Algoritmo *Particle Swarm Optmization* para resolução do problema da programação da produção Job-shop Flexível

Diego L. Cavalca, *Universidade Federal de São Carlos*

Resumo — A programação, ou escalonamento, da produção é um fator diferencial que pode aumentar a competividade industrial de diversos setores produtivos, através da mitigação de processos ineficientes. Esta programação visa otimizar os custos envolvidos na produção de produtos através de complexas análises combinatórias dos recursos envolvidos na manufatura, evitando gargalos operacionais que geram custos passíveis de correções neste processo. Na literatura é proposta uma modelagem deste problema de produção, o qual é chamado Job-shop Flexível (inglês FJSP), onde estabelece, além de outros fatores, que uma tarefa designada à produção pode ser processada por qualquer máquina em um determinado conjunto de opções de recursos produtivos disponíveis, gerando aumento significativo de soluções factíveis. Portanto, este trabalho apresenta uma solução computacional híbrida para o FJSP, apoiado na abordagem hierárquica, sendo o algoritmo resultante baseado em otimização por enxame de partículas (do inglês Particle Swarm Optmization, *PSO*) para a resolução do subproblema de roteamento e o método de busca local por arrefecimento simulado (do inglês *Simulated Annealing*, SA) para o subproblema de programação.

Palavras-chave—Produção, Otimização, Job-shop flexível, FJSP, Enxame de partículas, Arrefecimento Simulado.

1 Introdução

A programação, ou escalonamento, da produção é um fator diferencial que pode agregar a competitividade das empresas de diversos setores produtivos. Esta programação visa otimizar os custos envolvidos na produção de produtos através de complexas análises combinatórias dos recursos envolvidos na manufatura, mitigando as possibilidades de gastos envolvidos neste processo.

O problema de programação Job-Shop (do inglês, *Job-Shop Scheduling Problem*, ou JSP), é um ramo deste cenário que está entre os mais difíceis problemas de otimização combinatória conhecidos. [1] Esta abordagem consiste em programar um conjunto de tarefas em uma série de máquinas com o objetivo de minimizar um determinado critério de produção, sujeito à restrição de que cada tarefa tem uma ordem de processamento especificada *a priori*, além de que esta pode ser atribuída a todas as máquinas previamente estabelecidas. Assim, o JSP é um problema NP-difícil, que envolve cálculos combinatórios altamente complexos para a resolução desta abordagem.

Além do JSP clássico, que não possui uma solução ótima conhecida, uma nova versão ainda mais complexa deste problema vem chamando a atenção da comunidade científica, o qual é conhecido como programação Job-Shop Flexível (do inglês *Flexible Job-Shop Scheduling*).

No FJSP, além das características e restrições supracitadas presentes no JSP, a complexidade devido a detalhes adicionais. Conforme uma tarefa é designada à produção, esta pode ser processada por qualquer máquina em um determinado conjunto de opções de recursos produtivos disponíveis, respeitando as condições atuais desta, resultando em um aumento significativo de soluções factíveis.

Conforme [2], o FJSP a solução deste pode ser orientada à decomposição do problema original em dois subproblemas principais, o de roteamento e o de programação. O subproblema de roteamento consiste na atribuição individual das operações em um recurso dentre um conjunto de recursos válidos para a respectivas operações. O subproblema de programação é extraído a partir do subproblema supracitado e envolve a organização das operações atribuídas aos recursos, visando minimizar o objetivo global pré-estabelecido em detrimento de um cronograma produtivo factível. Esta abordagem subdividida é denominada representação.

Neste contexto de subproblemas decompostos, segundo [2], a resolução destes pode ser obtida através de dois métodos de otimização global, atuando de maneira individualizada.

Portanto, este trabalho visa apresentar uma solução computacional híbrida para o FJSP, apoiado na abordagem hierárquica, sendo o algoritmo resultante baseado em otimização por enxame de partículas (do inglês *Particle Swarm Optmization*, *PSO*) para a resolução do subproblema de roteamento e o método de busca local por arrefecimento simulado (do inglês *Simulated Annealing*, SA) para o subproblema de programação.

2 Modelagem do problema

No contexto deste trabalho, o FJSP será modelado de forma a conter um conjunto de n tarefas (*jobs*) a ser processadas em m recursos (máquinas), disponíveis dentro de um conjunto M destes. Cada job j é composto de uma sequência de n_j operações, $O_{j,1}$, $O_{j,2}$, …, $O_{j,nj}$. Assim, para um job ser complete, todas as operações que o compõe devem ser processadas.

A execução de cada operação i (denotada por $O_{j,i}$) deve ser realizada por uma máquina $M_{j,i}$ pertencente ao conjunto M. Portanto, o objetivo é determinar a atribuição

e a ordem de procedência de cada operação em uma específica máquina a fim de minimizar um determinado critério produtivo.

Neste trabalho, o critério de minimização a ser adotado será o tempo de completude de todas as operações, denominado na literatura por *makespan*.

