Submetido para TEMA

Algoritmo Híbrido para Resolver o Problema de Escalonamento Job Shop com Incertezas

Resumo. O problema de escalonamento do tipo *Job shop* é considerado *NP*-difícil. Em aplicações reais, o tempo de processamento de cada tarefa é muitas vezes impreciso. Por isso, neste trabalho é abordamos o problema de escalonamento *Job shop* com tempo de processamento *fuzzy* (JSSPF). O tempo de processamento de cada tarefa é modelado por números triangular *fuzzy* e o objetivo do problema é encontrar um escalonamento que minimize o *makespan fuzzy* do problema. Na abordagem proposta trabalhamos com o algoritmo memético (MA) e o algoritmo de sistema de colônia de formigas (ACS) para resolver o problema. Uma hibridização destas duas abordagens denominada MA-ACS(CC-MO) é proposta para resolver o problema *fuzzy*. Para comparar a eficiência da abordagem é realizada uma comparação entre três algoritmos AG-ACS, MA-ACS(MO) e MA-ACS(CC-MO), utilizando 8 problemas do OR-Library.

Palavras-chave. Problema de Escalonamento *Job Shop Fuzzy*, Sistema de Colônia de Formigas, Algoritmo Memético.

1. Introdução

Muitos métodos propostos na literatura baseiam-se na condição de que todos os dados do problema são conhecidos. Entretanto, esta suposição pode trazer dificuldades na prática, pois em problemas reais pode haver uma incerteza em um número de fatores, como disponibilidade de equipamentos, tempos de processamento, tempos de transporte e custos. Assim, nestes casos, as soluções geradas usando modelos com incertezas certamente deixará o problema mais próximo da realidade.

A lógica *fuzzy* introduzida por Zadeh em 1965 [12] é um meio de aproximar a precisão da matemática clássica à *inexatidão* do mundo real. Esta teoria consegue manipular e operar quantidades exatas e inexatas quantificadas através de valores linguísticos.

O problema de escalonamento *fuzzy* foi formulado por Dubois et al. em 1995 [4], como um problema de otimização com satisfação de restrições *fuzzy*, onde as restrições associadas ao problema são representadas por um conjunto *fuzzy*. No problema de escalonamento *job shop fuzzy* (JSSPF), os tempos de processamento das operações são imprecisos, desta forma a noção de "escalonamento ótimo"é considerada também imprecisa já que um escalonamento é avaliado através de um número *fuzzy*. Neste caso, a noção de escalonamento ótimo é avaliado seguindo um critério de ordenação entre os valores dos escalonamentos. Esse critério é importante e pode

ser utilizado em situações onde se tem vários escalonamentos e se quer saber qual é o melhor entre eles.

O JSSPF é um problema de alta complexidade computacional, visto que os tempos de processamentos das tarefas são números *fuzzy* e a comparação entre estes é dificíl, sendo às vezes impossível determinar qual o melhor escalonamento. Devido a isso, muitos trabalhos utilizam índices de *defuzzificação* existentes na literatura, para assim resolver um problema clássico (*crisp*). Outros utilizam algum método de ordenação (rankeamento) de números *fuzzy* que associa a cada número *fuzzy* um valor *crisp* e com este valor resolvem um problema clássico associado. Ou ainda, utilizam métodos de ordenação lexicográfica ou comparação baseada em α-cortes com o objetivo de otimizar um escalonamento em termos do *makespan fuzzy*.

Dentre os principais trabalhos da literatura, alguns que consideramos importantes são: Fortemps (1997), Lin (2000), Bonfim (2006), González et al. (2007) e Niu et al. (2008).

Uma das primeiras aplicações significativa que considera a incerteza nos parâmetros de tempo foi o trabalho de Fortemps [5]. Ele resolve o problema de *job shop* modelando o tempo de duração das tarefas como números de seis-pontos *fuzzy* com o objetivo de minimizar o *makespan*. O *makespan* é obtido através da comparação de números *fuzzy* utilizando o algoritmo *Simulated Annealing*.

Em Lin [8], as incertezas nos tempos de processamento são modeladas usando tanto números triangulares *fuzzy* como λ-*cortes*. Os autores minimizam o *makespan* resolvendo o problema do *job shop crisp* que resulta da *defuzzificação* dos tempos de processamento.

Bonfim [2], resolve o problema utilizando um algoritmo memético com população estruturada e utiliza o conceito de possibilidade para avaliar qual é o *makespan* de cada indivíduo. Neste trabalho o *makespan* é calculado através de uma janela de tempo onde cada valor é representado por um número *crip* associado.

