Aplicação do algoritmo memético no escalonamento *job shop* com paramêtros *fuzzy*

Tatiane Regina Bonfim*
Doutora em Engenharia Elétrica - UNICAMP
Professora e Coordenadora dos Cursos de Tecnologia em Programação de Computadores e Ciência da Computação da Faculdade Comunitária de Campinas - Unidade 2
e-mail: tatiane.regina@unianhanguera.edu.br

Akebo Yamakami
Doutor em Engenharia Elétrica - UNICAMP
Professor da Faculdade de Engenharia Elétrica e de Computação - UNICAMP
e-mail: akebo@dt.fee.unicamp.br

Resumo

Este trabalho apresenta uma implementação de um escalonador memético para resolver o problema de minimização do *makespan* no escalonamento *job shop* com parâmetros *fuzzy*. Para problemas com parâmetros com incertezas, torna-se difícil dizer a priori qual escalonamento é ótimo. A noção de ótimo torna-se imprecisa e o grau de otimalidade de um dado escalonamento pode ser representado por um número *fuzzy*. Na resolução do problema, o algoritmo memético foi utilizado para encontrar soluções, e o conceito de possibilidade foi aplicado para avaliar as melhores soluções. Os resultados mostram que o algoritmo memético apresenta uma boa performance na resolução do problema.

Palavras-chave: algoritmo memético, teoria da possibilidade, função de *fitness*, escalonamento *job shop* e números *fuzzy*.

Abstract

This work presents an implementation of a memetic scheduler for solving the makespan minimization problem in job shop scheduling with fuzzy parameters. For problems with parameters with uncertainties 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 can be characterized by a *fuzzy* number. For the resolution of the problem, the memetic algorithm has been used to find the solutions, and the concept of possibility has been used to evaluate the best solutions. Simulations showed that the memetic algorithm presents a good performance in the resolution of the problem.

Key-words: memetic algorithm, possibility theory, fitness function, job shop scheduling and fuzzy numbers.

**Bolsistas FUNADESP* - Fundação Nacional de Desenvolvimento do Ensino Superior Particular

Introdução

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.

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*, segundo (*Bellmann e Zadeh, 1970*), é um problema de satisfação de restrições *fuzzy*, no qual 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*, no qual alguns parâmetros são tratados com incertezas, até mesmo a noção de escalonamento ótimo é considerada imprecisa, pois se torna difícil estabelecer qual é o escalonamento ótimo das tarefas. Neste caso, em que a noção de ótimo também se torna imprecisa, 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 nas quais se têm vários escalonamentos e se quer definir qual é o melhor entre eles, segundo o critério do decisor.

O problema que é abordado é o de escalonamento *job shop* com parâmetros *fuzzy*. O escalonamento *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 irão processá-la e os correspondentes tempos de processamento. Cada operação precisa ser processada por uma dada máquina 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*. *Makespan* é o tempo total de processamento na máquina mais ocupada, que é a última a terminar a execução das tarefas a ela alocadas. Neste problema, cada parâmetro de tempo do problema é tratado na forma de um número fuzzy. O valor do *makespan* também é representado por um número *fuzzy*.

Neste trabalho, foi aplicado 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 os conceitos de possibilidade foram aplicados para avaliar as melhores soluções.

O algoritmo memético (*Moscato, 1989*) e (*Moscato e 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 (*Holland, 1973*) tem provado ser uma aproximação versátil e efetiva para a resolução de problemas de otimização. Contudo, existem muitas situações nas quais o algoritmo genético puro não oferece bons resultados e, sendo assim, vários métodos de hibridização têm sido propostos.

Em (*Chanas e 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". Em (*Bonfim e Yamakami, 2004*), foram aplicados os conceitos de possibilidade para encontrar soluções de um escalonamento "ótimo" para o problema de escalonamento de tarefas em máquinas paralelas idênticas com parâmetros *fuzzy* e foi aplicado o algoritmo memético com população estruturada para evoluir boas formas de escalonamento.