Com base em [1], este trabalho se guiará obedecendo as seguintes hipóteses no universo do problema de FJSP:

- O tempo de preparação e alternância das operações entre as máquinas são desconsiderados,
- As máquinas são independentes entre si no que tange os processamentos individuais,
- Os *jobs* são independentes entre si,
- Uma máquina executa uma, e apenas uma, operação por vez, estando disponível para novo processamento apenas ao término desta,
- Não há ordem de precedência na execução de operações entre *jobs* distintos.

3 PROPOSTA DE ALGORITMO

O *PSO*, segundo [3], é uma técnica computacional evolucionária que foi desenvolvida inspirado no comportamento de enxames de animais, como aves e abelhas, que utilizam a inteligência do enxame a fim de garantir a sobrevivência do bando. Neste sentido, o *PSO* busca resolver, de maneira heurística, um problema de otimização diante de uma medida de qualidade pré-estabelecida. Assim, considerando um grupo de agentes (enxame) explora iterativamente o espaço de busca, dado que cada indíviduo é considerado independente neste 'vôo', dado que a nuvem é regida pelas experiências obtidas pelos agentes.

Conforme citado, este trabalho apresenta uma versão híbrida do algoritmo *PSO* a fim de otimizar o *makespan* do FJSP, a partir da abordagem hierárquica deste problema.

3.1 *PSO* clássico

Partindo de uma população inicial aleatória, ou nuvem de partículas, o *PSO* considera cada elemento desta uma solução em potencial, dado que cada partícula tem a capacidade de 'voar' pelo espaço D-dimensional do problema explorado. Mimetizando o comportamento natural conhecido, esta exploração individual é regida por uma velocidade de vôo que é dinamicamente ajustada de acordo com, além da própria intuição individual, a melhor experiência pessoal da partícula (chamada de *pBest*) bem como a experiência global da nuvem (*gBest*) os quais consideram os melhores caminhos explorados dentro de cada contexto.

Conforme propõe [2], a fim de ponderar estes fatores, o movimento da nuvem é regido pelas equações:

$$V_{id} = W*V_{id} + C1*Rand()*(pBest_{id} - X_{id}) + C2*Rand()*(gBest - X_{id});$$

e

$$X_{id} = X_{id} + V_{id}.$$

A primeira equação, denominada aqui 1a, diz respeito ao cálculo da velocidade do indivíduo, a qual leva em conta:

- Primeiro termo: a exploração do espaço de busca pelo próprio indivíduo, chamado de fator de inércia,
- Segundo termo: a direção em relação a melhor experiência individual, ponderada por um fator aleatório [0,1] e um valor constante C1, representando o peso estocástico do indivíduo seguir em direção ao *pBest*,
- Terceiro termo: a direção em relação a melhor experiência conhecida pela nuvem até então, ponderada também por um valor aleatório entre 0 e 1, além de um valor constante estocástico C2 (similar a C1, mas em relação ao *gBest*) definido *a priori*.

Após o cálculo da equação 1a, o indíviduo já está pronto para se deslocar no espaço, o qual é consumado através da segunda equação apresentada, denominada aqui 1b.

Conforme pode ser observado na figura 1, diante das equações apresentadas, a nuvem possui a capacidade de explorar o espaço do problema estudado, convergindo para uma solução aproximada à qual se objetiva.

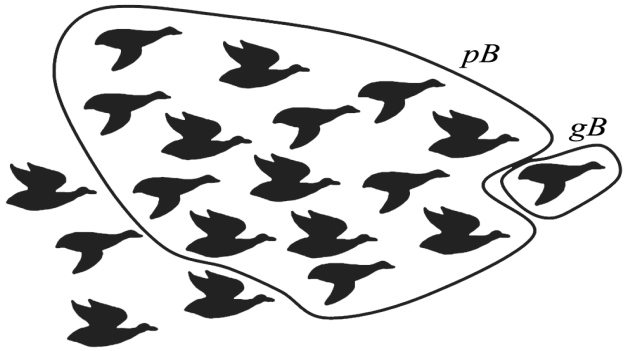

Fig. 1: Comportamento migratório da 'nuvem', explorando o espaço de busca do problema em busca da solução ótima. Adaptado de [2].

Portando, segundo [3], o algoritmo *PSO* é estruturado nas seguintes etapas:

1. Inicializa uma nuvem de partículas aleatórias,
2. Avalie a aptidão de cada partícula diante da função objetivo do problema em questão,
3. Compare os valores de aptidões das partículas com as respectivas aptidões estabelecidas no $pBest_{id}$. Se este valor for melhor, guarde este e também a posição da partícula em questão.
4. Compare os valores em *pBest* com o melhor valor encontrado pela nuvem até então (*gBest*). Se encontrar uma melhor aptidão, o valor e a posição desta serão os novos *gBest*,
5. Atualize a velocidade e posição de cada partícula, segundo as equações 1a e 1b.
6. Repita o passo (2) até um determinado critério de parada ser atingido, geralmente um valor ótimo de aptidão ou um determinado número de iterações.

3.2 Aplicação do *PSO* no FJSP

Conforme expõe [1], o sucesso do *PSO* para a resolução do FJSP está diretamente relacionada ao desenvolvimento de mecanismos eficientes de 'mapeamento do problema' e 'geração de soluções'. Tal afirmação fica comprovada ao levar em consideração o fato de que o *PSO* clássico foi desenvolvido para atuar em problemas contínuos, os quais são opostos ao estudado neste trabalho, o qual o FJSP denota-se um problema discreto. Todavia, de acordo com a implementação e ajustes citados, o desenvolvimento de um algoritmo híbrido permite que o *PSO* atinja resultado satisfatório na resolução de problemas deste tipo.