No trabalho de González et al. [6], os autores propõem um algoritmo memético para resolver o problema de *job shop fuzzy* e introduz um modelo de escalonamento baseado no valor experado do *makespan* ($E[C_{max}(x, \xi, \nu)]$). Este valor experado associa a cada variável *fuzzy* um valor esperado e assim resolve o problema com o objetivo de minimizar o *makespan* esperado.

Niu et al. [9], propõem um algoritmo de *Particle Swarm* combinado com operadores genéticos denominado GPSO, para resolver o problema de *job shop* com tempo de processamento *fuzzy*. Neste trabalho, o tempo de processamento das operações é descrito por números triangulares *fuzzy*, o objetivo do problema é encontrar um escalonamento que minimize o *makespan* e sua incerteza. Os autores utilizam um método de classificação de números *fuzzy* para estimar o valor do *makespan fuzzy* e sua incerteza.

Os trabalhos citados acima, possuem a desvantagem de lidarem somente com um problema associado, ou seja, em todos os trabalhos é utilizado algum método que associa um número real ao número *fuzzy* para então encontrar o *makespan* do problema. Neste trabalho é proposto um algoritmo que contorna esta dificuldade, mantendo as incertezas em todo processo de resolução do problema.

2. Modelagem matemática do *job shop fuzzy*

Seja m um conjunto de máquinas e n um conjunto de tarefas. Seja $O = 0, \cdots, n-1$ o conjunto de operações. 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. 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 $\tilde{ti}_{il}$, 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* $\tilde{M}$. A variável $\tilde{M}$ representa o tempo final $\tilde{tf}_{il}$ 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*. O modelo matemático para este problema é caracterizado como a seguir.

$$\text{Min } \tilde{M} = \underbrace{MAX}_{1 \leq l \leq m} \{\tilde{tf}_{il}\}, \forall i;\ 0 \leq i \leq n-1$$

$$\text{s.a: } \begin{array}{l} \tilde{ti}_{jk} - \tilde{ti}_{ik} \geq p_i; \forall (i,j) \in A, \forall i; 0 \leq i \leq n-1, \forall j; 0 \leq j \leq n-1 \\ \tilde{ti}_{jk} - \tilde{ti}_{ik} \geq p_i; \text{ ou } \tilde{ti}_{ik} - \tilde{ti}_{jk} \geq p_j; \forall (i,j) \in E_k, \forall k; 0 \leq k \leq m-1 \\ \tilde{ti}_{ik} \geq 0; \forall i \in O, \forall i; 0 \leq i \leq n-1, \forall k; 0 \leq k \leq m-1 \end{array}$$

Neste modelo matemático, a primeira restrição assegura que a sequência 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.

2.1. Tempo de processamento *fuzzy*

Em aplicações da vida real, frequentemente nos deparamos com situações onde o tempo que uma tarefa leva para ser processada numa determinada máquina não é conhecido. Desta forma, na literatura é comum utilizar números *fuzzy* para representar um tempo de processamento incerto. Esta incerteza dentro da teoria *fuzzy* pode ser modelada utilizando números triangulares *fuzzy* (NTF) da forma $\widetilde{A} = (a_i, a_m, a_s)$, onde a_m é o valor modal, e os espalhamentos inferiores e superiores são a_i e a_s, respectivamente.

A Tabela 1 apresenta uma instância gerada para o problema com 3 tarefas e 3 máquinas com tempos de processamento incerto. Para a geração desses números, utilizou o benckmark [11], onde o valor modal a_m corresponde ao valor *crip* deste benckmark e, os espalhamentos a esquerda $(a_m - a_i)$ e a direita $(a_s - a_m)$ foram gerados segundo uma distribuição uniforme no intervalo [0,1].

Tarefas	Ordem das operações (máquina, tempo de processamento)		
1	$O_{11}(1,[2.40,3,3.23])$	$O_{12}(2,[2.45,3,3.35])$	$O_{13}(3,[2.00,3,3.36])$
2	$O_{21}(1,[1.18,2,2.79])$	$O_{23}(3,[2.22,3,3.44])$	$O_{22}(2,[3.37,4,4.31])$
3	$O_{32}(2,[2.41,3,3.44])$	$O_{31}(1,[1.63,2,2.31])$	$O_{33}(3,[0.96,1,1.61])$

Tabela 1: Instância do problema *job shop fuzzy* (3×3).