Este artigo está dividido em sete Seções. Na Seção 2 é apresentada uma explanação a respeito do problema de escalonamento *job shop* com parâmetros *fuzzy*. Na Seção 3 são apresentados alguns conceitos de números *fuzzy*. Na Seção 4 é apresentado o algoritmo memético com população estruturada. A implementação do algoritmo memético com população estruturada na resolução do problema de escalonamento *job shop* com parâmetros imprecisos é descrita na Seção 5. Na Seção 6 são apresentadas as simulações e os resultados. Por último as conclusões são apresentadas na Seção 7.

Formulação do problema

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 irão processá-la e os correspondentes tempos de processamento. Cada operação precisa ser processada por uma dada máquina 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 NPcompleto 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)*). Este *benchmark* é composto por 10 máquinas e 10 tarefas, e promoveu a competição, por pelo menos 25 anos, entre pesquisadores procurando a melhor forma de solucioná-lo.

Modelo matemático do problema

Seja *m* um conjunto de máquinas e *n* um conjunto de tarefas. Seja $O = 0,...,n\text{-}1$ o conjunto de operações, no qual *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 $\tilde{p}_i$ é fixado, e o tempo possível para início do processamento da operação *i* na máquina *l* é $t\tilde{i}_{il}$, 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* $\tilde{M}$. A variável $\tilde{M}$ representa o tempo final $t\tilde{f}_{il}$ 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* triangular. O modelo matemático para este problema é caracterizado pela Equação 1.

$$
\begin{aligned}
&Min\ \tilde{M} = \underbrace{MAX}_{1\le l\le m} = \left\{t\tilde{f}_{il}\right\} \forall i;\ 0 \le i \le n-1. \\
&s.a: \quad t\tilde{i}_{jk} - t\tilde{i}_{ik} \ge p_i\ ; \forall (i,j) \in A,\ \forall i;\ 0 \le i \le n-1,\ \forall j;\ 0 \le j \le n-1. \\
&\qquad t\tilde{i}_{jk} - t\tilde{i}_{ik} \ge p_i \ \ ou\ \ t\tilde{i}_{ik} - t\tilde{i}_{jk} \ge p_j\ ; \forall \{i,j\} \in E_k, \forall k;\ 0 \le k \le m-1. \qquad (1) \\
&\qquad t\tilde{i}_{ik} \ge 0\ ;\ \forall i \in O, \forall i;\ 0 \le i \le n-1,\ \forall k;\ 0 \le k \le m-1.
\end{aligned}
$$

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 1 apresenta uma instância gerada para o problema 6 x 6 (m = 6 e n = 6) com parâmetros com incertezas.

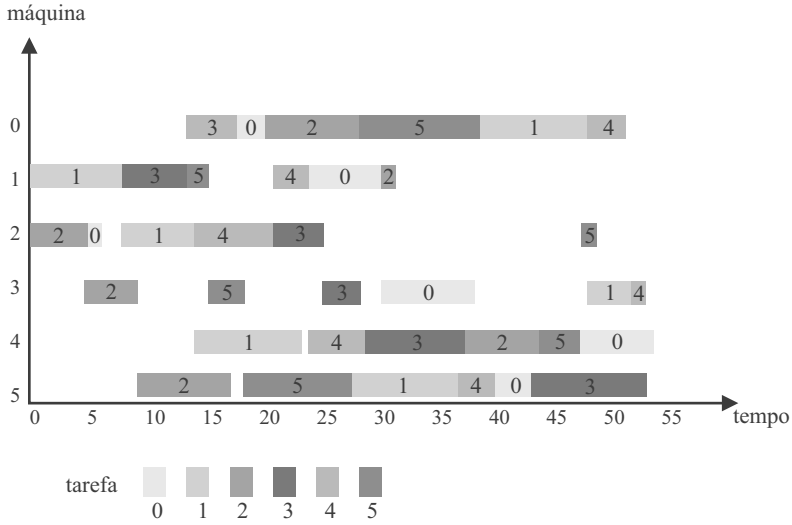
Tabela 1 - Instância do problema *job shop* 6x6 com parâmetros com incertezas

	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$	$(m,\widetilde{p})$
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, extraído de (*Muth and Thompson, 1963*), como valor modal p^0 e, os espalhamentos a esquerda $\left(p^0 - \underline{p}\right)$ e a direita $\left(\overline{p} - p^0\right)$, 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 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 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 1.

