudessem ser divididas, e um datilógrafo fizesse uma parte de uma tarefa e outro datilógrafo terminasse a outra parte. A função de atribuição de tarefas do supervisor seria fácil se todas as tarefas tivessem o mesmo tempo de duração e, neste caso, não faria diferença a escolha do datilógrafo para a execução da tarefa. O problema é que isto não é verdade porque uma tarefa poderia ser digitar uma pequena carta, enquanto outra tarefa poderia ser digitar um extenso relatório e, neste caso, caberia ao supervisor atribuir as tarefas aos datilógrafos de forma a obter o menor tempo possível para a sua execução.

3.1.1 Modelo matemático do problema

Seja m um conjunto de máquinas e n um conjunto de tarefas, onde cada tarefa possui um tempo de processamento t_i. A variável $x_{i,j}$ indica se a tarefa i é processada pela máquina j. Neste problema, cada máquina pode processar uma ou mais tarefas, a cada tempo, e cada tarefa pode ser processada por apenas uma máquina. O objetivo é minimizar o *makespan* M. O modelo matemático para este problema é caracterizado pela Formulação 3.1.

$$\begin{aligned} \text{Min M} = \quad & \underbrace{MAX}_{1\leq j\leq m} \textstyle\sum_{i=1}^{n} x_{i,j} t_i. \\ \text{s. a :} \quad & \textstyle\sum_{j=1}^{m} x_{i,j} = 1; \quad 1 \leq i \leq n. \\ & x_{i,j} = \begin{cases} 1, \text{ se a tarefa } i \text{ é atribuída à máquina } j. \\ 0, \text{ caso contrário.} \end{cases} \end{aligned} \tag{3.1}$$

O problema de escalonamento de tarefas em máquinas paralelas idênticas é tradicionalmente resolvido por vários métodos como *branch and bound* e programação dinâmica. São métodos que oferecem a solução ótima para problemas de tamanho razoável, mas apresentam algumas limitações na resolução de problemas de larga escala. Esta limitação é geralmente resolvida com a aplicação de procedimentos heurísticos. Este problema é classificado como um problema inteiro misto devido

à característica combinatória, e a dificuldade em resolvê-lo cresce de acordo com o seu tamanho. Um problema envolvendo 10 tarefas e 2 máquinas possui $2^{10} = 1024$ possíveis soluções a serem exploradas. Em um problema inteiro misto não é uma tarefa trivial encontrar a solução ótima. Sendo assim, trata-se de um problema NP-completo (Lenstra et al. (1977)) e, para resolvê-lo, é preciso um algoritmo em tempo polinomial. O desenvolvimento de algoritmos eficientes para resolver problemas NP-completo é uma tarefa muito difícil. Muitos estudos foram feitos, e podemos destacar alguns métodos heurísticos na subseção 3.1.2.

3.1.2 Métodos heurísticos aplicados ao problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros precisos

O princípio dos métodos heurísticos e meta-heurísticos é de prover, em tempo polinomial, uma solução que seja a melhor possível. Estes métodos procuram um compromisso entre a rapidez em obter um solução e a qualidade da solução. A qualidade da solução de um método heurístico é medida pela distância entre o valor obtido e o valor da solução ótima, que muitas vezes é desconhecido.

Os métodos heurísticos são procedimentos utilizados para resolver problemas através de um enfoque "intuitivo", no qual a estrutura do problema possa ser interpretada e explorada inteligentemente para obter uma boa solução.

3.1.2.1 Heurística construtiva

A heurística construtiva tem como objetivo gerar uma solução inicial para o problema de escalonamento de tarefas em máquinas paralelas idênticas. Uma heurística muito conhecida e aplicada a esse problema é a LPT (*Least Processing Time*)(Blocher and Chand (1991)). Este algoritmo primeiro arranja os tempos de processamento das tarefas em ordem decrescente e, então, iniciando com a primeira tarefa da lista, atribui as tarefas, uma a uma, às máquinas que têm menor quantidade de trabalho no momento. O comprimento do escalonamento é definido como o tempo de término de execução da última tarefa. A heurística construtiva LPT é apresentada pelo Algoritmo 2.

3.1.2.2 Heurística de melhoria

A heurística de melhoria é considerado um método de busca local que tem como objetivo melhorar uma solução do problema. Ela é baseada na troca de tarefas entre as máquinas (Jr. and Phillips (1981), Anderson (1996)). Os métodos de busca local possuem uma série de passos, percorrendo soluções vizinhas e procurando melhorar a solução corrente em cada estágio da busca. Determinar se uma determinada seqüência de atribuição de tarefas nas máquinas produz um *makespan* ótimo, é em

Algoritmo 2 Heurística construtiva LPT

Entrada: n: número de tarefas, m: número de máquinas, t_i: tempo de processamento da tarefa $i; 0 \leq i \leq n-1$.
para $j = 0$ até $j = m-1$ **faça**
$T(m_j) = 0$, tempo de processamento total na máquina j
fim para

- ordenar as tarefas em ordem decrescente de tempo de processamento - $t_k, t_l, ..., t_p$, onde $t_k \geq t_l \geq t_p$

para $i = 0$ até $i = n-1$ **faça**
$Min\{T(m_j)\}; 0 \leq j \leq m-1$
$T(m_j) = T(m_j) + t_i$
fim para
Saída: $Max\{T(m_j)\}; 0 \leq j \leq m-1$

geral muito difícil. Entretanto, é mais fácil determinar se uma determinada seqüência produzirá um *makespan* menor que o *makespan* obtido até um determinado momento da simulação.

A heurística de melhoria tem como objetivo promover a troca de tarefas entre duas máquinas se esta troca produzir um *makespan* menor que o *makespan* atual. A idéia geral da heurística é, a partir de uma solução factível $x \in X$, onde X é considerado o conjunto de todas as soluções possíveis, construir uma vizinhança de soluções $N(x)$, selecionar uma amostra de soluções da vizinhança $V \subseteq N(x)$, e depois escolher uma nova solução x' da amostra, de forma que essa solução seja melhor que a solução x. A vizinhança adotada para a máquina que possui o maior tempo de execução (*makespan*) são todas as tarefas das demais máquinas.

Existem dois mecanismos para percorrer a vizinhança adotada: o *first improving* e o *best improving*.

Na heurística de melhoria com mecanismo de percorrer a vizinhança do tipo *first improving*, o algoritmo termina sua busca na vizinhança assim que encontrar a primeira melhora no valor do *makespan*.

Na heurística de melhoria com mecanismo de percorrer a vizinhança do tipo *best improving*, o algoritmo faz uma busca em todas as máquinas, duas a duas, procurando trocas entre tarefas que melhorem o menor *makespan* encontrado até o momento. Ao final da execução, a melhor troca é a que será mantida. A melhor troca é aquela que produziu o menor *makespan* entre todas as trocas. No mecanismo do tipo *best improving*, pode-se percorrer toda a vizinhança na busca pela melhoria da solução ou pode-se percorrer uma vizinhança reduzida como, por exemplo, metade da vizinhança. O algoritmo de busca local tem convergência em tempo polinomial.

As heurísticas de melhoria dos tipos *first improving* e *best improving* são apresentadas pelo Algo-

ritmo 3.

Algoritmo 3 Heurística de melhoria dos tipos *first improving* e *best improving*

Entrada: n: número de tarefas, m: número de máquinas, p: máquina mais ocupada, $num_{tarefas}(p)$: número de tarefas alocadas à máquina p, t_i: tempo de processamento da tarefa $i; 0 \leq i \leq n-1$, M: *makespan* atual, $tipo_{alg}$: 1 (first improving) ou 2 (best improving), $T(m_j)$: tempo total de processamento na máquina $j, 0 \leq j \leq m-1$.

para $k = 0$ até $k = num_{tarefas}(p)$ **faça**
  para $w = 0$ até $w = m-1$ **faça**
    se $w\ != p$ **então**
      para $z = 0$ até $z = num_{tarefas}(w)$ **faça**
        $T(m_w) = T(m_w) - t_z + t_k$
        $T(m_p) = T(m_p) - t_k + t_z$
        se $Max\{T(m_j)\} < M$ **então**
          $M = Max\{T(m_j)\}; 0 \leq j \leq m-1$
          $p = \arg_j\{Max\{T(m_j)\}\}; 0 \leq j \leq m-1$
          se $(tipo_{alg} == 1)$ **então**
            quebra da execução dos *loops*
          senão
            $k = 0$
          fim se
        senão
          $T(m_w) = T(m_w) + t_z - t_k$
          $T(m_p) = T(m_p) + t_k - t_z$
        fim se
      fim para
    fim se
  fim para
fim para

Saída: M

3.1.2.3 *Simulated annealing*

O *Simulated Annealing* é um método de busca heurística que também é utilizado para obtenção de boas soluções em problemas de otimização combinatoriais. É uma aproximação heurística projetada para gerar uma boa, mas não necessariamente ótima solução, dentro de um tempo razoável de computação (Eglese (1990)).

O *Simulated Annealing* é um tipo de algoritmo de busca local. Uma forma simples de busca local é, para um solução inicial escolhida aleatoriamente, gerar um vizinho desta solução e calcular a mudança no custo. Se uma redução no custo for encontrada, a solução corrente é substituída pela vizinhança gerada e, caso contrário, a solução corrente é mantida. O processo é repetido até que ne-

nhuma melhoria seja encontrada na vizinhança da solução corrente e, então, o algoritmo termina sua execução com o mínimo local. Embora o algoritmo seja simples e rápido na execução, a desvantagem do método é que o mínimo local encontrado pode estar muito distante do mínimo global. Uma maneira de melhorar a solução é executar o algoritmo várias vezes, iniciando com soluções diferentes, e o melhor mínimo local encontrado é considerado.

O método do *Simulated Annealing* é apresentado pelo Algoritmo 4.

O *Simulated Annealing* tenta evitar parar em um ótimo local, movendo-se em uma vizinhança que minimize o valor do *makespan*. A aceitação ou rejeição de uma troca de tarefas é determinada por uma seqüência de números aleatórios, mas com uma probabilidade controlada.

A função $e^{-(T(m_p)-M)/V(k)}$ é chamada de probabilidade de aceitação de um movimento (de uma troca de tarefas entre máquinas) quando $(T(m_p)-M) \leq 0$ e $(T(m_w)-M) \leq 0$. Esta função implica que um acréscimo pequeno é mais provável de ser aceito do que grandes acréscimos. Além disso, quando $V(k)$(temperatura) é alta, a maioria das trocas é aceita. Quando $V(k)$ se aproxima de zero, a maioria é rejeitada.

O algoritmo do *simulated annealing* é iniciado com uma temperatura relativamente alta para evitar que a solução fique presa num mínimo local. O algoritmo prossegue, tentando um certo número de movimentos em cada temperatura, enquanto este parâmetro é gradualmente reduzido.

A idéia geral do algoritmo é efetuar trocas de tarefas entre a máquina mais cheia e as máquinas da vizinhança, de forma a obter um melhor resultado do *makespan*, ou seja, na tentativa de obter o mínimo global. A vizinhança é analisada um determinado número de vezes para cada temperatura. Os valores da temperatura vão decrescendo até chegar no valor 1.

3.1.2.4 Algoritmos genéticos

Os algoritmos genéticos foram propostos inicialmente por Holland, na década de 70 (Holland (1973)). Este algoritmo é baseado nos conceitos da evolução genética, de onde buscou-se importar os mecanismos de evolução, seleção natural das espécies e transmissão da informação genética para os ambientes computacionais.

O algoritmo genético é um método populacional, pois utiliza uma população de indivíduos, na qual cada um representa uma solução para o problema. A cada geração do algoritmo, os indivíduos mais aptos têm maior probabilidade de sobreviverem e, conseqüentemente, de transmitirem boas qualidades a seus descendentes.

Este método usa o vocabulário da genética. Um **indivíduo** é representado por um cromossomo, que é uma cadeia de genes, representando uma possível solução para o problema. Um **gen** representa uma certa característica no cromossomo. O valor de um gen é denominado **alelo**. O **genótipo** é o conjunto de genes que irá caracterizar o problema. Uma **população** é um conjunto de indivíduos

Algoritmo 4 Método de busca heurística *simulated annealing*

Entrada: n: número de tarefas, m: número de máquinas, p: máquina mais ocupada, $num_{tarefas}(p)$: número de tarefas alocadas à máquina p, t_i: tempo de processamento da tarefa $0 \leq i \leq n-1$, M: *makespan atual*, $V(t) = 50, 49, 48, 47, 46, ..., 1$: vetor com temperaturas em ordem decrescente de 50 a 1, $tipo_{alg}$: 1 (first improving) ou 2 (best improving).
para $r = 1$ até 50 **faça**
  para $k = 0$ até $k = num_{tarefas}(p)$ **faça**
    para $w = 0$ até $w = m - 1$ **faça**
      se $w\ != p$ **então**
        para $z = 0$ até $z = num_{tarefas}(w)$ **faça**
          $T(m_w) = T(m_w) - t_z + t_k$
          $T(m_p) = T(m_p) - t_k + t_z$
          se $(T(m_p) - M < 0$ e $T(m_w) - M < 0)$ **então**
            $M = Max\{T(m_j)\}; 0 \leq j \leq m-1$
            $p = \arg_j\{Max\{T(m_j)\}\}; 0 \leq j \leq m-1$
            se $(tipo_{alg} == 1)$ **então**
              quebra da execução dos *loops*
            senão
              $k = 0$
            fim se
          senão
            se (random(0,1) < $e^{-(T(m_p)-M)/V(r)}$) e (random(0,1) < $e^{-(T(m_w)-M)/V(r)}$) **então**
              $M = Max\{T(m_j)\}; 0 \leq j \leq m-1$
              $p = \arg_j\{Max\{T(m_j)\}\}; 0 \leq j \leq m-1$
              se $(tipo_{alg} == 1)$ **então**
                quebra da execução dos *loops*
              senão
                $k = 0$
              fim se
            senão
              $T(m_w) = T(m_w) + t_z - t_k$
              $T(m_p) = T(m_p) + t_k - t_z$
            fim se
          fim se
        fim para
      fim se
    fim para
  fim para
fim para
Saída: M

(**cromossomos**). A criação de uma população é chamada **geração**. A **função de aptidão** *(fitness function)* representa uma avaliação da qualidade da solução do indivíduo. O processo de reprodução

(***crossover***) envolve dois indivíduos (soluções do problema), dando origem a um novo, o qual herdará características genéticas de seus pais. O processo de **seleção** avalia cada indivíduo, selecionando os mais aptos para fazerem parte da próxima geração. A **mutação**, neste contexto, tem o objetivo de introduzir diversidade à população. A **mutação** e o ***crossover***, chamados operadores genéticos, são aplicados, geração a geração, de forma que a população sofra um processo evolutivo, gerando indivíduos melhor adaptados.

O método do *algoritmo genético* é apresentado pelo Algoritmo 5.

Algoritmo 5 Pseudocódigo do algoritmo genético

Entrada: n: número de tarefas, m: número de máquinas, t_i: tempo de processamento da tarefa i; $0 \leq i \leq n-1$, T_{pop}: tamanho da população, T_{subpop}: tamanho da subpopulação, N_{gera}: número de gerações.

- Gerar população inicial ($V(ind); 0 \leq ind \leq T_{pop} - 1$)
- Calcular vetor de *makespan* de cada indivíduo ($M(ind) = \underbrace{MAX}_{0 \leq j \leq m-1} \sum_{i=0}^{n-1} x_{i,j} t_i$), onde $x_{i,j}$ indica se a tarefa i é processada pela máquina j
- Avaliar função de fitness dos indivíduos, gerando $F(ind) = \frac{1}{1+M(ind)}$

para $k = 0$ até $k = N_{gera}$ **faça**
Passos:

- Selecionar 2 indivíduos da população inicial
- Aplicar *crossover* gerando 2 novos indivíduos para a subpopulação
- Selecionar 1 indivíduo da população
- Aplicar mutação gerando 1 novo indivíduo para a subpopulação
- Completar a geração da subpopulação ($V2(ind); 0 \leq ind \leq T_{subpop} - 1$)
- Calcular vetor de *makespan* de cada indivíduo ($M(ind) = \underbrace{MAX}_{0 \leq j \leq m-1} \sum_{i=0}^{n-1} x_{i,j} t_i$), onde $x_{i,j}$ indica se a tarefa i é processada pela máquina j
- Avaliar função de fitness dos indivíduos, gerando $F(ind) = \frac{1}{1+M(ind)}$
- Selecionar o melhor indivíduo da subpopulação e escolher os demais indivíduos aleatoriamente da subpopulação para compor a população da próxima geração

fim para
Saída: Melhor indivíduo da população

Na implementação do algoritmo genético para o problema de escalonamento de tarefas em má-

quinas paralelas idênticas (Bonfim et al. (2002b)), a representação genética utilizada para codificar as soluções do problema foi a seguinte: as posições do cromossomo representam as n tarefas que devem ser escalonadas nas m máquinas. O cromossomo foi representado por um vetor de dimensão igual ao número de tarefas. Cada alelo do cromossomo é representado pelo número da máquina ao qual a tarefa, associado ao alelo, está alocada. O primeiro passo no algoritmo é gerar a população inicial, sendo que um dos indivíduos foi gerado pela heurística construtiva LPT, e os demais foram gerados aleatoriamente, segundo a codificação especificada.

O segundo passo do algoritmo genético é a execução do cálculo do valor do *makespan* para cada indivíduo da população. Para efetuar o cálculo do *makespan*, é somado, para cada máquina, todos os tempos de processamento das tarefas a ela alocadas. O maior tempo entre todas as máquinas é considerado o *makespan* do indivíduo.

O passo seguinte do algoritmo é o cálculo da função de *fitness*. A função de aptidão (função de *fitness*), que avalia cada solução potencial, foi definida com base no *makespan* da solução. A função de *fitness* utilizada para resolução do problema foi $\frac{1}{1+makespan}$.

Em seguida, o algoritmo executa uma seqüência de passos enquanto a condição de parada não for atingida. A condição de parada aplicada ao problema foi o número de gerações, que é definido como dado de entrada do algoritmo. Essa seqüência de passos engloba a seleção de indivíduos para a realização das operações genéticas de *crossover* e de *mutação*, cálculo do *makespan* e avaliação da função de *fitness* dos indivíduos da população, geração da subpopulação e seleção de indivíduos dentro da subpopulação para compor a nova população para a geração seguinte do algoritmo.

Os operadores genéticos aplicados a esse problema foram:

- ***Crossover* simples entre indivíduos aleatórios:** nesta operação são escolhidos dois indivíduos da população inicial, por meio da roleta ponderada. Em seguida, é determinado o ponto de quebra e efetuado o *crossover* simples, que é a troca entre as partes dos dois cromossomos a partir do ponto de quebra.

- ***Crossover* simples entre um indivíduo aleatório e o melhor indivíduo:** nesta operação é escolhido um indivíduo da população inicial, por meio da roleta ponderada. O melhor indivíduo dentre a população inicial ou dentre a subpopulação é selecionado, o ponto de quebra é determinado e o *crossover* simples é realizado.

- ***Crossover* uniforme entre indivíduos aleatórios:** nesta operação são escolhidos dois indivíduos da população inicial, por meio da roleta ponderada. Em seguida, a porcentagem de genes que serão trocados é determinada e é realizado um sorteio de alelos na quantidade determinada. O *crossover* é realizado somente nos alelos envolvidos.

- ***Crossover* uniforme entre um indivíduo aleatório e o melhor indivíduo:** nesta operação é escolhido um indivíduo da população inicial, por meio da roleta ponderada. O melhor indivíduo dentre a população inicial ou dentre a subpopulação é selecioado e, em seguida, é determinada a porcentagem de genes que serão trocados. Em seguida, é realizado um sorteio de alelos na quantidade determinada e o *crossover* é efetuado somente nos alelos envolvidos.

- **Mutação Simples:** nesta operação é escolhido um indivíduo da população inicial de forma aleatória e, em seguida, é determinada a porcentagem de genes que serão trocados. Na sequência, é realizado um sorteio de alelos na quantidade determinada e, para cada alelo sorteado, é sorteado um valor dentro do limite permitido para o alelo, alterando o valor do mesmo.

- **Mutação Inversiva:** nesta operação é escolhido um indivíduo da população inicial de forma aleatória e, em seguida, é determinada a porcentagem de genes que serão trocados. Na sequência, é realizado um sorteio de alelos na quantidade determinada e, para cada alelo sorteado, o valor é permutado com o alelo adjacente.

A seleção dos indivíduos para a realização das operações genéticas é feita através do mecanismo chamado roleta ponderada (*roulette wheel*). No *roulette wheel*, a probabilidade de seleção de um cromossomo é diretamente proporcional a seu valor da função de *fitness*, sendo que, se o indivíduo possuir um *fitness* grande, maior será a chance de ser escolhido. Cromossomos mais aptos terão maior probabilidade de serem escolhidos. Em cada operação de *crossover* são gerados dois novos indivíduos para a subpopulação e, em cada operação de mutação é gerado um novo indivíduo para a subpopulação.

A subpopulação no algoritmo genético é composta pelos indivíduos criados através das operações genéticas aplicadas aos indivíduos da população e por novos indivíduos gerados aleatoriamente, seguindo o critério da codificação do cromossomo, buscando aumentar a diversidade.

Uma vez realizadas as operações genéticas e criada a subpopulação, os valores do *makespan* e da função de *fitness* de cada indivíduo são calculados. Em seguida, é realizado um método de seleção com o intuito de selecionar indivíduos da subpopulação para compor a nova população para a próxima geração do algoritmo. O mecanismo de seleção utilizado em (Bonfim et al. (2002b)) foi o método **Salvacionista**, onde é selecionado o melhor indíviduo, que é o que possui o menor valor de *fitness*, e os demais indivíduos são selecionados aleatoriamente, garantindo que a próxima geração será pelo menos igual à população anterior.

3.1.2.5 Algoritmos meméticos

O algoritmo memético, que foi introduzido por Moscato e Norman (Moscato (1989), Moscato and Normam (1992), Moscato and Berretta (1999)), é também chamado de algoritmo genético híbrido,

pois além de permitir a realização de operações genéticas, utiliza um (ou mais) operador(es) de busca local com o intuito de reforçar o processo de melhoria das soluções.

O algoritmo genético tem provado ser uma aproximação versátil e efetiva para a resolução de problemas de otimização. Contudo, existem muitas situações onde o algoritmo genético puro não oferece bons resultados e, sendo assim, vários métodos de hibridização têm sido propostos.

Segundo (Moscato and Normam (1992)), algoritmos meméticos são aproximações baseadas em população e em busca heurística, e são utilizados para resolver problemas de otimização combinatória. Testes têm demonstrado que, para alguns problemas, estes algoritmos são mais rápidos que os algoritmos genéticos tradicionais. Basicamente, estes algoritmos combinam heurísticas de busca local com operadores de *crossover*. O algoritmo memético também pode ser uma combinação de algoritmo genético com heurísticas construtivas, com o *simulated annealing* ou com métodos exatos.

Moscato e Norman (Moscato and Normam (1992)) têm introduzido o termo algoritmo memético para denominar algoritmos genéticos aos quais a busca local apresenta significante participação. Em algoritmos meméticos, um otimizador local é aplicado a cada filho, antes que o mesmo seja inserido na população, com o intuito de colocá-lo em um ótimo local.

Nesta abordagem híbrida, o algorítmo genético é utilizado para executar a exploração global na população, enquanto o otimizador local é utilizado para melhorar localmente a qualidade das soluções.

O pseudocódigo do *algoritmo memético* é apresentado pelo Algoritmo 6.

Na implementação do algoritmo memético para o problema de escalonamento de tarefas em máquinas paralelas idênticas (Bonfim et al. (2002b)), todos os passos do algoritmo genético foram executados conforme descritos no Algoritmo 5 e, para a execução do método híbrido, foi incluída uma busca local após a seleção dos indivíduos da subpopulação que irão compor a população na geração seguinte. O método de busca local aplicado neste problema foi a heurística de melhoria do tipo *first improving*, descrita no Algoritmo 3. Essa busca local foi aplicada apenas no melhor indivíduo da população, que é aquele com menor valor de *fitness*. O objetivo na aplicação da busca local é promover uma melhora no resultado encontrado até o momento.

3.2 Escalonamento *job shop* com parâmetros precisos

O escalonamento do tipo *job shop* consiste de um conjunto de n tarefas que precisam ser processadas em m máquinas. Neste problema, cada tarefa consiste de uma seqüência de operações que especificam a ordem das máquinas pelas quais a tarefa será processada. Cada tarefa é composta por uma lista ordenada de operações contendo a seqüência das máquinas que irá processá-la e os correspondentes tempos de processamento. Cada operação precisa ser processada por uma dada máquina

Algoritmo 6 Pseudocódigo do algoritmo memético

Entrada: n: número de tarefas, m: número de máquinas, t_i: tempo de processamento da tarefa i, $0 \leq i \leq n-1$, T_{pop}: tamanho da população, T_{subpop}: tamanho da subpopulação, N_{gera}: número de gerações.

- Gerar população inicial ($V(ind); 0 \leq ind \leq T_{pop} - 1$)
- Calcular vetor de *makespan* de cada indivíduo ($M(ind) = \underbrace{MAX}_{0 \leq j \leq m-1} \sum_{i=0}^{n-1} x_{i,j} t_i$), onde $x_{i,j}$ indica se a tarefa i é processada pela máquina j
- Avaliar função de fitness dos indivíduos, gerando $F(ind) = \frac{1}{1+M(ind)}$

para $k = 0$ até $k = N_{gera}$ **faça**

Passos:

- Selecionar 2 indivíduos da população inicial
- Aplicar *crossover* gerando 2 novos indivíduos para a subpopulação
- Selecionar 1 indivíduo da população
- Aplicar mutação gerando 1 novo indivíduo para a subpopulação
- Completar a geração da subpopulação ($V2(ind); 0 \leq ind \leq T_{subpop} - 1$)
- Calcular vetor de *makespan* de cada indivíduo ($M(ind) = \underbrace{MAX}_{0 \leq j \leq m-1} \sum_{i=0}^{n-1} x_{i,j} t_i$), onde $x_{i,j}$ indica se a tarefa i é processada pela máquina j
- Avaliar função de fitness dos indivíduos, gerando $F(ind) = \frac{1}{1+M(ind)}$
- Selecionar o melhor indivíduo da subpopulação e escolher os demais indivíduos aleatoriamente da subpopulação para compor a população da próxima geração
- Aplicar busca local no melhor indivíduo da população e atualizar seu valor de *makespan* e de *fitness*

fim para

Saída: Melhor indivíduo da população

durante um período de tempo, sem preempção.

O objetivo do problema é encontrar uma forma de escalonar as tarefas pelas máquinas que minimize o *makespan*. O problema está sujeito às seguintes restrições:

- Não existe restrição de precedência entre operações de tarefas diferentes.
- Existe restrição de precedência entre operações dentro de uma tarefa, indicando qual máquina a tarefa passará em cada tempo.
- As operações, quando estiverem sendo processadas, não poderão ser interrompidas (sem preempção).
- Cada tarefa poderá ser executada por uma máquina por vez.

De acordo com (Blazewicz et al. (1996)), o escalonamento do tipo *job shop* é um problema *NP-completo* e pertence ao conjunto dos problemas considerados difíceis de serem tratados. É um problema de considerável importância industrial. A dificuldade neste problema está no gerenciamento de um alto número de restrições.

A história do problema de escalonamento *job shop*, que se iniciou há mais de 40 anos, é também a história do *benchmark* proposto por (Fisher and Thompson (1963)). Esta instância é composta por 10 máquinas e 10 tarefas. Este *benchmark* promoveu a competição, por pelo menos 25 anos, entre pesquisadores procurando a melhor forma de solucioná-lo.

3.2.1 Modelo matemático do problema

Seja m um conjunto de máquinas e n um conjunto de tarefas. Seja $O = 0, ..., m-1$ o conjunto de operações, onde 0 é considerada a operação inicial de todas as tarefas e $m-1$ é considerada a operação final de todas as tarefas. Neste problema todas as tarefas possuem a mesma quantidade de operações, que é igual ao número de máquinas.

Seja A o conjunto de pares ordenados de operações restringidos por relações de precedência para cada tarefa. Para cada máquina k, o conjunto E_k descreve o conjunto de todos os pares de operações a serem executadas na máquina k, ou seja, operações que não poderão ser sobrepostas. Para cada operação i, seu tempo de processamento p_i é fixado, e o tempo possível para início do processamento da operação i na máquina l é ti_{il}, que é uma variável determinada durante a otimização do problema.

O objetivo é minimizar o tempo total gasto entre o início da primeira operação e o término da última operação, que representa o *makespan* M. A variável M representa o tempo final tf_{il} de execução da tarefa mais ocupada. O modelo matemático para este problema é caracterizado pela Formulação 3.2.

$$
\begin{aligned}
\text{Min M} = \quad & \underbrace{MAX}_{0\leq l\leq m-1} \{tf_{il}\}\,, \forall i; 0 \leq i \leq n-1. \\
\text{s. a :} \quad & ti_{jk} - ti_{ik} \geq p_i \;\; ; \forall (i,j) \in A \;\; , \forall i; 0 \leq i \leq n-1 \;\; , \forall j; 0 \leq j \leq n-1. \\
& ti_{jk} - ti_{ik} \geq p_i \;\; \text{ou} \;\; ti_{ik} - ti_{jk} \geq p_j \;\; ; \forall \{i,j\} \in E_k \;\; , \forall k; 0 \leq k \leq m-1. \\
& ti_{ik} \geq 0 \;\; ; \forall i \in O \;\; , \forall i; 0 \leq i \leq n-1\,, \forall k; 0 \leq k \leq m-1.
\end{aligned}
\tag{3.2}
$$

No modelo matemático do problema, a primeira restrição assegura que a seqüência de processamento das operações em cada tarefa corresponde a uma ordem predeterminada. A segunda restrição denota que existe somente uma tarefa sendo atendida por uma máquina em um determinado momento e, a terceira restrição assegura a execução de todas as operações. Qualquer solução que atenda essas três restrições é chamada de escalonamento.

A Tabela 3.1 apresenta o *benchmark* para o problema 6 x 6 ($m = 6$ e $n = 6$) de (Muth and Thompson (1963)).

Tab. 3.1: *Benchmark* do problema do *job shop* 6x6.

	(m,p)	*(m,p)*	*(m,p)*	*(m,p)*	*(m,p)*	*(m,p)*
Tarefa 0:	(2,1)	(0,3)	(1,6)	(3,7)	(5,3)	(4,6)
Tarefa 1:	(1,8)	(2,5)	(4,10)	(5,10)	(0,10)	(3,4)
Tarefa 2:	(2,5)	(3,4)	(5,8)	(0,9)	(1,1)	(4,7)
Tarefa 3:	(1,5)	(0,5)	(2,5)	(3,3)	(4,8)	(5,9)
Tarefa 4:	(2,9)	(1,3)	(4,5)	(5,4)	(0,3)	(3,1)
Tarefa 5:	(1,3)	(3,3)	(5,9)	(0,10)	(4,4)	(2,1)

Na tabela acima, m representa a máquina que a tarefa será processada e p indica o tempo de processamento da tarefa na máquina m. Nesse *benchmark*, a tarefa 0 precisa ser processada inicialmente pela máquina 2, por uma unidade de tempo e, em seguida, pela máquina 0, por três unidades de tempo. Cada tarefa apresenta uma seqüência de máquinas que irão processá-la. Um escalonamento factível é o escalonamento de uma seqüência de tarefas em cada máquina tal que a ordem das operações de cada tarefa seja preservada. Cada máquina não poderá processar mais de uma tarefa ao mesmo tempo e diferentes operações de uma mesma tarefa não poderão ser processadas simultaneamente em máquinas diferentes. O mínimo *makespan* encontrado para esse problema é o valor 55, e um escalonamento ótimo desse problema é representado pelo Diagrama de Grantt apresentado na Figura 3.1.

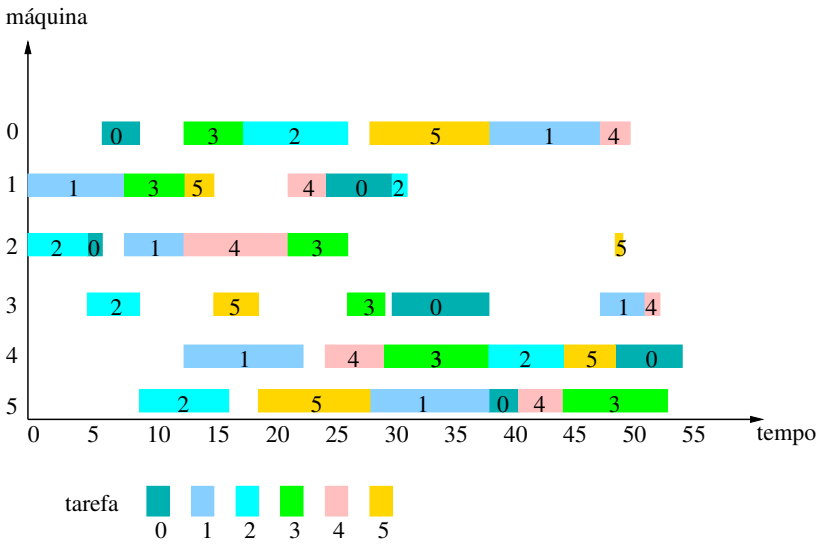

Fig. 3.1: Diagrama de Grantt do escalonamento ótimo para o problema do *job shop* 6x6.

3.2.2 Métodos heurísticos aplicados ao problema do *job shop* com parâmetros precisos

O problema de escalonamento do tipo *job shop* é um problema prático muito importante e é considerado um dos piores membros da classe de problemas NP-completos. Em geral, é difícil aplicar métodos de busca convencionais para encontrar a solução ótima ou próxima da ótima em um tempo razoável para esta classe de problemas. Por este motivo tem se aplicado métodos heurísticos na resolução deste tipo de problema, pois o método heurístico tem o compromisso de obter rapidamente uma solução de qualidade.

São destacados a seguir alguns métodos heurísticos aplicados ao problema do *job shop*.

3.2.2.1 Algoritmo memético com população estruturada

O algoritmo memético com população estruturada organiza a população de indivíduos em uma estrutura de árvore ternária com 3 níveis, com um número fixo de 13 agentes (Moscato and Berretta (1999)), conforme a Figura 3.2. Um dos primeiros passos na implementação de um algoritmo genético ou memético, é a definição da representação da solução do problema (representação do cromossomo). Com o intuito de adotarmos uma estrutura de dados eficiente e de rápida manipulação, optou-se pelo uso da população estruturada em árvores.

Nesta estrutura em árvore, a população é formada hierarquicamente por agentes líderes e subordinados. Esta classificação é feita de acordo com o valor da solução que cada agente apresenta. Todo nó raiz possui um valor melhor que os seus nós filhos. Sendo assim, o valor armazenado no agente 1

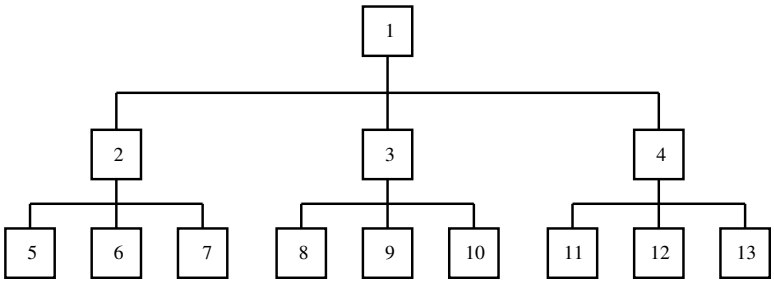

Fig. 3.2: População estruturada em árvore.

apresenta a melhor solução encontrada na população.

Podemos verificar que, nesta estrutura, o agente 1 é líder dos agentes 2, 3 e 4. O agente 2 tem como subordinados os agentes 5, 6 e 7. O agente 3 tem como subordinados os agentes 8, 9 e 10 e o agente 4 tem como subordinados os agentes 11, 12 e 13.

Cada agente na população representa dois indivíduos, um chamado de *pocket* e o outro de *current*. A solução que o indivíduo *pocket* possui é sempre melhor que a solução do indivíduo *current*. Quando um *current*, para um dado agente, possui uma solução melhor que o *pocket*, estes valores devem ser trocados.

O indivíduo *pocket* de um líder possui melhor valor que os indivíduos *pocket* de seus agentes subordinados. Quando isto não ocorre, os indivíduos *pocket* destes agentes são trocados de posição. Com estas trocas, garantimos que todo indivíduo *pocket* será melhor que o indivíduo *current* do mesmo agente, que todo *pocket* líder será melhor que os indivíduos *pocket* subordinados e que o agente 1 possuirá a melhor solução da população.