Na instância exemplificada acima, a ordem de processamento das tarefas nas máquinas é pré-estabelecida. De acordo com a Tabela 1, a tarefa 1 precisa ser processada inicialmente na máquina 1, com um tempo *fuzzy* de [2.40,3,3.23], em seguida pela máquina 2, com tempo *fuzzy* de [2.45,3,3.35] e assim por diante. Um escalonamento é factível quando a ordem das operações é preservada.

2.2. Relação de ordem

Para determinar o *makespan* do problema do *job shop fuzzy* é utilizado o algoritmo proposto por Hernandes, com algumas modificações. Este algoritmo tem como finalidade encontrar todos os caminhos não-dominados entre o nó origem e o nó destino, e utiliza a relação de ordem proposta por Okada e Soper [10]. No algoritmo proposto, utiliza-se números triangulares *fuzzy* e é determinado o seguinte critério de dominância parcial *fuzzy* [10].

Definition 2.1. *(Dominância Parcial) Sejam* $\widetilde{A} = (a_i, a_m, a_s)$ *e* $\widetilde{B} = (b_i, b_m, b_s)$ *dois números triangulares fuzzy e* $\varepsilon \in [0,1]$*, então* $\widetilde{A}$ *domina* $\widetilde{B}$ *com grau* ε*, denotado por* $\widetilde{A} \prec_\varepsilon \widetilde{B}$*, se e somente se* $a_m \leq b_m$*,* $(a_i)_\varepsilon \leq (b_i)_\varepsilon$*,* $(a_s)_\varepsilon \leq (b_s)_\varepsilon$ *e* $\widetilde{A} \neq \widetilde{B}$*.*

2.3. Comparação de números triangulares *fuzzy*

A comparação entre os números triangulares *fuzzy* é um recurso utilizado no desenvolvimento do algoritmo proposto neste artigo, pois é necessário, entre vários números *fuzzy*, definir qual é o maior ou menor. O método de comparação utilizado (Bortolan et al. [3]), consiste em definir um número real que represente o número triangular *fuzzy*. Uma observação importante, é que este critério de ordenação é utilizado apenas para ordenar a população de acordo com seu valor de *makespan fuzzy*.

3. Modelagem por grafo disjuntivo *fuzzy*

A característica *fuzzy* de um problema pode ser encontrada em diversos níveis: da estrutura do grafo (nós e arestas) aos parâmetros associados ao grafo (custo, tempo, etc). Problemas com estrutura do grafo *crisp* e parâmetros *fuzzy* é um dos mais estudados da literatura. Estes são problemas em que a estrutura do grafo é bem conhecida e rígida e os parâmetros associados são representados por números *fuzzy*.

Um dos problemas de grafos com parâmetros *fuzzy* mais estudados na literatura é o de caminho mínimo com parâmetros incertos. Um outro problema de otimização que pode ser modelado na forma de grafo com parâmetros *fuzzy*, bastante complexo e estudado na literatura, é o problema de escalonamento *job shop*.

Inicialmente estes dois problemas não apresentam nenhuma relação em comum, já que o objetivo do problema de caminho mínimo é encontrar um menor caminho entre dois nós do grafo, ou seja, minimizar a função objetivo ($min\{f(x)\}$) e o objetivo do problema de *job shop* é minimizar o *makespan* (caminho máximo) do grafo, ou seja, $min(max\{f(x)\})$. Entretando, podemos estabelecer uma relação entre estes dois problemas, o que é bastante interessante pois o problema de caminho máximo é NP-completo e não existe algoritmo exato para resolvê-lo.

No trabalho de Hernandes (2007) [7] é proposto um algoritmo de caminho mínimo, baseado no algoritmo clássico de Bellman-Ford-Moore (FBM), que pode ser aplicado em grafos com parâmetros negativos *fuzzy*. Desta forma, fizemos as modificações necessárias e utilizamos este algoritmo para calcular o caminho crítico (*makespan*) para o problema do *job shop fuzzy*.

Na modelagem do problema do *job shop fuzzy* através do grafo disjuntivo com parâmetros incertos, os tempos de processamento das tarefas nas máquinas é representado por números *fuzzy* (Figura 1). Na literatura não foi encontrado nenhum trabalho que resolve o problema de *job shop fuzzy* modelado através do grafo disjuntivo com parâmetros incertos, sendo esta mais uma justificativa para o algoritmo proposto neste trabalho.

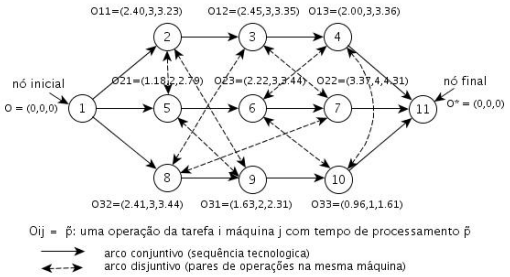

Figura 1: Grafo disjuntivo com parâmetros *fuzzy* do JSSPF ($n = 3$ e $m = 3$).