Figura 1: Diagrama de Grantt do escalonamento mínimo encontrado para o problema do *job shop* 6x6 com parâmetros com incertezas



Números *fuzzy*

Nesta seção serão descritas algumas definições com relação a números *fuzzy*.

Definição 1. Um número *fuzzy* $\widetilde{A}$ é um conjunto *fuzzy* no espaço dos números reais $\Re$ com função de pertinência $\mu_{\widetilde{A}} : \Re \rightarrow [0,1]$.

Definição 2. O α_{corte}, $\alpha \in [0,1]$, de um número *fuzzy* $\widetilde{A}$ é um conjunto descrito pela Equação 2:

$$\widetilde{A}^{\lambda} = \begin{cases} Supp(\widetilde{A}) & para\ \lambda = 0. \\ \{x : \widetilde{A}(x) \geq \lambda\} & para\ \lambda \in (0,1]. \end{cases} \quad (2)$$

$Supp(\widetilde{A}) = [\underline{a}, \overline{a}] = [\inf\{x : \widetilde{A}(x) \geq 0\}, \sup\{x : \widetilde{A}(x) \geq 0\}]$. $Supp(\widetilde{A})$ é chamado s*uporte*.

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

Um número *fuzzy* triangular, que é um caso

especial de número *fuzzy*, é denotado por $\tilde{t}_i = [\underline{t_i}, t_i^0, \overline{t_i}]$. Este número tem um valor modal t_i^0, um espalhamento à esquerda $t_i^0 - \underline{t_i}$ e um espalhamento à direita $\overline{t_i} - t_i^0$.

Uma outra forma de representar um número *fuzzy* é usando Intervalos de Confiança, como apresentado em (*Kaufmann e Gupta, 1991*). Neste caso, a função de pertinência pode ser denotada na forma de intervalo e usando $\alpha - cortes$ e, sendo assim, um número *fuzzy* triangular pode ser representado por $\tilde{t}_{i\alpha} = [t_{i1}^{\alpha}, t_{i2}^{\alpha}]$, onde $t_{i1}^{\alpha} = \underline{t_i} + a(t_i^0 - \underline{t_i})$ e $t_{i2}^{\alpha} = \overline{t_i} - a(\overline{t_i} - t_i^0)$.

O uso de $\alpha - cortes$, de acordo com (*Cantão, 2003*), nos permite operar com números *fuzzy* ponto-a-ponto e efetuar operações aritméticas em cada $\alpha - corte$, usando as operações algébricas usuais.

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 e Prade, 1980*). A construção algorítmica de um número *fuzzy* como intervalo de confiança pode ser verificada em (*Cantão, 2003*).

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

Definição 3. Sejam $\tilde{a}$ e $\tilde{b}$ dois números *fuzzy*, a possibilidade $Poss(\tilde{a}, \tilde{b})$ é definida como:

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

Definição 4. Sejam e dois números *fuzzy*, a necessidade é definida como:

A possibilidade fornece uma idéia de sobreposição e a necessidade fornece uma idéia de inclusão de B em A.

A definição de possibilidade pode ser extendida para comparação de números *fuzzy* e pode ser descrita de acordo com a Equação 3.

$$\begin{cases} Poss(\tilde{a} \geq \tilde{b}) = 1; \textit{se e somente se } a_0 \geq b_0 \\ Poss(\tilde{b} \geq \tilde{a}) = hgt(\tilde{a} \cap \tilde{b}) = \mu_{\tilde{a}}(d) \end{cases} \quad (3)$$

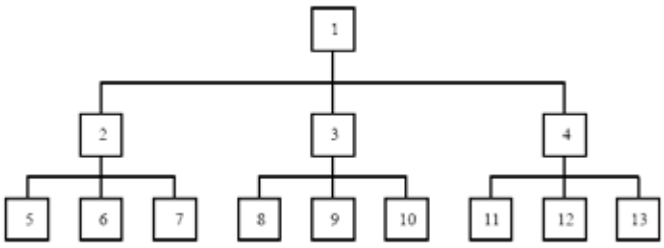
Escalonador memético

Para a resolução do problema de escalonamento *job shop* com parâmetros com incertezas, utilizou-se o algoritmo memético com população estruturada e os conceitos de possibilidade. O algoritmo memético foi utilizado para evoluir boas formas de escalonamento, e o conceito de possibilidade foi aplicado para analisar o grau de possibilidade de um determinado escalonamento ser ótimo.