A medida tomada para avaliar qual é a melhor solução da população é chamada de função de avaliação ou função de *fitness*. A função de *fitness* é responsável por atribuir um valor referente à qualidade desta solução, indicando o quão próximo um indivíduo está de ser a solução do problema. Assim, a cada indivíduo na população é atribuído um valor, denominado *fitness*, que representa o valor da solução na função objetivo do problema.

Neste tipo de estrutura em árvore, a operação de *crossover* é executada somente entre indivíduos *pocket* e segue o seguinte critério: cada *pocket* líder cruzará com todos os seus indivíduos *pocket* subordinados. O filho resultante deste cruzamento será armazenado no indivíduo *current* do subordinado. Suponha que seja executado um cruzamento entre o *pocket* do agente 1 (líder) e o *pocket* do agente 2 (subordinado), o filho resultante deste cruzamento é armazenado no *current* do agente 2. Como o cruzamento é efetuado para todos os indivíduos *pocket*, a cada geração são executados 12 cruzamentos.

A operação de mutação também é executada apenas em indivíduos *pocket*. Neste caso, é escolhido aleatoriamente um indivíduo *pocket*, com exceção ao *pocket* do agente 1, e a mutação é efetuada. O

filho gerado após esta operação é armazenado no indivíduo *current* deste *pocket*.

3.2.2.2 Aplicação do algoritmo memético com população estruturada ao problema do *job shop* com parâmetros precisos

Um conceito importante e crítico na execução de um algoritmo genético ou memético é a escolha da representação de possíveis soluções para o problema dentro de um cromossomo ou indivíduo.

Para o problema abordado do *job shop* com parâmetros precisos, o cromossomo foi codificado de duas formas. Inicialmente por um vetor V, de nxm colunas, onde n é o número de tarefas e m é o número de máquinas. Cada campo do vetor v_k, onde $0 \leq k \leq (nxm) - 1$, é preenchido por $num_t; 0 \leq num_t \leq n - 1$, que indica a tarefa a ser realizada. Este vetor contém a seqüência de tarefas que serão atendidas. Para a geração de cromossomos factíveis, é escolhida aleatoriamente uma tarefa num_t e então é verificado quantas vezes esta tarefa já foi sorteada, definindo assim o número de operação para essa tarefa. Para que o cromossomo seja factível, esse num_t tem que aparecer m vezes no vetor. O cromossomo foi codificado na forma de vetor, com a seqüência de atendimento das operações, para facilitar a aplicação dos operadores genéticos.

No vetor abaixo podemos verificar um exemplo da codificação do cromossomo para o problema 6 x 6 ($m = 6$ e $n = 6$) da Tabela 3.1. Esse é o cromossomo que resultou o escalonamento ótimo apresentado na Figura 3.1.

$$\mathbf{V} = \begin{matrix}(2 & 0 & 1 & 3 & 0 & 1 & 4 & 2 & 3 & 2 & 3 & 5 & 5 & 2 & 4 & 5 & 5 & 0 \\ 2 & 1 & 1 & 3 & 4 & 3 & 2 & 5 & 5 & 1 & 0 & 0 & 1 & 4 & 4 & 4 & 0 & 3)\end{matrix}$$

A seguir, o cromossomo foi codificado por uma matriz A_{mxn}, de m linhas e n colunas, onde m é o número de máquinas e n é o número de tarefas. Esta matriz é montada com base no vetor V e no *benchmark*. Cada elemento da matriz a_{ji}, onde $0 \leq j \leq m - 1$ e $0 \leq i \leq n - 1$, é preenchido por um número $num_t; 0 \leq num_t \leq n - 1$, que indica qual tarefa será processada pela máquina. Sendo assim, cada linha do cromossomo indicará a seqüência de operações que deverá ser processada em cada máquina. Nessa seqüência não existe repetição de tarefas.

Para a geração de cromossomos factíveis, é escolhida aleatoriamente uma tarefa num_t e então é verificado quantas vezes esta tarefa já foi sorteada, definindo assim o número de operação para essa tarefa. Com base na informação do número de operação, busca-se na matriz de operações o número j da máquina que irá processar a operação para a tarefa num_t. Em seguida, a matriz de cromossomos é preenchida na posição a_{jk}, onde k é a próxima posição vazia na linha j da matriz, com o valor num_t.

A matriz abaixo apresenta um exemplo de cromossomo codificado para o problema 6 x 6 ($m = 6$ e $n = 6$) da Tabela 3.1. Esse é o cromossomo que resultou o escalonamento ótimo apresentado na Figura 3.1.

$$\mathbf{A} = \begin{bmatrix} 0 & 3 & 2 & 5 & 1 & 4 \\ 1 & 3 & 5 & 4 & 0 & 2 \\ 2 & 0 & 1 & 4 & 3 & 5 \\ 2 & 5 & 3 & 0 & 1 & 4 \\ 1 & 4 & 3 & 2 & 5 & 0 \\ 2 & 5 & 1 & 0 & 4 & 3 \end{bmatrix}$$

De acordo com a matriz acima, a linha 1 do cromossomo, que representa a máquina 0, indica a seqüência das tarefas que serão atendidas pela máquina. Sendo assim, a máquina 0 atenderá inicialmente a tarefa 0, depois a tarefa 3, depois a tarefa 2, e assim por diante.

O primeiro passo na execução do algoritmo memético com população estruturada é a geração da população inicial. Foram gerados 26 indivíduos, 13 indivíduos *pocket* e 13 indivíduos *current*. Os 26 indíviduos foram codificados das duas maneiras descritas acima.

O passo seguinte na execução do algoritmo é a avaliação da função de *fitness*. O critério utilizado para a avaliação das soluções foi o *makespan*. Para calcular o *makespan* no problema do *job shop* é preciso montar a janela de tempo para cada cromossomo, conforme a Figura 3.1.

A janela de tempo é armazenada em uma matriz JT_{mxn}, de m linhas e n colunas, onde m é o número de máquinas e n é o número de tarefas. Cada elemento da matriz $jt_{ji} = (ti_{ji}/tf_{ji})$, onde $0 \leq j \leq m-1$ e $0 \leq i \leq n-1$, é preenchido por um par ordenado indicando o tempo de início do processamento da tarefa (ti) e o tempo final do processamento da tarefa (tf). Para o cálculo dessa janela de tempo é necessário obedecer as restrições do problema analisando a matriz de operações O_{nxm} e a matriz que codifica o cromossomo A_{mxn}. A janela de tempo para o problema $6x6$, representada na Figura 3.1, apresenta a matriz JT_{mxn} destacada abaixo.

$$\mathbf{JT} = \begin{bmatrix} 6/9 & 13/18 & 18/27 & 28/38 & 38/48 & 48/51 \\ 0/8 & 8/13 & 13/16 & 22/25 & 25/31 & 31/32 \\ 0/5 & 5/6 & 8/13 & 13/22 & 22/27 & 49/50 \\ 5/9 & 16/19 & 27/30 & 31/38 & 48/52 & 52/53 \\ 13/23 & 25/30 & 30/38 & 38/45 & 45/49 & 49/55 \\ 9/17 & 19/28 & 28/38 & 38/41 & 41/45 & 45/54 \end{bmatrix}$$

Para o cálculo da função de *fitness* $(F(ind))$ de um indivíduo, pega-se o tempo final máximo na janela de tempo do indivíduo $(MAX\{tf_j\}; 0 \leq j \leq m-1)$. Esse procedimento é realizado para cada indivíduo da população.

Foi considerado como valor de *fitness* o valor do *makespan*. Sendo assim, a cada geração, a

solução considerada melhor é aquela que apresenta o menor valor de *fitness*. Todos os indivíduos são avaliados e, aquele que possuir o menor valor de *fitness* é considerado o melhor indivíduo da população.

No Apêndice A é exemplificada a criação de cromossomos, através de um vetor V e uma matriz A_{mxn}, para um problema de escalonamento *job shop* de 3 tarefas e 2 máquinas. Nesse apêndice é também exemplificada a criação da matriz de janela de tempo JT_{mxn}.

O próximo passo na execução do algoritmo é a ordenação da árvore, seguindo os critérios de indivíduos *pocket* e *current*, de forma que o melhor indivíduo (com menor valor de *fitness*) fique na posição do agente 1.

Os passos seguintes, onde são executados os operadores genéticos, são realizados enquanto o critério de parada não for alcançado. O critério de parada utilizado neste trabalho foi o número de gerações. Os operadores genéticos executados neste algoritmo foram o *crossover* e mutação. Foram implementados e testados para este problema, 7 tipos de *crossover* e 3 tipos de *mutação*. Os operadores genéticos são aplicados no cromossomo codificado em vetor. A seguir são descritos os operadores de *crossover* e de *mutação*.

O *crossover* 1 é realizado entre indivíduos *pocket* líder e *pocket* subordinado. O filho gerado por este *crossover* é armazenado no indivíduo *current* do subordinado. Nesta operação genética, é gerada aleatoriamente uma taxa de pontos de cruzamento (p_{cruz}, $0 \leq p_{cruz} \leq ((nxm) - 1)/2)$), que representa a quantidade de pares de operações que serão permutadas. A seguir, para cada ponto de cruzamento, é efetuado o sorteio de duas tarefas distintas e de duas operações distintas. Após o sorteio, a operação 1 da tarefa 1 do indivíduo *pocket* líder é permutada com a operação 2 da tarefa 2 do indivíduo *pocket* subordinado, e a operação 2 da tarefa 2 do indivíduo *pocket* líder é permutada com a operação 1 da tarefa 1 do indivíduo *pocket* subordinado. Ao final desta troca, são gerados dois novos indivíduos. Com essa permuta o cromossomo mantém-se factível, pois é mantido o número correto de operações para cada tarefa. Este número de operações é igual ao número de máquinas. Após executadas todas as trocas para os pontos de cruzamento, é analisado qual é o melhor indivíduo gerado e este é armazenado no *current* do indivíduo subordinado. Este tipo de *crossover* pode ser classificado como um *crossover* simples.

O *crossover* 2 é realizado entre indivíduos *pocket* líder e *pocket* subordinado. O filho gerado por este *crossover* é armazenado no indivíduo *current* do subordinado. Nesta operação genética, é sorteada uma operação ($op; 0 \leq op \leq m - 1$) e, em seguida, todas as tarefas para esta operação do indivíduo *pocket* líder são colocadas, na mesma posição, no primeiro filho gerado. O restante das posições do primeiro filho são preenchidas com as operações do indivíduo *pocket* subordinado. Para a geração do segundo filho, todas as tarefas para a operação sorteada do indivíduo *pocket* subordinado são colocadas, na mesma posição, neste indivíduo filho. O restante das posições são preenchidas com

as operações do indivíduo *pocket* líder. Ao final da operação é analisado qual é o melhor indivíduo gerado e este é armazenado no *current* do indivíduo subordinado.

O *crossover* 3 é realizado entre indivíduos *pocket* líder e *pocket* subordinado. O filho gerado por este *crossover* é armazenado no indivíduo *current* do subordinado. Nesta operação genética, para cada posição no vetor do cromossomo dos filhos, um dos pais é escolhido, uma posição aleatória deste pai, ainda não escolhida, é escolhida e colocada no vetor do primeiro filho, e uma posição aleatória, ainda não escolhida, do outro cromossomo pai, é escolhida e colocada no vetor do segundo filho. Ao final da operação é analisado qual é o melhor indivíduo gerado e este é armazenado no *current* do indivíduo subordinado.

O *crossover* 4 é realizado entre indivíduos *pocket* líder e *pocket* subordinado. O filho gerado por este *crossover* é armazenado no indivíduo *current* do subordinado. Nesta operação genética, um ponto inicial e um ponto final (*substring*) são escolhidos aleatoriamente. Em seguida, é escolhida, no vetor do cromossomo do *pocket* líder, a operação na posição do ponto inicial, e o primeiro filho é gerado colocando a operação na primeira posição do cromossomo filho. Este procedimento é repetido para toda a *substring*. O restante do cromossomo filho é preenchido com as operações que faltam no filho e na ordem em que aparecem no cromossomo do *pocket* subordinado. Para a geração do segundo filho, é escolhida aleatoriamente uma *substring*, e é selecionada, no vetor do cromossomo do *pocket* subordinado, a operação na posição inicial da *substring*, que é colocada na primeira posição do cromossomo do filho. Este procedimento é repetido para toda a *substring*. O restante do cromossomo filho é preenchido com as operações que faltam no filho e na ordem em que aparecem no cromossomo do *pocket* líder. Ao final da operação é analisado qual é o melhor indivíduo gerado e este é armazenado no *current* do indivíduo subordinado.

O *crossover* 5 é realizado entre indivíduos *pocket* líder e *pocket* subordinado. O filho gerado por este *crossover* é armazenado no indivíduo *current* do subordinado. Nesta operação genética, um ponto inicial e um ponto final (*substring*) são escolhidos aleatoriamente. Em seguida, é selecionada, no vetor do cromossomo do *pocket* líder, a operação na posição do ponto inicial, e o primeiro filho é gerado colocando a operação na mesma posição da *substring*. Este procedimento é repetido para toda a *substring*. O restante do cromossomo filho é preenchido com as operações que faltam no filho e na ordem em que aparecem no cromossomo do *pocket* subordinado. Para a geração do segundo filho, é escolhida aleatoriamente uma *substring*. Em seguida, é selecionada, no vetor do cromossomo do *pocket* subordinado, a operação na posição inicial da *substring*, e colocada no cromossomo do filho na mesma posição em que aparece na *substring*. Este procedimento é repetido para toda a *substring*. O restante do cromossomo filho é preenchido com as operações que faltam no filho e na ordem em que aparecem no cromossomo do *pocket* líder. Ao final da operação é analisado qual é o melhor indivíduo gerado e este é armazenado no *current* do indivíduo subordinado. Este operador de *crossover* pode

ser classificado como um *crossover* uniforme.

O *crossover* 6 é realizado entre indivíduos *pocket* líder e *pocket* subordinado. O filho gerado por este *crossover* é armazenado no indivíduo *current* do subordinado. Nesta operação genética, é verificado qual máquina é menos ociosa nos dois indivíduos. Para verificar qual é a máquina menos ociosa, que é a máquina com menos tempo sem atender alguma tarefa, é somado o tempo ocioso em cada máquina. A matriz de janela de tempo deve ser analisada para somar este tempo ocioso. Para essa máquina menos ociosa, o filho é gerado colocando a seqüência de operações da máquina menos ociosa na mesma posição do indivíduo pai. Este procedimento é realizado para todas as máquinas e pode ser que alguma posição do vetor filho já tenha sido preenchida e, neste caso, a operação não é colocada no cromossomo. Após o procedimento, se alguma posição do vetor filho não estiver preenchida, é verificada qual a operação que está faltando, e esta é colocada na primeira posição vazia do cromossomo filho. O filho gerado é armazenado no *current* do indivíduo subordinado. Com este operador de *crossover* procura-se gerar um cromossomo filho com as melhores seqüências de operações de cada máquina, e essa seqüência é a melhor para a máquina menos ociosa.

O *crossover* 7 é realizado entre indivíduos *pocket* líder e *pocket* subordinado. O filho gerado por este *crossover* é armazenado no indivíduo *current* do subordinado. Nesta operação genética, é verificado para cada tarefa, nos dois indivíduos, qual é a menos ociosa. Para verificar qual é a tarefa menos ociosa, que é a tarefa com menos tempo sem ser atendida por alguma tarefa, é somado o tempo ocioso de cada tarefa entre as máquinas que a estão atendendo. A matriz de janela de tempo e a matriz A devem ser analisadas para somar este tempo ocioso. Para essa tarefa menos ociosa, o filho é gerado colocando a seqüência de operações da tarefa menos ociosa na mesma posição do indivíduo pai. Este procedimento é realizado para todas as tarefas e pode ser que alguma posição do vetor filho já tenha sido preenchida e, neste caso, a operação da tarefa não é colocada no cromossomo. Após o procedimento, se alguma posição do vetor filho não estiver preenchida, é verificada qual a operação que está faltando, e esta é colocada na primeira posição vazia do cromossomo filho. O filho gerado é armazenado no *current* do indivíduo subordinado. Com este operador de *crossover* procura-se gerar um cromossomo filho com as melhores seqüências de operações de cada tarefa, e essa seqüência é a melhor para a tarefa menos ociosa.

A mutação é realizada apenas em indivíduos *pocket* e o filho gerado é armazenado no *current* deste indivíduo. No primeiro operador de mutação implementado, duas tarefas distintas são sorteadas e, em seguida, é escolhida uma operação e verificada a posição desta operação na primeira tarefa sorteada. Em seguida é sorteada outra operação e verificada a posição desta operação na segunda tarefa sorteada. A seguir, é efetuada a troca, dentro do indivíduo, das duas operações. Este operador pode ser classificado como um operador de mutação simples.

No segundo operador de mutação implementado, duas tarefas distintas são sorteadas e todas as

operações entre as duas tarefas são trocadas, na seqüência das operações. O filho gerado pela mutação é armazenado no *current* do indivíduo *pocket*. Este operador pode ser classificado como um operador de mutação simples.

No terceiro operador de mutação implementado, são sorteadas 3 tarefas distintas e, em seguida, é sorteada uma operação e verificada a posição desta operação na primeira tarefa sorteada. Em seguida é sorteada outra operação e verificada a posição desta operação na segunda tarefa sorteada, e depois é sorteada outra operação e verificada a posição desta operação na terceira tarefa sorteada. A seguir, são gerados 3 cromossomos filhos trocando, respectivamente, as operações entre as tarefas 1 e 2, as operações entre as tarefas 1 e 3 e as operações entre as tarefas 2 e 3. Após efetuadas as trocas entre as operações, é verificado qual o melhor indivíduo gerado, que é armazenado no *current* do indivíduo *pocket* mutado.

No Apêndice B é exemplificada a execução dos operadores de *crossover* e de mutação para um problema de escalonamento *job shop* de 3 tarefas e 2 máquinas.

O operador de busca local utilizado foi a heurística de melhoria do tipo *first improving*, descrita na próxima seção, no Algoritmo 8. A busca local foi aplicada entre o agente *pocket* do melhor indivíduo (agente 1) com seus três filhos *pocket* subordinados com o intuito de promover uma melhora local no melhor indivíduo.

Após a execução, tanto dos operadores genéticos de *crossover* e de *mutação*, quanto do operador de busca local, os passos de geração da matriz A, geração da matriz de janela de tempo JT e cálculo da função de *fitness* são realizados para todos os indivíduos da população, e a árvore estruturada é ordenada novamente, mantendo o melhor indivíduo na posição do indivíduo *pocket* da raiz.

Um pseudocódigo para o algoritmo memético com população estruturada, aplicado ao problema de escalonamento do tipo *job shop*, é apresentado pelo Algoritmo 7.

3.2.2.3 Heurística de melhoria

A heurística de melhoria é considerado um método de busca local que tem como objetivo melhorar uma solução do problema.

Conforme detalhado na Seção 3.1.2.2, existem os métodos de *first improving* e *best improving* para percorrer a vizinhança na heurística de melhoria.

A idéia da busca local implementada é provocar trocas de operações para diferentes tarefas com o intuito de melhorar a solução encontrada até o momento.

A heurística de melhoria do tipo *first improving*, aplicada na resolução do problema de escalonamento *job shop*, é apresentada pelo Algoritmo 8.

Em (Vaessens et al. (1996)) são utilizados métodos de busca local determinísticos e aleatórios na resolução do problema de escalonamento do tipo *job shop*.

Algoritmo 7 Pseudocódigo do algoritmo memético com população estruturada aplicado ao problema de escalonamento *job shop* com parâmetros precisos

Entrada: n: número de tarefas, m: número de máquinas, O_{nxm}: matriz com a seqüência de operações para cada tarefa, $o_{ij} = (m_{ij}, p_{ij})$, onde $0 \leq i \leq n-1$ e $0 \leq j \leq m-1$: par ordenado de cada operação composto pela máquina que irá processá-la e o tempo de processamento da operação, $T_{pop} = 26$: tamanho da população, N_{gera}: número de gerações.

- Gerar população inicial ($V(ind)$ e $A_{mxn}(ind)$; $0 \leq ind \leq T_{pop} - 1$)
- Gerar a matriz de janela de tempo para cada indivíduo ($JT_{mxn}(ind)$; $0 \leq ind \leq T_{pop} - 1$)
- Calcular a funcão de *fitness* para cada indivíduo ($F(ind)$; $0 \leq ind \leq T_{pop} - 1$)
- Ordenar a árvore estruturada utilizando o valor da função de *fitness*

para $z = 0$ até $z = N_{gera}$ **faça**
Passos:

- Selecionar indivíduos *pocket* líder e subordinado
- Aplicar *crossover* gerando 2 novos indivíduos
- Recalcular a matriz $A_{mxn}(ind)$
- Gerar a matriz de janela de tempo para cada indivíduo ($JT_{mxn}(ind)$; $0 \leq ind \leq T_{pop} - 1$)
- Calcular a funcão de *fitness* para cada indivíduo ($F(ind)$; $0 \leq ind \leq T_{pop} - 1$)
- Ordenar a árvore estruturada utilizando o valor da função de *fitness*
- Selecionar 1 indivíduo *pocket*
- Aplicar mutação gerando 1 novo indivíduo
- Recalcular a matriz $A_{mxn}(ind)$
- Gerar a matriz de janela de tempo para cada indivíduo ($JT_{mxn}(ind)$; $0 \leq ind \leq T_{pop} - 1$)
- Calcular a funcão de *fitness* para cada indivíduo ($F(ind)$; $0 \leq ind \leq T_{pop} - 1$)
- Ordenar a árvore estruturada utilizando o valor da função de *fitness*
- Aplicar busca local
- Recalcular a matriz $A_{mxn}(ind)$
- Gerar a matriz de janela de tempo para cada indivíduo ($JT_{mxn}(ind)$; $0 \leq ind \leq T_{pop} - 1$)
- Calcular a funcão de *fitness* para cada indivíduo ($F(ind)$; $0 \leq ind \leq T_{pop} - 1$)
- Ordenar a árvore estruturada utilizando o valor da função de *fitness*

fim para
Saída: Melhor indivíduo da população

Algoritmo 8 Heurística de melhoria do tipo *first improving*

Entrada: n: número de tarefas, m: número de máquinas, O_{nxm}: matriz com a seqüência de operações para cada tarefa, $o_{ij} = (m_{ij}, p_{ij})$, onde $0 \leq i \leq n-1$ e $0 \leq j \leq m-1$: par ordenado de cada operação composto pela máquina que irá processá-la e o tempo de processamento da operação, V: vetor com representação do cromossomo do indivíduo, A_{mxn}: matriz com representação do cromossomo do indivíduo, JT_{mxn}: matriz de janela de tempo do indivíduo, F: *fitness* do indivíduo, M: *makespan* do indivíduo.

para $k = 0$ até $k = n - 1$ **faça**
  para $w = 0$ até $w = m - 1$ **faça**
    $job_{troca} = 0$
    enquanto $job_{troca} < n$ **faça**
      se $k\ != job_{troca}$ **então**
        pos1 = posição da operação w da tarefa k no cromossomo $V(ind)$
        pos2 = posição da operação w da tarefa job_{troca} no cromossomo $V(ind)$
        $V[pos1] = V[pos2]$ e $V[pos2] = V[pos1]$
        atualizar A_{mxn} com base em V
        $fitness_{ant} = F$
        atualizar janela de tempo JT_{mxn}
        atualizar *fitness* $F = M$
        se $F < fitness_{ant}$ **então**
          quebra da execução dos *loops*
        senão
          $V[pos1] = V[pos2]$ e $V[pos2] = V[pos1]$
        fim se
      fim se
    fim enquanto
  fim para
fim para

Saída: M

Capítulo 4

Escalonamento de tarefas com parâmetros incertos

Muitas vezes é necessário levar em consideração aspectos de incertezas inerentes aos processos, como tempos de processamento de tarefas, tempos de transporte, disponibilidade de matérias primas, probabilidades de falhas nas máquinas, etc. Considerando incertezas em alguns aspectos do problema, passa-se a tratá-lo de uma forma mais próxima da realidade.

Um problema de escalonamento *fuzzy* é um problema de satisfação de restrições *fuzzy*, onde cada restrição associada ao problema pode ser representada por um conjunto *fuzzy* e, sendo assim, cada critério tem uma função de pertinência associada. No problema de escalonamento *fuzzy*, onde alguns parâmetros são tratados com incertezas, até mesmo a noção de escalonamento ótimo também é considerada imprecisa, pois torna-se difícil estabelecer qual é o escalonamento ótimo das tarefas. Sendo assim, o escalonamento é avaliado através de um número *fuzzy*.

Neste capítulo são abordados dois tipos de problemas de escalonamento de tarefas com parâmetros com incertezas - o escalonamento de tarefas em máquinas paralelas idênticas e problema do *job shop*.

4.1 Escalonamento *fuzzy*

A tomada de decisão *fuzzy* originou da idéia proposta por (Bellmann and Zadeh (1970)), que introduziu os conceitos de restrições *fuzzy*, objetivo e decisão *fuzzy*.

O problema de escalonamento *fuzzy* foi formulado por (Dubois et al. (1995)) como um problema de satisfação de restrições *fuzzy*, onde as restrições são mais ou menos relaxáveis ou sujeitas à preferências. No escalonamento *fuzzy* existem dois tipos de restrições: restrições definidas no espaço de soluções admissíveis, e restrições que caracterizam a qualidade da decisão do escalonamento.

Segundo (Dubois et al. (1995)), cada restrição pode ser representada por um conjunto *fuzzy*.

Para que o escalonamento seja satisfeito, algumas restrições precisam ser satisfeitas, enquanto outras podem nem sempre ser satisfeitas e, neste caso, se for necessário, podem ser relaxadas. Sendo assim, um bom escalonamento satisfaz restrições pesadas e relaxa restrições leves com o intuito de otimizar a performance da resolução do problema. Cada restrição associada ao problema pode ser representada como um conjunto *fuzzy* e, sendo assim, cada critério tem uma função de pertinência que corresponda à uma regra intuitiva relacionada ao critério. A priorização dos critérios pode ser feita através da atribuição de pesos aos conjuntos *fuzzy* correspondentes. O peso assegura que o critério mais importante terá maior efeito na função objetivo que aquele critério menos importante.

Em casos onde os parâmetros do problema não são precisamente conhecidos, torna-se difícil estabelecer qual é o escalonamento ótimo das tarefas. Neste caso, a noção de **ótimo** também é considerada imprecisa, e passa-se a avaliar o grau de otimalidade de um escalonamento ("o quanto determinado escalonamento é ótimo") através de um número *fuzzy* no intervalo [0,1]. Este grau de otimalidade é importante e pode ser utilizado em situações onde se tem vários escalonamentos e se quer definir qual é o melhor entre eles, segundo o critério do decisor.

4.2 Escalonamento de tarefas em máquinas paralelas idênticas com parâmetros incertos

As características do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas são as mesmas do problema com parâmetros precisos, descrito na Seção 4.2.1. A única diferença é que, neste caso, foi considerado que os tempos de processamento das tarefas não são precisamente conhecidos, e são representados por números *fuzzy* triangulares.

4.2.1 Modelo matemático do problema

Seja m um conjunto de máquinas e n um conjunto de tarefas, onde cada tarefa possui um tempo de processamento $\widetilde{t_i}$. O tempo de processamento é um número *fuzzy* que foi representado conforme descrito na (Definição 2.3.1). A variável $x_{i,j}$ indica se a tarefa i é processada pela máquina j. Neste problema, cada máquina pode processar uma ou mais tarefas, a cada tempo, e cada tarefa pode ser processada por apenas uma máquina. O objetivo é minimizar o *makespan* M. O modelo matemático para este problema é caracterizado pela Formulação 4.1.

$$
\begin{aligned}
\text{Min } \widetilde{M} = \quad & \underbrace{MAX}_{1\leq j\leq m} \sum_{i=1}^{n} x_{i,j}\widetilde{t_i} \\
\text{s. a : } \quad & \sum_{j=1}^{m} x_{i,j} = 1; \quad 1 \leq i \leq n \\
& x_{i,j} = \begin{cases} 1, \text{ se a tarefa } i \text{ é} \\ \text{ atribuída à máquina } j \\ 0, \text{ caso contrário.} \end{cases}
\end{aligned} \tag{4.1}
$$

4.2.2 Métodos heurísticos aplicados ao problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas

O problema de escalonamento de tarefas em máquinas paralelas com parâmetros com incertezas é um problema *NP-completo* e então os métodos exatos nem sempre apresentam uma solução ótima em um tempo computacionalmente razoável. Em virtude disso, os métodos heurísticos são muito utilizados na resolução desse tipo de problema. Os métodos heurísticos têm como objetivo fornecer uma solução que seja a melhor possível e em tempo polinomial.

Por se tratar de um problema de escalonamento *fuzzy*, onde torna-se difícil classificar um determinado escalonamento como ótimo, pois o problema agora possui restrições *fuzzy*, foi aplicado, neste trabalho, o conceito de otimalidade possível para medir a possibilidade de um determinado escalonamento ser ótimo. Para evoluir bons escalonamentos, foi aplicado o algoritmo memético com população estruturada. Sendo assim, o algoritmo memético foi aplicado para encontrar soluções para o problema, e o conceito de possibilidade, descrito nas (Seções 2.3.3 e 2.3.4), para avaliar as melhores soluções (Bonfim and Yamakami (2004a)). O método heurístico aplicado ao problema foi o algoritmo memético com população estruturada, descrito na Seção 3.2.2.1.

4.2.2.1 Aplicação do algoritmo memético com população estruturada ao problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas

Um pseudocódigo para o algoritmo memético com população estruturada, aplicado ao problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas, é apresentado pelo Algoritmo 9.

Um conceito importante e crítico na execução de um algoritmo genético ou memético é a escolha da representação de possíveis soluções para o problema dentro de um cromossomo ou indivíduo.

Para o problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas, o cromossomo foi codificado por um vetor V, de n colunas, onde n é o número de

Algoritmo 9 Pseudocódigo do algoritmo memético com população estruturada aplicado ao problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas

Entrada: n: número de tarefas, m: número de máquinas, $\widetilde{t_i}$: tempo de processamento da tarefa $0 \leq i \leq n-1$, $T_{pop} = 26$: tamanho da população, N_{gera}: número de gerações.

- Gerar população inicial ($V(ind); 0 \leq ind \leq T_{pop} - 1$)
- Calcular vetor de *makespan* de cada indivíduo ($\widetilde{M}(ind) = Max\{\widetilde{T_j}\}; 0 \leq j \leq m-1$), avaliando (Poss($\widetilde{T_j} \geq \widetilde{T}_{j_{max}}); \forall j$), considerando $\widetilde{T}_{j_{max}}$ o *makespan* até o momento
- Ordenar a árvore estruturada, avaliando (Poss$(\widetilde{F}(k) < \widetilde{F}(k_{min})); \forall k, 0 \leq k \leq T_{pop} - 1$), onde $\widetilde{F}(k_{min})$ é o *fitness* do melhor indivíduo até o momento

para $z = 0$ até $z = N_{gera}$ **faça**
Passos:

- Selecionar indivíduos *pocket* líder e subordinado
- Aplicar *crossover* gerando 2 novos indivíduos
- Selecionar 1 indivíduo *pocket*
- Aplicar mutação gerando 1 novo indivíduo
- Calcular vetor de *makespan* de cada indivíduo ($\widetilde{M}(ind) = Max\{\widetilde{T_j}\}; 0 \leq j \leq m-1$), avaliando ($eqadnt$Poss($\widetilde{T_j} \geq \widetilde{T}_{j_{max}}); \forall j$), considerando $\widetilde{T}_{j_{max}}$ o *makespan* até o momento
- Ordenar a árvore estruturada, avaliando (Poss$(\widetilde{F}(k) < \widetilde{F}(k_{min})); \forall k, 0 \leq k \leq T_{pop} - 1$), onde $\widetilde{F}(k_{min})$ é o *fitness* do melhor indivíduo até o momento
- Aplicar busca local
- Calcular vetor de *makespan* de cada indivíduo ($\widetilde{M}(ind) = Max\{\widetilde{T_j}\}; 0 \leq j \leq m-1$), avaliando (Poss($\widetilde{T_j} \geq \widetilde{T}_{j_{max}}); \forall j$), considerando $\widetilde{T}_{j_{max}}$ o *makespan* até o momento
- Ordenar a árvore estruturada, avaliando (Poss$(\widetilde{F}(k) < \widetilde{F}(k_{min})); \forall k, 0 \leq k \leq T_{pop} - 1$), onde $\widetilde{F}(k_{min})$ é o *fitness* do melhor indivíduo até o momento

fim para
Saída: Melhor indivíduo da população

tarefas. Cada campo do vetor v_k, onde $0 \leq k \leq n-1$, é preenchido por um número $j; 0 \leq j \leq m-1$, que indica o número da máquina que estará executando a tarefa. Cada posição do vetor indica qual máquina irá atender determinada tarefa.

O primeiro passo na execução do algoritmo memético com população estruturada é a geração da população inicial. Foram gerados 26 indivíduos, 13 indivíduos *pocket* e 13 indivíduos *current*. Cada indivíduo foi representado por um vetor conforme detalhado no parágrafo acima. Na geração destes indivíduos, um deles foi gerado por uma heurística construtiva chamada *LPT*, conforme descrito pelo Algoritmo 10, e os demais foram gerados aleatoriamente.

A heurística *LPT* foi ajustada para atender as características de incerteza aplicadas ao problema. Esta heurística ordena decrescentemente os tempos de processamento das tarefas e começa a distribuí-las pelas máquinas. Na ordenação das tarefas, como seus tempos de processamento são números *fuzzy*, foi utilizado o conceito de possibilidade para analisar qual o maior tempo de processamento e ordená-las. O método de ordenação utilizado foi o método da bolha, que verifica pares ordenados de tempos das tarefas (Poss$(\widetilde{t_{j+1}} \geq \widetilde{t_j}); \forall j; 0 \leq j \leq n-1$) e, ao encontrar uma tarefa $j+1$ com tempo de processamento maior que a tarefa j, estas tarefas são trocadas de posição no vetor de tempos de processamento.

Para a alocação das tarefas às máquinas, foi analisada qual é a máquina com menor tempo total de processamento e, como este parâmetro é um número *fuzzy*, foi utilizado o conceito de possibilidade para encontrar essa máquina (Poss$(\widetilde{T}(m_j) < \widetilde{T}(m_{j_{min}}); \forall j, 0 \leq j \leq m-1$). Cada tarefa é atribuída à máquina menos ocupada no momento da alocação.

Algoritmo 10 Heurística construtiva LPT com parâmetros com incertezas

Entrada: n: número de tarefas, m: número de máquinas, $\widetilde{t_i}$: tempo de processamento da tarefa $i; 0 \leq i \leq n-1$.
para $j = 0$ até $j = m-1$ **faça**
  $\widetilde{T}(m_j) = 0$, tempo de processamento total na máquina j
fim para

- ordenar as tarefas em ordem decrescente de tempo de processamento - $\widetilde{t_k}, \widetilde{t_l}, ..., \widetilde{t_p}$, onde $t_k \geq t_l \geq t_p$

para $i = 0$ até $i = n-1$ **faça**
  $Min\{\widetilde{T}(m_j)\}; 0 \leq j \leq m-1$
  $l = \arg\{j\} Min\{\widetilde{T}(m_j)\}; 0 \leq j \leq m-1$
  $v_i = l$
  $\widetilde{T}(m_j) = \widetilde{T}(m_j) \oplus \widetilde{t_i}$
fim para
Saída: $V(0)$: primeiro indivíduo da população

No vetor abaixo podemos verificar um exemplo de um cromossomo que possui 10 *tarefas* e 5 *máquinas*.

$$\mathbf{V} = \begin{pmatrix} 3 & 1 & 2 & 2 & 4 & 5 & 1 & 3 & 4 & 1 \end{pmatrix}$$

O passo seguinte na execução do algoritmo é a avaliação da função de *fitness*. O critério utilizado para a avaliação das soluções foi o *makespan*.