Inicialmente o grafo disjuntivo *fuzzy* não representa uma solução para o JSSPF. Quando todas as direções dos arcos disjuntivos são definidas, obtém-se um grafo direcionado, se este grafo direcionado for acíclico, temos uma solução (escalonamento) factível para o problema. Para calcular o *makespan*, basta calcular o caminho crítico do grafo acíclico *fuzzy*. Este caminho é determinado pela maior distância entre o nó inicial e o nó final do grafo com parâmetros incertos e será calculado pelo algoritmo descrito na seção 3.1..

3.1. Algoritmo para calcular o *makespan* do problema *job shop fuzzy*

Hernandes propõe um algoritmo para encontrar o caminho mínimo em grafos com parâmetros *fuzzy*. Tal algoritmo é baseado no algoritmo clássico de Bellman-Ford-Moore [1] e utiliza a relação de ordem (descrita na seção 2.2.) para determinar o conjunto de caminhos não-dominados. Desta forma, iremos adaptar o algoritmo proposto por Hernandes a fim de calcular o *makespan* do problema *job shop fuzzy*.

4. Abordagem proposta

Devido a complexidade do problema de escalonamento do tipo *job shop*, as estratégias heurísticas são métodos recomendados para resolvê-lo. Se tratando do problema onde o tempo de processamento é representado por números *fuzzy*, essa complexidade aumenta, sendo necessário uma reformulação significativa do problema e métodos eficientes para solucioná-lo.

Desta forma, neste trabalho propomos um algoritmo híbrido, que trabalha com o algoritmo de colônia de formigas(ACS) e o algoritmo genético (AG) com busca local, para resolver o problema de escalonamento *job shop* com tempo de processamento incerto. As técnicas de buscas locais serão descritas na Seção 4.1.. O algoritmo de colônia de formigas é utilizado para criar uma população inicial de escalonamentos factíveis e o algoritmo genético com busca local é utilizado para evoluir esses escalonamentos.

O algoritmo híbrido é utilizado para encontrar boas soluções para o problema e, para avaliar a melhor solução é utilizado o método de comparação descrito na Seção 2.3., que têm a finalidade de definir entre vários números *fuzzy* qual é o maior ou menor.

4.1. Algoritmo híbrido de sistema de colônia de formigas e genético para o problema do *job shop fuzzy*

Um conceito importante e crítico na execução de um algoritmo é a escolha da representação de possíveis soluções para o problema. Neste trabalho, o problema do *job shop* com parâmentros incertos (Tabela 1) é representado por um grafo disjuntivo *fuzzy* (Figura 1) e, cada solução é representada através de um vetor V de $n.m + 2$ colunas, onde n é o número de tarefas e m o número de máquinas. Cada campo do vetor v_k com $1 \leq k \leq n.m + 2$, é preenchido por um número k que indica o número do nó no grafo disjuntivo. Este vetor possui uma sequência que corresponde a ordem que as tarefas são atendidas pelas máquinas. Como dito anteriormente, na literatura não foi encontrado nenhum trabalho que resolve este problema através desta representação *fuzzy*, sendo essa mais uma razão para esta abordagem.

1. Geração da população inicial: O primeiro passo do algoritmo híbrido é a geração da população inicial e esta foi gerada utilizando o algoritmo de colônia de formigas.

2. Calculo do *makespan*: Para calcular o *makespan* do JSSPF utilizamos o algoritmo descrito na seção 3.1.. Este algoritmo encontra todos os caminhos não-dominados entre o nó inicial e o nó final, sendo necessário a utilização de um método capaz de decidir qual é o maior entre os valores encontrados, pois este representa o *makespan* do JSSPF.

3. Passos seguintes do algoritmo híbrido: Os passos seguintes do algoritmo híbrido, onde são executados os operadores genéticos e os métodos de busca local, são realizados enquanto o critério de parada não for alcançado. O critério

de parada utilizado foi o número de gerações. Os operadores genéticos executados neste algoritmo foram o *crossover* e a mutação, eles são aplicados no grafo disjuntivo *fuzzy*.

4. Operadores genéticos: O *crossover* é realizado entre dois indivíduos pais da população, gerando dois novos indivíduos filhos. Nesta operação genética, é sorteado uma tarefa dos indivíduos pais, e a ordem de localização que estas tarefas aparecem nos indivíduos filhos é preservada. O restante do cromossomo do filho é preenchido de acordo com a ordem que as outras tarefas aparecem no outro pai.