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 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).

Figura 2: População estruturada em árvore



Com o intuito de adotar 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 apresenta a melhor solução encontrada na população.

Pode-se 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, garante-se 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 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*.

■ 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 1.

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 *Amxn*. 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 ***V*** pode-se verificar um exemplo da codificação do cromossomo para o problema 6 x 6 ($m = 6$ e $n = 6$) da Tabela 1. Esse é o cromossomo que resultou o escalonamento ótimo apresentado na Figura 1.

V = (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)

A seguir, o cromossomo foi codificado por uma matriz *Amxn*, 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 .

A matriz **A** apresenta um exemplo da outra codificação do cromossomo para o problema 6 x 6 ($m = 6$ e $n = 6$) da Tabela 1. Esse é o cromossomo que resultou o escalonamento ótimo apresentado na Figura 1.

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

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 indivíduos 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 1.

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 $\widetilde{jt}_{ji} = \left(\widetilde{ti}_{ji} / \widetilde{ti}_{ji}\right)$, onde $0 \le j \le m - 1$ e $0 \le i \le n - 1$, 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 *Onxm* e a matriz que codifica o cromossomo *Amxn*. A janela de tempo para o problema 6x6, representado na Figura 1, apresenta a matriz $\widetilde{JT}_{mxn}$ destacada.

$$\widetilde{JT} = \begin{bmatrix} 12.76/17.66 & 17.66/20.66 & 20.66/29.66 & 29.66/39.51 & 39.51/49.53 & 49.53/52.53 \\ 0/8.04 & 8.04/12.76 & 12.76/15.61 & 21.89/24.89 & 24.89/30.89 & 30.89/31.89 \\ 0/5.00 & 5.00/6.00 & 8/04/12.89 & 12.89/21.89 & 21.89/26.90 & 48.76/49.83 \\ 5.00/9.00 & 15.61/18.77 & 26.90/29.73 & 30.89/37.89 & 49.53/53.29 & 53.29/54.29 \\ 12.89/23.06 & 24.89/29.89 & 29.89/37.80 & 37.80/44.80 & 44.80/48.76 & 48.76/54.76 \\ 9.00/17.00 & 18.77/27.67 & 27.67/37.52 & 37.52/41.52 & 41.52/44.52 & 44.52/53.63 \end{bmatrix}$$

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{ $\widetilde{tf}_j$ }; $0 \le j \le m-1$), 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 $Poss(\widetilde{tf}_j \ge \widetilde{tf}_{j_{\max}}); \forall j$, 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 3.

Para identificar o melhor indivíduo, foi analisado $Poss(\widetilde{F}(k) < \widetilde{F}(k_{\min})); \forall k,\ 0 \le k \le Tpop\text{-}1$, 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 *Tpop* 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 (Bonfim e Yamakami, 2006)*.

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 apresentar 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)*). Este método de *defuzzificação* foi escolhido por ser o mais preciso.

Algoritmo 1 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, Onxm: 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*(F , 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

• 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*

• selecionar 1 indivíduo *pocket*

• aplicar mutação gerando 1 novo indivíduo

• recalcular a matriz Amxn(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 Amxn(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

Exemplos numéricos

Nas avaliações da aplicação do algoritmo memético com população estruturada para o problema de escalonamento *job shop* 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, pode-se comparar os resultados ótimos com os resultados obtidos pelo método heurístico.

Como o tempo de processamento da operação é um parâmetro com incerteza, este valor 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, 1].

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 memético são apresentados na Tabela 2.