Para o cálculo da função de *fitness*, cada indivíduo tem o tempo total de execução das tarefas de suas máquinas calculado. Primeiramente foi calculado o tempo total de execução em cada máquina. Como os tempos de processamento das tarefas são parâmetros *fuzzy*, o tempo total de processamento de cada máquina ($\widetilde{T_j}$) foi obtido segundo a Equação 4.2. A operação de adição de números *fuzzy* é descrita pela Equação 2.7.

$$\begin{gathered} \widetilde{T_j} = x_{0,j}\widetilde{t_0} \oplus x_{1,j}\widetilde{t_1} \oplus \ldots \oplus x_{n-1,j}\widetilde{t}_{n-1}; \\ \forall i; 0 \leq i \leq n-1 \; ; \; \forall j; 0 \leq j \leq m-1 \; ; x_{i,j} \in \{0,1\}. \end{gathered} \tag{4.2}$$

De acordo com a Equação 4.2, é realizado em todas as máquinas o cálculo do tempo total de execução das tarefas a ela alocadas. Com base nesse cálculo, o *makespan* é definido como $Max\{\widetilde{T_j}\}; 0 \leq j \leq m-1$. Como $\widetilde{T_j}$ é um número *fuzzy*, foi utilizada a possibilidade para avaliar qual é o *makespan* de cada indivíduo.

Para identificar o *makespan* de cada indivíduo, foi analisado Poss$(\widetilde{T_j} \geq \widetilde{T}_{j_{max}}); \forall j$, considerando $\widetilde{T}_{j_{max}}$ o *makespan* até o momento.

Foi considerado como valor de *fitness* o valor do *makespan*. Sendo assim, a cada geração, a solução considerada melhor é aquela que apresenta o menor valor de *fitness*. Para avaliar qual o menor valor de *fitness*, foram aplicados os conceitos de possibilidade descritos pela Equação 2.17.

Para identificar o melhor indivíduo, foi analisado Poss$(\widetilde{F}(k) < \widetilde{F}(k_{min})); \forall k, 0 \leq k \leq T_{pop} - 1$, onde $\widetilde{F}$ é o vetor de *fitness* da população de indivíduos, $\widetilde{F}(k_{min})$ é o *fitness* do melhor indivíduo até o momento e T_{pop} é o tamanho da população.

O próximo passo na execução do algoritmo é a ordenação da árvore, seguindo os critérios de indivíduos *pocket* e *current*, de forma que o melhor indivíduo (com menor valor de *fitness*) fique na posição do agente 1.

Os passos seguintes, onde são executados os operadores genéticos, são realizados enquanto o critério de parada não for alcançado. O critério de parada utilizado neste trabalho foi o número de gerações. Os operadores genéticos executados neste algoritmo foram o *crossover* simples, *crossover* uniforme, mutação simples e mutação inversiva.

O *crossover* simples é realizado entre indivíduos *pocket* líder e *pocket* subordinado. O filho gerado por este *crossover* é armazenado no indivíduo *current* do subordinado. Nesta operação genética, é gerado aleatoriamente 1 ponto de cruzamento i, $0 \leq i \leq n-1$ e, em seguida é efetuada a permuta das posições (genes) i até $n-1$ entre os indivíduos *pocket* líder e subordinado.

O *crossover* uniforme é realizado entre indivíduos *pocket* líder e *pocket* subordinado. O filho gerado por este *crossover* é armazenado no indivíduo *current* do subordinado. Nesta operação genética, é gerada aleatoriamente a taxa de pontos de cruzamento (p) ou número de genes que serão permutados entre os cromossomos do *pocket* líder com o do *pocket* subordinado. Em seguida, é escolhida aleatoriamente a posição dos p genes (número da tarefa i, $0 \leq i \leq n-1$), e a permuta destas posições é efetuada entre os vetores dos indivíduos *pocket* líder e *pocket* subordinado.

O *crossover* uniforme é aplicado em uma taxa de cruzamento que varia entre 1 e $n/2$, onde n é número de tarefas.

A mutação é realizada apenas em indivíduos *pocket* e o filho gerado é armazenado no *current* deste indivíduo. Na mutação simples são escolhidas duas posições do vetor que representa o indivíduo, e a troca entre essas posições é efetuada.

Na mutação inversiva é escolhida uma posição do vetor que representa o indíviduo e, em seguida, é efetuada a troca entre posições adjacentes do vetor.

O operador de busca local utilizado foi a heurística de melhoria do tipo *first improving*, descrita na Seção 3.1.2.2 e com alguns ajustes para atender as características de incerteza aplicadas ao problema. A busca local foi aplicada entre o agente *pocket* do melhor indivíduo (agente 1) com seus três filhos *pocket* subordinados com o intuito de promover uma melhora local no melhor indivíduo.

A heurística de melhoria do tipo *first improving*, com parâmetros com incertezas é apresentada no Algoritmo 11. Esta heurística tem como objetivo promover a troca de tarefas entre duas máquinas se esta troca produzir um *makespan* menor que o *makespan* atual. Na estratégia *first improving*, o algoritmo termina sua execução ao encontrar a primeira melhora. Para comparar dois valores de *makespan* foi utilizado o conceito de possibilidade.

Após a execução, tanto dos operadores genéticos de *crossover* e de *mutação*, quanto do operador de busca local, os passos de cálculo do vetor de *makespan* e cálculo da função de *fitness* são realizados para todos os indivíduos da população, e a árvore estruturada é ordenada novamente, mantendo o melhor indivíduo na posição do indivíduo *pocket* da raiz.

Como saída do algoritmo é apresentado o melhor indivíduo da população, que será o indivíduo *pocket* do agente 1 (líder) da população. Este indivíduo tem seu valor de *fitness* representado por um número *fuzzy* triangular. Para apresentarmos seu valor *crisp*, foi aplicada a *defuzzificação*, que é o mapeamento de informações *fuzzy* em valores *crisp*. O método de *defuzzificação* aplicado foi o Método da Centróide (Centro de Gravidade) (Sugeno (1985)), descrito na Seção 2.3.5.

Algoritmo 11 Heurística de melhoria do tipo *first improving* com parâmetros com incertezas

Entrada: n: número de tarefas, m: número de máquinas, p: máquina mais ocupada, $num_{tarefas}(p)$: número de tarefas alocadas a máquina p, $\widetilde{t_i}$: tempo de processamento da tarefa $0 \leq i \leq n-1$, $\widetilde{M}$: *makespan* atual.
para $k = 0$ até $k = num_{tarefas}(p)$ **faça**
  para $w = 0$ até $w = m - 1$ **faça**
    se $w! = p$ **então**
      para $z = 0$ até $z = num_{tarefas}(w)$ **faça**
        $\widetilde{T}(m_w) = \widetilde{T}(m_w) \ominus \widetilde{t_z} \oplus \widetilde{t_k}$
        $\widetilde{T}(m_p) = \widetilde{T}(m_p) \ominus \widetilde{t_k} \oplus \widetilde{t_z}$
        se Poss$(Max\{\widetilde{T}(m_j)\} < \widetilde{M})$ **então**
          $\widetilde{M} = Max\{\widetilde{T}(m_j)\}; 0 \leq j \leq m-1$
          $p = j$
          quebra da execução dos *loops*
        senão
          $\widetilde{T}(m_w) = \widetilde{T}(m_w) \oplus \widetilde{t_z} \ominus \widetilde{t_k}$
          $\widetilde{T}(m_p) = \widetilde{T}(m_p) \oplus \widetilde{t_k} \ominus \widetilde{t_z}$
        fim se
      fim para
    fim se
  fim para
fim para
Saída: $\widetilde{T}$

4.3 Escalonamento *job shop* com parâmetros com incertezas

O problema de escalonamento *job shop* aqui considerado é o mesmo descrito na Seção 3.2.

Neste trabalho, foi considerado que os tempos de processamento das operações e os tempos possíveis para o início do processamento das operações não são precisamente conhecidos, e são representados por números triangulares *fuzzy*.

4.3.1 Modelo matemático do problema

Seja m um conjunto de máquinas e n um conjunto de tarefas. Seja $O = 0, ..., n-1$ o conjunto de operações, onde 0 é considerada a operação inicial de todas as tarefas e $n-1$ é considerada a operação final de todas as tarefas. Neste problema todas as tarefas possuem a mesma quantidade de operações, que é igual ao número de máquinas. Seja A o conjunto de pares ordenados de operações restringidos por relações de precedência para cada tarefa. Para cada máquina k, o conjunto E_k descreve o conjunto de todos os pares de operações a serem executadas na máquina k, ou seja, operações que não poderão ser sobrepostas. Para cada operação i, seu tempo de processamento $\widetilde{p_i}$ é fixado, e o tempo possível

para início do processamento da operação i na máquina l é $\widetilde{ti}_{il}$, que é uma variável que é determinada durante a otimização do problema.

O objetivo é minimizar o tempo total gasto entre o início da primeira operação e o término da última operação, que representa o *makespan* $\widetilde{M}$. A variável $\widetilde{M}$ representa o tempo final $\widetilde{tf}_{il}$ de execução da tarefa mais ocupada. Cada parâmetro de tempo do problema é um número *fuzzy*, e o valor do *makespan* também é representado por um número *fuzzy*. Os números *fuzzy* foram representados conforme descrito em 2.3.1. O modelo matemático para este problema é caracterizado pela Formulação 4.3.

$$
\begin{aligned}
\text{Min } \widetilde{M} = \quad & \underbrace{MAX}_{1\leq l\leq m}\{\widetilde{tf}_{il}\}\,, \forall i; 0 \leq i \leq n-1. \\
\text{s. a :} \quad & \widetilde{ti}_{jk} - \widetilde{ti}_{ik} \geq p_i \;\; ; \forall (i,j) \in A \;\; , \forall i; 0 \leq i \leq n-1 \;\; , \forall j; 0 \leq j \leq n-1. \\
& \widetilde{ti}_{jk} - \widetilde{ti}_{ik} \geq p_i \;\; \text{ou} \;\; \widetilde{ti}_{ik} - \widetilde{ti}_{jk} \geq p_j \;\; ; \forall \{i,j\} \in E_k \;\; , \forall k; 0 \leq k \leq m-1. \\
& \widetilde{ti}_{ik} \geq 0 \;\; ; \forall i \in O \;\; , \forall i; 0 \leq i \leq n-1\,, \forall k; 0 \leq k \leq m-1.
\end{aligned}
\tag{4.3}
$$

A descrição das restrições está relatada na Seção 3.2.1.

A Tabela 4.1 apresenta uma instância gerada para o problema 6 x 6 ($m = 6$ e $n = 6$) com parâmetros com incertezas.

Tab. 4.1: Instância do problema do *job shop* 6x6 com parâmetros com incertezas.

	(m,$\widetilde{p}$)	*(m,$\widetilde{p}$)*	*(m,$\widetilde{p}$)*	*(m,$\widetilde{p}$)*	*(m,$\widetilde{p}$)*	*(m,$\widetilde{p}$)*
Tarefa 0:	(2,[0.01,1,1.99])	(0,[2.77,3,3.23])	(1,[5.50,6,6.50])	(3,[6.93,7,7.07])	(5,[2.89,3,3.11])	(4,[5.07,6,6.93])
Tarefa 1:	(1,[7.90,8,8.22])	(2,[4.34,5,5.20])	(4,[9.82,10,10.69])	(5,[9.52,10,10.03])	(0,[9.99,10,10.06])	(3,[3.22,4,4.04])
Tarefa 2:	(2,[4.01,5,5.99])	(3,[3.83,4,4.17])	(5,[7.68,8,8.32])	(0,[8.64,9,9.36])	(1,[0.29,1,1.71])	(4,[6.08,7,7.92])
Tarefa 3:	(1,[4.03,5,5.13])	(0,[4.19,5,5.52])	(2,[4.50,5,5.55])	(3,[2,3,3.49])	(4,[7.22,8,8.5])	(5,[8.44,9,9.88])
Tarefa 4:	(2,[8.57,9,9.43])	(1,[2.35,3,3.65])	(4,[4.9,5,5.1])	(5,[3.92,4,4.08])	(0,[2.15,3,3.85])	(3,[0.73,1,1.27])
Tarefa 5:	(1,[2.22,3,3.33])	(3,[2.7,3,3.79])	(5,[8.61,9,9.08])	(0,[9.17,10,10.38])	(4,[3.75,4,4.14])	(2,[0.26,1,1.96])

Na tabela acima, m representa a máquina que a tarefa será processada e $\widetilde{p}$ indica o tempo de processamento da tarefa na máquina m. O tempo de processamento é considerado um parâmetro incerto e, por isso, foi representado por um número *fuzzy* triangular. Para a geração desses números *fuzzy*, utilizou-se o valor p do *benchmark* original, apresentado na Tabela 3.1, como valor modal p^0 e, os espalhamentos a esquerda ($p_i{}^0 - \underline{p}$) e a direita ($\overline{p} - p^0$), foram gerados segundo uma distribuição uniforme no intervalo [0,1].

Na instância exemplificada, a tarefa 0 precisa ser processada inicialmente pela máquina 2, por um tempo *fuzzy* de [0.01,1,1.99] e, em seguida, pela máquina 0, por um tempo também *fuzzy* de [2.77,3,3.23]. Cada tarefa apresenta uma seqüência de máquinas que irão processá-la. Um escalo-

namento factível é o escalonamento de uma seqüência de tarefas em cada máquina tal que a ordem das operações de cada tarefa seja preservada. Cada máquina não poderá processar mais de uma tarefa ao mesmo tempo e diferentes operações de uma mesma tarefa não poderão ser processadas simultaneamente em máquinas diferentes. O mínimo *makespan* encontrado para esse problema foi o valor 54.76, com número *fuzzy* triangular [50.64, 55.00, 57.21], e o Diagrama de Grantt desse problema para este valor mínimo encontrado, é apresentado na Figura 4.1.

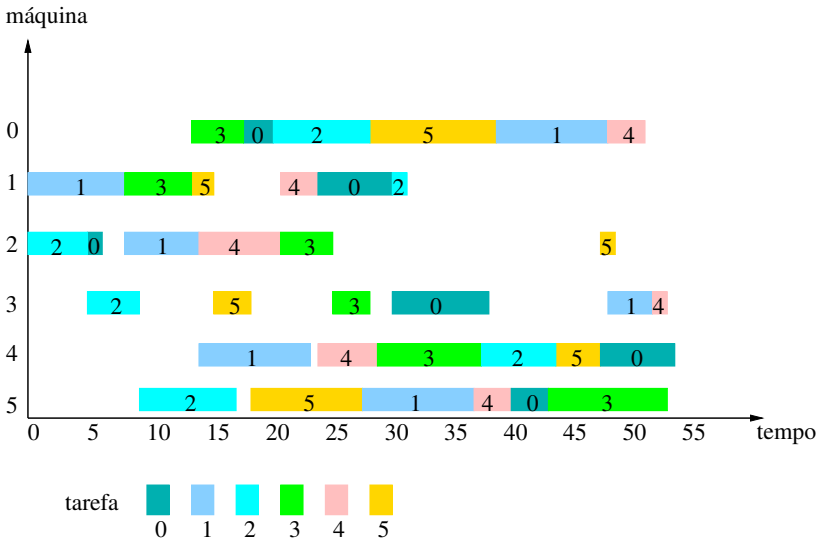

Fig. 4.1: Diagrama de Grantt do escalonamento mínimo encontrado para o problema do *job shop* 6x6 com parâmetros com incertezas.

4.3.2 Métodos heurísticos aplicados ao problema de escalonamento do *job shop* com parâmetros com incertezas

Por se tratar de um problema de escalonamento *fuzzy*, onde torna-se difícil classificar um determinado escalonamento como ótimo, pois o problema agora possui restrições *fuzzy*, foi aplicado, neste trabalho, o conceito de otimalidade possível para medir a possibilidade de um determinado escalonamento ser ótimo. Para evoluir bons escalonamentos, foi aplicado o algoritmo memético com população estruturada. Sendo assim, foi aplicado o algoritmo memético para encontrar soluções para o problema, e o conceito de possibilidade, descrito nas (Seções 2.3.3 e 2.3.4), para avaliar as melhores soluções. O método heurístico aplicado ao problema foi o algoritmo memético com população estruturada, descrito na Seção 3.2.2.1.

4.3.2.1 Aplicação do algoritmo memético com população estruturada ao problema de escalonamento *job shop* com parâmetros com incertezas

Um pseudocódigo para o algoritmo memético com população estruturada, aplicado ao problema de escalonamento *job shop* com parâmetros com incertezas, é apresentado pelo Algoritmo 12.

Um conceito importante e crítico na execução de um algoritmo genético ou memético é a escolha da representação de possíveis soluções para o problema dentro de um cromossomo ou indivíduo.

Para o problema do *job shop* com parâmetros com incertezas, o cromossomo foi codificado através de um vetor V e através de uma matriz A_{mxn}. Estas duas formas estão descritas na Seção 3.2.2.2, onde o problema tratado possui parâmetros precisos.

No vetor abaixo podemos verificar um exemplo da codificação do cromossomo para o problema 6 x 6 ($m = 6$ e $n = 6$) da Tabela 4.1. Esse é o cromossomo que resultou o escalonamento ótimo apresentado na Figura 4.1.

$$\mathbf{V} = \begin{matrix}(1 & 2 & 3 & 2 & 0 & 5 & 5 & 1 & 2 & 4 & 3 & 1 & 0 & 2 & 4 & 5 & 1 & 5 \\ 0 & 4 & 3 & 2 & 3 & 3 & 2 & 4 & 0 & 1 & 1 & 5 & 5 & 0 & 3 & 0 & 4 & 4)\end{matrix}$$

A matriz abaixo apresenta um exemplo da outra codificação do cromossomo para o problema 6 x 6 ($m = 6$ e $n = 6$) da Tabela 4.1. Esse é o cromossomo que resultou o escalonamento ótimo apresentado na Figura 4.1.

$$\mathbf{A} = \begin{bmatrix} 3 & 0 & 2 & 5 & 1 & 4 \\ 1 & 3 & 5 & 4 & 0 & 2 \\ 2 & 0 & 1 & 4 & 3 & 5 \\ 2 & 5 & 3 & 0 & 1 & 4 \\ 1 & 4 & 3 & 2 & 5 & 0 \\ 2 & 5 & 1 & 4 & 0 & 3 \end{bmatrix}$$

De acordo com a matriz acima, a linha 1 do cromossomo, que representa a máquina 0, indica a seqüência das tarefas que serão atendidas pela máquina. Sendo assim, a máquina 0 atenderá inicialmente a tarefa 3, depois a tarefa 0, depois a tarefa 2, e assim por diante.

O primeiro passo na execução do algoritmo memético com população estruturada é a geração da população inicial. Foram gerados 26 indivíduos, 13 indivíduos *pocket* e 13 indivíduos *current*. Os 26 indíviduos foram codificados das duas maneiras descritas na Seção 3.2.2.2.

O passo seguinte na execução do algoritmo é a avaliação da função de *fitness*. O critério utilizado para a avaliação das soluções foi o *makespan*. Para calcular o *makespan* no problema do *job shop* é preciso montar a janela de tempo para cada cromossomo, conforme a Figura 4.1.

Algoritmo 12 Pseudocódigo do algoritmo memético com população estruturada aplicado ao problema de escalonamento *job shop* com parâmetros com incertezas

Entrada: n: número de tarefas, m: número de máquinas, O_{nxm}: matriz com a seqüência de operações para cada tarefa, $o_{ij} = (m_{ij}, \widetilde{p}_{ij})$, onde $0 \leq i \leq n-1$ e $0 \leq j \leq m-1$: par ordenado de cada operação composto pela máquina que irá processá-la e o tempo de processamento da operação, $T_{pop} = 26$: tamanho da população, N_{gera}: número de gerações.

- Gerar população inicial ($V(ind)$ e $A_{mxn}(ind)$; $0 \leq ind \leq T_{pop} - 1$)
- Gerar a matriz de janela de tempo para cada indivíduo
- Calcular a funcão de *fitness* para cada indivíduo
- Ordenar a árvore estruturada utilizando o valor da função de *fitness*, avaliando Poss$(\widetilde{F}(k) < \widetilde{F}(k_{min})); \forall k, 0 \leq k \leq T_{pop} - 1$, onde $(\widetilde{F}(k_{min}))$ é o *fitness* do melhor indivíduo até o momento

para $z = 0$ até $z = N_{gera}$ **faça**
Passos:

- Selecionar indivíduos *pocket* líder e *pocket* subordinado
- Aplicar *crossover* gerando 2 novos indivíduos
- Selecionar 1 indivíduo *pocket*
- Aplicar mutação gerando 1 novo indivíduo
- Recalcular a matriz $A_{mxn}(ind)$
- Gerar a matriz de janela de tempo para cada indivíduo
- Calcular a funcão de *fitness* para cada indivíduo
- Ordenar a árvore estruturada utilizando o valor da função de *fitness*
- Aplicar busca local
- Recalcular a matriz $A_{mxn}(ind)$
- Gerar a matriz de janela de tempo para cada indivíduo
- Calcular a funcão de *fitness* para cada indivíduo
- Ordenar a árvore estruturada utilizando o valor da função de *fitness*

fim para
Saída: Melhor indivíduo da população

A janela de tempo é armazenada em uma matriz $\widetilde{JT}_{mxn}$, de m linhas e n colunas, onde m é o número de máquinas e n é o número de tarefas. Cada elemento da matriz $\widetilde{jt}_{ji} = (\widetilde{ti}_{ji}/\widetilde{tf}_{ji})$, onde $0 \leq j \leq m-1$ e $0 \leq i \leq n-1$, é preenchido por um par ordenado indicando o tempo de início do processamento da tarefa ($\widetilde{ti}$) e o tempo final do processamento da tarefa ($\widetilde{tf}$). Para o cálculo dessa janela de tempo é necessário obedecer as restrições do problema analisando a matriz de operações O_{nxm} e a matriz que codifica o cromossomo A_{mxn}. A janela de tempo para o problema $6x6$, representada na Figura 4.1, apresenta a matriz $\widetilde{JT}_{mxn}$ destacada abaixo.

$$\widetilde{\mathbf{JT}} = \begin{bmatrix} 12.76/17.66 & 17.66/20.66 & 20.66/29.66 & 29.66/39.51 & 39.51/49.53 & 49.53/52.53 \\ 0/8.04 & 8.04/12.76 & 12.76/15.61 & 21.89/24.89 & 24.89/30.89 & 30.89/31.89 \\ 0/5.00 & 5.00/6.00 & 8.04/12.89 & 12.89/21.89 & 21.89/26.90 & 48.76/49.83 \\ 5.00/9.00 & 15.61/18.77 & 26.90/29.73 & 30.89/37.89 & 49.53/53.29 & 53.29/54.29 \\ 12.89/23.06 & 24.89/29.89 & 29.89/37.80 & 37.80/44.80 & 44.80/48.76 & 48.76/54.76 \\ 9.00/17.00 & 18.77/27.67 & 27.67/37.52 & 37.52/41.52 & 41.52/44.52 & 44.52/53.63 \end{bmatrix}$$

Para o cálculo da função de *fitness* ($\widetilde{F}(ind)$) de um indivíduo, é utilizado o tempo final máximo na janela de tempo do indivíduo ($MAX\{\widetilde{tf}_j\}; 0 \leq j \leq m-1$), que é o valor do *makespan*. Esse procedimento é realizado para cada indivíduo da população. Como os tempos de processamento e de atendimento das tarefas são parâmetros *fuzzy*, o tempo final da execução de cada operação também é um número *fuzzy*. Como $\widetilde{tf}_j$ é um número *fuzzy*, foi utilizado o conceito de possibilidade para avaliar qual é o *makespan* de cada indivíduo.

Para identificar o *makespan* de cada indivíduo, foi analisado Poss$(\widetilde{tf}_j \geq \widetilde{tf}_{j_{max}}); \forall j$, considerando $\widetilde{tf}_{j_{max}}$ o *makespan* até o momento.

Foi considerado como valor de *fitness* o valor do *makespan*. Sendo assim, a cada geração, a solução considerada melhor é aquela que apresenta o menor valor de *fitness*. Todos os indivíduos são avaliados e, aquele que possuir o menor valor de *fitness* é considerado o melhor indivíduo da população. Para avaliar qual o menor valor de *fitness*, foram aplicados os conceitos de possibilidade descritos pela Equação 2.17.

Para identificar o melhor indivíduo, foi analisado Poss$(\widetilde{F}(k) < \widetilde{F}(k_{min})); \forall k, 0 \leq k \leq T_{pop}-1$, onde $\widetilde{F}$ é o vetor de *fitness* da população de indivíduos, $\widetilde{F}(k_{min})$ é o *fitness* do melhor indivíduo até o momento e T_{pop} é o tamanho da população.

Os passos seguintes, onde são executados os operadores genéticos, são realizados enquanto o critério de parada não for alcançado. O critério de parada utilizado neste trabalho foi o número de gerações. Os operadores genéticos executados neste algoritmo foram o *crossover* e a mutação. Foram implementados e testados, para este problema, 7 tipos de *crossover* e 3 tipos de *mutação*. Estes operadores estão descritos na Seção 3.2.2.2, onde o problema tratado possui parâmetros precisos.

Como saída do algoritmo é apresentado o melhor indivíduo da população, que será o indivíduo *pocket* do agente 1 (líder) da população. Este indivíduo tem seu valor de *fitness* representado por um número *fuzzy* triangular. Para apresentarmos seu valor *crisp*, foi aplicada a *defuzzificação*, que é o mapeamento de informações *fuzzy* em valores *crisp*. O método de *defuzzificação* aplicado foi o Método da Centróide (Centro de Gravidade) (Sugeno (1985)), descrito na Seção 2.3.5.

Capítulo 5

Escalonamento neuro-memético de tarefas com parâmetros precisos

Neste capítulo são abordados dois tipos de problemas de escalonamento de tarefas com parâmetros precisos - o escalonamento de tarefas em máquinas paralelas idênticas e o problema de escalonamento *job shop*. Na resolução destes problemas de escalonamento foi aplicado o algoritmo memético com população estruturada em coevolução com redes neurais.

Nesta coevolução, o algoritmo memético é um método heurístico que tem como objetivo evoluir boas formas de escalonamento na busca de soluções para os problemas, e a rede neural tem como objetivo a obtenção do valor da função de *fitness* para cada indivíduo gerado pelo algoritmo memético. Foram aplicadas duas redes neurais nesta coevolução - a rede neural do tipo *backpropagation* e a rede neural com aprendizado por reforço.

5.1 Arquitetura da rede neural

Tanto na resolução do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros precisos, quanto na resolução do problema de escalonamento *job shop* com parâmetros precisos, foram utilizadas redes neurais coevoluindo com o algoritmo memético com população estruturada. O algoritmo memético com população estruturada foi descrito na Seção 3.2.2.1. O algoritmo memético foi utilizado para evoluir boas formas de escalonamento, e a rede neural foi utilizada para calcular a função de *fitness* de cada indivíduo que compõe a população no algoritmo memético. A rede neural age na coevolução do algoritmo memético. Este esquema de coevolução é apresentado na Figura. 5.1.

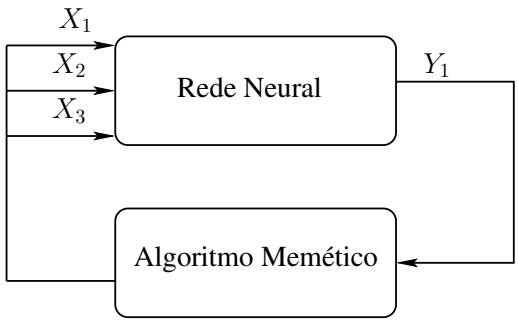

Fig. 5.1: Esquema de coevolução do algoritmo memético.

5.1.1 Redes neurais

A arquitetura da rede neural do escalonador neuro-memético, conforme apresentada pela Figura. 5.2, possui uma camada de entrada com três neurônios, sendo que a primeira entrada (X_1) representa o tempo total de processamento das tarefas. No problema de escalonamento *job shop*, esta entrada representa a soma dos tempos finais de cada máquina. A segunda entrada (X_2) representa o tempo médio (tempo total de processamento / número de máquinas). A terceira entrada (X_3), no problema de escalonamento de tarefas em máquinas paralelas idênticas, representa o tempo total de processamento na máquina mais vazia, ou seja, a máquina que é a primeira a terminar a execução das tarefas alocadas a ela. A terceira entrada, no problema de escalonamento *job shop*, representa o menor tempo final da última tarefa entre todas as últimas tarefas executadas pelas máquinas.

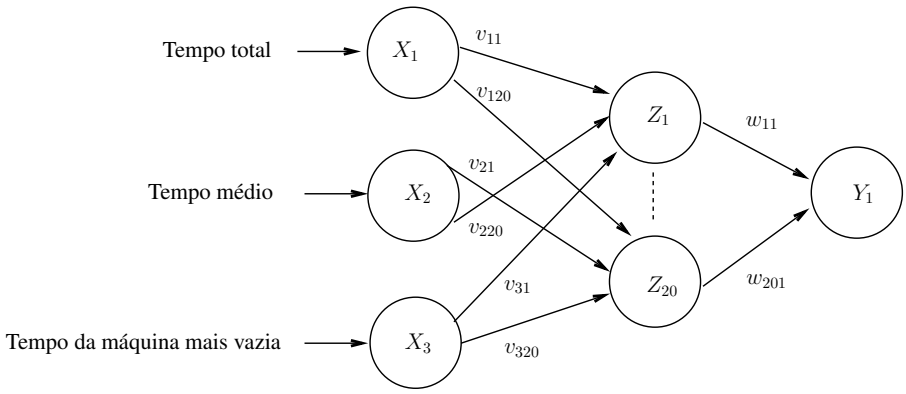

Fig. 5.2: Arquitetura da rede neural.

A camada intermediária da rede neural possui 20 neurônios. Foi utilizada apenas uma camada intermediária pois, segundo (Bertsekas and Tsitsiklis (1996)), para a maioria das aplicações uma camada é suficiente para oferecer um bom mapeamento.

A camada de saída (Y_1) tem apenas um neurônio que representa o valor da função de *fitness* do indivíduo no algoritmo memético.

Na fase de aprendizagem, a rede neural utiliza as entradas, que são valores obtidos através do escalonamento realizado pelo algoritmo memético, para aprender o valor de *fitness* associado à essas entradas. Foram aplicadas duas redes neurais com essa finalidade. A primeira foi a *backpropagation* e a segunda foi com aprendizado por reforço.

5.1.1.1 Rede neural *backpropagation*

A rede neural do tipo *backpropagation*, segundo (Bertsekas and Tsitsiklis (1996)), é um método de treinamento baseado no gradiente e tem como objetivo minimizar o erro quadrático médio da saída computada pela rede.

Aplicações que utilizam este tipo de rede geralmente tratam de problemas que envolvem o mapeamento de um conjunto de entradas a um conjunto específico de saídas desejadas. Sendo assim, este método é baseado no treinamento supervisionado.

O principal objetivo deste método é treinar a rede para que ela seja capaz de responder corretamente às entradas-padrão que são usadas para o treinamento. E, após o treinamento, é esperado que a rede seja capaz de obter bons resultados para entradas similares àquelas usadas no treinamento.

O treinamento da rede neural *backpropagation* envolve três fases:

1. *Feedforward* da entrada de treinamento.

2. Cálculo e *backpropagation* do erro.

3. Ajuste de pesos.

Após o treinamento, somente a fase de *feedforward* é executada.

No problema de escalonamento *job shop*, como os valores da camada de entrada são números grandes, foi utilizado o maior valor das entradas para fazer a normalização das entradas. Cada entrada foi dividida pelo maior valor e em seguida foi multiplicada por 2.

No início da execução do algoritmo de *backpropagation*, dado que p é o número de neurônios da camada de entrada, q é o número de neurônios da camada intermediária e u é o número de neurônios da camada de saída, todos os pesos entre a camada de entrada e a camada intermediária ($v_{ij}; 1 \leq i \leq p; 1 \leq j \leq q$) e entre a camada intermediária e a camada de saída ($w_{jk}; 1 \leq j \leq q; 1 \leq k \leq u$), são gerados aleatoriamente, obedecendo o intervalo [-0.5, 0.5].

De acordo com (Bertsekas and Tsitsiklis (1996)), a escolha dos pesos iniciais influenciará a rede no alcance de um mínimo global (ou somente um mínimo local) do erro e, sendo assim, convergirá rapidamente. O ajuste de pesos entre duas unidades depende da derivada da função de ativação

da unidade da camada superior e da ativação da unidade da camada mais baixa. Por esta razão, é importante evitar escolhas de pesos iniciais que façam com que as ativações ou derivadas das ativações sejam zero. Os valores para os pesos iniciais não devem ser muito altos, pois os sinais de entrada iniciais para a camada intermediária ou para a camada de saída irão cair na região onde a derivada da função sigmóide assume um valor muito pequeno (região chamada de saturação). Por outro lado, se os pesos iniciais forem muito pequenos, os sinais de entrada da rede para a camada intermediária ou para a camada de saída ficarão em torno de zero, o que causará um baixo aprendizado. Um procedimento comum é inicializar os pesos com valores aleatórios entre [-0.5, 0.5]. Os valores podem ser positivos ou negativos, pois os pesos finais após o treinamento podem ser positivos ou negativos também.

Após a inicialização dos pesos, na fase de *feedforward*, cada entrada ($X_i; 1 \leq i \leq p$) recebe um sinal de entrada (x_i) e envia este sinal para cada unidade da camada intermediária ($Z_j; 1 \leq j \leq q$). Os valores das entradas - tempo total, tempo médio e tempo da máquina menos ocupada - são calculados através do escalonamento realizado pelo algoritmo memético.

Em seguida, cada unidade intermediária (Z_j) soma seu sinal de entrada (z_in_j) de acordo com a Equação 5.1, e aplica a função de ativação para calcular seu sinal de saída (z_j) conforme a Equação 5.2. Este sinal de saída é enviado para todas as unidades da camada de saída.

$$z\_in_j = v_{0j} + \sum_{i=1}^{p} x_i v_{ij}; \forall j, 1 \leq j \leq q. \tag{5.1}$$

$$z_j = f(z\_in_j). \tag{5.2}$$

Foi utilizada como função de ativação a função bipolar sigmóide. A escolha desta função deve-se ao fato dela ser contínua, diferenciável e monotonicamente decrescente, que são características necessárias pela rede *backpropagation*. Do ponto de vista computacional, é desejável que a derivada da função seja fácil de calcular. Para a maioria das funções de ativação geralmente utilizadas, o valor da derivada pode ser expressa em termos do valor da função. Usualmente, é esperado que a função sature aproximando-se dos valores máximo e mínimo. O sinal gerado pela função bipolar sigmóide tem limite entre (-1,1).

A função de ativação bipolar sigmóide é definida de acordo com a Equação 5.3 e sua derivada é definida de acordo com a Equação 5.4.

$$f(x) = \frac{2}{1 + \exp(-x)} - 1 \tag{5.3}$$

$$f'(x) = 0.5[1 + f(x)][1 - f(x)] \tag{5.4}$$

Cada unidade de saída ($Y_k, 1 \leq k \leq u$) soma seu sinal de entrada (y_in_k) conforme a Equação 5.5, e aplica a função de ativação para computar seu sinal de saída (y_k), que representa a resposta da rede a uma dada entrada, conforme a Equação 5.6. A função de ativação utilizada foi a bipolar sigmóide.

$$y\_in_k = w_{0k} + \sum_{j=1}^{q} z_j w_{jk}; \forall k, 1 \leq k \leq u. \tag{5.5}$$

$$y_k = f(y\_in_k). \tag{5.6}$$

Durante o treinamento, nas fases de cálculo e *backpropagation* do erro, cada unidade de saída compara o seu sinal saída (y_k) com o valor desejado t_k para determinar o fator de erro δ_k associado àquele padrão, que é calculado de acordo com a Equação 5.7. Este fator δ_k é utilizado para distribuir o erro da unidade de saída Y_k para as unidades da camada intermediária.

$$\delta_k = (t_k - y_k) f'(y\_in_k). \tag{5.7}$$

O valor de t_k, usado no treinamento da rede *backpropagation* para o problema de escalonamento de tarefas em máquinas paralelas idênticas, é o tempo total de processamento das tarefas alocadas à máquina mais ocupada. Para o valor de t_k, no problema de escalonamento *job shop*, foi utilizado o maior tempo final da última tarefa entre todas as últimas tarefas executadas pelas máquinas. Para o cálculo do valor de δ_k foi utilizado o valor da derivada da função bipolar sigmóide sobre o valor de y_in_k. O fator de erro δ_k é também utilizado para fazer o ajuste de pesos entre a camada de saída e a camada intermediária. Para fazer o ajuste de pesos, são calculados os termos de correção mostrados na Equação 5.8.

$$\begin{aligned} \Delta w_{jk} &= \alpha \delta_k z_j. \\ \Delta w_{0k} &= \alpha \delta_k. \end{aligned} \tag{5.8}$$

Na Equação 5.8, α é a taxa de aprendizagem. Foi utilizado, nos dois problemas, $\alpha = 0.5$. Ainda no treinamento, é calculado o fator de erro δ_j para cada unidade da camada intermediária Z_j. Este termo é calculado conforme a Equação 5.9.

$$\delta_j = \delta\_in_j f'(z\_in_j). \tag{5.9}$$

O valor de δ_in_j é obtido conforme a Equação 5.10.

$$\delta\_in_j = \sum_{k=1}^{r} \delta_k w_{jk}. \tag{5.10}$$

O fator de erro δ_j é utilizado para fazer o ajuste de pesos entre a camada intermediária e a camada de entrada. Vale ressaltar que não é necessário propagar o erro da camada intermediária para a camada de entrada. Para fazer o ajuste de pesos entre a camada intermediária e a camada de saída, são calculados os termos de correção apresentados pela Equação 5.11.

$$\begin{aligned} \Delta v_{ij} &= \alpha \delta_j x_i. \\ \Delta v_{0j} &= \alpha \delta_j. \end{aligned} \tag{5.11}$$

Após todos os fatores δ terem sido determinados, os pesos de todas as camadas são ajustados simultaneamente. Na fase de ajuste de pesos, o ajuste dos pesos w_{jk}, entre a camada intermediária Z_j e a camada de saída Y_k, é baseado no fator δ_k e na ativação z_j da camada intermediária Z_j.