Seguindo a proposta de [1], a partícula será representada por um vetor de *i* colunas - correspondentes ao total de operações a ser otimizado – as quais terão como valores as máquinas designadas para o processamento de cada operação $O_{j,i}$. Diante deste pressuposto, a composição da partícula aqui abordada se dá através do processo estocástico de designação de uma máquina M_i factível para a operação, dado que existe uma lista ordenada de máquinas de acordo com o tempo de processamento que estas levam para a execução da operação em questão. Como exemplo, pode ser observado na tabela 1 a distribuição das máquinas $M_{j,i}$ para cada operação $O_{j,i}$. Neste exemplo, é apresentado um problema FJSP 3x4, que consiste em 3 *jobs* e 4 máquinas, contendo 7 operações na produção.

		Machine			
		M_1	M_2	M_3	M_4
J1	$O_{1,1}$	2	3	4	1
	$O_{1,2}$	3	1	8	2
J2	$O_{2,1}$	1	4	1	2
	$O_{2,2}$	5	3	2	9
	$O_{2,3}$	3	1	1	4
J3	$O_{3,1}$	7	6	3	5
	$O_{3,2}$	4	5	6	2

Tabela 1: Exemplo de um problema FJSP 3x4. Adaptado de [1].

Partindo deste exemplo, é possível extrair então uma lista de prioridades, apresentada na tabela 2, a qual representa, conforme citado, a ordem de prioridade das máquinas a ser levada em consideração no processo estocástico de designação, onde 1 representa o nível da máquina que utiliza o menor de tempo de processamento, sendo então 1 < 2 < 3 < 4 a ordem crescente das melhores máquinas disponíveis neste problema.

		Priority order				
		1	2	3	⋮	4
J1	$O_{1,1}$	M_4	M_1	M_2		M_3
	$O_{1,2}$	M_2	M_4	M_1		M_3
J2	$O_{2,1}$	M_1	M_3	M_4		M_2
	$O_{2,2}$	M_3	M_2	M_1		M_4
	$O_{2,3}$	M_2	M_3	M_1		M_4
J3	$O_{3,1}$	M_3	M_4	M_2		M_1
	$O_{3,2}$	M_4	M_1	M_2		M_3

Tabela 2. Lista ordenada de máquinas factíveis para cada operação $O_{j,i}$, conforme apresentado em [1].

Com esta simples manipulação da estrutura de dados, é possível gerar uma representação eficiente da partícula, que será a adotada nos processos de geração da população, cálculo da velocidade e atualização da posição, que irá compôr assim o roteamento, o qual consideraremos o primeiro subproblema a ser resolvido dentro da abordagem hierárquica proposta.

Portanto, cada coluna da partícula contém o índice (ou nível) da coluna de prioridade – o qual indiretamente representa a máquina que processa a operação $O_{j,i}$ – conforme pode ser visualizado no exemplo da figura 2.

	Job1		Job2			Job3	
Operation	$O_{1,1}$	$O_{1,2}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$	$O_{3,1}$	$O_{3,2}$
Particle Position	2	1	3	2	2	4	4
	↓	↓	↓	↓	↓	↓	↓
Processing Machine	M_1	M_2	M_4	M_2	M_3	M_1	M_3

Fig 2. Exemplo de partícula (do inglês *particle position*), apresentado em [1].

3.3 Fator de inércia

Conforme [3], o fator de inércia *W* presente na equação 1a controla a independência no vôo da partícula, permitindo-a explorar o espaço de busca do problema de maneira individual. Um valor muito alto permite a liberdade individual, enquanto um valor muito baixo faz a partícula se guiar pelos melhores valores obtidos (*pBest* e *gBest*), conforme a equação em questão.

Assim, uma solução interessante é permitir que a partícula explore de maneira ampla o problema nas primeiras iterações, porém que iterativamente diminua essa 'liberdade' e se aproxime das melhores direções da nuvem. Tal proposta é formulada na seguinte equação apresentada em [1]:

$$W = W_{max} * ((W_{max} - W_{min})/Iter_{max}) * Iter,$$

Onde:

- W_{max}: valor máximo permitido para o fator de inércia,
- W_{min}: valor mínimo para o mesmo fator anterior,
- $Iter_{max}$: número máximo de iterações pro *PSO*,
- *Iter*: iteração corrente do *PSO*.

3.4 Subproblema de programação

Com a representação do subproblema de roteamento, através da partícula, onde esta contém a designação de uma máquina $M_{j,i}$ para cada operação $O_{j,i}$, é necessário validar qual a melhor sequência de operações a serem executadas dentro de cada máquina. Esta ideia deriva o conceito de subproblema de programação: a partir do roteamento estabelecido, o algoritmo deve explotar as várias possibilidades de ordenação para obter então uma solução ótima que minimize o *makespan*.