A mutação é realizada em um indivíduo pai, gerando um indivíduo filho. Neste trabalho a mutação utilizada foi a denominada mutação inversiva. Nela escolhe-se aleatoriamente duas sequências de tarefas e faz a troca entre elas. As demais tarefas do cromossomo permanecem na mesma ordem. Os operadores de *crossover* e mutação são aplicados com probabilidades P_c e P_m respectivamente.

5. Metodos de busca local: Na literatura, existem diferentes regras de busca local que são adicionadas ao algoritmo genético, para melhorar a qualidade das soluções do problema *job shop*. Neste artigo, o algoritmo proposto inicialmente cria uma população de soluções factíveis, ordena de acordo com seu valor de *fitness*, e então, aplica-se os operadores de *crossover* e mutação para gerar a população da próxima geração.

O objetivo da busca local é melhorar uma solução obtida na população inicial ou pelos operadores de *crossover* e mutação. Por isso, a busca local é realizada apenas no melhor indivíduo da população, sendo que, este novo indivíduo só é aceito para a próxima geração se melhorar a qualidade da solução corrente, caso contrário ele é descartado. Neste trabalho propomos duas técnicas de buscas locais e elas serão descritas a seguir.

1. *Troca de Tarefas Adjacentes no Caminho Crítico (CC)*

Nesta regra de busca local, calculamos o caminho crítico em cada solução. A seguir, dentro deste caminho crítico, são identificados os pares de tarefas adjacentes pertencentes a mesma máquina e então é realizada a mudança de direção entre os arcos que ligam estas tarefas adjacentes.

Observe que esta mudança de direção dos arcos pertencentes ao caminho crítico de cada solução, representa a mudança de direção dos arcos disjuntivos (arcos pertencentes à mesma máquina) no gráfico disjuntivo *fuzzy*. É importante lembrar que esta mudança de direção dos arcos, pode resultar em um escalonamento infactível. Por esta razão, antes de aceitar a troca dos arcos, é necessário verificar a factibilidade do escalonamento. Na Figura 2, ilustramos um exemplo de solução para o problema da Tabela 1.

Seja, V_1 uma solução do problema.

$$V_1 = (1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 6 \rightarrow 4 \rightarrow 9 \rightarrow 7 \rightarrow 10 \rightarrow 11)$$

e seja cc_1 seu caminho critíco calculado através do gráfico disjuntivo (Fig. 2(a)).

$$cc_1 = (1 \rightarrow 5 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 9 \rightarrow 10 \rightarrow 11)$$

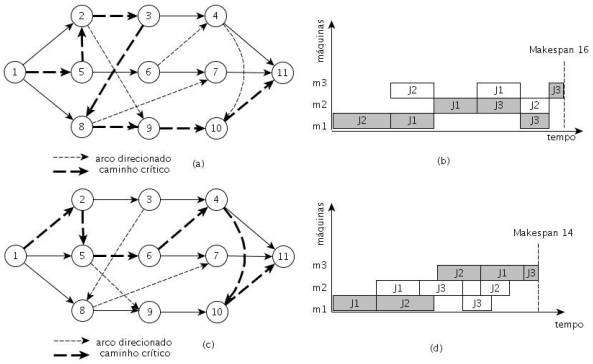

Figura 2: Gráfico de Gantt e grafo disjuntivo: (a), (b) antes de aplicar a busca local, (c) e (d) depois de aplicar a busca local CC.

O *makespan* de V_1 é 16 unidades como ilustrado nos blocos críticos (blocos sombreados, Fig. 2(b)). A busca local CC é realizada nas tarefas adjacentes pertencentes à mesma máquina, no caso os nós 5 e 2. Uma mudança de direção entre os arcos que ligam estes nós (Fig. 2(c)) é realizada, dando origem a uma nova solução V_2.

$$V_2 = (1 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 8 \rightarrow 6 \rightarrow 4 \rightarrow 9 \rightarrow 7 \rightarrow 10 \rightarrow 11)$$

Esta nova solução (escalonamento) têm *makespan* de 14 unidades e caminho crítico cc_2, representado pelo grafo disjuntivo (Figura 2(c)) e pelos blocos sombreados (Figura 2(d)).

$$cc_2 = (1 \rightarrow 2 \rightarrow 5 \rightarrow 6 \rightarrow 4 \rightarrow 10 \rightarrow 11)$$

Como pode ser visto nas Figuras 2(a) e 2(c), a mudança de direção do arco que liga o nó 5 ao nó 2, resultou na inversão da direção entre os arcos que pertencem a m_1. Isto fez com que a solução resultante V_2 obtivesse um valor de *fitness* menor.