O ajuste dos pesos v_{ij}, entre a camada de entrada X_i e a camada intermediária Z_j, é baseado no fator δ_j e na ativação x_i da camada de entrada X_i.

Os novos pesos são calculados conforme a Equação 5.12.

$$\begin{aligned} w_{jk}(novo) &= w_{jk}(velho) + \Delta w_{jk}. \\ v_{ij}(novo) &= v_{ij}(velho) + \Delta v_{ij}. \end{aligned} \tag{5.12}$$

O algoritmo padrão do *backpropagation* modifica pesos na direção da diminuição mais rápida da superfície do erro para os pesos atuais.

Segundo (Bertsekas and Tsitsiklis (1996)), uma época é um ciclo do treinamento da rede. Muitas épocas são necessárias para o treinamento da rede neural *backpropagation*. A base matemática do algoritmo do *backpropagation* é a técnica de otimização conhecida como gradiente. O gradiente da função (neste caso, a função é o erro e as variáveis são os pesos da rede) dá a direção na qual a função cresce mais rapidamente. O negativo do gradiente dá a direção na qual a função decresce mais rapidamente.

O treinamento foi realizado enquanto o fator de erro δ_k decrescia. Quando o erro começa a crescer, o treinamento é interrompido.

Após o treinamento, a rede neural *backpropagation* é aplicada utilizando apenas a fase de *feedforward* do algoritmo de treinamento. A motivação na utilização da rede *backpropagation* é a obtenção de um contrapeso entre respostas corretas aos testes padrão de treinamento e boas respostas aos novos padrões de entrada, ou seja, um balanço entre memorização e generalização. Segundo (Bertsekas and Tsitsiklis (1996)), não é necessariamente vantajoso continuar o treinamento até que o erro quadrático alcance um mínimo. (Hecht-Nielsen (1989)) sugere o uso de dois conjuntos de dados durante o treinamento: um conjunto de padrões de treinamento e um conjunto de padrões de teste e treinamento. Estes dois conjuntos são disjuntos. O ajuste de pesos é baseado nos padrões de treinamento e, nos intervalos durante o treinamento, o erro é computado usando os padrões de teste e

treinamento. O treinamento continua enquanto o erro para os padrões de treinamento e teste decresce. Quando o erro começa a aumentar, a rede está começando a memorizar os padrões de treinamento muito especificamente, começando a perder a habilidade de generalizar. Neste ponto, o treinamento é encerrado.

Ao final da execução do algoritmo, para o problema de escalonamento *job shop*, a saida (y_k) é multiplicada pelo maior valor da entrada e depois dividida por 2 para recuperar o seu valor original, obtido antes pela normalização das entradas.

Um pseudocódigo para o treinamento da rede neural *backpropagation* é apresentado pelo Algoritmo 13.

5.1.1.2 Rede neural com aprendizado por reforço

A história do aprendizado por reforço, de acordo com (Barto and Sutton (1998)), tem duas linhas principais. A primeira linha tem interesse no aprendizado por tentativa e erro e teve origem na psicologia de aprendizado animal. A outra linha tem interesse no problema de controle ótimo e na solução usando função de valor e programação dinâmica. O termo controle ótimo é utilizado para descrever o problema de projetar um controlador para minimizar uma medida do comportamento de um sistema dinâmico sobre o tempo. A Programação Dinâmica é considerado um dos métodos para resolver problemas de controle ótimo e o aprendizado por reforço também pode ser considerado um método de resolução de problemas de controle ótimo. Vários métodos de programação dinâmica são incrementais e iterativos e, igualmente aos métodos de aprendizado, eles alcançam gradualmente a resposta correta através de sucessivas aproximações.

A linha que foca o aprendizado por tentativa e erro é mais familiar e foi a linha principal que conduziu ao aprendizado por reforço. Esta linha teve início na psicologia, onde a teoria de aprendizado por reforço é comum. Edward Thorndike foi o primeiro a expressar a essência do aprendizado por tentativa e erro, utilizando a idéia de que ações são seguidas de bons ou maus resultados. Segundo (Thorndike (1911)), de várias respostas dadas a uma mesma situação, aquelas que são acompanhadas de uma satisfação do animal são mais conectadas à situação de modo que, quando esta situação retornar, estas ações serão tomadas. Em contrapartida, se as ações tomadas gerarem uma insatisfação no animal, para outras situações semelhantes essas ações serão menos prováveis de ocorrer novamente porque terão uma conexão mais enfraquecida com a situação. Quanto maior for a satisfação mais forte será a ligação entre a ação e a situação e, quanto menor for a satisfação, mais enfraquecida será esta ligação. Thorndike chamou esta teoria de a "Lei do Efeito" porque ela descreve os efeitos dos eventos do reforço provocados pela seleção de determinadas ações.

A Lei do Efeito inclui os dois aspectos mais importantes do aprendizado por tentativa e erro. O primeiro aspecto é o da seleção, que envolve tentar alternativas e selecioná-las através da comparação

Algoritmo 13 Pseudocódigo do treinamento da rede neural *backpropagation*

Entrada: p: número de neurônios da camada de entrada, q: número de neurônios da camada intermediária, u: número de neurônios da camada de saída, X_i: neurônio da camada de entrada da rede neural, onde $1 \leq i \leq p$, Z_j: neurônio da camada intermediária da rede neural, onde $1 \leq j \leq q$, Y_k: neurônio da camada de saída da rede neural, onde $1 \leq k \leq u$, t_k: valor de saída desejado, α: taxa de aprendizagem.

- Inicializar os pesos ($v_{ij}; 1 \leq i \leq p; 1 \leq j \leq q$) e ($w_{jk}; 1 \leq j \leq q; 1 \leq k \leq u$)

repita

Fase do *feedforward*:

- Cada entrada ($X_i; 1 \leq i \leq p$) recebe um sinal de entrada (x_i) e envia este sinal para cada unidade da camada intermediária ($Z_j; 1 \leq j \leq q$)
- Cada unidade intermediária (Z_j) calcula seu sinal de entrada (z_in_j) e aplica a função de ativação para calcular seu sinal de saída (z_j). Este sinal de saída é enviado para todas as unidades da camada de saída
- Cada unidade de saída ($Y_k, 1 \leq k \leq u$) calcula seu sinal de entrada (y_in_k) e aplica a função de ativação para computar seu sinal de saída (y_k)

Fase de *backpropagation* do erro:

- Cada unidade de saída compara o seu sinal de saída (y_k) com o valor desejado (t_k) para determinar o erro (δ_k) associado àquele padrão e utiliza este fator de erro para calcular os termos de correção (Δw_{jk}) e (Δw_{0k})
- Cada unidade intemediária calcula o fator de erro (δ_j) e, com base nesse fator, calcula os termos de correção (Δv_{ij}) e (Δv_{0j})

Fase de ajuste de pesos:

- Cada unidade de saída ajusta seu peso ($w_{jk}(novo)$)
- Cada unidade intermediária ajusta seu peso ($v_{ij}(novo)$)

até (δ_k) cresça

Saída: Saída da rede (y_k)

de suas conseqüências. O segundo aspecto é o da associação, significando que as alternativas encontradas pela seleção estão associadas com as situações particulares. A seleção natural na evolução é um exemplo de processo de seleção, mas que não é associativo. O aprendizado supervisionado é associativo, mas não segue o processo de seleção. A combinação destes dois aspectos é essencial para a Lei do Efeito e para o aprendizado por tentativa e erro.

A Lei do Efeito é também um modo de combinar busca e memória - *busca* na forma de tentar e selecionar entre muitas ações em cada situação, e *memória* na forma de relembrar quais ações são melhores, associando-as com situações nas quais foram melhores. A combinação de busca de memória é essencial para o aprendizado por reforço.

Segundo (Barto and Sutton (1998)), a idéia de programar um computador para que ele aprenda por tentativa e erro surgiu na década de 50 através de trabalhos de Minsky, que apresentou uma discussão de modelos computacionais de aprendizado por reforço, e Farley e Clark, que apresentaram uma rede neural projetada para o aprendizado por tentativa e erro. Na década de 60 os termos "reforço" e "aprendizado por reforço" foram amplamente utilizados na engenharia.

O aprendizado por reforço, segundo (Barto and Sutton (1998)), é um método onde existe uma interação entre um agente e um ambiente com o intuito de alcançar o objetivo. Esta interação pode ser verificada pela Figura. 5.3.

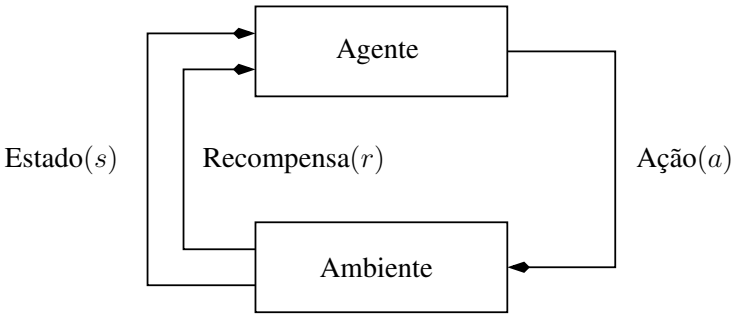

Fig. 5.3: Interação entre agente e ambiente no aprendizado por reforço.

Nesta interação, o ambiente é responsável por responder a ação tomada pelo agente, podendo dar uma recompensa ou não por esta ação. O ambiente é definido por um conjunto de estados.

O aprendizado por reforço ensina como mapear estados em ações e como maximizar um sinal de recompensa numérico (recompensa final). O agente não sabe qual ação tomar. Ele vai descobrindo qual ação gera uma melhor recompensa. Na maioria dos casos, as ações não afetam somente a recompensa imediata, mas também recompensas subseqüentes. As duas características mais importantes do aprendizado por reforço são - tentativa e erro, e busca e recompensa (Barto and Sutton (1998)).

O agente, no aprendizado por reforço, é quem toma a ação/decisão e interage com o ambiente

com o objetivo de maximizar sua recompensa total. O agente seleciona ações e o ambiente responde a estas ações apresentando novas situações ao agente. O agente deve ser capaz de captar o estado do ambiente e de tomar ações que afetem o estado do ambiente.

O aprendizado por reforço é diferente do aprendizado supervisionado. O aprendizado supervisionado é o aprendizado de exemplos providos pelo conhecimento do supervisor externo. Esta é uma classe importante de aprendizado, mas não é adequada sozinha na aprendizagem por interação. Em problemas interativos não é muito comum obter exemplos de ações desejadas que representem todas as situações nas quais o agente pode agir. Neste caso, é esperado que o agente seja capaz de aprender com sua própria experiência.

Um dos desafios do aprendizado por reforço em relação a outras classes de aprendizado é o balanço entre a exploração e explotação. Para obter uma boa recompensa, o agente deve preferir ações tomadas no passado e que produziram uma boa recompensa mas, para que o agente tome essa ações, ele deve tentar ações ainda não tomadas no passado. O agente tem que explorar ações que ele já conhece com o objetivo de obter a recompensa, mas também tem que explorar novas ações com o intuito de melhorar a recompensa. O agente precisa tentar uma série de ações e progressivamente favorecer aquelas que parecem ser melhores. Outra característica chave do aprendizado por reforço é explicitar o problema como um todo de um agente guiado por um objetivo, e que interage com um ambiente incerto.

O agente da aprendizagem por reforço tem um objetivo explícito, podendo detectar aspectos de seu ambiente e escolher ações que irão influenciar diretamente no ambiente. Além disso, é considerado que no início o agente opera no ambiente com uma incerteza significativa.

A seguir são destacadas algumas aplicações do aprendizado por reforço:

- No movimento de um jogador de xadrez. A escolha é verificada através de um planejamento - antecipando possíveis respostas - e através de um julgamento imediato e intuitivo de posições e movimentos particulares.

- Na operação de ajuste de parâmetros em tempo real do controlador adaptativo de uma refinaria de petróleo. O controlador otimiza esses parâmetros levando em consideração uma relação de rendimento, custo e qualidade. A decisão é tomada com base nos custos especificados e nos pontos sugeridos originalmente pelos engenheiros.

- Na decisão de um robô móvel - se ele entrará em uma nova sala na busca de mais lixo para coletar ou se começará a encontrar uma maneira de retornar à sua estação para recarregar a bateria. O robô toma sua decisão baseada em quão rapidamente e facilmente pôde encontrar o recarregador.

Todos estes exemplos envolvem interação entre um agente que irá tomar uma decisão e seu ambiente, dentro do qual o agente procurará conseguir um objetivo, apesar da incerteza existente sobre o ambiente. As ações do agente permitem afetar o estado futuro do ambiente (dentro dos exemplos acima, a próxima posição no jogo de xadrez, o nível dos reservatórios da refinaria, a próxima localização do robô) e, desse modo, afetando as opções e oportunidades disponíveis posteriormente ao agente. A escolha correta requer um exame de conseqüências indiretas e posteriores às ações tomadas e, por isso, requer um planejamento. O efeito das ações não pode ser completamente predito e, portanto, o agente precisa monitorar seu ambiente frequentemente e reagir apropriadamente. Em todos estes exemplos o agente pode usar sua experiência para melhorar sua performance ao longo do tempo. O jogador de xadrez, por exemplo, refina a intuição que se usa para avaliar as posições, melhorando desse modo seu jogo.

O conhecimento que o agente traz à tarefa no início, seja da experiência prévia de tarefas relacionadas ou construídas dentro do projeto ou pela evolução, pode influenciar na decisão do que é útil ou fácil de aprender, mas a interação com o ambiente é essencial para ajustar o comportamento a fim de explorar características específicas da tarefa.

Além do agente e do ambiente, podemos destacar outros 4 elementos da rede neural com aprendizado por reforço: uma política, uma função de recompensa, uma função de valor e, opcionalmente, um modelo do ambiente.

Uma política define a maneira que o agente deverá se comportar em um dado momento. Uma política é o mapeamento de estados percebidos do ambiente às ações que podem ser executadas naqueles estados. Isto poderia ser chamado de associações ou regras de estímulo e resposta. Em alguns casos a política pode ser uma função simples ou uma tabela. A política é o núcleo do agente na aprendizagem por reforço no sentido que ele sozinho é suficiente para determinar o comportamento. Em geral as políticas podem ser estocásticas.

A função de recompensa define o objetivo no aprendizado por reforço. Ela mapeia cada estado percebido do ambiente para um número simples, uma recompensa, indicando o quão desejável é este estado. O objetivo do agente na aprendizagem por reforço é maximizar a recompensa total recebida durante o processo de aprendizagem. A função de recompensa define o que são bons e maus eventos para o agente. A função de recompensa não pode ser alterada pelo agente. Ela serve de base para alterar a política. Por exemplo, se uma ação selecionada pela política levar a uma baixa recompensa, a política pode ser mudada para selecionar alguma outra ação em uma situação futura. Em geral as funções da recompensa podem ser estocásticas. Enquanto a função de recompensa indica o que é bom em um sentido imediato, a função de valor especifica o que é bom em um sentindo global e a longo prazo.

A função de valor de um estado é a quantidade total de recompensa que um agente pode esperar

acumular sobre o futuro, partindo deste estado. Enquanto a recompensa determina o quão é desejável um determinado estado no sentindo imediato, o valor indica o quão é desejável um determinado estado a longo prazo, após a escolha de outros estados e das recompensas por estas escolhas. Por exemplo, um estado pode sempre render uma recompensa imediata baixa mas ainda ter um valor elevado porque se segue regularmente por outros estados que possuem recompensas elevadas. O inverso também é verdadeiro. As recompensas estão em um plano primário e os valores, que são como predições das recompensas, são secundários. Sem recompensas não poderia haver valor algum, e a única finalidade de estimar valores é conseguir mais recompensa. Contudo, o interesse maior é com os valores ao fazer e avaliar decisões.

O conceito de função de valor é característica chave do método de aprendizado por reforço. A função de valor é essencial para a busca eficiente no espaço de políticas.

As escolhas das ações são feitas com base em julgamentos do valor. Procura-se as ações que causam estados com valor mais elevado, e não necessariamente com a recompensa mais elevada, porque estas ações obtêm uma quantidade maior de recompensa sobre todo o aprendizado. Na tomada de decisão e no planejamento, o interesse é na quantidade derivada chamada valor. Infelizmente, é muito mais difícil determinar valores do que determinar recompensas. As recompensas são dadas basicamente diretamente pelo ambiente, mas os valores precisam ser estimados e reestimados através de seqüência de observações feitas pelo agente. De fato, o componente mais importante do aprendizado por reforço é um método eficiente para estimar valores.

O quarto elemento do aprendizado por reforço é o modelo do ambiente. O modelo representa o comportamento do ambiente. Por exemplo, dado um estado e uma ação, o modelo pode predizer o estado seguinte resultante e a recompensa seguinte. Os modelos são usados para o planejamento e para decidir o curso de uma ação, considerando situações futuras possíveis, antes que a ação seja executada. A incorporação de modelos e de planejamento em sistemas de aprendizagem por reforço é um desenvolvimento relativamente novo.

No algoritmo de aprendizado por reforço, aplicado ao escalonador neuro-memético, com o objetivo de minimizar o *makespan* no escalonamento de tarefas em máquinas paralelas idênticas e no escalonamento *job shop* com parâmetros precisos, o algoritmo memético é executado para cada indivíduo da população e fornece as três entradas para a rede neural. A rede neural é aplicada com o intuito de encontrar o valor de *fitness* para as entradas fornecidas pelo algoritmo memético.

Inicialmente, a rede neural inicializa os pesos entre as camadas de entrada e intermediária e entre as camadas intermediária e de saída. Os pesos são gerados aleatoriamente, obedecendo ao intervalo [-0.5, 0.5].

Depois da inicialização dos pesos, a cada iteração, a rede neural realiza a fase de *feedforward*, seguindo os mesmos procedimentos da rede *backpropagation*, obtendo a saída da rede neural (y_k).

Após o cálculo da saída da rede neural, é verificado em qual estado a rede se encontra e qual a recompensa que ela deverá receber, se uma taxa de admissão ou uma taxa de rejeição. O valor de (r) é calculado com base na diferença entre a saída esperada (t_k) e a saída obtida pela rede (y_k). O cálculo da *função de recompensa* é efetuado conforme a Equação 5.13.

$$r = \frac{1}{1 + (t_k - y_k)} \tag{5.13}$$

A função de recompensa define quais eventos são bons e quais eventos são ruins.

O valor de (r) é utilizado para calcular a recompensa atual do agente. O cálculo desta recompensa é realizado de acordo com a **função valor** definida pela Equação 5.14.

$$V_l = V_{l-1} + r\gamma^l \tag{5.14}$$

A função valor especifica a quantidade total de recompensa que o agente pode esperar acumular para o futuro.

Na Equação 5.14, o valor de l representa o número atual da iteração da rede neural e γ é uma taxa de desconto. Foi utilizado para γ o valor 0.7.

Assim que a recompensa é calculada, se o critério de parada não for satisfeito, o ajuste de pesos é realizado de acordo com a Equação 5.15 ou Equação 5.16. Em caso de admissão, que é quando ocorre um aumento na recompensa, somente a Equação 5.15 é executada e, em caso de rejeição, que é quando ocorre uma diminuição no valor da recompensa, somente a Equação 5.16 é executada.

$$\begin{aligned} w_{jk}(novo) &= (1-\alpha)w_{jk}(velho) + \alpha(r + V_{l-1}). \\ v_{ij}(novo) &= (1-\alpha)v_{ij}(velho) + \alpha(r + V_{l-1}). \end{aligned} \tag{5.15}$$

$$\begin{aligned} w_{jk}(novo) &= (1-\alpha)w_{jk}(velho) + \alpha(r + V_{l}). \\ v_{ij}(novo) &= (1-\alpha)v_{ij}(velho) + \alpha(r + V_{l}). \end{aligned} \tag{5.16}$$

O critério de parada utilizado é $\epsilon \leq 0.001$, assumindo que $\epsilon = V_l - V_{l-1}$. Foi utilizado $\alpha = 0.0005$ como taxa de aprendizagem.

Ao final da execução do algoritmo, para o problema de escalonamento *job shop*, a saída (y_k) é multiplicada pelo maior valor da entrada e depois dividida por 2 para recuperar o seu valor original, obtido antes pela normalização das entradas.

Um pseudocódigo para o treinamento da rede neural com aprendizado por reforço é apresentado pelo Algoritmo 14.

Algoritmo 14 Pseudocódigo do treinamento da rede neural com aprendizado por reforço

Entrada: p: número de neurônios da camada de entrada, q: número de neurônios da camada intermediária, u: número de neurônios da camada de saída, X_i: neurônio da camada de entrada da rede neural, onde $1 \leq i \leq p$, Z_j: neurônio da camada intermediária da rede neural, onde $1 \leq j \leq q$, Y_k: neurônio da camada de saída da rede neural, onde $1 \leq k \leq u$, t_k: valor de saída desejado, α: taxa de aprendizagem, γ: taxa de desconto.

- Inicializar os pesos $(v_{ij}; 1 \leq i \leq p; 1 \leq j \leq q)$ e $(w_{jk}; 1 \leq j \leq q; 1 \leq k \leq u)$
- Inicializar $num = 1$

repita

Fase do *feedforward*:

- Cada entrada $(X_i; 1 \leq i \leq p)$ recebe um sinal de entrada (x_i) e envia este sinal para cada unidade da camada intermediária $(Z_j; 1 \leq j \leq q)$
- Cada unidade intermediária (Z_j) calcula seu sinal de entrada (z_in_j) e aplica a função de ativação para calcular seu sinal de saída (z_j). Este sinal de saída é enviado para todas as unidades da camada de saída
- Cada unidade de saída $(Y_k, 1 \leq k \leq u)$ calcula seu sinal de entrada (y_in_k) e aplica a função de ativação para computar seu sinal de saída (y_k)

Cálculo da recompensa:

- Cada unidade de saída calcula o valor da recompensa (r) com base no valor de saída (y_k) e no valor desejado (t_k)
- A recompensa atual V_{num} do agente é calculado com base no valor de (r)

Fase de ajuste de pesos:

- Cada unidade de saída ajusta seu peso $(w_{jk}(novo))$
- Cada unidade intermediária ajusta seu peso $(v_{ij}(novo))$

$num = num + 1$
$\epsilon = V_{num} - V_{num-1}$
até $\epsilon \leq 0.0001$
Saída: Saída da rede (y_k)

Capítulo 6

Escalonamento neuro-memético de tarefas com parâmetros com incertezas

Neste capítulo são abordados dois tipos de problemas de escalonamento de tarefas com parâmetros com incertezas - o escalonamento de tarefas em máquinas paralelas idênticas e o problema de escalonamento *job shop*. Na resolução destes problemas de escalonamento são aplicados o algoritmo memético com população estruturada em coevolução com redes neurais. Nesta coevolução, o algoritmo memético é um método heurístico que tem como objetivo evoluir boas formas de escalonamento na busca de soluções para os problemas, e a rede neural tem como objetivo a obtenção do valor da função de *fitness* para cada indivíduo gerado pelo algoritmo memético. Foram aplicadas duas redes neurais nesta coevolução - a rede neural do tipo *backpropagation* e a rede neural com aprendizado por reforço.

6.1 Arquitetura da rede neural

Tanto na resolução do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas, quanto na resolução do problema de escalonamento *job shop*, foram utilizadas redes neurais coevoluindo com o algoritmo memético com população estruturada. O algoritmo memético com população estruturada foi descrito na Seção 3.2.2.1. Sua aplicação ao problema de escalonamento de tarefas em máquinas paralelas idênticas foi detalhada na Seção 4.2.2.1, e sua aplicação ao problema de escalonamento *job shop* foi detalhada na Seção 4.3.2.1. O algoritmo memético foi utilizado para evoluir boas formas de escalonamento, e a rede neural foi utilizada para calcular a função de *fitness* de cada indivíduo que compõe a população no algoritmo memético. A rede neural age na coevolução do algoritmo memético. Este esquema de coevolução é apresentado na Figura. 6.1.

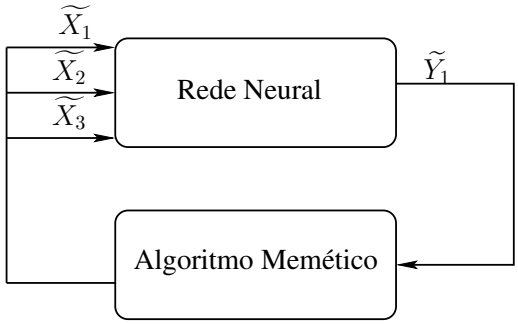

Fig. 6.1: Esquema de coevolução do algoritmo memético com a rede neural com incertezas.

6.1.1 Redes neurais com incertezas

A arquitetura da rede neural do escalonador neuro-memético, conforme apresentada pela Figura 6.2, possui uma camada de entrada com três neurônios, sendo que a primeira entrada representa o tempo total de processamento das tarefas. Para o problema de escalonamento *job shop*, foram somados os tempos finais da última tarefa de cada máquina. A segunda entrada representa o tempo médio (tempo total de processamento / número de máquinas). A terceira entrada, no problema de escalonamento de tarefas em máquinas paralelas idênticas, representa o tempo total de processamento na máquina mais vazia, ou seja, a máquina que é a primeira a terminar a execução das tarefas alocadas a ela. A terceira entrada, no problema de escalonamento *job shop*, representa o menor tempo final da última tarefa entre todas as últimas tarefas executadas pelas máquinas. As três entradas são números *fuzzy* triangulares e foram representados utilizando intervalos de confiança (com o uso de α-cortes), conforme descrito na Equação 2.3.1. Para encontrar a terceira entrada foi utilizado o conceito de possibilidade, conforme descrito na Seção 2.3.4, pois tratam-se de números *fuzzy*.

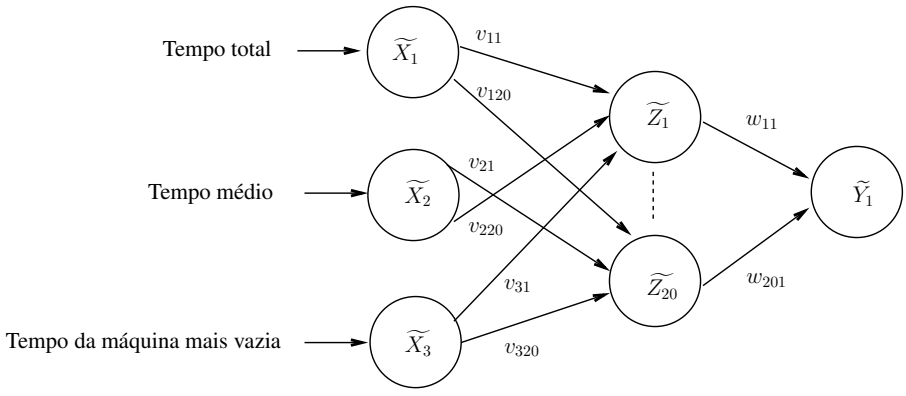

Fig. 6.2: Arquitetura da rede neural com incertezas.

As camadas intermediária e de saída da rede neural foram definidas da mesma forma que na rede

neural para os problemas com parâmetros precisos, conforme descrito na Seção 5.1.1. A diferença é que, para este caso, estes parâmetros contém incertezas e são representados por números *fuzzy* triangulares.

Da mesma forma que no problema com parâmetros precisos, foram aplicadas duas redes neurais - *backpropagation* e com aprendizado por reforço.

6.1.1.1 Rede neural *backpropagation* com incertezas

As descrições da rede neural do tipo *backpropagation* podem ser vistas na Seção 5.1.1.1.

Para o problema de escalonamento *job shop*, como os valores da camada de entrada são números grandes, foi utilizado o maior valor das entradas para fazer a normalização das entradas. Como as entradas são valores *fuzzy*, foi utilizado o conceito de possibilidade para encontrar a maior entrada, conforme descrito na Seção 2.3.4. Cada entrada foi dividida pelo maior valor e em seguida multiplicada por 2. Essas operações de divisão e multiplicação foram realizadas de acordo com as Equações 2.14 e 2.13, respectivamente.

No início da execução do algoritmo de *backpropagation*, dado que p é o número de neurônios da camada de entrada, q é o número de neurônios da camada intermediária e u é o número de neurônios da camada de saída, todos os pesos entre a camada de entrada e a camada intermediária ($v_{ij}; 1 \leq i \leq p; 1 \leq j \leq q$) e entre a camada intermediária e a camada de saída ($w_{jk}; 1 \leq j \leq q; 1 \leq k \leq u$), são gerados aleatoriamente, obedecendo o intervalo [-0.5, 0.5].

Após a inicialização dos pesos, na fase de *feedforward*, cada entrada ($\widetilde{X_i}; 1 \leq i \leq p$) recebe um sinal de entrada ($\widetilde{x_i}$) e envia este sinal para cada unidade da camada intermediária ($\widetilde{Z_j}; 1 \leq j \leq q$). Os valores das entradas - tempo total, tempo médio e tempo da máquina menos ocupada - são calculados através do escalonamento realizado pelo algoritmo memético e são números *fuzzy* triangulares. Cada número *fuzzy* foi representado na forma de intervalos de confiança, conforme descrito na Equação 2.3.1. O algoritmo da construção do número *fuzzy* com intervalos de confiança está descrito no Algoritmo 1.

Em seguida, cada unidade intermediária ($\widetilde{Z_j}$) soma seu sinal de entrada ($\widetilde{z\_in_j}$) de acordo com a Equação 6.1, e aplica a função de ativação para calcular seu sinal de saída ($\widetilde{z_j}$) conforme a Equação 6.2. Este sinal de saída é enviado para todas as unidades da camada de saída.

$$\widetilde{z\_in_j} = v_{0j} \oplus \sum_{i=1}^{p} \widetilde{x_i} \otimes v_{ij}; \forall j, 1 \leq j \leq q. \tag{6.1}$$

$$\widetilde{z_j} = f(\widetilde{z\_in_j}). \tag{6.2}$$

Foi utilizada como função de ativação a função bipolar sigmóide, que é definida de acordo com a

Equação 6.3 e sua derivada é definida de acordo com a Equação 6.4.

$$f(\widetilde{x}) = (2 \oslash (1 \oplus \exp(-\widetilde{x}))) \ominus 1 \tag{6.3}$$

$$f'(\widetilde{x}) = 0.5 \otimes [1 \oplus f(\widetilde{x})] \otimes [1 \ominus f(\widetilde{x})] \tag{6.4}$$

Cada unidade de saída ($\widetilde{Y_k}, 1 \leq k \leq u$) soma seu sinal de entrada ($\widetilde{y\_in_k}$) conforme a Equação 6.5, e aplica a função de ativação para computar seu sinal de saída ($\widetilde{y_k}$), que representa a resposta da rede a uma dada entrada, conforme a Equação 6.6. A função de ativação utilizada foi a bipolar sigmóide.

$$\widetilde{y\_in_k} = w_{0k} \oplus \sum_{j=1}^{q} \widetilde{z_j} \otimes w_{jk}; \forall k, 1 \leq k \leq u. \tag{6.5}$$

$$\widetilde{y_k} = f(\widetilde{y\_in_k}). \tag{6.6}$$

Durante o treinamento, nas fases de cálculo e *backpropagation* do erro, cada unidade de saída compara o seu sinal saída ($\widetilde{y_k}$) com o valor desejado $\widetilde{t_k}$ para determinar o fator de erro $\widetilde{\delta_k}$ associado àquele padrão, que é calculado de acordo com a Equação 6.7. Este fator $\widetilde{\delta_k}$ é utilizado para distribuir o erro da unidade de saída $\widetilde{Y_k}$ para as unidades da camada intermediária.

$$\widetilde{\delta_k} = (\widetilde{t_k} \ominus \widetilde{y_k}) \otimes f'(\widetilde{y\_in_k}). \tag{6.7}$$

O valor de $\widetilde{t_k}$, usado no treinamento da rede *backpropagation* para o problema de escalonamento de tarefas em máquinas paralelas idênticas, é o tempo total de processamento das tarefas alocadas à máquina mais ocupada. Para a valor de $\widetilde{t_k}$, no problema de escalonamento *job shop*, foi utilizado o maior tempo final da última tarefa entre todas as últimas tarefas executadas pelas máquinas. O fator de erro $\widetilde{\delta_k}$ é também utilizado para fazer o ajuste de pesos entre a camada de saída e a camada intermediária. Para fazer o ajuste de pesos, são calculados os termos de correção mostrados na Equação 6.8.

$$\begin{array}{l} \widetilde{\Delta w_{jk}} = \alpha \otimes \widetilde{\delta_k} \otimes \widetilde{z_j}. \\ \widetilde{\Delta w_{0k}} = \alpha \otimes \widetilde{\delta_k}. \end{array} \tag{6.8}$$

Na Equação 6.8, α é a taxa de aprendizagem. Ainda no treinamento, é calculado o fator de erro $\widetilde{\delta_j}$ para cada unidade da camada intermediária $\widetilde{Z_j}$. Este termo é calculado conforme a Equação 6.9.

$$\widetilde{\delta_j} = \widetilde{\delta\_in_j} \otimes f'(\widetilde{z\_in_j}). \tag{6.9}$$

O valor de $\widetilde{\delta\_in_j}$ é obtido conforme a Equação 6.10.

$$\widetilde{\delta\_in_j} = \sum_{k=1}^{r} \widetilde{\delta}_k \otimes w_{jk}. \tag{6.10}$$

O fator de erro $\widetilde{\delta}_j$ é utilizado para fazer o ajuste de pesos entre a camada intermediária e a camada de entrada. Vale ressaltar que não é necessário propagar o erro da camada intermediária para a camada de entrada. Para fazer o ajuste de pesos entre a camada intermediária e a camada de saída, são calculados os termos de correção apresentados pela Equação 6.11.

$$\begin{aligned} \widetilde{\Delta v_{ij}} &= \alpha \otimes \widetilde{\delta}_j \otimes \widetilde{x}_i. \\ \widetilde{\Delta v_{0j}} &= \alpha \otimes \widetilde{\delta}_j. \end{aligned} \tag{6.11}$$

Após todos os fatores $\widetilde{\delta}$ terem sido determinados, os pesos de todas as camadas são ajustados simultaneamente. Na fase de ajuste de pesos, o ajuste dos pesos w_{jk}, entre a camada intermediária $\widetilde{Z_j}$ e a camada de saída $\widetilde{Y_k}$, é baseado no fator $\widetilde{\delta_k}$ e na ativação $\widetilde{z_j}$ da camada intermediária $\widetilde{Z_j}$.

O ajuste dos pesos v_{ij}, entre a camada de entrada $\widetilde{X_i}$ e a camada intermediária $\widetilde{Z_j}$, é baseado no fator $\widetilde{\delta_j}$ e na ativação $\widetilde{x_i}$ da camada de entrada $\widetilde{X_i}$.

Os novos pesos são calculados conforme a Equação 6.12.

$$\begin{aligned} w_{jk}(novo) &= w_{jk}(velho) + \text{defuzzificacao}\{\widetilde{\Delta w_{jk}}\}. \\ v_{ij}(novo) &= v_{ij}(velho) + \text{defuzzificacao}\{\widetilde{\Delta v_{ij}}\}. \end{aligned} \tag{6.12}$$

Na Equação 6.12 foi preciso utilizar o valor *crisp* dos termos de correção e, como eles são números *fuzzy*, foi aplicada a *defuzzificação*, que é o mapeamento de informações *fuzzy* em valores *crisp*. O método de *defuzzificação* aplicado foi o Método da Centróide (Centro de Gravidade), detalhado na Seção 2.3.5.