Tabela 2: 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*

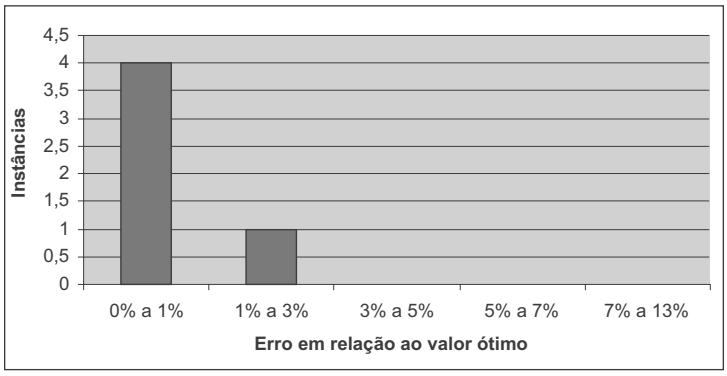
Na Tabela 3 pode-se verificar o erro médio dos valores encontrados pelo algoritmo memético com população estruturada na resolução do problema de escalonamento *job shop* com parâmetros imprecisos. Este erro médio foi calculado através da comparação do valor ótimo obtido pelo escalonador memético com o valor ótimo extraído na literatura. Nesta tabela está especificada a porcentagem de instâncias, em relação ao número total, que alcançaram o valor ótimo.

Tabela 3: Porcentagem 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 3, pode-se 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%.

Figura 3: Erro médio em relação ao valor ótimo usando o algoritmo memético com população estruturada



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 4 apresenta a quantidade de indivíduos com 90% de possibilidade de serem ótimos para cada instância testada.

Tabeba 4: 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*
15.10la27	*24*

Pela Tabela 4, pode-se 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.

Através das simulações pode-se analisar que o uso do algoritmo memético com população estruturada, na resolução do problema do escalonamento *job shop*, é bem eficiente, pois obteve um bom número de instâncias com valor ótimo aproximado do valor ótimo utilizado para comparação.

Conclusão

Neste trabalho foi apresentado o uso dos conceitos de otimalidade possível para analisar a possibilidade de um escalonamento ser ótimo e foi aplicado o algoritmo memético com população estruturada para evoluir bons escalonamentos e para determinar a função de *fitness* das soluções do problema. Com o uso do algoritmo memético, e dos conceitos de possibilidade foi possível resolver o problema de minimização do *makespan* no escalonamento do *job shop* com parâmetros *fuzzy*.

As contribuições inéditas neste trabalho são a aplicação dos conceitos de possibilidade na análise do grau de possibilidade de um escalonamento ser ótimo, e a forma de representação de um indivíduo no algoritmo memético com população estruturada.

Outras contribuições podem ser analisadas em (*Bonfim, T. R. e Yamakami, A. (2006)*). Através de simulações e comparações com os valores ótimos, verifica-se que a aplicação do algoritmo memético com população estruturada, juntamente com os conceitos de possibilidade, na resolução do problema, foi bem eficiente.

Referências Bibliográficas

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

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

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

Bonfim, T. R.; Yamakami, A. (2006). *Escalonamento memético e neuro-memético de tarefas*. 2006. 162 páginas. Tese de Doutorado em Engenharia Elétrica, Faculdade de Engenharia Elétrica e de Computação, UNICAMP, Campinas, Brasil.

Cantão, L. A. P. (2003). *Nonlinear Programming with Fuzzy Parameters: theory and algorithms*, PhD thesis, University of Campinas. 2003.

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

Dubois, D.; Prade, H. (1980). *Fuzzy Sets and Systems: theory and applications*, Academic Press. 1980.

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

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

Kaufmann, A.; Gupta, M. M. (1991). *Introduction to Fuzzy Arithmetic: theory and applications*, Van Nostre Reinhold. 1991.

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

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

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

Moscato, P.; Berretta, R. (1999). *The number partitioning problem: an open challenge for Evolutionary Computation*, McGraw-Hill, chapter 17. 1999.

Moscato, P.; Normam, M. (1992). *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 e Transportation Applications. 1992.

Muth, J. F.; Thompson, G. L. (1963). *Industrial Scheduling*. Prentice Hall, 1963.

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