2. *Troca de Tarefa na Máquina mais Ociosa (MO)*

Resolver o problema do *job shop* geralmente gera soluções onde o escalonamento tem *gaps* entre as tarefas nas máquinas. Estes *gaps* surgem no escalonamento devido às restrições de precedências impostas às tarefas. Uma grande quantidade de *gaps* numa máquina faz com que ela fique ociosa, o que aumenta o valor de *makespan* do escalonamento.

A busca local MO, identifica a máquina ociosa no escalonamento e a tarefa candidata que será remanejada para dentro de um *gap*, a fim de obter um melhor escalonamento, reduzindo o valor do *makespan*. Esta regra de troca de tarefas adjacentes na máquina somente será permitida se, o escalonamento resultante for factível.

A Figura 3 ilustra a busca local MO. No escalonamento representado pelo gráfico de Gantt(Figura 3(a)) a máquina ociosa é a m_3. Desta forma, a busca local será aplicada nesta máquina. Como o j_3 é processado após j_1 nesta máquina, podemos alocar o j_3 antes do j_1, como mostra a Figura 3(b). Como pode ser visto, a troca das tarefas na máquina mais ociosa, levou a uma redução no valor do *makespan* do novo escalonamento.

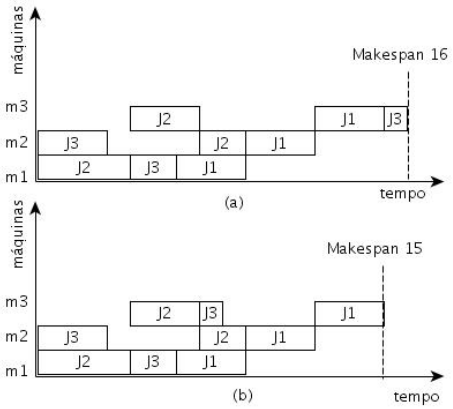

Figura 3: Exemplo para MO: (a) antes da busca local, (b) depois da busca MO.

Como saída do algoritmo é apresentado o melhor indivíduo da população. Este indivíduo tem seu valor de *fitness* representado por um número triangular *fuzzy*. Para apresentarmos seu valor *crisp*, foi aplicada a *defuzzificação*, que é o mapeamento de informações *fuzzy* em valores *crip*. O método de *defuzzificação* utilizado foi o método da centróide. Um pseudocódigo do algoritmo híbrido aplicado ao problema de escalonamento *job shop* com parâmetros incertos, é apresentado no Algoritmo 2.

Algoritmo 2: Pseudocódigo do algoritmo híbrido aplicado ao JSSPF

Entrada: m: número de máquinas, n: número de tarefas, $O_{m\times n}$: matriz com sequência de operações para cada tarefa, $o_{ij} = (m_{ij}, \widetilde{p}_{ij})_{1\leq i\leq n, 1\leq j\leq m}$: par ordenado de cada operação composto pela máquina que irá processa-lá e o tempo de processamento da operação, T_{pop}: tamanho da população, N_{it}: número de iterações, N_{form}: número de formigas, ρ: nível de feromônio, β: informação heurística, α: informação de feromônio, q_0: informação heurística, N_{gera}: número de gerações, P_c: probabilidade de *crossover* e P_m: probabilidade de mutação.

- Gera a população inicial através do algoritmo de colônia de formigas;
- Calcular o valor de *makespan* de cada indivíduo;
- Ordena a população utilizando o valor de *makespan* através do método descrito na Seção 2.3.;

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

- Selecionar 2 indivíduos da população inicial;
- Aplicar *crossover* gerando 2 novos indivíduos para a memória;
- Selecionar 1 indivíduo da população inicial;
- Aplicar mutação gerando 1 novo indivíduo para a memória;
- calcular o valor de *makespan* da memória, através do algoritmo descrito na seção **??**;
- Aplicar supressão na memória e verificar diversidade;
- Ordena a população utilizando o valor de *makespan* através do método descrito na Seção 2.3.
- Selecionar os x% melhores indivíduos da memória para a próxima geração;
- Atualizar a memória da próxima geração completando com novos indivíduos, se necessário;
- Aplicar busca local no melhor indivíduo da população e atualizar seu valor de *fitness*.

fim para

Saída: Melhor escalonamento da população e seu valor de *makespan*.