O treinamento foi realizado enquanto o fator de erro $\widetilde{\delta_k}$ decrescia. Quando o erro começa a crescer, o treinamento é interrompido.

Após o treinamento, a rede neural *backpropagation* é aplicada utilizando apenas a fase de *feed-forward* do algoritmo de treinamento.

Um pseudocódigo para o treinamento da rede neural *backpropagation* com incertezas é apresentado pelo Algoritmo 15.

6.1.1.2 Rede neural com aprendizado por reforço com incertezas

As descrições da rede neural com aprendizado por reforço e a interação entre seus componentes podem ser vistas na Seção 5.1.1.2.

Algoritmo 15 Pseudocódigo do treinamento da rede neural *backpropagation* com incertezas

Entrada: p: número de neurônios da camada de entrada, q: número de neurônios da camada intermediária, u: número de neurônios da camada de saída, $\widetilde{X_i}$: neurônio da camada de entrada da rede neural, onde $1 \leq i \leq p$, $\widetilde{Z_j}$: neurônio da camada intermediária da rede neural, onde $1 \leq j \leq q$, $\widetilde{Y_k}$: neurônio da camada de saída da rede neural, onde $1 \leq k \leq u$, $\widetilde{t_k}$: valor de saída desejado, α: taxa de aprendizagem.

- Inicializar os pesos ($v_{ij}; 1 \leq i \leq p; 1 \leq j \leq q$) e ($w_{jk}; 1 \leq j \leq q; 1 \leq k \leq u$)

repita

Fase do *feedforward*:

- Cada entrada ($\widetilde{X_i}; 1 \leq i \leq p$) recebe um sinal de entrada ($\widetilde{x_i}$) e envia este sinal para cada unidade da camada intermediária ($\widetilde{Z_j}; 1 \leq j \leq q$)
- Cada unidade intermediária ($\widetilde{Z_j}$) calcula seu sinal de entrada ($\widetilde{z\_in_j}$) e aplica a função de ativação para calcular seu sinal de saída ($\widetilde{z_j}$). Este sinal de saída é enviado para todas as unidades da camada de saída
- Cada unidade de saída ($\widetilde{Y_k}, 1 \leq k \leq u$) calcula seu sinal de entrada ($\widetilde{y\_in_k}$) e aplica a função de ativação para computar seu sinal de saída ($\widetilde{y_k}$)

Fase de *backpropagation* do erro:

- Cada unidade de saída compara o seu sinal de saída ($\widetilde{y_k}$) com o valor desejado ($\widetilde{t_k}$) para determinar o erro ($\widetilde{\delta_k}$) associado àquele padrão e utiliza este fator de erro para calcular os termos de correção ($\widetilde{\Delta w_{jk}}$) e ($\widetilde{\Delta w_{0k}}$)
- Cada unidade intemediária calcula o fator de erro ($\widetilde{\delta_j}$) e, com base nesse fator, calcula os termos de correção ($\widetilde{\Delta v_{ij}}$) e ($\widetilde{\Delta v_{0j}}$)

Fase de ajuste de pesos:

- Cada unidade de saída ajusta seu peso ($w_{jk}(novo)$)
- Cada unidade intermediária ajusta seu peso ($v_{ij}(novo)$)

até ($\widetilde{\delta_k}$) cresça

Saída: Saída da rede ($\widetilde{y_k}$)

Inicialmente, a rede neural inicializa os pesos entre as camadas de entrada e intermediária e entre as camadas intermediária e de saída. Os pesos são gerados aleatoriamente, obedecendo ao intervalo [-0.5, 0.5].

Depois da inicialização dos pesos, a cada iteração, a rede neural realiza a fase de *feedforward*, seguindo os mesmos procedimentos da rede *backpropagation*, obtendo a saída da rede neural ($\widetilde{y_k}$). Após o cálculo da saída da rede neural, é verificado em qual estado a rede se encontra e qual a recompensa ($\widetilde{r}$) que ela deverá receber, se uma taxa de admissão ou uma taxa de rejeição. O valor de ($\widetilde{r}$) é calculado com base na diferença entre a saída esperada ($\widetilde{t_k}$) e a saída obtida pela rede ($\widetilde{y_k}$). O cálculo da **função de recompensa** é efetuado conforme Equação 6.13.

$$\widetilde{r} = 1 \oslash (1 \oplus (\widetilde{t_k} \ominus \widetilde{y_k})) \tag{6.13}$$

O valor de ($\widetilde{r}$) é utilizado para calcular a recompensa atual ($\widetilde{V_l}$) do agente. O cálculo da recompensa é realizado de acordo com a **função valor** definida pela Equação 6.14.

$$\widetilde{V_l} = \widetilde{V_{l-1}} \oplus (\widetilde{r} \otimes \gamma^l) \tag{6.14}$$

Na Equação 6.14, o valor de l representa o número atual da iteração da rede neural e γ é uma taxa de desconto. Foi utilizado para γ o valor 0.9.

Assim que a recompensa é calculada, se o critério de parada não for satisfeito, o ajuste de pesos é realizado de acordo com a Equação 6.15 ou Equação 6.16. Em caso de admissão, que é quando ocorre um aumento na recompensa, somente a Equação 6.15 é executada e, em caso de rejeição, que é quando ocorre uma diminuição no valor da recompensa, somente a Equação 6.16 é executada.

$$\begin{aligned} w_{jk}(novo) &= (1-\alpha)w_{jk}(velho) + defuzzificacao(\alpha \otimes (\widetilde{r} \oplus \widetilde{V_{l-1}})). \\ v_{ij}(novo) &= (1-\alpha)v_{ij}(velho) + defuzzificacao(\alpha \otimes (\widetilde{r} \oplus \widetilde{V_{l-1}})). \end{aligned} \tag{6.15}$$

$$\begin{aligned} w_{jk}(novo) &= (1-\alpha)w_{jk}(velho) + defuzzificacao(\alpha \otimes (\widetilde{r} \oplus \widetilde{V_l})). \\ v_{ij}(novo) &= (1-\alpha)v_{ij}(velho) + defuzzificacao(\alpha \otimes (\widetilde{r} \oplus \widetilde{V_l})). \end{aligned} \tag{6.16}$$

Nas Equações 6.15 e 6.16 foi preciso utilizar o valor *crisp* das recompensas e, como são números *fuzzy*, foi aplicada a *defuzzificação*, que é o mapeamento de informações *fuzzy* em valores *crisp*. O método de *defuzzificação* aplicado foi o Método da Centróide (Centro de Gravidade), detalhado na Seção 2.3.5.

Foi utilizado $\alpha = 0.0005$ como taxa de aprendizagem. O critério de parada utilizado é $\epsilon \leq 0.001$, assumindo que $\epsilon = defuzzificacao(V_l) - defuzzificacao(V_{l-1})$.

Um pseudocódigo para o treinamento da rede neural com aprendizado por reforço com incertezas é apresentado pelo Algoritmo 16.

Algoritmo 16 Pseudocódigo do treinamento da rede neural com aprendizado por reforço com incertezas

Entrada: p: número de neurônios da camada de entrada, q: número de neurônios da camada intermediária, u: número de neurônios da camada de saída, $\widetilde{X_i}$: neurônio da camada de entrada da rede neural, onde $1 \leq i \leq p$, $\widetilde{Z_j}$: neurônio da camada intermediária da rede neural, onde $1 \leq j \leq q$, $\widetilde{Y_k}$: neurônio da camada de saída da rede neural, onde $1 \leq k \leq u$, $\widetilde{t_k}$: valor de saída desejado, α: taxa de aprendizagem, γ: taxa de desconto.

- Inicializar os pesos $(v_{ij}; 1 \leq i \leq p; 1 \leq j \leq q)$ e $(w_{jk}; 1 \leq j \leq q; 1 \leq k \leq u)$
- Inicializar $num = 1$

repita

Fase do *feedforward*:

- Cada entrada $(\widetilde{X_i}; 1 \leq i \leq p)$ recebe um sinal de entrada $(\widetilde{x_i})$ e envia este sinal para cada unidade da camada intermediária $(\widetilde{Z_j}; 1 \leq j \leq q)$
- Cada unidade intermediária $(\widetilde{Z_j})$ calcula seu sinal de entrada $(\widetilde{z\_in_j})$ e aplica a função de ativação para calcular seu sinal de saída $(\widetilde{z_j})$. Este sinal de saída é enviado para todas as unidades da camada de saída
- Cada unidade de saída $(\widetilde{Y_k}, 1 \leq k \leq u)$ calcula seu sinal de entrada $(\widetilde{y\_in_k})$ e aplica a função de ativação para computar seu sinal de saída $(\widetilde{y_k})$

Cálculo da recompensa:

- Cada unidade de saída calcula o valor da recompensa (r) com base no valor de saída $(\widetilde{y_k})$ e no valor desejado $(\widetilde{t_k})$
- A recompensa atual V_{num} do agente é calculado com base no valor de (r)

Fase de ajuste de pesos:

- Cada unidade de saída ajusta seu peso $(w_{jk}(novo))$
- Cada unidade intermediária ajusta seu peso $(v_{ij}(novo))$

$num = num + 1$

$\epsilon = V_{num} - V_{num-1}$

até $\epsilon \leq 0.001$

Saída: Saída da rede $(\widetilde{y_k})$

Capítulo 7

Resultados Numéricos

7.1 Escalonamento de tarefas em máquinas paralelas idênticas com parâmetros precisos

Na resolução do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros precisos, que foi descrito na Seção 3.1.1, foram utilizadas as seguintes abordagens:

1. Heurística construtiva, que foi descrita na Seção 3.1.2.1.
2. Heurística de melhoria, que foi descrita na Seção 3.1.2.2.
3. O *Simulated annealing*, que foi descrito na Seção 3.1.2.3.
4. Algoritmo genético, que foi descrito na Seção 3.1.2.4.
5. Algoritmo memético com a estratégia de busca local do tipo *first improving*, que foi descrito na Seção 3.1.2.5.

Nas avaliações dos métodos para o problema foram usadas 390 instâncias, extraídas de (Muller (1993)). As instâncias são caracterizadas pelas seguintes informações: número de máquinas, número de tarefas para serem escalonadas nessas máquinas e tempo de processamento de cada tarefa. O valor ótimo do mínimo *makespan* de cada instância é conhecido. Assim, podemos comparar os resultados ótimos com os resultados obtidos com as heurísticas.

O número de processadores, nestas instâncias, é 5, 10 ou 25. O número de tarefas é 10, 50, 100, 500 ou 1000. Os tempos de processamento das tarefas foram gerados aleatoriamente segundo uma distribuição uniforme dentro de três intervalos, [1, 100], [1, 1000] e [1, 10000].

A Tabela 7.1 apresenta uma comparação dos algoritmos desenvolvidos com outras abordagens para o problema. Nesta tabela, **Número de Ótimos** significa o total de ótimos alcançados pela heurística. **Porcentagem de Ótimos** significa a relação entre o número de instâncias para as quais foram obtidas soluções ótimas e o número total de instâncias.

Tab. 7.1: Valores ótimos obtidos pelos algoritmos.

Algoritmo	*Número de ótimos*	*Porcentagem de ótimos*
Heurística construtiva (LPT)	88	22,56%
Heurística de melhoria	226	58,12%
Simulated annealing	306	78,5%
Algoritmo genético	217	56,6%
Algoritmo memético (first improving)	326	83,59%

A Figura 7.1 apresenta uma representação visual para os resultados mostrados na Tabela 7.1. Podemos observar que o algoritmo memético possui uma porcentagem maior de casos para os quais foram obtidas soluções ótimas, em relação aos demais métodos.

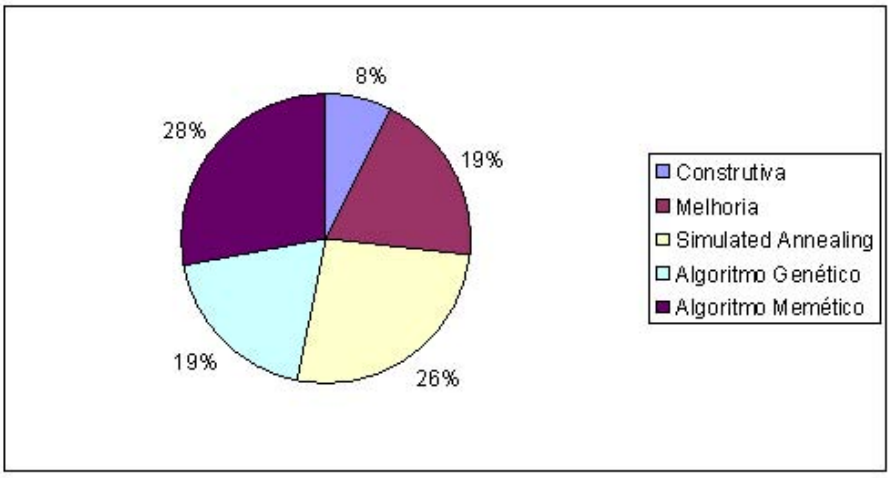

Fig. 7.1: Comparação entre os métodos com relação à porcentagem de ótimos.

A Tabela 7.2 apresenta um detalhamento dos resultados da simulação dos algoritmos genéticos e meméticos, especificando a porcentagem de valores ótimos encontrados. Os resultados estão divididos de acordo com o número de máquinas e o número de tarefas. Pode-se verificar, através da tabela, como o algoritmo memético obteve um melhor desempenho em relação ao genético.

Estes resultados foram publicados em (Bonfim et al. (2002b)).

Tab. 7.2: Número de ótimos encontrados em cada grupo.

Núm. de máquinas	*Núm. de tarefas*	*Porc. de ótimos do alg. gen.*	*Porc. de ótimos do alg. mem.*
5	10	63,3%	63,3%
5	50	60,0%	73,3%
5	100	66,7%	90,0%
5	500	70,0%	100%
5	1000	73,3%	100%
10	50	33,3%	63,3%
10	100	43,3%	83,3%
10	500	60,0%	100%
10	1000	63,3%	100%
25	50	70,0%	73,3%
25	100	26,7%	56,6%
25	500	36,7%	100%
25	1000	56,7%	93,3%

7.2 Escalonamento neuro-memético de tarefas em máquinas paralelas idênticas com parâmetros precisos

Na resolução do problema de escalonamento neuro-memético de tarefas em máquinas paralelas idênticas com parâmetros precisos, que foi descrito no Capítulo 5, foram utilizadas as seguintes abordagens:

1. Algoritmo memético coevoluindo com a rede neural *backpropagation*, que foi descrito no Algoritmo 13.

2. Algoritmo memético coevoluindo com a rede neural com aprendizado por reforço, que foi descrito no Algoritmo 14.

Nas avaliações dos métodos para o problema foram usadas as mesmas 390 instâncias, extraídas de (Muller (1993)), utilizadas no problema de escalonamento de tarefas em máquina paralelas idênticas com parâmetros precisos descrito na Seção 7.1.

Os parâmetros adotados na execução do escalonador neuro-memético são apresentados na Tabela 7.3.

Pela Tabela 7.4 podemos verificar o erro médio dos valores encontrados com as redes neurais *backpropagation* e com aprendizado por reforço. Este erro médio foi calculado através da comparação do valor ótimo obtido pelo escalonador neuro-memético com o valor ótimo fornecido na instância extraída de (Muller (1993)). Nesta tabela está especificada a porcentagem de instâncias, em relação ao número total, que alcançaram o valor ótimo. Pode-se notar que, com o uso de ambas as redes

Tab. 7.3: Parâmetros aplicados ao escalonador neuro-memético.

Parâmetro	*Valor*
Tamanho da população	26
Probabilidade de *crossover*	0.5
Probabilidade de mutação	0.06
Número de gerações	40
α	0.005
γ	0.9

neurais para o cálculo da função de *fitness*, foi encontrado um bom número de resultados ótimos ou próximos do valor ótimo.

Tab. 7.4: Poncentagem de valores ótimos e erro médio em relação ao valor ótimo.

Erro médio	*Rede neural backpropagation*	*Rede neural com aprendizado por reforço*
0% - 1%	88,7%	88,2%
1% - 3%	3,8%	4,9%
3% - 5%	3%	1,8%
5% - 7%	2%	0%
7% - 10%	0,8%	0%
10% - 20%	1,7%	5,10%

Pela Figura 7.2, podemos verificar o erro médio que os escalonadores neuro-memético *backpropagation* e com aprendizado por reforço apresentaram em relação ao valor ótimo referente a cada instância. Por exemplo, para 346 instâncias, o escalonador neuro-memético *backpropagation* obteve um valor de *makespan* com um erro médio entre 0% a 1% do valor do *makespan* ótimo. Para 14 instâncias, o algoritmo obteve um erro médio entre 1% a 3%. O escalonador neuro-memético com aprendizado por reforço obteve, por exemplo, para 344 instâncias, um valor de *makespan* com um erro médio entre 0% a 1% do valor do *makespan* ótimo. Para 19 instâncias, o algoritmo obteve um erro médio entre 1% a 3%. Podemos verificar, através desta figura, que os escalonadores obtiveram um bom número de escalonamentos ótimos ou próximo dos resultados ótimos.

Pelas Figuras 7.3 e 7.4, podemos verificar os valores da função de *fitness* obtidos pelas redes neurais e o valor de *fitness* ótimo de cada instância, respectivamente, utilizando a rede neural *backpropagation* e a rede neural com aprendizado por reforço.

Na Tabela 7.5 podemos observar os resultados obtidos pelo escalonador neuro-memético *backpropagation*, escalonador neuro-memético com aprendizado por reforço, o algoritmo memético, o algoritmo genético, o *Simulated Anneling*, a estratégia de melhoria *first improving* e a heurística construtiva LPT na resolução do problema de escalonamento de tarefas em máquinas paralelas idên-

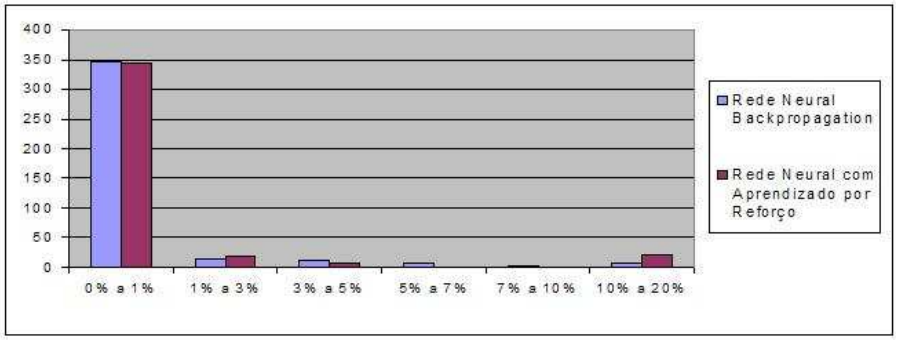

Fig. 7.2: Número de instâncias e erro médio usando *backpropagation* e aprendizado por reforço.

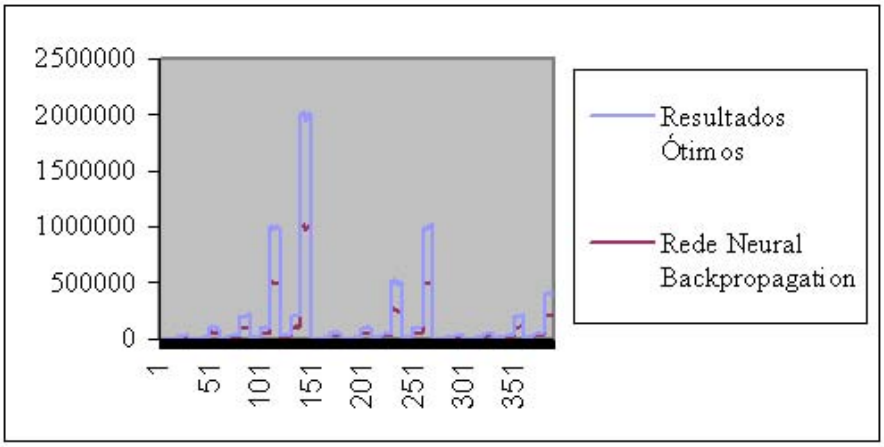

Fig. 7.3: Valor da função de *fitness* e número de instâncias usando *backpropagation*.

ticas com parâmetros precisos. Com esta tabela podemos comparar a eficiência de vários métodos aplicados ao problema e podemos verificar que a aplicação das redes neurais para encontrar a função de *fitness*, coevoluindo com o algoritmo memético, tem funcionado muito bem.

Pela Tabela 7.5 podemos verificar que a utilização das redes neurais, coevoluindo com o algoritmo memético, tem se comportado muito bem, oferecendo um número de instâncias com valor ótimo aproximado do algoritmo memético. Isto significa que, tanto na rede neural *backpropagation* quanto na rede neural com aprendizado por reforço, a rede tem demonstrado que está aprendendo um valor aproximado e bom para a função de *fitness*.

Estes resultados foram publicados em (Bonfim et al. (2002a)), (Bonfim and Yamakami (2002b)) e (Bonfim and Yamakami (2002a)).

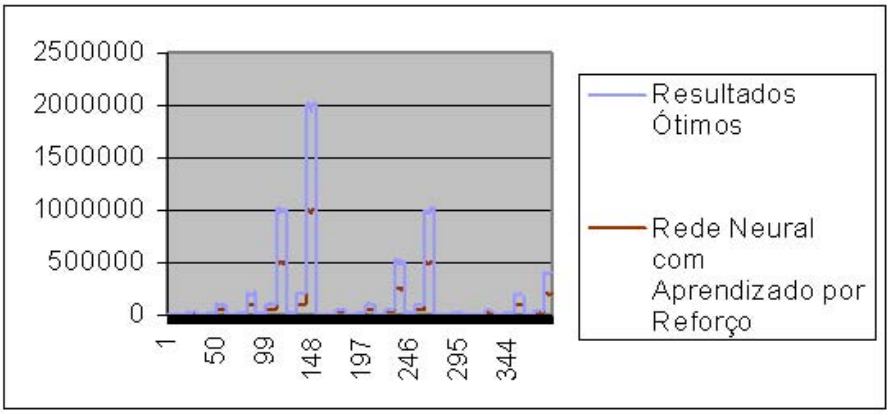

Fig. 7.4: Valor da função de *fitness* e número de instâncias usando aprendizado por reforço.

Tab. 7.5: Valores ótimos obtidos pelos algoritmos.

Algoritmo	*Número de ótimos*	*Porcentagem de ótimos*
Heurística construtiva (LPT)	88	22,56%
Heurística de melhoria	226	58,12%
Simulated annealing	306	78,5%
Algoritmo genético	217	56,6%
Algoritmo memético (first improving)	326	83,59%
Rede neural *backpropagation*	310	79,5%
Rede neural com aprendizado por reforço	330	84,6%

7.3 Escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas

O algoritmo memético com população estruturada foi aplicado para resolver o problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas de 390 instâncias, extraídas de (Muller (1993)), onde o mínimo *makespan* é conhecido. O problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas foi descrito na Seção 4.2, e a aplicação do algoritmo memético com população estruturada na resolução deste problema foi descrita na Seção 4.2.2.1. O número de processadores, nestas instâncias, é 5, 10 ou 25. O número de tarefas é 10, 50, 100, 500 ou 1000. Os tempos de processamento das tarefas foram gerados aleatoriamente segundo uma distribuição uniforme dentro de três intervalos, [1, 100], [1, 1000] e [1, 10000]. Como o tempo de processamento da tarefa é um parâmetro com incerteza, este valor gerado foi utilizado como valor modal $t_i{}^0$ e, os espalhamentos à esquerda ($t_i{}^0 - \underline{t_i}$) e à direita ($\overline{t_i} - t_i{}^0$), foram gerados segundo uma distribuição uniforme no intervalo [0, 15]. Com os espalhamentos à esquerda e à direita, os números *fuzzy* foram gerados conforme Equação 2.3.1.

Os parâmetros adotados na execução do algoritmo memético são apresentados na Tabela 7.6.

Tab. 7.6: Parâmetros aplicados ao algoritmo memético.

Parâmetro	*Valor*
Tamanho da população	26
Probabilidade de *crossover*	0.5
Probabilidade de mutação	0.06
Número de gerações	40

Pela Tabela 7.7 podemos verificar o erro médio dos valores encontrados pelo algoritmo memético com população estruturada na resolução do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros imprecisos. Este erro médio foi calculado através da comparação do valor ótimo obtido pelo escalonador memético com o valor ótimo fornecido na instância extraída de (Muller (1993)). Nesta tabela está especificada a porcentagem de instâncias, em relação ao número total, que alcançaram o valor ótimo.

Tab. 7.7: Poncentagem de valores ótimos e erro médio em relação ao valor ótimo.

Erro médio	*Algoritmo memético com população estruturada*
0% - 1%	57,4%
1% - 3%	7,95%
3% - 5%	2,56%
5% - 7%	16,41%
7% - 10%	12,82%
10% - 20%	2,82%

Pela Figura 7.5, podemos verificar o erro médio que o algoritmo memético com população estruturada apresentou em relação ao valor ótimo referente a cada instância. Por exemplo, para 224 instâncias, o algoritmo obteve um valor de *makespan* com um erro médio entre 0% a 1% do valor do *makespan* ótimo. Para 31 instâncias, o algoritmo obteve um erro médio entre 1% a 3%.

A seguir é apresentada a solução ótima e uma das soluções encontradas pelo algoritmo proposto para uma das 390 instâncias testadas. Esta instância exemplificada apresenta 5 máquinas e 10 tarefas. Os tempos de processamento das tarefas no caso clássico, com parâmetros precisos, são: 8513, 5858, 5107, 4887, 4096, 4071, 1656, 1315, 1162 e 621. Para o algoritmo proposto, com parâmetros fuzzy, os tempos de processamento *defuzzificados* são: 8513.99, 5862, 5113, 4890.49, 4102.99, 4076.49, 1656.49, 1319.98, 1169, 623.49. Os resultados encontrados pelos algoritmos estão apresentados na Tabela 7.8.

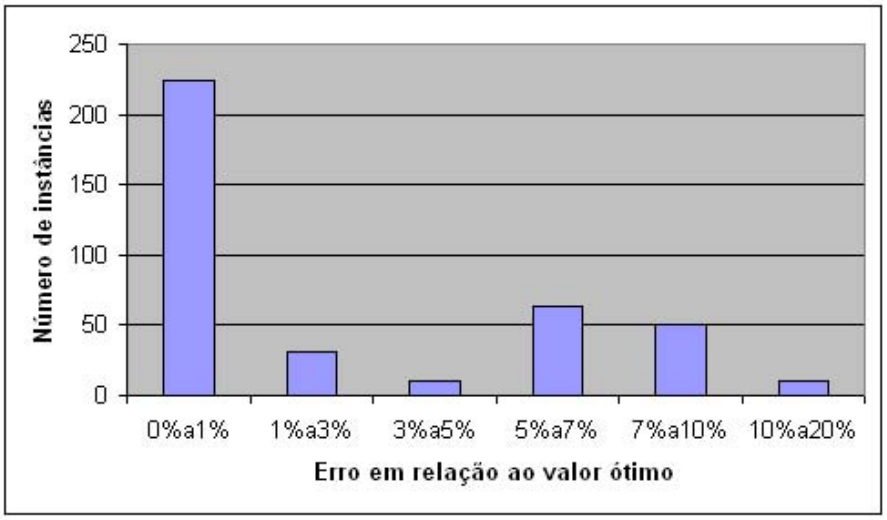

Fig. 7.5: Erro médio em relação ao valor ótimo.

	Alg. memético com par. precisos		Alg. memético com par. com incertezas	
Proc.	*Tarefas*	*Tempo*	*Tarefas*	*Tempo*
0	8513	**8513**	8513.99	**8513.99**
1	5858 1162	7020	5862 1169	7031
2	5107 1312 621	7040	5113 1319.98 623.49	7056.47
3	4887 1656	6543	4890.49 1656.49	6546.98
4	4096 4071	8167	4102.99 4076.49	8179.48

Tab. 7.8: Resultados do alg. memético para o problema com parâmetros precisos e do alg. memético com parâmetros com incertezas para uma instância com 5 máquinas e 10 tarefas.

Pela Tabela 7.8, o cromossomo da solução ótima foi [0 1 2 3 4 4 3 2 1 2] e o *makespan* obtido foi 8513. Pelo algoritmo proposto, o cromossomo da melhor solução foi [0 1 2 3 4 4 3 2 1 2] e o *makespan* obtido foi 8513.99.

Na execução de cada instância, foi verificado quantos indivíduos ficaram com seus valores até 90% do valor do *makespan* ótimo encontrado pelo algoritmo. A Figura 7.6 apresenta a quantidade de indivíduos com 90% de possibilidade de serem ótimos.

Pela Figura 7.6, podemos verificar que, em 237 instâncias, 25 indivíduos tiveram seus valores de *makespan* até 90% do valor do melhor indivíduo encontrado pelo algoritmo. Para 64 instâncias, esta margem foi obtida por 24 indivíduos. Para 46 instâncias, foi obtida por 23 indivíduos. Isto considerando um universo de 26 indivíduos.

Estes resultados foram publicados em (Bonfim and Yamakami (2004a)).

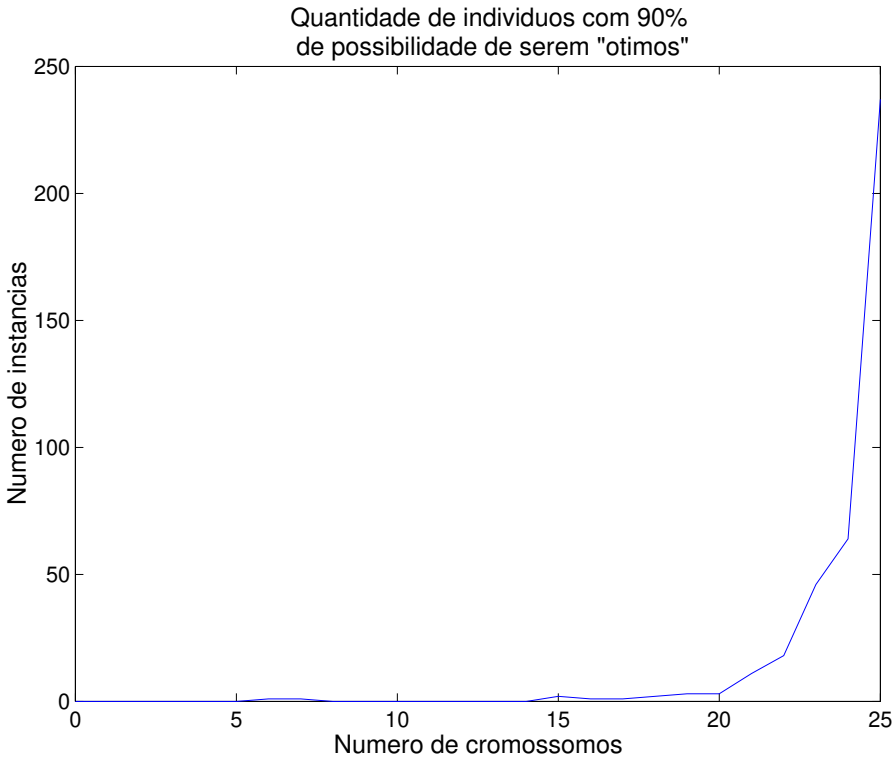

Fig. 7.6: Quantidade de indivíduos com 90% de possibilidade de serem ótimos.

7.4 Escalonamento neuro-memético de tarefas em máquinas paralelas idênticas com parâmetros com incertezas

O escalonador neuro-memético foi aplicado para resolver o problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas de 210 instâncias, extraídas de (Muller (1993)), onde o *makespan* com incertezas é conhecido através do problema detalhado na Seção 7.3.

O problema de escalonamento neuro-memético de tarefas em máquinas paralelas idênticas com parâmetros com incertezas foi descrito no Capítulo 6, e a aplicação do escalonador neuro-memético na resolução deste problema foi descrito nas Seções 6.1.1.1 e 6.1.1.2.

O número de processadores, nestas instâncias, é 5, 10 ou 25. O número de tarefas é 10, 50 ou 100. Os tempos de processamento das tarefas foram gerados aleatoriamente segundo uma distribuição uniforme dentro de três intervalos, [1, 100], [1, 1000] e [1, 10000]. Como o tempo de processamento da tarefa é um parâmetro *fuzzy*, este valor gerado foi utilizado como valor modal $t_i{}^0$ e, os espalhamentos à esquerda ($t_i{}^0 - \underline{t_i}$) e à direita ($\overline{t_i} - t_i{}^0$), foram gerados segundo uma distribuição uniforme no intervalo [0, 15]. Com os espalhamentos à esquerda e à direita, e os α-cortes, os números *fuzzy* foram gerados com intervalos *fuzzy*, conforme detalhado na Equação 2.3.1. O valor ótimo do mínimo

makespan fuzzy de cada instância é conhecido através do problema detalhado na Seção 7.3.

Os parâmetros aplicados na execução do escalonador neuro-memético são apresentados na Tabela 7.9.

Tab. 7.9: Parâmetros aplicados ao escalonador neuro-memético.

Parâmetro	*Valor*
Tamanho da população	26
Probabilidade de *crossover*	0.5
Probabilidade de mutação	0.06
Número de gerações	30
Número de neurônios na camada intermediária da rede neural	20
Número de neurônios na camada de entrada da rede neural	3
Número de neurônios na camada de saída da rede neural	1

Pela Tabela 7.10 podemos verificar o erro médio dos valores encontrados pelas redes neurais *backpropagation* e com aprendizado por reforço. Este erro médio foi calculado através da comparação do valor ótimo obtido pelo escalonador neuro-memético com o valor ótimo obtido pelo escalonador memético apresentado na Seção 7.3 e publicado em (Bonfim and Yamakami (2004a)), onde foram aplicados os conceitos de possibilidade para encontrar soluções de um escalonamento "ótimo" no problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas, e foi aplicado o algoritmo memético com população estruturada para evoluir boas formas de escalonamento.

Tab. 7.10: Poncentagem de valores ótimos e erro médio em relação ao valor ótimo.

Erro médio	*Rede neural backpropagation*	*Rede neural com aprendizado por reforço*
0% - 1%	72,82%	70,48%
1% - 3%	12,38%	14,29%
3% - 5%	1,9%	6,67%
5% - 7%	1,9%	1,9%
7% - 10%	1%	2,38%
10% - 20%	10%	4,28%

Pela Figura 7.7, podemos verificar o erro médio que o escalonador neuro-memético *backpropagation* apresentou em relação ao valor ótimo referente a cada instância. Por exemplo, para 150 instâncias, o escalonador obteve um valor de *makespan* com um erro médio entre 0% a 1% do valor do *makespan* ótimo. Para 28 instâncias, o algoritmo obteve um erro médio entre 1% a 3%.

Pela Figura 7.8, podemos verificar o erro médio que o escalonador neuro-memético com aprendizado por reforço apresentou em relação ao valor ótimo referente a cada instância. Por exemplo, para 148 instâncias, o escalonador obteve um valor de *makespan* com um erro médio entre 0% a 1% do valor do *makespan* ótimo. Para 30 instâncias, o algoritmo obteve um erro médio entre 1% a 3%. Podemos verificar, através das figuras, que os escalonadores obtiveram um bom número de escalonamentos ótimos ou próximo dos resultados ótimos.

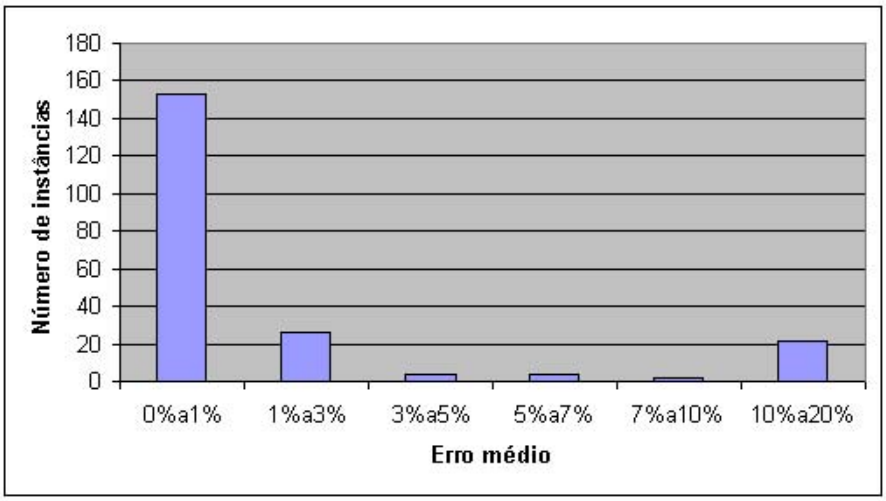

Fig. 7.7: Número de instâncias e erro médio usando *backpropagation*.