Assim, a programação pode ser representada em uma estrutura similar a um vetor, na qual coluna equivale a uma máquina M_i e o valor desta coluna contém uma sequência ordenada de operações $O_{j,i}$ extraídas com base no vetor de partícula (roteamento). Tal estrutura pode ser visualizada na figura abaixo, a qual representa um subproblema de programação.

Machine	M_1		M_2		M_3		M_4
Operation	$O_{1,1}$	$O_{3,1}$	$O_{1,2}$	$O_{2,2}$	$O_{2,3}$	$O_{3,2}$	$O_{2,1}$

Fig 3. Exemplo de programação extraída a partir de um roteamento pré-estabelecido, apresentado em [1].

3.5 Explotação da programação

Dada a característica meta-heurística do *PSO*, é possível, segundo [1] e [2], combiná-lo com um método de busca local a fim de procurar esta solução ótima de programação. Com este intuito, é proposto neste trabalho uma abordagem semelhante a [2], o qual utiliza o método *Simulated Annealing* (SA) a fim de explotar o espaço de busca a partir da programação extraída do roteamento disposto na nuvem de partículas.

O SA é um método de busca local inspirado no processo de arrefecimento presente no setor metalúrgico, o qual através da exposição do ferro a uma alta temperatura e consecutivo resfriamento, é possível alterar suas propriedades moleculares a fim de se obter uma versão mais pura e maleável deste material [3]. Logo, a partir de uma solução inicial, o SA submete esta ao processo de arrefecimento, o qual partindo de uma alta temperatura, busca novas soluções a partir destas, que tendem a ser melhores que a inicial. À medida que nova solução é gerada, a temperatura sofre decréscimo até chegar a um limiar mínimo, o qual encerrará o processo de arrefecimento, resultando em uma solução igual ou melhor que a inicial.

É importante ressaltar que a geração de uma nova solução (chamada de vizinhança) é parte importante do SA, uma vez que o modo que se perturba a solução original pode influenciar nos resultados obtidos pelo método.

Neste trabalho, o método de vizinhança implementado consiste em, a partir de uma programação base, escolher uma máquina M_i aleatória que tenha ao menos 2 operações em sua escala e, a partir desta máquina, escolher um ponto pivô. Se este ponto escolhido for o último dentro da sequência de operações $O_{j,i}$ dispostas, é feita a troca com a respectiva operação presente na primeira posição. Caso contrário, a troca é feita então com a operação subsequente. O funcionamento desta função de vizinhança pode ser visto na figura 4 a seguir.

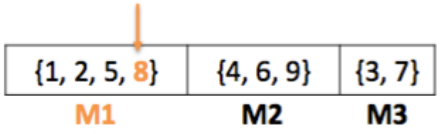

Fig 4. Neste exemplo, a máquina M1 foi escolhida aleatoriamente, e o ponto pivô 4 foi escolhido para a troca. No caso, a operação 8 irá trocar de posição com a presente na posição 1.

3.6 Controle antiestagnação

Diante da característica de exploração do espaço de busca que define o *PSO*, através das equações 1a e 1b, que regem o movimento da nuvem, caso os valores de posição em *pBest* e *gBest* sejam similares os indivíduos tendem a seguir a mesma direção, estacionando em ótimos locais, não atingindo a convergência desejada.

Neste sentido, mesmo com ajustes dos parâmetros da equação 1a, conforme já apresentados, este cenário pode ocorrer, cabendo então desenvolver mecanismos auxiliares a fim de evitar tal comportamento. Assim, este trabalho propõe a implementação de um controle antiestagnação da nuvem, a qual consiste em um registro de partículas já exploradas, para que, a cada iteração do algoritmo *PSO*, faça com que a nuvem valide cada partícula a fim de obter um roteamento único a cada iteração. Caso

Portanto, deste modo, o mecanismo proposto segue os seguintes passos:

1. Criar um registro de soluções R;
2. A cada iteração do *PSO*, após atualizar a nuvem de partículas, faça:
   2.1. Se a partícula S_i está contida em R:
      2.1.1. Enquanto S_i ainda pertencer ao registro R faça:
         2.1.1.1. Escolhe um ponto pivô (índice do vetor) em S_i aleatoriamente;
         2.1.1.2. Realiza a troca factível do nível no ponto pivô escolhio;
         2.1.1.3. Retorna nova partícula S_i.

Assim, o mecanismo antiestagnação causa uma perturbação aleatória em cada partícula já observada em momento passado pela nuvem a fim de ampliar o espaço de busca, evitando convergência precoce do algoritmo.

3.7 Algoritmo resultante

Portanto, diante ds detalhes supracitados, o algoritmo híbrido proposto neste trabalho, de forma resumida, consiste na implementação do *PSO* a fim de explorar o subproblema de roteamento, representado pelas partículas que compõem a nuvem. Então, cada partícula em questão é submetida ao método de busca local SA, que visa extrair a melhor aptidão possível a partir do roteamento individual recebido inicialmente, consistindo na explotação do espaço de busca do problema representado através do subproblema de programação.

Mais detalhes sobre a implementação deste trabalho podem ser encontrados em

http://github.com/diegocavalca/mestrado/CCO-727/Trabalho3_PSO-FJSP/.