5. Resultados numéricos da abordagem *fuzzy*

Neste capítulo são apresentados e discutidos os resultados obtidos com a aplicação do algoritmo híbrido MA-ACS(CC-MO) proposto neste trabalho, para resolver o JSSPF. Para comparar a eficiência de MA-ACS(CC-MO), os resultados são comparados com o algoritmo AG-ACS(algoritmo genético com população inicial gerado pelo ACS) e MA-ACS(MO). Todos os experimentos foram realizados com os mesmos parâmetros. Na avaliação da abordagem, foram utilizadas 8 instâncias com diferentes números de tarefas e máquinas, extraídas do OR-Library [11]. As instâncias são caracterizadas pelas seguintes informações: número de tarefas, número de máquinas e uma tabela contendo a sequência de operações de cada tarefa, incluindo o número da máquina que irá processá-la e o tempo de processamento da operação. Como aqui os tempos de processamento das tarefas são considerados parâmetros com incertezas, então, foram gerados os números *fuzzy* com espalhamentos à esquerda e à direita como descrito na Seção 2.1..

Cada experimento foi executado 10 vezes e implementado em Matlab 8 com a plataforma Linux, Intel Core 2 Duo 2.0GHz e 4Gb. Os parâmetros adotados na execução dos algoritmos são apresentados a seguir.

- Número de formigas $= 15$, $\alpha = 0,1$, $\beta = 2$, $\rho = 0,01$, $q_0 = 0,7$.
- Número de gerações: 500, tamanho da população: 40 indivíduos, probabilidade de crossover: $0,8$ e probabilidade de mutação: $0,6$.
- Supressão: elimina indivíduos de mesmo *makespan*, diversidade: Insere novos indivíduos criados pelo ACS, taxa miníma de diversidade populacional: 50%.

Na Tabela 2, são apresentados todos os resultados encontrados pelos algoritmos AG-ACS, MA-ACS(MO) e MA-ACS(CC-MO). Nela são apresentados os valores do *makespan fuzzy* encontrados por cada algoritmo, a *defuzzificação* pelo método da centróide dos *makespan fuzzy* e os valores dos *makespan crisp* de cada problema estraído do Or-Library.

De acordo com a Tabela 2, a abordagem proposta MA-ACS(CC-MO) mostra desempenho superior sobre AG-ACS e MA-ACS(MO), especialmente quando a dimensão do problema é grande ou o problema é muito difícil, como La23, mas quando o tamanho do problema é pequeno, como a instância do Ft06 os resultados são semelhantes.

Na execução de cada instância, foram verificados quantos indivíduos ficaram com seus valores até 80% do valor do *makespan* ótimo encontrado pelo algoritmo. A Tabela 3 apresenta a quantidade de indivíduos com 80% de possibilidade de serem ótimos para cada instância testada.

Pela Tabela 3, podemos verificar quantos indivíduos tiveram seus valores de *makespan* até 80% do valor do melhor indivíduo encontrado pelos algoritmos. Isso é um ponto muito positivo, pois mostra que o algoritmo encontra não apenas uma solução ótima, mas um conjunto de soluções com alto grau de possibilidade de serem ótima.

Problema e tamanho	**Algoritmo**	**Makespan fuzzy**	**Defuzzi-ficação**	**Makespan crisp**
Ft06 (6x6)	AG-ACS	[50,6 55 57,75]	54,4	55
	MA-ACS(MO)	[50,6 55 57,75]	54,4	
	MA-ACS(CC-MO)	[50,6 55 57,75]	54,4	
La01 (10x5)	AG-ACS	[612,7 666 699,3]	659,3	666
	MA-ACS(MO)	[612,7 666 699,3]	659,3	
	MA-ACS(CC-MO)	[612,7 666 699,3]	659,3	
La08 (15x5)	AG-ACS	[793,9 863 906,1]	854,3	863
	MA-ACS(MO)	[793,9 863 906,1]	854,3	
	MA-ACS(CC-MO)	[793,9 863 906,1]	854,3	
La11 (20x5)	AG-ACS	[1124,2 1222 1283,1]	1209,8	1222
	MA-ACS(MO)	[1124,2 1222 1283,1]	1209,8	
	MA-ACS(CC-MO)	[1124,2 1222 1283,1]	1209,8	
La17 (10x10)	AG-ACS	[729,56 793 832,65]	785,07	784
	MA-ACS(MO)	[724 787 826,35]	779,13	
	MA-ACS(CC-MO)	[722,2 785 824,2]	777,13	
Abz6 (10x10)	AG-ACS	[894,24 972 1020,6]	962,28	943
	MA-ACS(MO)	[871,2 947 994,3]	937,5	
	MA-ACS(CC-MO)	[871,2 947 994,3]	937,5	
Orb02 (10x10)	AG-ACS	[859,28 934 980,7]	924,66	888
	MA-ACS(MO)	[834,4 907 952,35]	897,93	
	MA-ACS(CC-MO)	[828, 900, 945]	891	
La23 (15x10)	AG-ACS	[1004,64 1092 1146,6]	1081,1	1032
	MA-ACS(MO)	[969,68 1054 1106,7]	1043,5	
	MA-ACS(CC-MO)	[966 1050 1102,5]	1039,3	

Tabela 2: Comparação dos resultados para minimizar o *makespan fuzzy*.