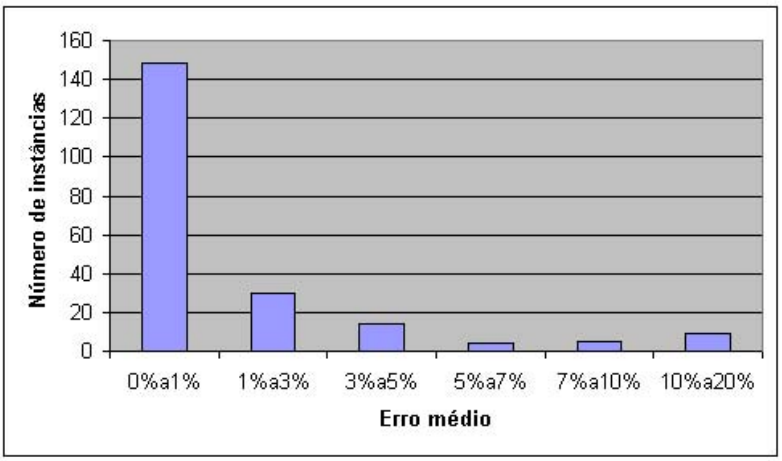

Fig. 7.8: Número de instâncias e erro médio usando aprendizado por reforço.

Pelas Figuras 7.9 e 7.10, podemos verificar os valores da função de *fitness* obtidos pelas redes neurais e o valor de *fitness* ótimo de cada instância, respectivamente, utilizando a rede neural *back-propagation* e a rede neural com aprendizado por reforço.

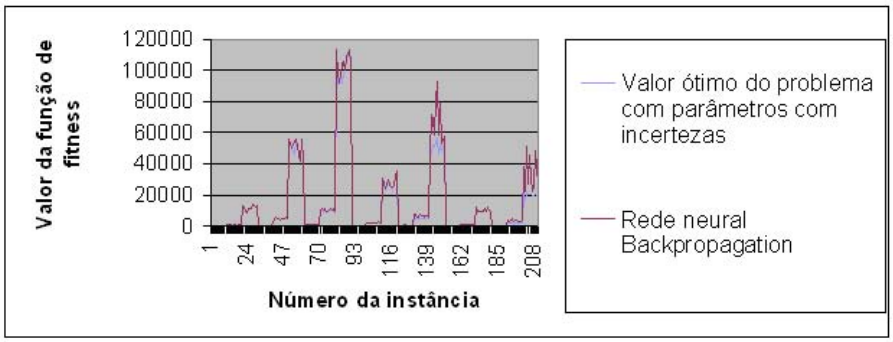

Fig. 7.9: Valor da função de *fitness* e número de instâncias usando *backpropagation*.

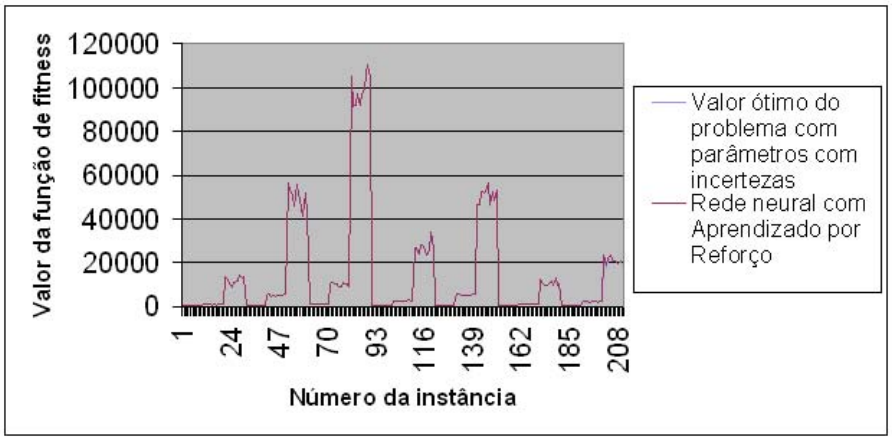

Fig. 7.10: Valor da função de *fitness* e número de instâncias usando aprendizado por reforço.

Podemos verificar, através das Figuras 7.9 e 7.10, que o escalonador com a rede neural *backpropagation* foi um pouco mais eficiente que o escalonador com a rede neural com aprendizado. Na rede neural *backpropagation*, em quase todos os pontos do gráfico, o valor obtido é igual ou muito próximo do valor ótimo de *fitness*.

Pela Tabela 7.11 podemos verificar os resultados obtidos utilizando a rede neural *backpropagation*, a rede neural com aprendizado por reforço e o algoritmo memético com população estruturada para a resolução do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas. Com esta tabela podemos comparar a eficência destes três métodos aplicados ao problema, e podemos verificar que a aplicação das redes neurais para encontrar o valor da função de *fitness*, coevoluindo com o algoritmo memético, funciona muito bem.

Através das simulações podemos observar que o uso de redes neurais, coevoluindo com o algoritmo memético, é bem eficiente, pois obteve um bom número de instâncias com valor ótimo aproximado do valor ótimo utilizado para comparação. Podemos concluir que a rede neural é capaz de

Tab. 7.11: Comparação entre métodos para o problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas.

Algoritmo	*Número de ótimos*	*Porcentagem de ótimos*
Rede neural *backpropagation*	153	72,86%
Rede neural com aprendizado por reforço	148	70,48%
Algoritmo memético com população estruturada	105	50,0%

aprender o valor da função de *fitness*.

Estes resultados foram publicados em (Bonfim and Yamakami (2004b)).

7.5 Escalonamento do problema *job shop* com parâmetros precisos

Na resolução do problema de escalonamento *job shop* com parâmetros precisos, que foi descrito na Seção 3.2, foi utilizado o algoritmo memético com população estruturada, que foi descrito na Seção 3.2.2.1 e sua aplicação ao problema foi descrita na Seção 3.2.2.2.

Nas avaliações do método para o problema foram usadas 5 instâncias, extraídas de (Muth and Thompson (1963), Lawrence (1984) e Michel and Hentenryck (1998)). As instâncias são caracterizadas pelas seguintes informações: número de tarefas, número de máquinas, e uma tabela contendo a seqüência de operações de cada tarefa, incluindo o número da máquina que irá processá-la e o tempo de processamento da operação. O valor ótimo do mínimo *makespan* de cada instância é conhecido. Assim, podemos comparar os resultados ótimos com os resultados obtidos pelo método heurístico.

A primeira instância trabalhada, extraída de (Muth and Thompson (1963)), possui 6 tarefas e 6 máquinas. A segunda, terceira e quarta instâncias trabalhadas, extraídas de (Lawrence (1984)), possuem 15 tarefas e 10 máquinas. A quinta instância trabalhada, extraída de (Lawrence (1984)), possui 20 tarefas e 10 máquinas.

Os parâmetros adotados na execução do algoritmo memético são apresentados na Tabela 7.12.

Tab. 7.12: Parâmetros aplicados ao algoritmo memético.

Parâmetro	*Valor*
Tamanho da população	26
Probabilidade de *crossover*	0.5
Probabilidade de mutação	0.06
Número de gerações	200

Pela Tabela 7.13 podemos verificar o erro médio dos valores encontrados pelo algoritmo memético com população estruturada na resolução do problema de escalonamento *job shop* com parâmetros

precisos. Este erro médio foi calculado através da comparação do valor ótimo obtido pelo escalonador memético com o valor ótimo extraído na literatura. Nesta tabela está especificada a porcentagem de instâncias, em relação ao número total, que alcançaram o valor ótimo.

Tab. 7.13: Poncentagem de valores ótimos e erro médio em relação ao valor ótimo.

Erro médio	*Algoritmo memético com população estruturada*
0% - 1%	60%
1% - 3%	40%

Pela Figura 7.11, podemos verificar o erro médio que o algoritmo memético com população estruturada apresentou em relação ao valor ótimo referente a cada instância. Por exemplo, para 3 instâncias, o algoritmo obteve um valor de *makespan* com um erro médio entre 0% a 1% do valor do *makespan* ótimo. Para 2 instâncias, o algoritmo obteve um erro médio entre 1% a 3%.

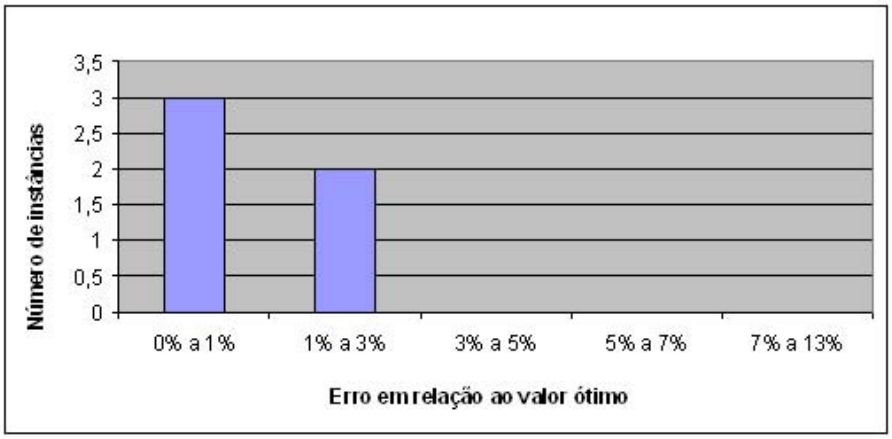

Fig. 7.11: Erro médio em relação ao valor ótimo usando algoritmo memético com população estruturada.

A seguir é apresentada uma das soluções encontradas pelo algoritmo memético proposto para uma das 5 instâncias testadas. Esta instância exemplificada apresenta 6 máquinas e 6 tarefas. Os tempos de processamento das tarefas no caso clássico, com parâmetros precisos, são apresentados na Tabela 7.14. Nestes parâmetros, para cada tarefa, é descrita a seqüência de operações com a máquina que irá processá-la e com o seu tempo de processamento.

O cromossomo que representa o indivíduo que obteve o escalonamento ótimo da instância exemplificada acima é apresentado pela matriz A.

Tab. 7.14: Tempos de processamento das operações no problema do *job shop* 6x6.

	(m,p)	*(m,p)*	*(m,p)*	*(m,p)*	*(m,p)*	*(m,p)*
Tarefa 0:	(2,1)	(0,3)	(1,6)	(3,7)	(5,3)	(4,6)
Tarefa 1:	(1,8)	(2,5)	(4,10)	(5,10)	(0,10)	(3,4)
Tarefa 2:	(2,5)	(3,4)	(5,8)	(0,9)	(1,1)	(4,7)
Tarefa 3:	(1,5)	(0,5)	(2,5)	(3,3)	(4,8)	(5,9)
Tarefa 4:	(2,9)	(1,3)	(4,5)	(5,4)	(0,3)	(3,1)
Tarefa 5:	(1,3)	(3,3)	(5,9)	(0,10)	(4,4)	(2,1)

$$\mathbf{A} = \begin{bmatrix} 0 & 3 & 2 & 5 & 1 & 4 \\ 1 & 3 & 5 & 4 & 0 & 2 \\ 2 & 0 & 1 & 4 & 3 & 5 \\ 2 & 5 & 3 & 0 & 1 & 4 \\ 1 & 4 & 3 & 2 & 5 & 0 \\ 2 & 5 & 1 & 0 & 4 & 3 \end{bmatrix}$$

Pela matriz A apresentada, o cromossomo da solução ótima obteve para a máquina 0 a seqüência para as tarefas que nela serão processadas: $0, 3, 2, 5, 1$ e 4.

A janela de tempo obtida pelo algoritmo memético com população estruturada, para a instância exemplificada está apresentada na matriz JT abaixo.

$$\mathbf{JT} = \begin{bmatrix} 6/9 & 13/18 & 18/27 & 28/38 & 38/48 & 48/51 \\ 0/8 & 8/13 & 13/16 & 22/25 & 25/31 & 31/32 \\ 0/5 & 5/6 & 8/13 & 13/22 & 22/27 & 49/50 \\ 5/9 & 16/19 & 27/30 & 31/38 & 48/52 & 52/53 \\ 13/23 & 25/30 & 30/38 & 38/45 & 45/49 & 49/55 \\ 9/17 & 19/28 & 28/38 & 38/41 & 41/45 & 45/54 \end{bmatrix}$$

Os valores apresentados na janela de tempo são os tempos de início e de término da execução das operações em cada máquina. Por exemplo, para a máquina 0, a tarefa 0 iniciará seu processamento no tempo 6 e finalizará no tempo 9. Na seqüência, ainda para a máquina 0, a tarefa 3 iniciará seu processamento no tempo 13 e finalizará no tempo 18. A janela de tempo é obtida com base no cromossomo que representa o indivíduo. De acordo com a matriz JT, o *makespan* obtido foi 55 pela máquina 4.

7.6 Escalonamento job shop neuro-memético com parâmetros precisos

Na resolução do problema de escalonamento neuro-memético *job shop* com parâmetros precisos, que foi descrito no Capítulo 5, foram utilizadas as seguintes abordagens:

1. Algoritmo memético coevoluindo com a rede neural *backpropagation*, que foi descrito no Algoritmo 13.

2. Algoritmo memético coevoluindo com a rede neural com aprendizado por reforço, que foi descrito no Algoritmo 14.

Nas avaliações do método para o problema foram usadas as mesmas 5 instâncias utilizadas na Seção 7.5, onde o problema foi solucionado apenas pelo algoritmo memético.

Os parâmetros adotados na execução do escalonador neuro-memético são apresentados na Tabela 7.3.

Pela Tabela 7.15 podemos verificar o erro médio dos valores encontrados com as redes neurais *backpropagation* e com aprendizado por reforço. Este erro médio foi calculado através da comparação do valor ótimo obtido pelo escalonador neuro-memético com o valor ótimo. Nesta tabela está especificada a porcentagem de instâncias, em relação ao número total, que alcançaram o valor ótimo. Na tabela também são apresentados os resultados obtidos pelo algoritmo memético para efeito de comparação. Pode-se notar que, com o uso de ambas as redes neurais para o cálculo da função de *fitness*, foi encontrado um bom número de resultados ótimos ou próximos do valor ótimo.

Tab. 7.15: Poncentagem de valores ótimos e erro médio em relação ao valor ótimo.

Erro médio	*RN backpropagation*	*RN com aprendizado por reforço*	*Alg. mem.*
0% - 1%	80%	80%	60%
1% - 3%	20%	20%	40%

Pela Figura 7.12 podemos verificar o erro médio que os escalonadores neuro-memético *backpropagation* e com aprendizado por reforço apresentaram em relação ao valor ótimo referente a cada instância. Por exemplo, para 4 instâncias, tanto o escalonador neuro-memético *backpropagation* como o escalonador neuro-memético com aprendizado por reforço obtiveram um valor de *makespan* com um erro médio entre 0% a 1% do valor do *makespan* ótimo. Para 1 instância, os algoritmos obtiveram um erro médio entre 1% a 3%. Podemos verificar, através da figura, que os escalonadores obtiveram um bom número de escalonamentos ótimos ou próximo dos resultados ótimos.

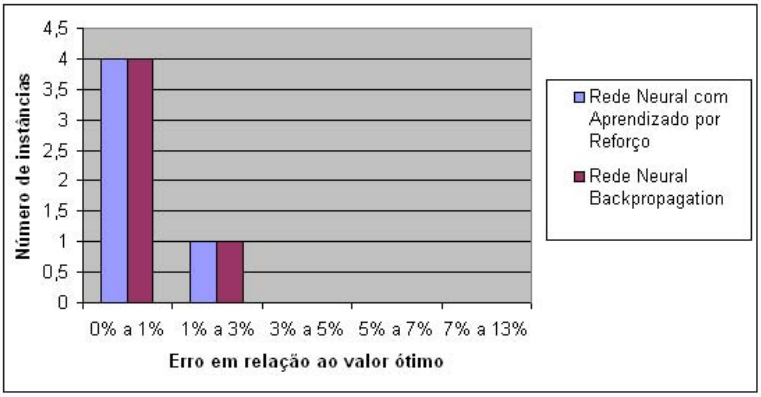

Fig. 7.12: Número de instâncias e erro médio usando *backpropagation* e aprendizado por reforço.

A seguir é apresentada uma das soluções encontradas pelo algoritmo neuro-memético utilizando a rede neural com aprendizado por reforço proposto para a instância com 6 máquinas e 6 tarefas. Os tempos de processamento das tarefas no caso clássico, com parâmetros precisos, são apresentados na Tabela 7.14.

O cromossomo que representa o indivíduo que obteve o escalonamento ótimo da instância exemplificada acima é apresentado pela matriz A abaixo.

$$\mathbf{A} = \begin{bmatrix} 0 & 3 & 2 & 5 & 1 & 4 \\ 1 & 3 & 5 & 4 & 0 & 2 \\ 2 & 0 & 1 & 4 & 3 & 5 \\ 2 & 5 & 3 & 0 & 1 & 4 \\ 1 & 4 & 3 & 5 & 2 & 0 \\ 2 & 5 & 1 & 4 & 0 & 3 \end{bmatrix}$$

Pela matriz A apresentada, o cromossomo da solução ótima obteve para a máquina 0 a seqüência de tarefas que nela serão processadas: $0, 3, 2, 5, 1$ e 4. Observa-se que o cromossomo obtido nesta resolução foi o mesmo obtido pelo algoritmo memético, apresentado na Seção 7.5. Isto comprova a eficiência da coevolução do algoritmo memético com a rede neural, pois com a coevolução foi possível obter o valor ótimo para o problema.

A janela de tempo obtida pelo algoritmo memético com população estruturada, para a instância exemplificada está apresentada na matriz JT.

$$\mathbf{JT} = \begin{bmatrix} 6/9 & 13/18 & 18/27 & 28/38 & 38/48 & 48/51 \\ 0/8 & 8/13 & 13/16 & 22/25 & 25/31 & 31/32 \\ 0/5 & 5/6 & 8/13 & 13/22 & 22/27 & 42/43 \\ 5/9 & 16/19 & 27/30 & 31/38 & 48/52 & 52/53 \\ 13/23 & 25/30 & 30/38 & 38/42 & 42/49 & 49/55 \\ 9/17 & 19/28 & 28/38 & 38/42 & 42/45 & 45/54 \end{bmatrix}$$

De acordo com a matriz JT, o *makespan* obtido foi 55 pela máquina 4. Na execução do algoritmo neuro-memético, utilizando a rede neural *backpropagation*, para a mesma instância apresentada acima, foram obtidos os mesmos resultados do algoritmo memético descritos na Seção 7.5.

Pelas Figuras 7.13 e 7.14 podemos verificar os valores da função de *fitness* obtidos pelas redes neurais e o valor de *fitness* ótimo de cada instância, respectivamente, utilizando a rede neural *backpropagation* e a rede neural com aprendizado por reforço.

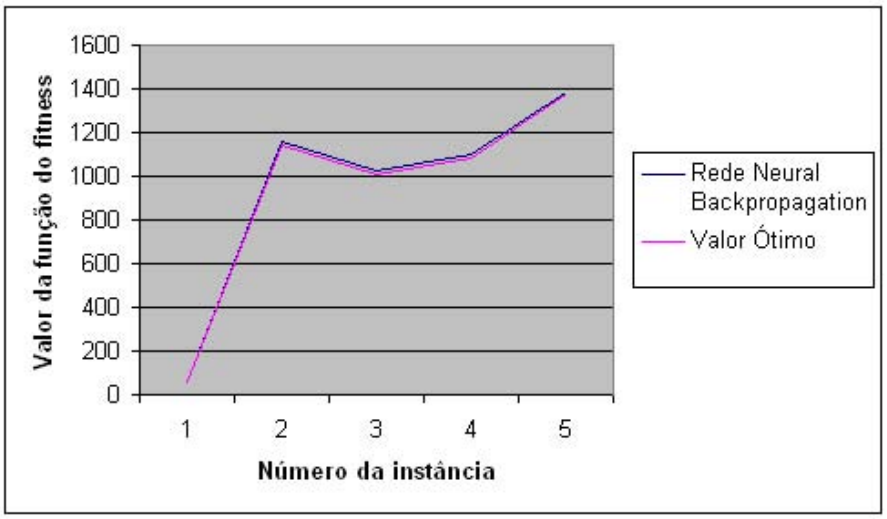

Fig. 7.13: Valor da função de *fitness* e número de instâncias usando *backpropagation*.

Podemos verificar que a utilização das redes neurais, coevoluindo com o algoritmo memético, tem se comportado muito bem, oferecendo um número de instâncias com valor ótimo aproximado do algoritmo memético. Isto significa que, tanto na rede neural *backpropagation* quanto na rede neural com aprendizado por reforço, a rede tem demonstrado que está aprendendo um valor aproximado e bom para a função de *fitness*.

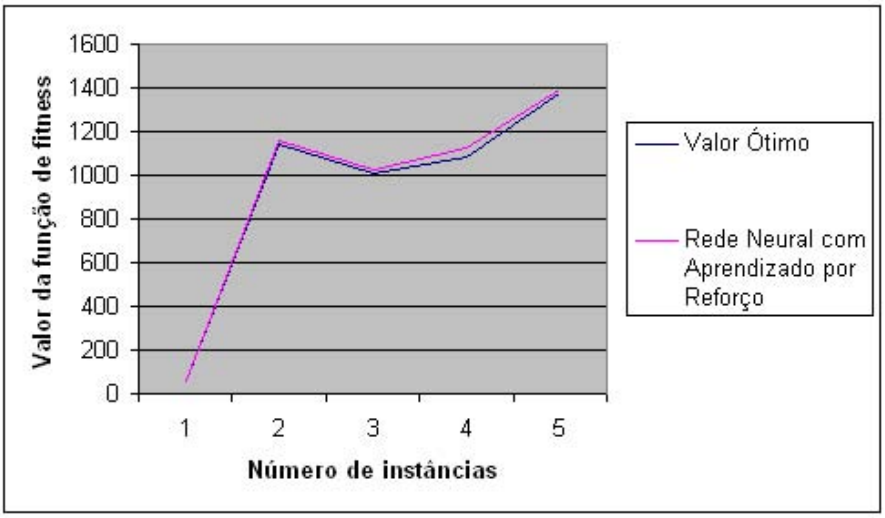

Fig. 7.14: Valor da função de *fitness* e número de instâncias usando aprendizado por reforço.

7.7 Escalonamento do problema *job shop* com parâmetros com incertezas

Na resolução do problema de escalonamento *job shop* com parâmetros com incertezas, que foi descrito na Seção 4.3, foram utilizados o algoritmo memético com população estruturada e os conceitos de possibilidade. A aplicação do algoritmo memético, com os conceitos de possibilidade, ao problema foi descrita na Seção 4.3.2.

Nas avaliações do método para o problema foram usadas as mesmas 5 instâncias utilizadas na Seção 7.5, onde o problema foi solucionado apenas pelo algoritmo memético.

Como o tempo de processamento das tarefas é um parâmetro com incertezas, foram gerados os números *fuzzy* com espalhamentos à esquerda e à direita, da mesma maneira como foi realizado na Seção 7.3, para o problema de máquinas paralelas idênticas.

A primeira instância trabalhada, extraída de (Muth and Thompson (1963)), possui 6 tarefas e 6 máquinas e será referenciada por $(6.6.mu)$. A segunda, terceira e quarta instâncias trabalhadas, extraídas de (Lawrence (1984)), possuem 15 tarefas e 10 máquinas e serão referenciadas por ($15.10la21$, $15.10.la24$ e $15.10l25$). A quinta instância trabalhada, extraída de Lawrence (1984), possui 20 tarefas e 10 máquinas e será referenciada por $(20.10la27)$.

Os parâmetros adotados na execução do algoritmo são apresentados na Tabela 7.12.

Pela Tabela 7.16 podemos verificar o erro médio dos valores encontrados pelo algoritmo memético com população estruturada na resolução do problema de escalonamento *job shop* com parâmetros imprecisos. Este erro médio foi calculado através da comparação do valor ótimo obtido pelo escalona-

dor memético com o valor ótimo extraído na literatura. Nesta tabela está especificada a porcentagem de instâncias, em relação ao número total, que alcançaram o valor ótimo.

Tab. 7.16: Poncentagem de valores ótimos e erro médio em relação ao valor ótimo.

Erro médio	*Algoritmo memético com população estruturada*
0% - 1%	80%
1% - 3%	20%

Pela Figura 7.15, podemos verificar o erro médio que o algoritmo memético com população estruturada apresentou em relação ao valor ótimo referente a cada instância. Por exemplo, para 4 instâncias, o algoritmo obteve um valor de *makespan* com um erro médio entre 0% a 1% do valor do *makespan* ótimo. Para 1 instância, o algoritmo obteve um erro médio entre 1% a 3%.

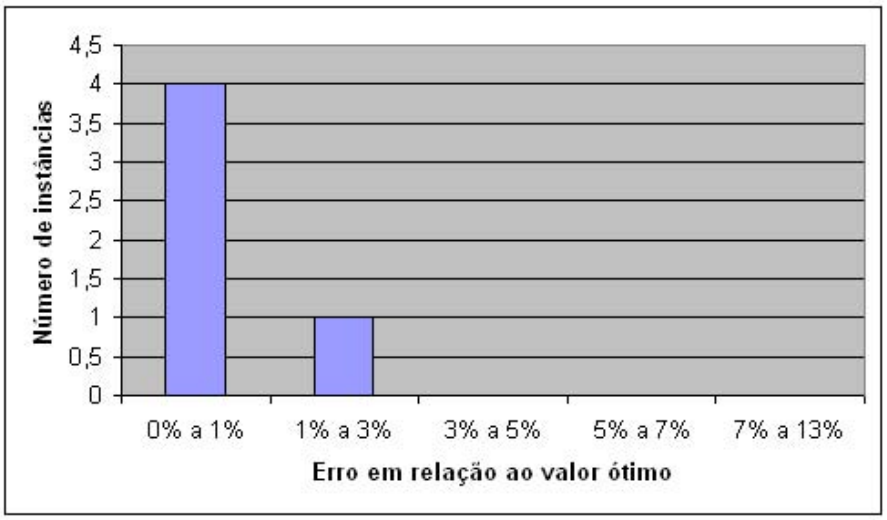

Fig. 7.15: Erro médio em relação ao valor ótimo usando o algoritmo memético com população estruturada.

A seguir é apresentada uma das soluções encontradas pelo algoritmo proposto para a instância que apresenta 6 máquinas e 6 tarefas. Os tempos de processamento das tarefas com parâmetros com incertezas, são apresentados na Tabela 7.17. Nestes parâmetros, para cada tarefa, é descrita a seqüência de operações com a máquina que irá processá-la e com o seu tempo de processamento, que é um número *fuzzy* triangular.

O cromossomo que representa o indivíduo que obteve o escalonamento ótimo da instância exemplificada é apresentado pela matriz A.

Tab. 7.17: Tempos de processamento das operações no problema do *job shop* 6x6 com parâmetros com incertezas.

	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$
Tarefa 0:	(2,[0.01,1,1.99])	(0,[2.77,3,3.23])	(1,[5.50,6,6.50])	(3,[6.93,7,7.07])	(5,[2.89,3,3.11])	(4,[5.07,6,6.93])
Tarefa 1:	(1,[7.90,8,8.22])	(2,[4.34,5,5.20])	(4,[9.82,10,10.69])	(5,[9.52,10,10.03])	(0,[9.99,10,10.06])	(3,[3.22,4,4.04])
Tarefa 2:	(2,[4.01,5,5.99])	(3,[3.83,4,4.17])	(5,[7.68,8,8.32])	(0,[8.64,9,9.36])	(1,[0.29,1,1.71])	(4,[6.08,7,7.92])
Tarefa 3:	(1,[4.03,5,5.13])	(0,[4.19,5,5.52])	(2,[4.50,5,5.55])	(3,[2,3,3.49])	(4,[7.22,8,8.5])	(5,[8.44,9,9.88])
Tarefa 4:	(2,[8.57,9,9.43])	(1,[2.35,3,3.65])	(4,[4.9,5,5.1])	(5,[3.92,4,4.08])	(0,[2.15,3,3.85])	(3,[0.73,1,1.27])
Tarefa 5:	(1,[2.22,3,3.33])	(3,[2.7,3,3.79])	(5,[8.61,9,9.08])	(0,[9.17,10,10.38])	(4,[3.75,4,4.14])	(2,[0.26,1,1.96])

$$\mathbf{A} = \begin{bmatrix} 3 & 0 & 2 & 5 & 1 & 4 \\ 1 & 3 & 5 & 4 & 0 & 2 \\ 2 & 0 & 1 & 4 & 3 & 5 \\ 2 & 5 & 3 & 0 & 1 & 4 \\ 1 & 4 & 3 & 2 & 5 & 0 \\ 2 & 5 & 1 & 4 & 0 & 3 \end{bmatrix}$$

Pela matriz A apresentada, o cromossomo da solução ótima obteve para a máquina 0 a seqüência para as tarefas que nela serão processadas: $3, 0, 2, 5, 1$ e 4. Observa-se que o cromossomo aqui obtido difere-se em alguns pontos dos cromossomos obtidos nos casos anteriores. Isto influenciará na formação da janela de tempo, o que pode provocar um acréscimo ou decréscimo no valor do *makespan* obtido. Neste caso, com este cromossomo, foi possível obter um valor bem próximo do valor ótimo.

A janela de tempo obtida pelo algoritmo memético com população estruturada, para a instância exemplificada está apresentada na matriz JT abaixo.

$$\widetilde{\mathbf{JT}} = \begin{bmatrix} 12.76/17.66 & 17.66/20.66 & 20.66/29.66 & 29.66/39.51 & 39.51/49.53 & 49.53/52.53 \\ 0/8.04 & 8.04/12.76 & 12.76/15.61 & 21.89/24.89 & 24.89/30.89 & 30.89/31.89 \\ 0/5.00 & 5.00/6.00 & 8.04/12.89 & 12.89/21.89 & 21.89/26.90 & 48.76/49.83 \\ 5.00/9.00 & 15.61/18.77 & 26.90/29.73 & 30.89/37.89 & 49.53/53.29 & 53.29/54.29 \\ 12.89/23.06 & 24.89/29.89 & 29.89/37.80 & 37.80/44.80 & 44.80/48.76 & 48.76/54.76 \\ 9.00/17.00 & 18.77/27.67 & 27.67/37.52 & 37.52/41.52 & 41.52/44.52 & 44.52/53.63 \end{bmatrix}$$

Os valores apresentados na janela de tempo são os tempos de início e de término da execução das operações em cada máquina. Por exemplo, para a máquina 0, a tarefa 3 iniciará seu processamento no tempo 12.76 e finalizará no tempo 17.66. Na seqüência, ainda para a máquina 0, a tarefa 0 iniciará seu processamento no tempo 17.66 e finalizará no tempo 20.66. A janela de tempo é obtida com base

no cromossomo que representa o indivíduo. Como os tempos de início e término da execução das operações em cada máquina, são números *fuzzy*, eles foram defuzzificados na formação da janela de tempo. De acordo com a matriz JT, o *makespan* obtido foi 54.76 pela máquina 4.

Na execução de cada instância, foram verificados quantos indivíduos ficaram com seus valores até 90% do valor do *makespan* ótimo encontrado pelo algoritmo. A Tabela 7.18 apresenta a quantidade de indivíduos com 90% de possibilidade de serem ótimos para cada instância testada.

Tab. 7.18: Quantidade de indivíduos com 90% de possibilidade de serem ótimos.

Instância	*Quantidade de indivíduos*
$6.6.mu$	13
$15.10la21$	20
$15.10la24$	18
$15.10la25$	22
$20.10la27$	24

Pela Tabela 7.18, podemos verificar que, por exemplo, para a instância 20.10la27, 24 indivíduos tiveram seus valores de *makespan* até 90% do valor do melhor indivíduo encontrado pelo algoritmo. Isto para uma população de 26 indivíduos.

7.8 Escalonamento job shop neuro-memético com parâmetros com incertezas

Na resolução do problema de escalonamentoalg neuro-memético *job shop* com parâmetros com incertezas, que foi descrito no Capítulo 6, foram utilizados o algoritmo memético com população estruturada, as redes neurais *backpropagation* e com aprendizado por reforço, e os conceitos de possibilidade. A aplicação do algoritmo neuro-memético com os conceitos de possibilidade ao problema foi descrita nas Seções 6.1.1.1 e 6.1.1.2.

Nas avaliações do método para o problema foram usadas as mesmas 5 instâncias utilizadas na Seção 7.5, onde o problema foi solucionado apenas pelo algoritmo memético.

O valor ótimo do mínimo *makespan* com incertezas de cada instância é conhecido através do problema detalhado na Seção 7.7. Como o tempo de processamento das tarefas é um parâmetro *fuzzy*, os números *fuzzy* foram gerados com intervalos de confiança, da mesma maneira como foi realizado na Seção 7.4, para o problema de máquinas paralelas idênticas.

As instâncias trabalhadas foram as mesmas utilizadas na Seção 7.7.

Os parâmetros aplicados na execução do escalonador neuro-memético são apresentados na Tabela 7.9.

Pela Tabela 7.19 podemos verificar o erro médio dos valores encontrados pelas redes neurais *backpropagation* e com aprendizado por reforço. Na tabela também são apresentados os resultados obtidos pelo algoritmo memético para efeito de comparação.

Este erro médio foi calculado através da comparação do valor ótimo obtido pelo escalonador neuro-memético com o valor ótimo obtido pelo escalonador memético apresentado na Seção 7.7, onde foram aplicados os conceitos de possibilidade para encontrar soluções de um escalonamento "ótimo" no problema de escalonamento *job shop* com parâmetros com incertezas, e foi aplicado o algoritmo memético com população estruturada para evoluir boas formas de escalonamento.

Tab. 7.19: Poncentagem de valores ótimos e erro médio em relação ao valor ótimo.

Erro médio	*RN backpropagation*	*RN com aprendizado por reforço*	*Alg. mem.*
0% - 1%	100%	80%	80%
1% - 3%	0%	20%	20%

Pela Figura 7.16, podemos verificar o erro médio que os escalonadores neuro-memético *backpropagation* e com aprendizado por reforço apresentaram em relação ao valor ótimo referente a cada instância. Por exemplo, para 5 instâncias, o escalonador neuro-memético *backpropagation* obteve um valor de *makespan* com um erro médio entre 0% a 1% do valor do *makespan* ótimo. O escalonador neuro-memético com aprendizado por reforço obteve, por exemplo, para 4 instâncias, um valor de *makespan* com um erro médio entre 0% a 1% do valor do *makespan* ótimo. Para 1 instância, este último algoritmo obteve um erro médio entre 1% a 3%. Podemos verificar, através da figura, que os escalonadores obtiveram um bom número de escalonamentos ótimos ou próximo dos resultados ótimos.

A seguir é apresentada uma das soluções encontradas pelo algoritmo neuro-memético utilizando a rede neural com aprendizado por reforço proposto para a instância com 6 máquinas e 6 tarefas. Os tempos de processamento das tarefas com parâmetros com incertezas, são apresentados na Tabela 7.20. Nestes parâmetros, para cada tarefa, é descrita a seqüência de operações com a máquina que irá processá-la e com o seu tempo de processamento, que é um número *fuzzy* triangular.

O cromossomo que representa o indivíduo que obteve o escalonamento ótimo da instância exemplificada é apresentado pela matriz A.

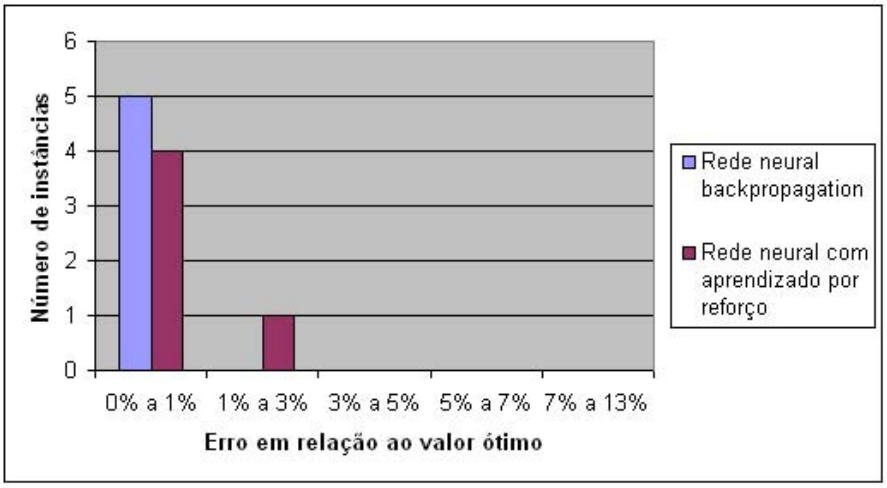

Fig. 7.16: Número de instâncias e erro médio usando *backpropagation* e aprendizado por reforço.