4 Resultados

A fim de validar o desempenho do algoritmo proposto, foi utilizado os *benchmarks* estabelecidos por [4] para o FJSP, além de comparação com resultados obtidos por [1] e [2] para o mesmo cenário diante de implementações de *PSO* + SA para a resolução deste mesmo problema.

O algoritmo proposto neste trabalho foi implementado utilizando a ferramenta MATLAB 2015a, testados sob o sistema operacional OSX El Captain 10.11.6, em um computador com processador de 2.26GHz Intel Core 2 Duo, memória RAM de 8gb 1067MHz DDR3 e placa de vídeos NVIDIA GeForce 9400 256MB e 1Tb de HD.

A configuração dos parâmetros do algoritmo híbrido foi dada como segue, proposto de acordo com [1] e [2]: C1 e C2 = 2; $W = 0.4 \sim 1.2$ (*PSO*); $\alpha = 0.9 \sim 0.95$, *Iterações por temperatura* = 20 (SA).

4.1 Cenário 8x8

De acordo com [4], este cenário é o único com a característica parcialmente flexível na base de dados em questão. Na implementação do algoritmo, assim como [2], os recursos produtivos com tempo nulo são descartados na distribuição estocástica do roteamento, no momento da geração e cálculo do movimento das partículas. Este cenário conta com 27 operações, 8 *jobs* e 8 recursos produtivos, os tempos estão apresentados na tabela abaixo.

job	$O_{i,j}$	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
1	$O_{1,1}$	5	3	5	3	3	-	10	9
	$O_{1,2}$	10	-	5	8	3	9	9	6
	$O_{1,3}$	-	10	-	5	6	2	4	5
2	$O_{2,1}$	5	7	3	9	8	-	9	-
	$O_{2,2}$	-	8	5	2	6	7	10	9
	$O_{2,3}$	-	10	-	5	6	4	1	7
	$O_{2,4}$	10	8	9	6	4	7	-	-
3	$O_{3,1}$	10	-	-	7	6	5	2	4
	$O_{3,2}$	-	10	6	4	8	9	10	-
	$O_{3,3}$	1	4	5	6	-	10	-	7
4	$O_{4,1}$	3	1	6	5	9	7	8	4
	$O_{4,2}$	12	11	7	8	10	5	6	9
	$O_{4,3}$	4	6	2	10	3	9	5	7
5	$O_{5,1}$	3	6	7	8	9	-	10	-
	$O_{5,2}$	10	-	7	4	9	8	6	-
	$O_{5,3}$	-	9	8	7	4	2	7	-
	$O_{5,4}$	11	9	-	6	7	5	3	6
6	$O_{6,1}$	6	7	1	4	6	9	-	10
	$O_{6,2}$	11	-	9	9	9	7	6	4
	$O_{6,3}$	10	5	9	10	11	-	10	-
7	$O_{7,1}$	5	4	2	6	7	-	10	-
	$O_{7,2}$	-	9	-	9	11	9	10	5
	$O_{7,3}$	-	8	9	3	8	6	-	10
8	$O_{8,1}$	2	8	5	9	-	4	-	10
	$O_{8,2}$	7	4	7	8	9	-	10	-
	$O_{8,3}$	9	9	-	8	5	6	7	1
	$O_{8,4}$	9	-	3	7	1	5	8	-

Tabela 3: Cenário 8x8 do FJSP proposto em [4]. Adaptado de [2].

Os parâmetros testados neste cenário foram: Número de iterações = 30, Número de partículas = 50 (PSO); *Temp. inicial* = 3, *Temp. final* = 0.01 (SA). O algoritmo convergiu para o melhor resultado encontrado na literatura, o qual consiste no *makespan* de valor 14. O resultado, representado pelo gráfico de *Gantt*, pode ser observado na figura 5.

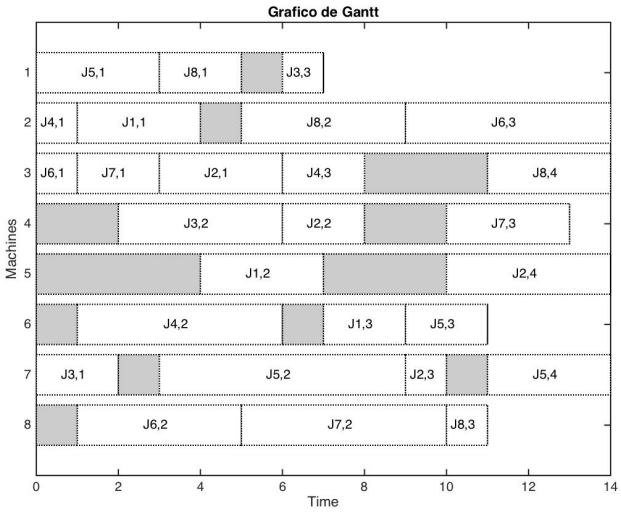

Fig. 5: Gráfico de *Gantt* com a melhor solução encontrada pelo algoritmo proposto para o cenário 8x8.