Instância	AG-ACS	MA-ACS(MO)	MA-ACS(CC-MO)
Ft06 (6×6)	3	3	6
La01 (10×5)	20	20	20
La08 (15×5)	22	20	22
La11 (20×5)	23	26	22
La17 (10×10)	21	19	21
Abz6 (10×10)	24	20	20
Orb02(10×10)	21	22	20
La23 (15×10)	21	21	20

Tabela 3: Quantidade de indivíduos com +80% de possibilidade de serem ótimos.

6. Conclusões

Verificando os resultados acima, observamos que os algoritmos propostos possuem a vantagem de lidar com o problema do *job shop* com parâmetros *fuzzy* na sua integra, sem necessidade de métodos de *defuzzificação* ou comparação de números *fuzzy* para encontrar o *makespan*. Uma outra vantagem bastante interessante e útil da abordagem proposta neste trabalho, é que os algoritmos são capazes de encontrar um conjunto de soluções com alto grau de possibilidade de serem ótimas.

Abstract. The *job shop* scheduling problem is considered *NP*-hard. In real applications, the processing time of each task is often imprecise. Therefore, this paper deals with the *job shop* scheduling problem with *fuzzy* processing time (JSSPF).The processing time of each task is modeled by triangular *fuzzy* numbers and aim of the

problem is to find a schedule that minimizes the *fuzzy makespan* of the problem. In this approach worked with the memetic algorithm (MA) and the algorithm of ant colony system (ACS) to solve the problem. A hybridization of these two approaches called MA-ACS (CC-MO) is proposed to solve the *fuzzy* problem. To compare the efficiency of the approach is realized a comparison between three algorithms AG-ACS, ACS-MA (MO) and MA-ACS (CC-MO), using eight of the OR-Library problems.

Referências

[1] Bellman, R. E. (1958). On a routing problem, *Quarterly Applied Mathematics* (16): 87–90.

[2] Bonfim, T. R. (2006). "Escalonamento Memético e Neuro-Memético de Tarefas". Tese de Doutorado da Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas.

[3] Bortolan, G. & Degani, R. (1993). A review of some methods for ranking fuzzy subsets, in: Dubois, D., Prade, H., Yager, R. (eds.), *Readings in Fuzzy Sets for Intelligent Systems*, Morgan Kaufmann, San Francisco, CA, pp. 149-158.

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

[5] Fortemps, P.(1997). Jobshop scheduling with imprecise durations: A fuzzy aproach. *IEEE Trans. Fuzzy Syst.*, vol.(4), pp. 557-569.

[6] González Rodrígues, I., Vela, C. R., Puente, J. (2007). A memetic approach to fuzzy job shop based on expectation model. In: Proceedings of IEEE International Conference on Fuzzy Systems, *FUZZ-IEEE*, London, pp. 692-697.

[7] Hernandes, F. (2007). "Algoritmos para Problemas de Grafos com Incertezas". Tese de doutorado da Faculdade de Engenharia Elétrica e Computação, Universidade Estadual de Campinas.

[8] Lin, F.-T. (2000). Fuzzy job-shop scheduling based on ranking level $(\lambda, 1)$ interval-valued fuzzy numbers, *IEEE Trans. Fuzzy Syst.*, vol.10(4), pp. 510-522.

[9] Niu Q., Jiao B., Gu X. (2008). Particle swarm optimization combined with genetic operators for job shop scheduling problem with fuzzy processing time. *Applied Mathematics and Computation*, 205 (1), pp. 148-158.

[10] Okada, S. e Soper, T. (2000). A shortest path problem on a network with fuzzy arc lengths. *Fuzzy Sets and Systems*, 109, pp. 129-140.

[11] Tamura, N.(2007). OR-library. Available online at: http://bach.istc.kobe-u.ac.jp/csp2sat/jss/. Accessed 15 july 2007.

[12] Zadeh, L. (1965). Fuzzy Sets. *Information and Control*, 8:338-353.