Tab. 7.20: Tempos de processamento das operações no problema do *job shop* 6x6 com parâmetros com incertezas.

	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$
Tarefa 0:	(2,[0,1,2])	(0,[2,3,4])	(1,[5,6,7])	(3,[6,7,8])	(5,[2,3,4])	(4,[5,6,7])
Tarefa 1:	(1,[7,8,9])	(2,[4,5,6])	(4,[9,10,11])	(5,[9,10,11])	(0,[9,10,11])	(3,[3,4,5])
Tarefa 2:	(2,[4,5,6])	(3,[3,4,5])	(5,[7,8,9])	(0,[8,9,10])	(1,[0,1,2])	(4,[6,7,8])
Tarefa 3:	(1,[4,5,6])	(0,[4,5,6])	(2,[4,5,6])	(3,[2,3,4])	(4,[7,8,9])	(5,[8,9,10])
Tarefa 4:	(2,[8,9,10])	(1,[2,3,4])	(4,[4,5,6])	(5,[3,4,5])	(0,[2,3,4])	(3,[0,1,2])
Tarefa 5:	(1,[2,3,4])	(3,[2,3,4])	(5,[8,9,10])	(0,[9,10,11])	(4,[3,4,5])	(2,[0,1,2])

$$\mathbf{A} = \begin{bmatrix} 0 & 3 & 2 & 5 & 1 & 4 \\ 1 & 5 & 3 & 0 & 4 & 2 \\ 0 & 2 & 1 & 4 & 3 & 5 \\ 2 & 5 & 3 & 0 & 1 & 4 \\ 1 & 4 & 3 & 2 & 5 & 0 \\ 2 & 5 & 1 & 4 & 0 & 3 \end{bmatrix}$$

Pela matriz A apresentada, o cromossomo da solução ótima obteve para a máquina 0 a seqüência para as tarefas que nela serão processadas: $0, 3, 2, 5, 1$ e 4.

A janela de tempo obtida pelo algoritmo memético com população estruturada, para a instância exemplificada está apresentada na matriz JT.

$$\widetilde{\mathbf{JT}} = \begin{bmatrix} 1.00/4.00 & 16.00/21.00 & 21.00/30.00 & 30.00/40.00 & 40.00/50.00 & 50.00/53.00 \\ 0.00/8.00 & 8.00/11.00 & 11.00/16.00 & 16.00/22.00 & 22.00/25.00 & 30.00/31.00 \\ 0.00/1.00 & 1.00/6.00 & 8.00/13.00 & 13.00/22.00 & 22.00/27.00 & 49.00/50.00 \\ 6.00/10.00 & 11.00/14.00 & 27.00/30.00 & 30.00/37.00 & 50.00/54.00 & 54.00/55.00 \\ 13.00/23.00 & 25.00/30.00 & 30.00/38.00 & 38.00/45.00 & 45.00/49.00 & 49.00/55.00 \\ 10.00/18.00 & 28.00/27.00 & 27.00/37.00 & 37.00/41.00 & 41.00/44.00 & 44.00/53.00 \end{bmatrix}$$

De acordo com a matriz JT, o *makespan* obtido foi 55.00 pelas máquinas 3 e 4.

Pelas Figuras 7.17 e 7.18 podemos verificar os valores da função de *fitness* obtidos pelas redes neurais e o valor de *fitness* ótimo de cada instância, respectivamente, utilizando a rede neural *back-propagation* e a rede neural com aprendizado por reforço.

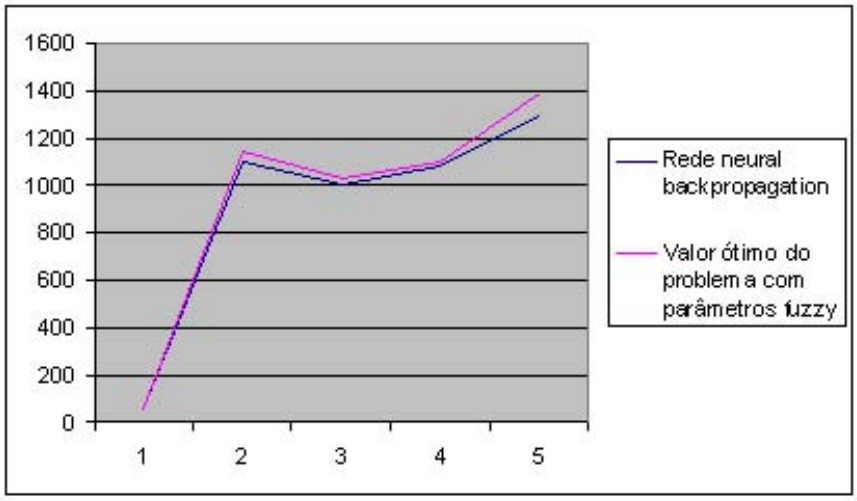

Fig. 7.17: Valor da função de *fitness* e número de instâncias usando *backpropagation*.

Podemos verificar, através das Figuras 7.17 e 7.18, que o escalonador com rede neural com aprendizado por reforço foi um pouco mais eficiente que o escalonador com a rede neural *backpropagation*.

Na execução de cada instância, foram verificados quantos indivíduos ficaram com seus valores até 90% do valor do *makespan* ótimo encontrado pelo algoritmo. A Tabela 7.21 apresenta a quantidade de indivíduos com 90% de possibilidade de serem ótimos para cada instância testada, considerando um universo de 26 indivíduos.

Pela Tabela 7.21, podemos verificar que, por exemplo, para a instância 20.10la27, 4 indivíduos tiveram seus valores de *makespan* até 90% do valor do melhor indivíduo encontrado pelo algoritmo neuro-memético utilizando a rede neural *backpropagation* e 10 indivíduos tiveram seus valores de

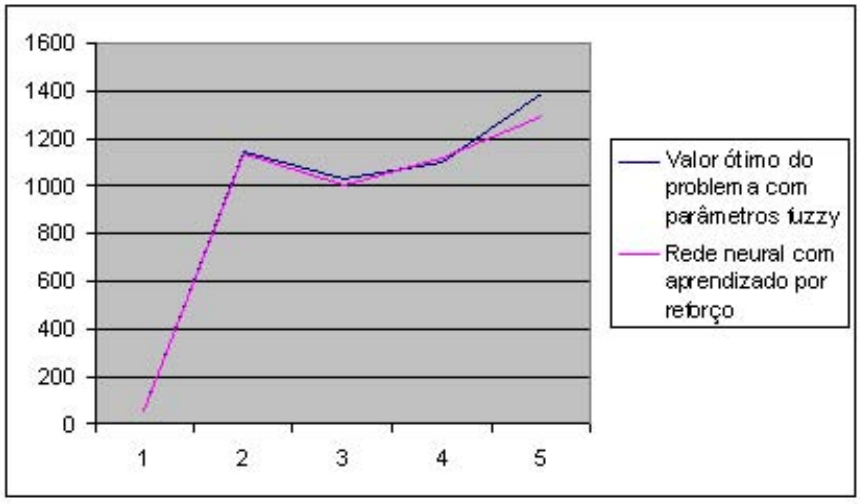

Fig. 7.18: Valor da função de *fitness* e número de instâncias usando aprendizado por reforço.

Tab. 7.21: Quantidade de indivíduos com 90% de possibilidade de serem ótimos.

Instância	*Num. de ind. na rede backpropagation*	*Num. de ind. na rede com ap. por reforço*
6.6.mu	2	6
15.10la21	7	8
15.10la24	2	6
15.10la25	3	8
20.10la27	4	10

makespan até 90% do valor do melhor indivíduo encontrado pelo algoritmo neuro-memético utilizando a rede neural com aprendizado por reforço.

Capítulo 8

Conclusão Geral

Neste trabalho foram abordados dois tipos problemas de escalonamento de tarefas considerando-os primeiramente com parâmetros precisos, e depois com parâmetros com incertezas.

O primeiro problema abordado foi o de escalonamento de tarefas em máquinas paralelas idênticas e o objetivo foi encontrar o escalonamento que minimizasse o *makespan*, que é o tempo total de processamento na máquina mais ocupada, que é a última a terminar a execução das tarefas a ela alocadas.

No problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas, a incerteza foi aplicada aos tempos de processamento das tarefas, na forma de números *fuzzy*. Cada tempo de processamento das tarefas foi representado por um número *fuzzy* triangular.

O segundo problema abordado foi o *job shop*. O objetivo do problema foi encontrar uma forma de escalonar as tarefas pelas máquinas que minimize o *makespan*, que é o tempo total gasto entre o início da primeira operação e o término da última operação.

No problema de escalonamento *job shop* com parâmetros com incertezas, foi considerado que os tempos de processamento das operações e os tempos possíveis para o início do processamento das operações não eram precisamente conhecidos, e eram representados por números triangulares *fuzzy*.

Foram apresentadas duas abordagens para a resolução de cada um dos problemas com parâmetros precisos. A primeira abordagem foi utilizando o algoritmo memético e, na segunda abordagem, utilizando o algoritmo neuro-memético.

Para a resolução do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros precisos, na primeira abordagem, foi aplicado o algoritmo memético para evoluir boas formas de escalonamento. O algoritmo memético aplicado neste caso foi o algoritmo genético acoplado com uma busca local do tipo *first improving*. Na segunda abordagem, foi aplicada a coevolução de redes neurais com o algoritmo memético. Neste caso, foi aplicado o algoritmo memético para encontrar soluções para o problema, e a rede neural foi aplicada para encontrar a função de *fitness* destas

soluções. O algoritmo memético utilizado foi o algoritmo genético acoplado com uma busca local do tipo *first improving*. As redes neurais aplicadas foram a *backpropagation* e a com aprendizado por reforço.

Na resolução do problema de escalonamento *job shop* com parâmetros precisos, na primeira abordagem, foi utilizado o algoritmo memético com população estruturada para encontrar um escalonamento que minimizasse o *makespan*. Na segunda abordagem, foi aplicada a coevolução de redes neurais com o algoritmo memético. Neste caso, foi aplicado o algoritmo memético com população estruturada para encontrar soluções para o problema, e a rede neural foi aplicada para encontrar a função de *fitness* destas soluções. As redes neurais aplicadas foram a *backpropagation* e a com aprendizado por reforço.

Para a resolução de cada um dos problemas com parâmetros com incertezas também foram apresentadas duas abordagens. A primeira abordagem utilizando o algoritmo memético e, na segunda abordagem, utilizando o algoritmo neuro-memético.

Tanto para a resolução do problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros com incertezas quanto para a resolução do problema de escalonamento *job shop* com parâmetros com incertezas, na primeira abordagem, foi aplicado o algoritmo memético com população estruturada para evoluir boas formas de escalonamento e foi utilizado o conceito de otimalidade possível para medir a possibilidade de um determinado escalonamento ser ótimo. Na segunda abordagem, foi aplicada a coevolução de redes neurais com o algoritmo memético com população estruturada. Neste caso, foi aplicado o algoritmo memético para encontrar soluções para o problema, a rede neural foi aplicada para encontrar a função de *fitness* destas soluções, e o conceito de possibilidade foi aplicado para avaliar as melhores soluções. As redes neurais aplicadas foram a *backpropagation* e a com aprendizado por reforço.

Neste trabalho foi apresentado o uso dos conceitos de otimalidade possível para analisar a possibilidade de um escalonamento ser ótimo, pois um problema de escalonamento *fuzzy* é um problema de satisfação de restrições *fuzzy*, onde as restrições são mais ou menos relaxáveis ou sujeitas à preferências. Como as restrições do problema não são precisamente conhecidas, torna-se difícil estabelecer qual é o escalonamento ótimo das tarefas e, neste caso, a noção de ótimo também torna-se imprecisa, e passamos a avaliar o grau de otimalidade de um escalonamento (o quanto determinado escalonamento é **ótimo**) através de um número *fuzzy* no intervalo [0,1]. Este grau de otimalidade é importante e pode ser utilizado em situações onde se tem vários escalonamentos e se quer definir qual é o melhor entre eles, segundo o critério do decisor.

Neste trabalho também foram aplicadas redes neurais coevoluindo com o algoritmo memético para evoluir bons escalonamentos e para determinar a função de *fitness* das soluções do problema. O uso de redes neurais deve-se ao fato de que nem sempre se conhece a função de *fitness* de um

problema e, desta forma, pretendeu-se apresentar que, com o uso de redes neurais, coevoluindo com o algoritmo memético, foi possível estimar a função de *fitness* de um problema.

Através de simulações e comparações com os valores ótimos, foi verificado que as abordagens propostas se mostraram bem eficientes. Com este trabalho foi possível mostrar que a rede neural trabalha muito bem na determinação do valor da função de *fitness* de um indivíduo. Através dos resultados obtidos, foi possível perceber que a rede neural age muito bem na coevolução do algoritmo memético, se comparado com os resultados obtidos pelo algoritmo memético sem a rede neural. Os resultados das simulações mostraram que as duas redes neurais implementadas aprenderam bem e encontraram bons valores para a função de *fitness*.

Foi verificado que, com o uso do algoritmo memético e de redes neurais foi possível resolver o problema de minimização do *makespan* nos problemas de escalonamento de tarefas em máquinas paralelas idênticas e do *job shop* com parâmetros precisos. Foi verificado também que, com o uso do algoritmo memético, de redes neurais e dos conceitos de possibilidade foi possível resolver o problema de minimização do *makespan* nos problemas de escalonamento de tarefas em máquinas paralelas idênticas e do *job shop* com parâmetros com incertezas.

As contribuições deste trabalho estão na utilização do algoritmo memético com população estruturada na resolução dos problemas, a utilização da abordagem neuro-memética e a utilização do conceito de possibilidade na análise da otimalidade de um escalonamento. Outra contribuição está na forma de representação do indivíduo (cromossomo) no caso do problema de escalonamento do *job shop* e a implementação dos operadores genéticos.

Como trabalhos futuros pretende-se sugerir novas operações de *crossover* e de mutação. Pretende-se propor a utilização de outras redes neurais e com mudança de parâmetros como, por exemplo, no número de camadas. Pretende-se propor outras formas de algoritmos meméticos. Para casos com parâmetros com incertezas, pretende-se fazer um estudo da robustez das melhores soluções quanto às pertubações. Sugere-se também como trabalho futuro outras abordagens como a utilização de algoritmos meméticos com busca tabu.

Referências Bibliográficas

E. J. Anderson. Theory and methodology: Mechanisms for local search. *European Journal of Operational Research*, 88:139–151, 1996.

A. G. Barto and R. S. Sutton. *Reinforcement Learning - An Introduction*. A Bradford Book, The MIT Press, 1998.

R. E. Bellmann and L. A. Zadeh. Decision-making in a fuzzy environment. *Management Sci.*, 17: 141–164, 1970.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

J. Blazewicz, G. Finke, R. Haupt, and G. Schmidt. New trends in machine scheduling. *European Journal of Operational Research*, 37(3):300–317, 1988.

J. Blazewicz, W. Domschke, and E. Pesch. The job shop scheduling problem: Conventional and new solution techniques. *European Journal of Operational Research*, 93(3):1–33, 1996.

J. D. Blocher and S. Chand. Scheduling of parallel processors: A posterior bound on lpt sequencing and a two-step algorithm. *Naval Research Logistics*, 38:273–287, 1991.

T. R. Bonfim and A. Yamakami. Rede neural aplicada na co-evolução de algoritmos meméticos para resolução do problema de minimização do makespan no escalonamento de máquinas paralelas idênticas. In *VII Brazilian Symposium on Neural Networks (SBRN'02)*, pages 137–145, Pernambuco - Brazil, 2002a. Sociedade Brasileira de Computação.

T. R. Bonfim and A. Yamakami. Neural network applied to the coevolution of the memetic algorithm for solving the makespan minimization problem in parallel machine scheduling. In *IEEE papers - VII Brazilian Symposium on Neural Networks (SBRN'02)*, Pernambuco - Brazil, 2002b. Sociedade Brasileira de Computação.

T. R. Bonfim and A. Yamakami. Escalonamento de tarefas em máquinas paralelas idênticas com parâmetros fuzzy. In *Anais do XVI Congresso Brasileiro de Automática*, Gramado, Brasil, 2004a.

T. R. Bonfim and A. Yamakami. Escalonador neuro-memético de tarefas em máquinas paralelas idênticas com parâmetros fuzzy. In *Anais do XXXVI Simpósio Brasileiro de Pesquisa Operacional (SBPO)*, São João Del Rei, Brasil, 2004b.

T. R. Bonfim, A. Yamakami, and G. C. Cardoso. Memetic-neural scheduler of jobs in identical parallel machines. In Dynamic Publishers, editor, *Second International Workshop on Intelligent Systems Design and Applications*, pages 125–130, Atlanta, U.S.A., 2002a. Computational Intelligence and Applications.

T. R. Bonfim, E. Yoshimoto, and C. L. Filho. Aplicação de algoritmos genético e memético no escalonamento de tarefas em máquinas paralelas idêntica. In *Anais do XIV Congresso Brasileiro de Automática*, pages 1212–1217, Natal, Brasil, 2002b.

L. A. P. Cantão. *Programação Não-Linear com Parâmetros Fuzzy: Teoria e Algoritmos*. PhD thesis, Universidade Estadual de Campinas, 2003.

S. Chanas and A. Kasperski. Possible and necessary optimality of solutions in the single machine scheduling problem with fuzzy parameters. *Fuzzy Sets and Systems*, 142:359–371, 2004.

T. C. E. Cheng and C. C. S. Sin. A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3):271–292, 1990.

D. Dubois and H. Prade. *Fuzzy Sets and Systems: Theory and Applications*. Academic Press, 1980.

D. Dubois, H. Fargier, and H. Prade. Fuzzy constraints in job-shop scheduling. *Journal of Intelligent Manufacturing*, 6(4):215–234, 1995.

R. W. Eglese. Simulated anneling: A tool for operational research. *European Journal of Operational Research*, 46:271–281, 1990.

H. Fisher and G. L. Thompson. *Probabilistic learning combinations of local job-shop scheduling rules*. Prentice-Hall, 1963.

R. Hecht-Nielsen. Theory of the backpropagation neural network. In *International Joint Conference on Neural Networks*, pages 593–605, Washington, 1989.

J. C. Ho and J. S. Wong. Makespan minimization for m parallel identical processors. *Naval Research Logistics*, 42:935–948, 1995.

J. H. Holland. Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput*, 2:88–105, 1973.

J. H. Backstone Jr. and D. T. Phillips. An improved heuristic for minimizing makespan among m identical parallel processors. *Computers and Industrial Engineering*, 5(4):279–287, 1981.

A. Kaufmann and M. M. Gupta. *Introduction to Fuzzy Arithmetic: Theory and Applications*. Van Nostrand Reinhold, 1991.

S. Lawrence. *Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques*. PhD thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1984.

J.K. Lenstra, K. A. H. G. Rinnooy, and P. Brucker. Complexity machine scheduling problems. *Ann. Discrete Math.*, 1:343–362, 1977.

L. Michel and P. V. Hentenryck. Job-shop scheduling in localizer. Technical Report Technical Report CS-98-03, Brown University, 1998.

P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P Report 826, Caltech Concurrent Computation Program, 1989.

P. Moscato and R. Berretta. *The number partitioning problem: An open challenge for Evolutionary Computation*, chapter 17. McGraw-Hill, 1999.

P. Moscato and M. Normam. A memetic approach for the travelling salesman problem: implementation of a computational ecology for combinatorial optimization on message-passing systems. *Proc. Intl. Conf. on Parallel Computing and Transportation Applications*, 1992.

F. M. Muller. *Algoritmos heurísticos e exatos para resolução do problema de sequenciamento em processadores paralelos*. PhD thesis, Universidade Estadual de Campinas, 1993.

J. F. Muth and G. L. Thompson. *Industrial Scheduling*. Prentice Hall, 1963.

R. A. Nichols, R. L. Bulfin, and R. G. Parker. An interactive procedure for minimizing makespan on parallel processors. *Int. J. Prod. Res.*, pages 77–81, 1978.

M. Sugeno. An introductory survey of fuzzy control. *Inf. Science*, 36:59–83, 1985.

E. L. Thorndike. *Animal Intelligence*. Hafner, Darien, CT, 1911.

R. J. M. Vaessens, E. H. L. Aarts, and J. K. Lenstra. Job-shop scheduling by local search. *INFORMS Journal on Computing*, 8:302–317, 1996.

L. Zadeh. The concept of a linguistic variable and its application to approximate reasoning. part 1. *Information Sciences*, 8:199–249, 1975a.

L. Zadeh. The concept of a linguistic variable and its application to approximate reasoning. part 2. *Information Sciences*, 8:301–357, 1975b.

L. Zadeh. The concept of a linguistic variable and its application to approximate reasoning. part 3. *Information Sciences*, 9:43–80, 1975c.

L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

Apêndice A

Apêndices

Neste apêndice será apresentada, para um exemplo de 3 tarefas e 2 máquinas, a maneira de codificação do cromossomo na aplicação do algoritmo memético com população estruturada na resolução do problema de escalonamento *job shop* com parâmetros precisos, descrito na Seção 3.2.2.2. Neste apêndice também será apresentada a maneira de criar uma matriz de janela de tempo através da codificação do cromossomo.

A Tabela A.1 apresenta um *benchmark* exemplo para um problema de escalonamento *job shop*, com parâmetros precisos, de 3 tarefas e 2 máquinas ($m = 2$ e $n = 3$).

Tab. A.1: *Benchmark* do problema do *job shop* 3x2.

	(m,p)	*(m,p)*
Tarefa 0:	(0,1)	(1,1)
Tarefa 1:	(0,2)	(1,1)
Tarefa 2:	(1,1)	(0,2)

Segundo o *benchmark* acima, a tarefa 0 deve ser executada primeiramente pela máquina 0, com tempo de processamento de uma unidade (operação 1), e depois pela máquina 2, com tempo de processamento de 1 unidade (operação 2).

Os cromossomos são codificados de duas maneiras. A primeira codificação é feita através de um vetor (V), de nxm colunas, onde n é o número de tarefas e m é o número de máquinas. Este vetor é gerado aleatoriamente com os números das tarefas. Sua factibilidade é garantida, pois o número de operações por tarefa não pode ultrapassar o número de máquinas. Para o *benchmark* exemplo, cada número de tarefa tem que aparecer 2 vezes no vetor (V), pois o problema tem 2 máquinas.

Supondo que a seqüência gerada seja representada pelo vetor V abaixo.

$$\mathbf{V} = \begin{pmatrix} 0 & 1 & 1 & 2 & 0 & 2 \end{pmatrix}$$

A segunda forma de codificação do cromossomo é uma matriz A_{mxn}, de m linhas e n colunas, onde m é o número de máquinas e n é o número de tarefas. Esta matriz é montada através do vetor V e no *benchmark*, da seguinte maneira:

No vetor V, a primeira operação a ser executada é a tarefa 0. Olhando no *benchmark*, a primeira operação da tarefa 0 deve ser executada pela máquina 0 e, com essa informação, marcamos a primeira coluna na linha referente à máquina 0 (primeira linha).

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & & \\ & & \end{bmatrix}$$

No vetor V, a segunda operação a ser executada é a tarefa 1. Olhando no *benchmark*, a primeira operação da tarefa 1 deve ser executada pela máquina 0 e, com essa informação, marcamos a segunda coluna (próxima disponível) na linha referente à máquina 0.

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

No vetor V, a terceira operação a ser executada é a tarefa 1. Olhando no *benchmark*, a segunda operação da tarefa 1 deve ser executada pela máquina 1 e, com essa informação, marcamos a primeira coluna (próxima disponível) na linha referente à máquina 1.

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

No vetor V, a quarta operação a ser executada é a tarefa 2. Olhando no *benchmark*, a primeira operação da tarefa 2 deve ser executada pela máquina 1 e, com essa informação, marcamos a segunda coluna (próxima disponível) na linha referente à máquina 1.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & \\ 1 & \mathbf{2} & \end{bmatrix}$$

No vetor V, a quinta operação a ser executada é a tarefa 0. Olhando no *benchmark*, a segunda operação da tarefa 0 deve ser executada pela máquina 1 e, com essa informação, marcamos a terceira coluna (próxima disponível) na linha referente à máquina 1.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & \\ 1 & 2 & \mathbf{0} \end{bmatrix}$$

No vetor V, a sexta operação a ser executada é a tarefa 2. Olhando no *benchmark*, a segunda operação da tarefa 2 deve ser executada pela máquina 0 e, com essa informação, marcamos a terceira coluna (próxima disponível) na linha referente à máquina 0.

A matriz de janela de tempo JT_{mxn}, de m linhas e n colunas, onde m é o número de máquinas e n é o número de tarefas, é construída com base na matriz A_{mxn} e no *benchmark*.

Começando pela máquina 0, que processa primeiro a tarefa 0, e olhando no *benchmark*, a tarefa 0 pode ser executada primeiro pela máquina 0 e com 1 unidade de tempo. Em seguida, a máquina 0 processa a tarefa 1, e olhando no *benchmark*, a tarefa 1 pode ser executada pela máquina 0 e com 2 unidades de tempo. Em seguida, a máquina 0 processa a tarefa 2 e, olhando no *benchmark*, a tarefa 2 deve ser executada primeiro pela máquina 1, para depois ser executada pela máquina 0. Desta forma, é precido olhar, na matriz A_{mxn}, qual tarefa deverá ser executada na máquina 1. Olhando na matriz, a tarefa 2 só poderá ser executada pela máquina 1, com 1 unidade de tempo, após a execução da tarefa 1, também com 1 unidade de tempo. Desta forma, o início da execução da tarefa 1 na máquina 1 só ocorrerá após o término da execução da tarefa 1 na máquina 0. Na construção da matriz de janela de tempo é importante levar em consideração a restrição de precedência.

A matriz da janela de tempo para este exemplo está representada em JT. A primeira linha é referente à máquina 0, e a segunda linha é referente à máquina 1.

$$\mathbf{JT} = \begin{bmatrix} 0/1 & 1/3 & 5/7 \\ 3/4 & 4/5 & 5/6 \end{bmatrix}$$

Segundo a janela de tempo JT, o *makespan* obtido para este indivíduo é igual a 7 pela máquina 0.

O Diagrama de Grantt do escalonamento deste cromossomo, gerado para a resolução do problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura A.1.

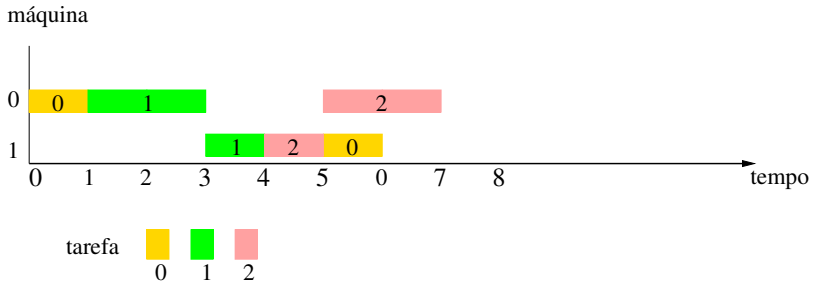

Fig. A.1: Diagrama de Grantt de um dos indivíduos na resolução do problema do *job shop* 3x2.

Apêndice B

Apêndices

Neste apêndice será apresentada, para um exemplo de 3 tarefas e 2 máquinas, a execução de alguns operadores de *crossover* e de *mutação* na aplicação do algoritmo memético com população estruturada na resolução do problema de escalonamento *job shop* com parâmetros precisos, descrito na Seção 3.2.2.2.

A Tabela B.1 apresenta um *benchmark* exemplo para um problema de escalonamento *job shop*, com parâmetros precisos, de 3 tarefas e 2 máquinas ($m = 2$ e $n = 3$).

Tab. B.1: *Benchmark* do problema do *job shop* 3x2.

	(m,p)	*(m,p)*
Tarefa 0:	(0,1)	(1,1)
Tarefa 1:	(0,2)	(1,1)
Tarefa 2:	(1,1)	(0,2)

Segundo o *benchmark* acima, a tarefa 0 deve ser executada primeiramente pela máquina 0, com tempo de processamento de uma unidade (operação 1), e depois pela máquina 2, com tempo de processamento de 1 unidade (operação 2).

B.1 *Crossover* 1

A operação de *crossover 1* foi aplicada na resolução do problema de escalonamento *job shop* e foi descrita na Seção 3.2.2.2. Considerando que os dois indivíduos, representados por $V1$ e $V2$, sejam respectivamente os indivíduo *pocket* líder e *pocket* subordinado, e que realizarão a operação de *crossover 1*.

$$\mathbf{V1} = \begin{pmatrix} 0 & 1 & 2 & 1 & 0 & 2 \end{pmatrix}$$

$$\mathbf{V2} = \begin{pmatrix} 0 & 1 & 1 & 2 & 0 & 2 \end{pmatrix}$$

Pelo *crossover* 1, é gerada aleatoriamente uma taxa de pontos de cruzamento de 0 à 2, para o exemplo considerado.

Supondo que seja gerado o número 1. Isso quer dizer que será feita apenas 1 troca de operações entre os indivíduos. Para realizar esta troca, são geradas aleatoriamente 2 tarefas distintas e 2 operações distintas.

Se as tarefas escolhidas forem a tarefa 0 e tarefa 2, e as operações escolhidas forem 1 e 2, na geração dos filhos, serão trocadas a operação 1 da tarefa 0 do indivíduo $V1$ com a operação 2 da tarefa 2 do indivíduo $V2$, e trocadas a operação 2 da tarefa 2 do invivíduo $V1$ com a operação 1 da tarefa 0 do indivíduo $V2$.

$$\mathbf{V1} = \begin{pmatrix} \mathbf{0} & 1 & 2 & 1 & 0 & \mathbf{2} \end{pmatrix}$$

$$\mathbf{V2} = \begin{pmatrix} \mathbf{0} & 1 & 1 & 2 & 0 & \mathbf{2} \end{pmatrix}$$

Os dois novos filhos, gerados após a troca, ficarão com os vetores codificados conforme $F1$ e $F2$.

$$\mathbf{F1} = \begin{pmatrix} \mathbf{2} & 1 & 2 & 1 & 0 & \mathbf{0} \end{pmatrix}$$

$$\mathbf{F2} = \begin{pmatrix} \mathbf{2} & 1 & 1 & 2 & 0 & \mathbf{0} \end{pmatrix}$$

A factibilidade nos cromossomos filhos gerados é mantida porque é garantida que a quantidade de operações de cada tarefa que aparecem no vetor é igual ao número de máquinas.

Tanto para o primeiro filho quanto para o segundo filho, a matriz A_{mxn} ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este exemplo está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/2 & 2/4 & 4/5 \\ 0/1 & 2/3 & 5/6 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pela máquina 1, é igual a 6.

O Diagrama de Grantt do escalonamento do filho gerado após a execução da operação de *crossover 1*, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.1.

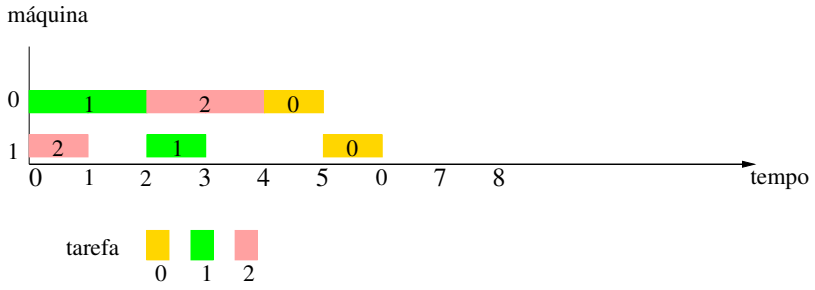

Fig. B.1: Diagrama de Grantt do escalonamento do filho gerado após a execução do *crossover 1* para o problema do *job shop* 3x2.

B.2 *Crossover 2*

A operação de *crossover 2* foi aplicada na resolução do problema de escalonamento *job shop* e foi descrita na Seção 3.2.2.2. Considerando que os dois indivíduos, representados por $V1$ e $V2$, sejam respectivamente os indivíduo *pocket* líder e *pocket* subordinado, e que realizarão a operação de *crossover 2*.

$$\mathbf{V1} = (0 \quad 1 \quad 2 \quad \mathbf{1} \quad \mathbf{0} \quad \mathbf{2})$$

$$\mathbf{V2} = (0 \quad 1 \quad \mathbf{1} \quad 2 \quad \mathbf{0} \quad \mathbf{2})$$

Pelo *crossover* 2, é sorteada uma operação e, neste exemplo, existem duas opções.

Supondo que seja escolhida a operação 2. Isso quer dizer que trocaremos todas as segundas operações entre os dois indivíduos. As segundas operações das tarefas estão marcadas em negrito nos vetores $V1$ e $V2$.

Para gerar o primeiro filho, as segundas operações de todas as tarefas do indivíduo *pocket* líder são colocadas, na mesma posição, no indivíduo filho. O restante das operações são retiradas do indivíduo *pocket* subordinado, na mesma ordem em que aparecem no indivíduo pai. O primeiro filho gerado é representado pelo vetor $F1$.

$$\mathbf{F1} = \begin{pmatrix} 0 & 1 & 2 & \mathbf{1} & \mathbf{0} & \mathbf{2} \end{pmatrix}$$

Para gerar o segundo filho, as segundas operações de todas as tarefas do indivíduo *pocket* subordinado são colocadas, na mesma posição, no indivíduo filho. O restante das operações são retiradas do indivíduo *pocket* líder, na mesma ordem em que aparecem no indivíduo pai. O segundo filho gerado é representado pelo vetor $F2$.

$$\mathbf{F2} = \begin{pmatrix} 0 & 1 & \mathbf{1} & 2 & \mathbf{0} & \mathbf{2} \end{pmatrix}$$

Os dois novos filhos, gerados após a troca, ficarão com os vetores codificados conforme $F1$ e $F2$. Neste caso, os filhos gerados mantiveram a mesma codificação dos indivíduos pais.

A factibilidade nos cromossomos filhos gerados é mantida porque é garantida que a quantidade de operações de cada tarefa que aparecem no vetor é igual ao número de máquinas.

A matriz A_{mxn}, para o primeiro filho $F1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 1 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este exemplo está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/1 & 1/3 & 3/5 \\ 0/1 & 3/4 & 4/5 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pelas máquinas 0 e 1, é igual a 5.

O Diagrama de Grantt do escalonamento do primeiro filho gerado após a execução da operação de *crossover 2*, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.2.

A matriz A_{mxn}, para o segundo filho $F2$, ficará representada conforme $A2$.

$$\mathbf{A1} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este exemplo está representada em $JT2$.

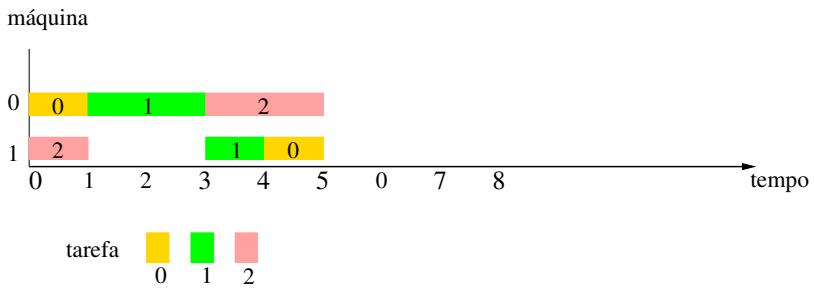

Fig. B.2: Diagrama de Grantt do escalonamento do primeiro filho gerado após a execução do *crossover 2* para o problema do *job shop* 3x2.

$$\mathbf{JT2} = \begin{bmatrix} 0/1 & 1/3 & 5/7 \\ 3/4 & 4/5 & 5/6 \end{bmatrix}$$

Segundo a janela de tempo $JT2$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 7.

O Diagrama de Grantt do escalonamento do segundo filho gerado após a execução da operação de *crossover 2*, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.3.

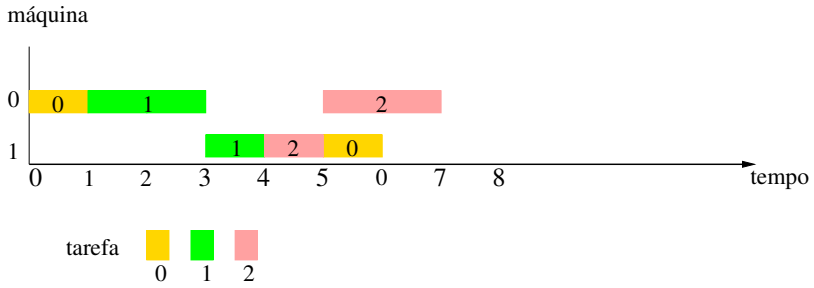

Fig. B.3: Diagrama de Grantt do escalonamento do segundo filho gerado após a execução do *crossover 2* para o problema do *job shop* 3x2.