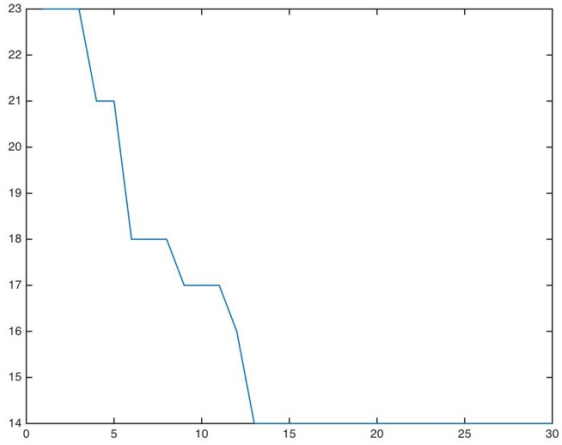

Fig. 6: Gráfico de convergência para o cenário 8x8.

É interessante notar que o algoritmo proposto mostrou um desempenho estável ao ser observado o resultado de 10 testes distintos para o cenário proposto, o qual, a partir da figura 7, expõe a eficiência da convergência de resultados próximos ao ótimo encontrado, mesmo com a característica parcialmente flexível presente o cenário 8x8.

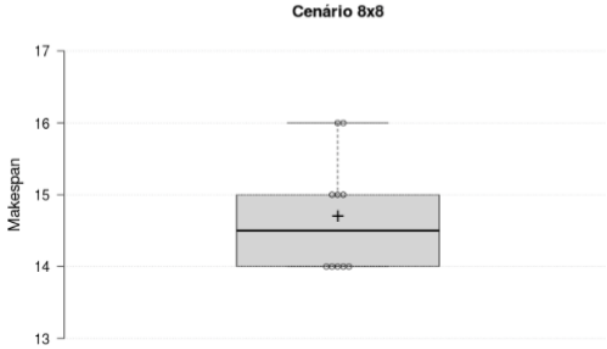

Fig. 7: *Boxplot* gerado para 10 execuções do algoritmo no cenário 8x8.

4.2 Cenário 10x10

Este cenário, contendo 10 *jobs* processados em 10 recursos, composto de 30 operações, apresenta uma complexidade um pouco mais elevada em relação ao anterior, dado que - além de ser maior - este é totalmente flexível, aumentando assim assim a quantidade de soluções factíveis. Os tempos de processamento de cada recurso estão presentes na Tabela 4.

job	$O_{i,j}$	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
1	$O_{1,1}$	1	4	6	9	3	5	2	8	9	5
	$O_{1,2}$	4	1	1	3	4	8	10	4	11	4
	$O_{1,3}$	3	2	5	1	5	6	9	5	10	3
2	$O_{2,1}$	2	10	4	5	9	8	4	15	8	4
	$O_{2,2}$	4	8	7	1	9	6	1	10	7	1
	$O_{2,3}$	6	11	2	7	5	3	5	14	9	2
3	$O_{3,1}$	8	5	8	9	4	3	5	3	8	1
	$O_{3,2}$	9	3	6	1	2	6	4	1	7	2
	$O_{3,3}$	7	1	8	5	4	9	1	2	3	4
4	$O_{4,1}$	5	10	6	4	9	5	1	7	1	6
	$O_{4,2}$	4	2	3	8	7	4	6	9	8	4
	$O_{4,3}$	7	3	12	1	6	5	8	3	5	2
5	$O_{5,1}$	7	10	4	5	6	3	5	15	2	6
	$O_{5,2}$	5	6	3	9	8	2	8	6	1	7
	$O_{5,3}$	6	1	4	1	10	4	3	11	13	9
6	$O_{6,1}$	8	9	10	8	4	2	7	8	3	10
	$O_{6,2}$	7	3	12	5	4	3	6	9	2	15
	$O_{6,3}$	4	7	3	6	3	4	1	5	1	11
7	$O_{7,1}$	1	7	8	3	4	9	4	13	10	7
	$O_{7,2}$	3	8	1	2	3	6	11	2	13	3
	$O_{7,3}$	5	4	2	1	2	1	8	14	5	7
8	$O_{8,1}$	5	7	11	3	2	9	8	5	12	8
	$O_{8,2}$	8	3	10	7	5	13	4	6	8	4
	$O_{8,3}$	6	2	13	5	4	3	5	7	9	5
9	$O_{9,1}$	3	9	1	3	8	1	6	7	5	4
	$O_{9,2}$	4	6	2	5	7	3	1	9	6	7
	$O_{9,3}$	8	5	4	8	6	1	2	3	10	12
10	$O_{10,1}$	4	3	1	6	7	1	2	6	20	6
	$O_{10,2}$	3	1	8	1	9	4	1	4	17	15
	$O_{10,3}$	9	2	4	2	3	5	2	4	10	23

Tabela 4: Cenário 10x10 do FJSP proposto em [4]. Adaptado de [2].

Os parâmetros testados neste cenário foram os mesmo do cenário anterior, alterando-se apenas a *Temp. inicial (SA)*, definindo esta com o valor 5 para o teste.

Neste cenário, o melhor *makespan* atingido foi 8, divergindo do valor 7 disposto na literatura como o melhor encontrado para o cenário em questão. Abaixo, na figura 8, é possível visualizar o gráfico contendo a melhor programação encontrado pelo algoritmo.