B.3 *Crossover* 3

A operação de *crossover 3* foi aplicada na resolução do problema de escalonamento *job shop* e foi descrita na Seção 3.2.2.2. Considerando que os dois indivíduos, representados por $V1$ e $V2$, sejam respectivamente os indivíduo *pocket* líder e *pocket* subordinado, e que realizarão a operação de *crossover 3*.

$$\mathbf{V1} = \begin{pmatrix} 0 & 1 & 2 & 1 & 0 & 2 \end{pmatrix}$$

$$\mathbf{V2} = \begin{pmatrix} 0 & 1 & 1 & 2 & 0 & 2 \end{pmatrix}$$

Segundo este *crossover*, para cada posição no vetor do cromossomo do filho, é escolhido um dos cromossomos pais e, em seguida, é escolhida uma posição deste pai que ainda não tenha sido escolhida. A operação desta posição escolhida é colocada no vetor do primeiro filho. Para gerar o segundo filho, é escolhida uma posição do outro cromossomo pai e colocada no cromossomo do segundo filho.

Para gerar o primeiro filho, de acordo com o *crossover* 3, é sorteado de qual pai será escolhida a operação.

Supondo que seja sorteado o cromossomo pai $V2$, e que seja sorteada a posição 4. O cromossomo do filho é representado por $F1$, onde a primeira posição foi preenchida com a operação da quarta posição do cromossomo $V2$.

$$\mathbf{F1} = \begin{pmatrix} \mathbf{2} & & & & & \end{pmatrix}$$

Supondo que as próximas escolhas sejam as seguintes: posição 3 do cromossomo $V1$, posição 2 do cromossomo $V1$, posição 6 do cromossomo $V2$ (não é permitida porque já existem duas operações para a tarefa 2), posição 5 do cromossomo $V2$, posição 1 do cromossomo $V1$, posição 2 do cromossomo $V2$. Com esta seqüência de escolhas, o cromossomo do primeiro filho é representado por $F1$.

$$\mathbf{F1} = \begin{pmatrix} 2 & 2 & 1 & 0 & 0 & 1 \end{pmatrix}$$

A factibilidade nos cromossomos filhos gerados é mantida porque é garantida que a quantidade de operações de cada tarefa que aparecem no vetor é igual ao número de máquinas.

A matriz A_{mxn}, para o primeiro filho $F1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 2 & 1 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

A matriz da janela de tempo para este exemplo está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 1/3 & 3/5 & 5/6 \\ 0/1 & 6/7 & 7/8 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pela máquina 1, é igual a 8.

O Diagrama de Grantt do escalonamento do primeiro filho gerado após a execução da operação de *crossover 3*, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.4.

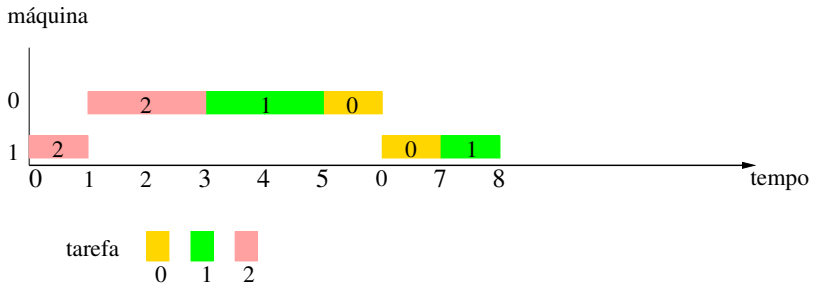

Fig. B.4: Diagrama de Grantt do escalonamento do primeiro filho gerado após a execução do *crossover 3* para o problema do *job shop* 3x2.

B.4 *Crossover* 4

A operação de *crossover 4* foi aplicada na resolução do problema de escalonamento *job shop* e foi descrita na Seção 3.2.2.2. Considerando que os dois indivíduos, representados por $V1$ e $V2$, sejam respectivamente os indivíduo *pocket* líder e *pocket* subordinado, e que realizarão a operação de *crossover 4*.

$$\mathbf{V1} = (0 \quad \mathbf{1} \quad \mathbf{2} \quad \mathbf{1} \quad \mathbf{0} \quad 2)$$

$$\mathbf{V2} = (0 \quad \mathbf{1} \quad \mathbf{1} \quad \mathbf{2} \quad \mathbf{0} \quad 2)$$

Segundo este *crossover*, é escolhida uma *substring* (uma posição inicial e uma posição final) dentro do cromossomo. Supondo que tenham sido sorteadas as posições inicial 2 e final 5, então as posições marcadas em negrito nos vetores $V1$ e $V2$, serão colocadas nos cromossomos filhos gerados.

Para gerar o primeiro filho $F1$, de acordo com o *crossover* 4, a *substring* sorteada do indivíduo $V1$ é colocada nas primeiras posições do cromossomo filho. O restante das posições são preenchidas com as tarefas que faltam e são retiradas do cromossomo pai $V2$. Como faltam as tarefas 0 e 2, elas são retiradas do cromossomo pai $V2$ e nessa mesma ordem porque estão dispostas desta maneira no cromossomo pai $V2$.

$$\mathbf{F1} = \ (\mathbf{1} \quad \mathbf{2} \quad \mathbf{1} \quad \mathbf{0} \quad 0 \quad 2)$$

Para gerar o segundo filho $F2$, de acordo com o *crossover* 4, a *substring* sorteada do indivíduo $V2$ é colocada nas primeiras posições do cromossomo filho. O restante das posições são preenchidas com as tarefas que faltam e são retiradas do cromossomo pai $V1$. Como faltam as tarefas 0 e 2, elas são retiradas do cromossomo pai $V1$ e nessa mesma ordem porque estão dispostas desta maneira no cromossomo pai $V1$.

$$\mathbf{F2} = \ (\mathbf{1} \quad \mathbf{1} \quad \mathbf{2} \quad \mathbf{0} \quad 0 \quad 2)$$

A factibilidade nos cromossomos filhos gerados é mantida porque é garantida que a quantidade de operações de cada tarefa que aparecem no vetor é igual ao número de máquinas.

A matriz A_{mxn}, para o primeiro filho $F1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este exemplo está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/2 & 2/3 & 3/5 \\ 0/1 & 2/3 & 3/4 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 5.

O Diagrama de Grantt do escalonamento do primeiro filho gerado após a execução da operação de *crossover 4*, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.5.

A matriz A_{mxn}, para o segundo filho $F2$, ficará representada conforme $A2$.

$$\mathbf{A2} = \begin{bmatrix} 1 & 0 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este exemplo está representada em $JT2$.

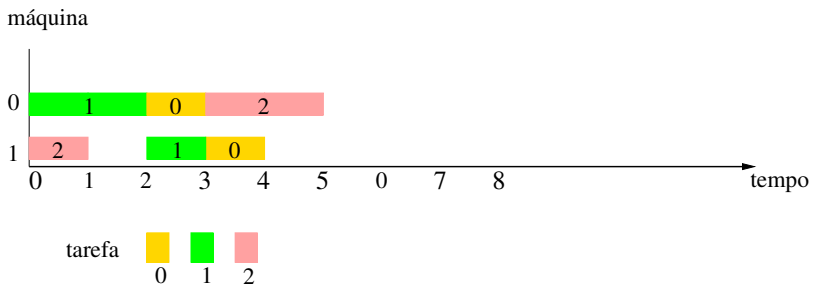

Fig. B.5: Diagrama de Grantt do escalonamento do primeiro filho gerado após a execução do *crossover 4* para o problema do *job shop* 3x2.

$$\mathbf{JT2} = \begin{bmatrix} 0/2 & 2/3 & 4/6 \\ 2/3 & 3/4 & 4/5 \end{bmatrix}$$

Segundo a janela de tempo $JT2$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 6.

O Diagrama de Grantt do escalonamento do segundo filho gerado após a execução da operação de *crossover 4*, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.6.

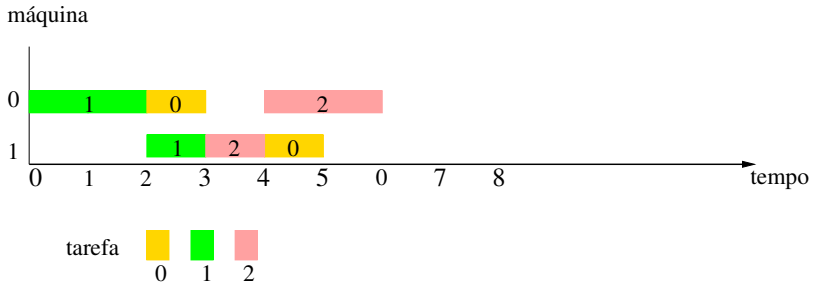

Fig. B.6: Diagrama de Grantt do escalonamento do segundo filho gerado após a execução do *crossover 4* para o problema do *job shop* 3x2.

B.5 *Crossover 5*

A operação de *crossover 5* foi aplicada na resolução do problema de escalonamento *job shop* e foi descrita na Seção 3.2.2.2. Considerando que os dois indivíduos, representados por $V1$ e $V2$, sejam respectivamente os indivíduo *pocket* líder e *pocket* subordinado, e que realizarão a operação de *crossover 5*.

$$\mathbf{V1} = \ (0 \quad 1 \quad 2 \quad 1 \quad \mathbf{0} \quad \mathbf{2})$$

$$\mathbf{V2} = \ (0 \quad 1 \quad 1 \quad 2 \quad \mathbf{0} \quad \mathbf{2})$$

Segundo este *crossover*, é escolhida uma *substring* (uma posição inicial e uma posição final) dentro do cromossomo. Supondo que tenham sido sorteadas as posições inicial 5 e final 6, então as posições marcadas em negrito nos vetores $V1$ e $V2$, serão colocadas nos cromossomos filhos gerados.

Para gerar o primeiro filho, que é representado por $F1$, de acordo com o *crossover* 5, a *substring* sorteada do indivíduo $V1$ é colocada na mesma posição em que aparece no vetor $V1$. O restante das posições são preenchidas com as tarefas que faltam e são retiradas do cromossomo pai $V2$. Como faltam as tarefas 0, 1, 1 e 2, elas são retiradas do cromossomo pai $V2$ e na ordem em que estão dispostas no cromossomo pai $V2$.

$$\mathbf{F1} = \ (0 \quad 1 \quad 1 \quad 2 \quad \mathbf{0} \quad \mathbf{2})$$

Para gerar o segundo filho, que é representado por $F2$, de acordo com o *crossover* 5, a *substring* sorteada do indivíduo $V2$ é colocada na mesma posição em que aparece no vetor $V2$. O restante das posições são preenchidas com as tarefas que faltam e são retiradas do cromossomo pai $V1$. Como faltam as tarefas 0, 1, 1 e 2, elas são retiradas do cromossomo pai $V2$ e na ordem em que estão dispostas no cromossomo pai $V1$.

$$\mathbf{F2} = \ (0 \quad 1 \quad 2 \quad 1 \quad \mathbf{0} \quad \mathbf{2})$$

A factibilidade nos cromossomos filhos gerados é mantida porque é garantida que a quantidade de operações de cada tarefa que aparecem no vetor é igual ao número de máquinas.

A matriz A_{mxn}, para o primeiro filho $F1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

A matriz A_{mxn}, para o segundo filho $F2$, ficará representada conforme $A2$.

$$\mathbf{A2} = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 1 & 0 \end{bmatrix}$$

Neste caso, os filhos gerados são iguais ao pais e então não houve nenhuma alteração no valor do *makespan*.

B.6 *Crossover 6*

A operação de *crossover 6* foi aplicada na resolução do problema de escalonamento *job shop* e foi descrita na Seção 3.2.2.2. Considerando que os dois indivíduos, representados por $V1$ e $V2$, sejam respectivamente os indivíduo *pocket* líder e *pocket* subordinado, e que realizarão a operação de *crossover 6*.

$$\mathbf{V1} = \begin{pmatrix} 0 & 1 & 2 & 1 & 0 & 2 \end{pmatrix}$$

$$\mathbf{V2} = \begin{pmatrix} 0 & 1 & 1 & 2 & 0 & 2 \end{pmatrix}$$

A matriz A_{mxn}, do vetor $V1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 1 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo $V1$ está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/1 & 1/3 & 3/5 \\ 0/1 & 3/4 & 4/5 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pelas máquinas 0 e 1, é igual a 5.

O Diagrama de Grantt do escalonamento do cromossomo $V1$, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.7.

A matriz A_{mxn}, do vetor $V2$, ficará representada conforme $A2$.

$$\mathbf{A2} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo $V2$ está representada em $JT2$.

$$\mathbf{JT2} = \begin{bmatrix} 0/1 & 1/3 & 5/7 \\ 3/4 & 4/5 & 5/6 \end{bmatrix}$$

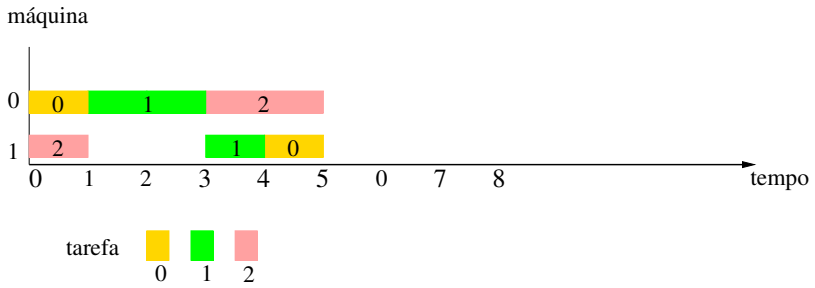

Fig. B.7: Diagrama de Grantt do escalonamento do cromossomo $V1$ para o problema do *job shop* 3x2.

Segundo a janela de tempo $JT2$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 7.

O Diagrama de Grantt do escalonamento do cromossomo $V2$, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.8.

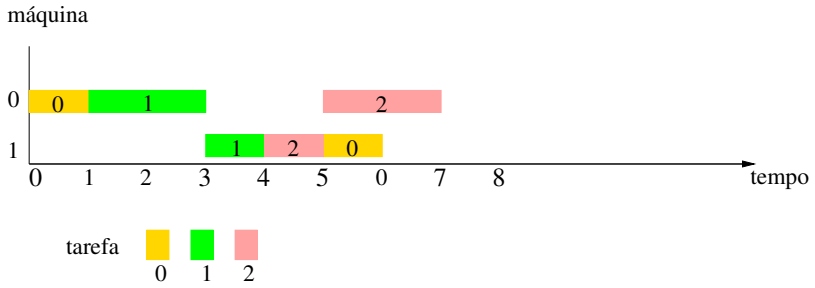

Fig. B.8: Diagrama de Grantt do escalonamento do cromossomo $V2$ para o problema do *job shop* 3x2.

Segundo este *crossover*, é verificado, para os dois indivíduos, qual é a máquina menos ociosa. Para verificar qual é a máquina menos ociosa, que é a máquina com menos tempo sem atender alguma tarefa, é somado o tempo ocioso em cada máquina. A matriz de janela de tempo deve ser analisada para somar este tempo ocioso.

Para o indivíduo $V1$, na qual a matriz de janela de tempo é representada por $JT1$, o tempo ocioso da máquina 0 é igual a 0, pois todas as tarefas são atendidas em seqüência. O tempo ocioso da máquina 1 é de 1 à 3. Sendo assim, o tempo ocioso na máquina 1 é igual a 2.

Para o indivíduo $V2$, na qual a matriz de janela de tempo é representada por $JT2$, o tempo ocioso da máquina 0 de 3 à 5. Sendo assim, o tempo ocioso na máquina 0 é igual a 2. O tempo ocioso da máquina 1 é de 0 à 3. Sendo assim, o tempo ocioso na máquina 1 é igual a 3.

A idéia do *crossover 6* é cruzar as melhores seqüências das máquinas menos ociosas. Começando com a máquina 0, que é a menos ociosa para o invidíduo $V1$, e tem a seqüência de tarefas (012) na

matriz $A1$.

Olhando no *benchmark* o número da operação destas tarefas (012) na máquina 0, é verificado que se tratam da operação 1 para a tarefa 0, da operação 1 para a tarefa 1 e da operação 2 para a tarefa 2.

Para gerar o filho, são retirados do indivíduo $V1$ a primeira operação da tarefa 0, a primeira operação da tarefa 1, e a segunda operação da tarefa 2. Estas operações são colocadas na mesma posição em que estavam no vetor $V1$. O filho gerado é representado pelo vetor $F1$ e estas operações retiradas do vetor $V1$ estão marcadas em negrito.

$$\mathbf{F1} = (\mathbf{0} \quad \mathbf{1} \qquad\quad \mathbf{2})$$

Em seguida, na geração do filho $F1$, é verificada a próxima máquina menos ociosa e, como existem duas máquinas com o mesmo tempo ocioso (máquina 0 para $V2$ e máquina 1 para $V1$), é escolhida a máquina 1 para o indivíduo $V1$, que tem a seqüência de tarefas (210) na matriz $A1$.

Olhando no *benchmark* o número da operação destas tarefas (210) na máquina 1, é verificado que se tratam da operação 2 para a tarefa 0, da operação 2 para a tarefa 1 e da operação 1 para a tarefa 2.

Para continuar a geração do filho, são retirados do indivíduo $V1$ a segunda operação da tarefa 0, a segunda operação da tarefa 1, e a primeira operação da tarefa 2. Estas operações são colocadas nas posições vagas do vetor $F1$ e na ordem em que aparecem no vetor $V1$. O filho gerado é representado pelo vetor $F1$ e estas operações retiradas do vetor $V1$ estão marcadas em negrito.

$$\mathbf{F1} = (0 \quad 1 \quad \mathbf{1} \quad \mathbf{0} \quad \mathbf{2} \quad 2)$$

A factibilidade no cromossomo filho gerado é mantida porque é garantida que a quantidade de operações de cada tarefa que aparecem no vetor é igual ao número de máquinas.

A matriz A_{mxn}, para o primeiro filho $F1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 2 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo filho $F1$ está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/1 & 1/3 & 6/8 \\ 3/4 & 4/5 & 5/6 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 8.

O Diagrama de Grantt do escalonamento do filho gerado após a execução da operação de *crosso-*

ver 6, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.9.

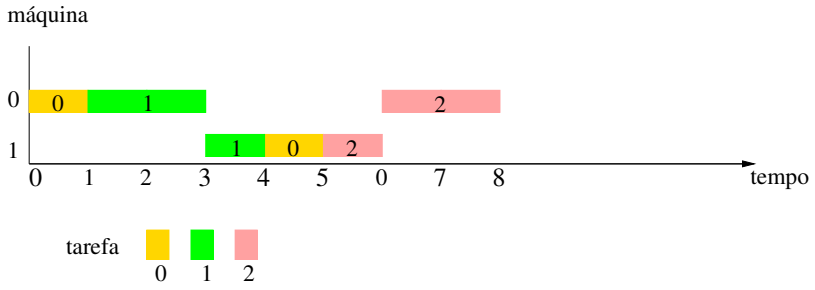

Fig. B.9: Diagrama de Grantt do escalonamento do filho gerado após a execução do *crossover 6* para o problema do *job shop* 3x2.

B.7 *Crossover 7*

A operação de *crossover 7* foi aplicada na resolução do problema de escalonamento *job shop* e foi descrita na Seção 3.2.2.2. Considerando que os dois indivíduos, representados por $V1$ e $V2$, sejam respectivamente os indivíduo *pocket* líder e *pocket* subordinado, e que realizarão a operação de *crossover 7*.

$$\mathbf{V1} = \begin{pmatrix} 0 & 1 & 2 & 1 & 0 & 2 \end{pmatrix}$$

$$\mathbf{V2} = \begin{pmatrix} 0 & 1 & 1 & 2 & 0 & 2 \end{pmatrix}$$

A matriz A_{mxn}, do vetor $V1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 1 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo $V1$ está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/1 & 1/3 & 3/5 \\ 0/1 & 3/4 & 4/5 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pelas máquinas 0 e 1, é igual a 5.

O Diagrama de Grantt do escalonamento do cromossomo $V1$, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.7.

A matriz A_{mxn}, do vetor $V2$, ficará representada conforme $A2$.

$$\mathbf{A2} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo $V2$ está representada em $JT2$.

$$\mathbf{JT2} = \begin{bmatrix} 0/1 & 1/3 & 5/7 \\ 3/4 & 4/5 & 5/6 \end{bmatrix}$$

Segundo a janela de tempo $JT2$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 7.

Segundo este *crossover*, é verificado, para os dois indivíduos, qual é a tarefa menos ociosa. Para verificar qual é a tarefa menos ociosa, que é a tarefa com menos tempo sem ser atendida por alguma tarefa, é somado o tempo ocioso de cada tarefa entre as máquinas que a estão atendendo. A matriz de janela de tempo e a matriz A devem ser analisadas para somar este tempo ocioso.

Para o indivíduo $V1$, na qual a matriz de janela de tempo é representada por $JT1$ e a matriz de codificação é representada por $A1$, o tempo ocioso da tarefa 0 ($0/1$ na máquina 0 - $4/5$ na máquina 1) é igual a 3. O tempo ocioso da tarefa 1 ($1/3$ na máquina 0 - $3/4$ na máquina 1) é igual a 1. O tempo ocioso da tarefa 2 ($0/1$ na máquina 1 - $3/5$ na máquina 0) é igual a 2.

Para o indivíduo $V2$, na qual a matriz de janela de tempo é representada por $JT2$ e a matriz de codificação é representada por $A2$, o tempo ocioso da tarefa 0 ($0/1$ na máquina 0 - $5/6$ na máquina 1) é igual a 4. O tempo ocioso da tarefa 1 ($1/3$ na máquina 0 - $3/4$ na máquina 1) é igual a 1. O tempo ocioso da tarefa 2 ($4/5$ na máquina 1 - $5/7$ na máquina 0) é igual a 4.

A idéia do *crossover 7* é cruzar as melhores seqüências das tarefas menos ociosas. Começando com a tarefa 1, que é a menos ociosa para o invidíduo $V1$. A seqüência de operações da tarefa 1 em $V1$ é colocada no vetor do filho gerado, na mesma posição em que aparecem no vetor $V1$. O vetor do filho gerado é representado por $F1$, onde as operações da tarefa 1, retiradas de $V1$, estão marcadas em negrito.

$$\mathbf{F1} = \ (\ \mathbf{1} \quad \mathbf{1} \quad)$$

Em seguida, é analisada a tarefa 2 do invidíduo $V1$, que é a próxima menos ociosa. A seqüência de operações da tarefa 1 em $V1$ é colocada no vetor do filho gerado, na mesma posição em que aparecem no vetor $V1$. As operações da tarefa 2, retiradas de $V1$, estão marcadas em negrito no cromossomo

filho $F1$.

$$\mathbf{F1} = (\ 1\ \ \mathbf{2}\ \ 1\ \ \ \mathbf{2})$$

Em seguida, é analisada a tarefa 0 do invidíduo $V1$, que é a próxima menos ociosa. A seqüência de operações da tarefa 0 em $V1$ é colocada no vetor do filho gerado, na mesma posição em que aparecem no vetor $V1$. As operações da tarefa 0, retiradas de $V1$, estão marcadas em negrito no cromossomo filho $F1$.

$$\mathbf{F1} = (\mathbf{0}\ \ 1\ \ 2\ \ 1\ \ \mathbf{0}\ \ 2)$$

Se a mesma posição na qual for inserida a operação já estiver ocupada, a operação é colocada na primeira posição vazia do cromossomo filho.

Neste caso, o cromossomo filho gerado é igual ao cromossomo pai $V1$. Isto indica que a seqüência de atendimento das tarefas no indivíduo $V1$ é melhor que a seqüência de atendimento das tarefas no indivíduo $V2$.

A factibilidade no cromossomo filho gerado é mantida porque é garantida que a quantidade de operações de cada tarefa que aparecem no vetor é igual ao número de máquinas.

B.8 *Mutação* 1

A operação de *mutação 1* foi aplicada na resolução do problema de escalonamento *job shop* e foi descrita na Seção 3.2.2.2. Considerando que o indivíduo, representados por $V1$, seja o indivíduo *pocket* que realizará a operação de *mutação 1*.

$$\mathbf{V1} = (0\ \ 1\ \ 1\ \ 2\ \ 0\ \ 2)$$

A matriz A_{mxn}, do vetor $V1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo $V1$ está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/1 & 1/3 & 5/7 \\ 3/4 & 4/5 & 5/6 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 7.

O Diagrama de Grantt do escalonamento do cromossomo $V1$, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.8.

Segundo este operador de *mutação*, são sorteadas duas tarefas e, para cada tarefa, é sorteada uma operação. Supondo que tenham sido sorteadas as tarefas 1 e 0 e, para a tarefa 1, tenha sido sorteada a operação 1 e, para a tarefa 0, tenha sido sorteada a operação 2.

Para gerar o filho, a primeira operação da tarefa 1 é trocada com a segunda operação da tarefa 0. O filho gerado é representado pelo vetor $F1$, e as tarefas trocadas são marcadas em negrito.

$$\mathbf{F1} = (0 \quad \mathbf{0} \quad 1 \quad 2 \quad \mathbf{1} \quad 2)$$

A matriz A_{mxn}, do vetor $F1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 0 & 1 & 2 \\ 0 & 2 & 1 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo $F1$ está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/1 & 1/3 & 3/5 \\ 1/2 & 2/3 & 3/4 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 5.

O Diagrama de Grantt do escalonamento do cromossomo $F1$, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.10.

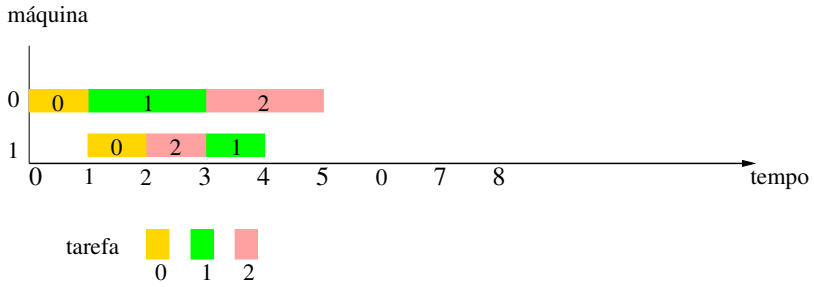

Fig. B.10: Diagrama de Grantt do escalonamento do filho gerado após a execução da *mutação 1* para o problema do *job shop* 3x2.

B.9 *Mutação 2*

A operação de *mutação 2* foi aplicada na resolução do problema de escalonamento *job shop* e foi descrita na Seção 3.2.2.2. Considerando que o indivíduo, representados por $V1$, seja o indivíduo *pocket* que realizará a operação de *mutação 2*.

$$\mathbf{V1} = \begin{pmatrix} 0 & 1 & 1 & 2 & 0 & 2 \end{pmatrix}$$

A matriz A_{mxn}, do vetor $V1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo $V1$ está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/1 & 1/3 & 5/7 \\ 3/4 & 4/5 & 5/6 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 7.

O Diagrama de Grantt do escalonamento do cromossomo $V1$, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.8.

Segundo este operador de *mutação*, são sorteadas duas tarefas distintas e, em seguida, são trocadas todas as operações entre as tarefas.

Para gerar o filho, supondo que tenham sido sorteadas as tarefas 1 e 0, as primeiras operações das tarefas 1 e 0 são trocadas. Em seguida, as segundas operações das tarefas 1 e 0 são trocadas. O filho gerado é representado pelo vetor $F1$, e as tarefas trocadas são marcadas em negrito.

$$\mathbf{F1} = \begin{pmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} & 2 & \mathbf{1} & 2 \end{pmatrix}$$

A matriz A_{mxn}, do vetor $F1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 2 & 1 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo $F1$ está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/2 & 2/3 & 5/7 \\ 3/4 & 4/5 & 5/6 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 7.

O Diagrama de Grantt do escalonamento do cromossomo $F1$, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.11.

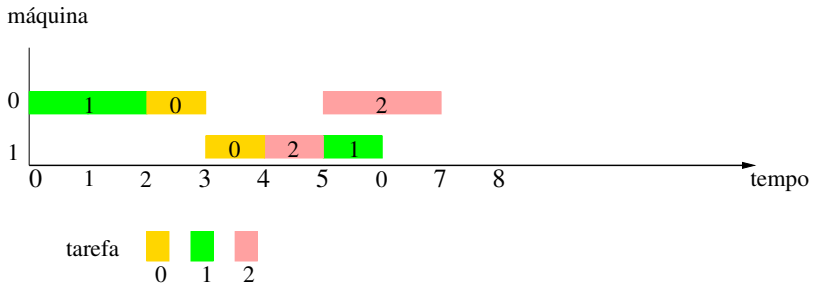

Fig. B.11: Diagrama de Grantt do escalonamento do filho gerado após a execução da *mutação 2* para o problema do *job shop* 3x2.

B.10 *Mutação* 3

A operação de *mutação 3* foi aplicada na resolução do problema de escalonamento *job shop* e foi descrita na Seção 3.2.2.2. Considerando que o indivíduo, representados por $V1$, seja o indivíduo *pocket* que realizará a operação de *mutação 3*.

$$\mathbf{V1} = \begin{pmatrix} 0 & 1 & 1 & 2 & 0 & 2 \end{pmatrix}$$

A matriz A_{mxn}, do vetor $V1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo $V1$ está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/1 & 1/3 & 5/7 \\ 3/4 & 4/5 & 5/6 \end{bmatrix}$$

O primeiro filho, representado pelo vetor $F1$, é gerado após a troca da operação 1 da tarefa 0 com a operação 1 da tarefa 1. As tarefas trocadas, no vetor $F1$, estão marcadas em negrito.

$$\mathbf{F1} = (\mathbf{1} \quad \mathbf{0} \quad 1 \quad 2 \quad 0 \quad 2)$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 7.

O Diagrama de Grantt do escalonamento do cromossomo $V1$, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.8.

Segundo este operador de *mutação*, são sorteadas três tarefas distintas e, em seguida, é sorteada um operação para cada uma das tarefas.

Supondo que tenham sido sorteadas as tarefas 0, 1 e 2, e que tenha sido sorteada a operação 1 para a tarefa 0, a operação 1 para a tarefa 1 e a operação 2 para a tarefa 2. São gerados três filhos realizando a troca entre as tarefas 0 e 1, a troca entre as tarefas 0 e 2 e a troca entre as tarefas 1 e 2.

O primeiro filho, representado pelo vetor $F1$, é gerado após a troca da operação 1 da tarefa 0 com a operação 1 da tarefa 1. As tarefas trocadas, no vetor $F1$, estão marcadas em negrito.

$$\mathbf{F1} = (\mathbf{1} \quad \mathbf{0} \quad 1 \quad 2 \quad 0 \quad 2)$$

O segundo filho, representado pelo vetor $F2$, é gerado após a troca da operação 1 da tarefa 0 com a operação 2 da tarefa 2. As tarefas trocadas, no vetor $F2$, estão marcadas em negrito.

$$\mathbf{F2} = (\mathbf{2} \quad 1 \quad 1 \quad 2 \quad 0 \quad \mathbf{0})$$

O terceiro filho, representado pelo vetor $F3$, é gerado após a troca da operação 1 da tarefa 1 com a operação 2 da tarefa 2. As tarefas trocadas, no vetor $F3$, estão marcadas em negrito.

$$\mathbf{F1} = (0 \quad \mathbf{2} \quad 1 \quad 2 \quad 0 \quad \mathbf{1})$$

A matriz A_{mxn}, do vetor $F1$, ficará representada conforme $A1$.

$$\mathbf{A1} = \begin{bmatrix} 1 & 0 & 2 \\ 1 & 2 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo $F1$ está representada em $JT1$.

$$\mathbf{JT1} = \begin{bmatrix} 0/2 & 2/3 & 4/6 \\ 2/3 & 3/4 & 4/5 \end{bmatrix}$$

Segundo a janela de tempo $JT1$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 6.

O Diagrama de Grantt do escalonamento do cromossomo $F1$, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.12.

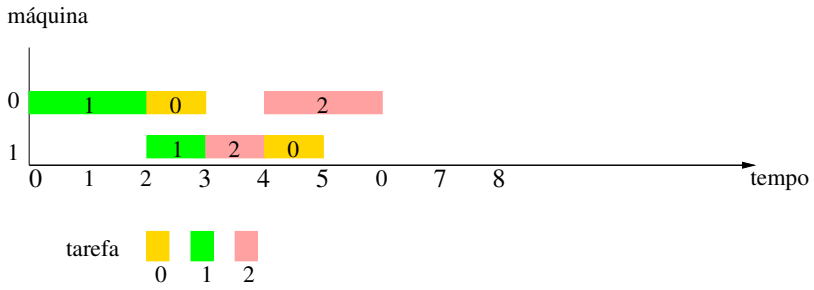

Fig. B.12: Diagrama de Grantt do escalonamento do primeiro filho gerado após a execução da *mutação 3* para o problema do *job shop* 3x2.

A matriz A_{mxn}, do vetor $F2$, ficará representada conforme $A2$.

$$\mathbf{A2} = \begin{bmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \end{bmatrix}$$

A matriz da janela de tempo para este cromossomo $F2$ está representada em $JT2$.

$$\mathbf{JT2} = \begin{bmatrix} 0/2 & 2/4 & 4/5 \\ 0/1 & 2/3 & 5/6 \end{bmatrix}$$

Segundo a janela de tempo $JT2$, o *makespan* obtido para este indivíduo, pela máquina 1, é igual a 6.

O Diagrama de Grantt do escalonamento do cromossomo $F2$, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.13.

A matriz A_{mxn}, do vetor $F3$, ficará representada conforme $A3$.

$$\mathbf{A3} = \begin{bmatrix} 0 & 1 & 2 \\ 2 & 0 & 1 \end{bmatrix}$$

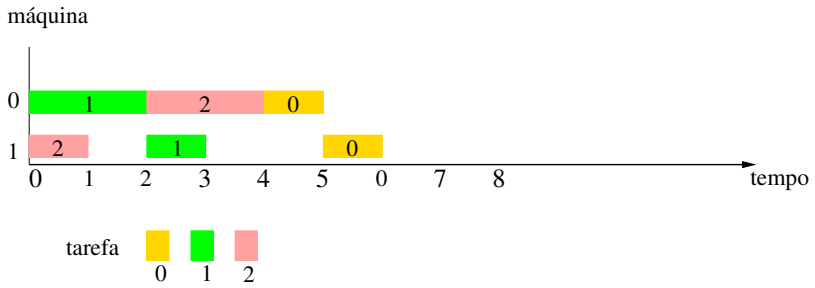

Fig. B.13: Diagrama de Grantt do escalonamento do segundo filho gerado após a execução da *mutação 3* para o problema do *job shop* 3x2.

A matriz da janela de tempo para este cromossomo $F3$ está representada em $JT3$.

$$\mathbf{JT3} = \begin{bmatrix} 0/1 & 1/3 & 3/5 \\ 0/1 & 1/2 & 3/4 \end{bmatrix}$$

Segundo a janela de tempo $JT3$, o *makespan* obtido para este indivíduo, pela máquina 0, é igual a 5. Este é o melhor indivíduo gerado pelo operador de *mutação 3*.

O Diagrama de Grantt do escalonamento do cromossomo $F3$, para o problema do *job shop* de 3 tarefas e 2 máquinas, é apresentado na Figura B.14.

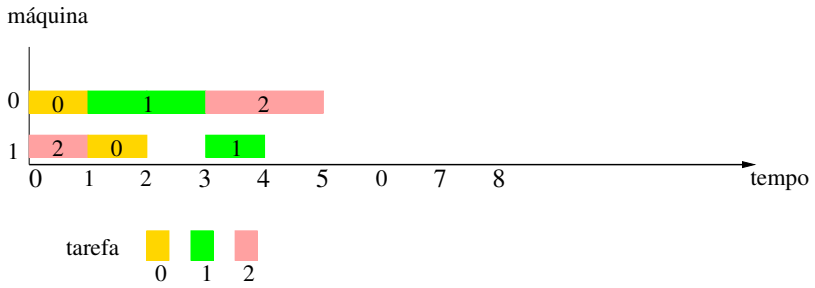

Fig. B.14: Diagrama de Grantt do escalonamento do terceiro filho gerado após a execução da *mutação 3* para o problema do *job shop* 3x2.