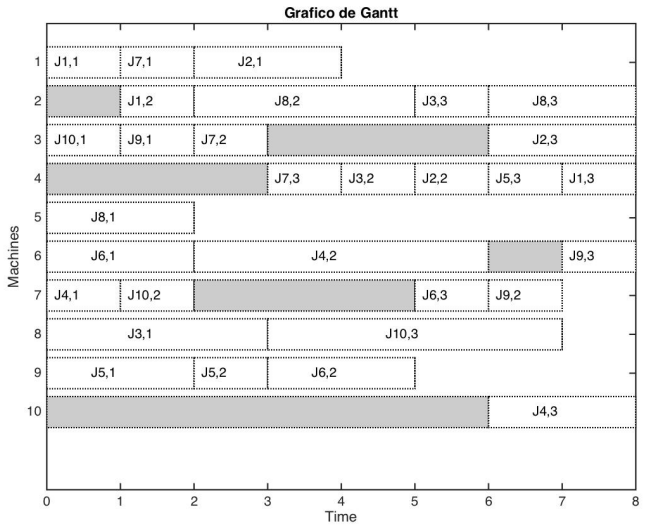

Fig. 8: Gráfico de *Gantt* com a melhor solução encontrada pelo algoritmo proposto para o cenário 10x10.

Todavia, apesar de não atingir o melhor resultado, o algoritmo exibe o mesmo comportamento coeso, após 10 execuções, o qual sempre obtém resultados muito próximo do ótimo, como pode ser observado na figura 9.

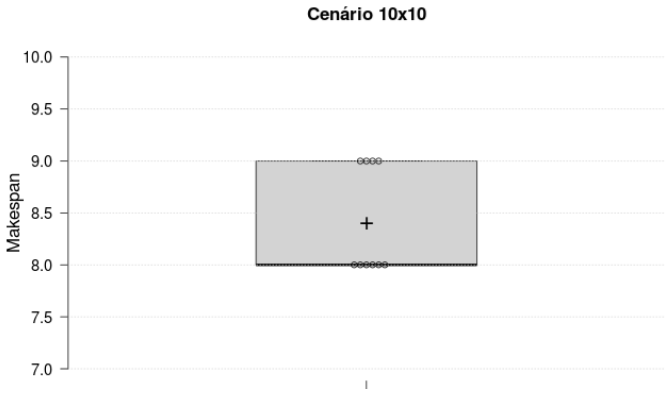

Fig. 9: *Boxplot* gerado para 10 execuções do algoritmo no cenário 10x10.

4.3 Cenário 15x10

Por fim, o algoritmo foi submetido ao teste da base 15x10, o qual apresenta a maior complexidade do conjunto apresentado por [4], que contém 15 *jobs*, compostos de 56 operações, a serem processados em 10 máquinas, diante de um cenário totalmente flexível. Os tempos desta base estão dispostos na tabela 5.

Dada a complexidade do cenário, os parâmetros testados foram alterados, sendo definidos: Número de iterações = 15, Número de partículas = 100 (PSO); *Temp. inicial* = 10, *Temp. final* = 0.01 (SA).

Devido a complexidade computacional resultante deste cenário, apenas 2 testes completos foram executados, os quais obtiveram como resultado uma solução com *makespan* no valor 14, aquém do melhor disposto na literatura, estabelecido em 11 para este cenário.

O melhor resultado obtido, então, é disposto através do gráfico de *Gantt* na figura 10.

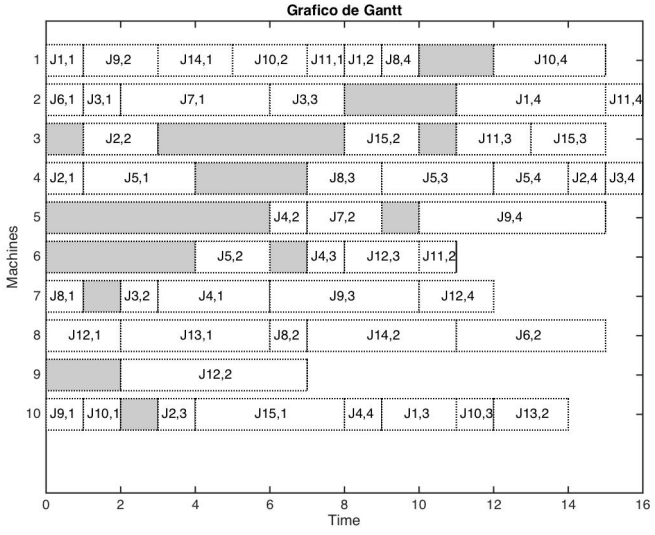

Fig. 10: Gráfico de *Gantt* com a melhor solução encontrada pelo algoritmo proposto para o cenário 15x10.

job	O_{ij}	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
1	$O_{1,1}$	1	4	6	9	3	5	2	8	9	4
	$O_{1,2}$	1	1	3	4	8	10	4	11	4	3
	$O_{1,3}$	2	5	1	5	6	9	5	10	3	2
	$O_{1,4}$	10	4	5	9	8	4	15	8	4	4
2	$O_{2,1}$	4	8	7	1	9	6	1	10	7	1
	$O_{2,2}$	6	11	2	7	5	3	5	14	9	2
	$O_{2,3}$	8	5	8	9	4	3	5	3	8	1
	$O_{2,4}$	9	3	6	1	2	6	4	1	7	2
3	$O_{3,1}$	7	1	8	5	4	9	1	2	3	4
	$O_{3,2}$	5	10	6	4	9	5	1	7	1	6
	$O_{3,3}$	4	2	3	8	7	4	6	9	8	4
	$O_{3,4}$	7	3	12	1	6	5	8	3	5	2
4	$O_{4,1}$	6	2	5	4	1	2	3	6	5	4
	$O_{4,2}$	8	5	7	4	1	2	36	5	8	5
	$O_{4,3}$	9	6	2	4	5	1	3	6	5	2
	$O_{4,4}$	11	4	5	6	2	7	5	4	2	1
5	$O_{5,1}$	6	9	2	3	5	8	7	4	1	2
	$O_{5,2}$	5	4	6	3	5	2	28	7	4	5
	$O_{5,3}$	6	2	4	3	6	5	2	4	7	9
	$O_{5,4}$	6	5	4	2	3	2	5	4	7	5
6	$O_{6,1}$	4	1	3	2	6	9	8	5	4	2
	$O_{6,2}$	1	3	6	5	4	7	5	4	6	5
7	$O_{7,1}$	1	4	2	5	3	6	9	8	5	4
	$O_{7,2}$	2	1	4	5	2	3	5	4	2	5
8	$O_{8,1}$	2	3	6	2	5	4	1	5	8	7
	$O_{8,2}$	4	5	6	2	3	5	4	1	2	5
	$O_{8,3}$	3	5	4	2	5	49	8	5	4	5
	$O_{8,4}$	1	2	36	5	2	3	6	4	11	2
9	$O_{9,1}$	6	3	2	22	44	11	10	23	5	1
	$O_{9,2}$	2	3	2	12	15	10	12	14	18	16
	$O_{9,3}$	20	17	12	5	9	6	4	7	5	6
	$O_{9,4}$	9	8	7	4	5	8	7	4	56	2
10	$O_{10,1}$	5	8	7	4	56	3	2	5	4	1
	$O_{10,2}$	2	5	6	9	8	5	4	2	5	4
	$O_{10,3}$	6	3	2	5	4	7	4	5	2	1
	$O_{10,4}$	3	2	5	6	5	8	7	4	5	2
11	$O_{11,1}$	1	2	3	6	5	2	1	4	2	1
	$O_{11,2}$	2	3	6	3	2	1	4	10	12	1
	$O_{11,3}$	3	6	2	5	8	4	6	3	2	5
	$O_{11,4}$	4	1	45	6	2	4	1	25	2	4
12	$O_{12,1}$	9	8	5	6	3	6	5	2	4	2
	$O_{12,2}$	5	8	9	5	4	75	63	6	5	21
	$O_{12,3}$	12	5	4	6	3	2	5	4	2	5
	$O_{12,4}$	8	7	9	5	6	3	2	5	8	4
13	$O_{13,1}$	4	2	5	6	8	5	6	4	6	2
	$O_{13,2}$	3	5	4	7	5	8	6	6	3	2
	$O_{13,3}$	5	4	5	8	5	4	6	5	4	2
	$O_{13,4}$	3	2	5	6	5	4	8	5	6	4
14	$O_{14,1}$	2	3	5	4	6	5	4	85	4	5
	$O_{14,2}$	6	2	4	5	8	6	5	4	2	6
	$O_{14,3}$	3	25	4	8	5	6	3	2	5	4
	$O_{14,4}$	8	5	6	4	2	3	6	8	5	4
15	$O_{15,1}$	2	5	6	8	5	6	3	2	5	4
	$O_{15,2}$	5	6	2	5	4	2	5	3	2	5
	$O_{15,3}$	4	5	2	3	5	2	8	4	7	5
	$O_{15,4}$	6	2	11	14	2	3	6	5	4	8

Tabela 5: Cenário 15x10 do FJSP proposto em [4]. Adaptado de [2].

5 Conclusão

Diante da relevância do objetivo desta pesquisa, que busca obter soluções ótimas em tempos viáveis para problemas complexos do cotidiano, a proposta do algoritmo de Enxame de Partículas integrado com a meta-heurística auxiliar de Arrefecimento Simulado, se mostra uma abordagem eficiente para a solução da programação da produção em cenários parcialmente ou totalmente flexíveis, como o FJSP.

Neste contexto, foi possível identificar a dificuldade do solução proposta resolver problemas altamente complexos, como os cenários 10x10 e 15x10, apesar de resultar em programações próximas as ótimas conhecidas. Uma hipótese a ser estudada em trabalhos futuros, a fim de mitigar este problema, é implementar uma função de vizinhança capaz de explorar com mais eficiência o espaço de soluções factíveis, utilizando o conceito de caminhos críticos, conforme proposto em [2], onde - a partir do subproblema de programação - é possível extrair um caminho de produção o qual, quando perturbado de maneira específica, tende a gerar soluções eficientes.

Portanto, o algoritmo híbrido apresentado neste trabalho conseguiu atingir os objetivos estabelecidos *a priori* na formulação do trabalho de pesquisa, qualificando-se como uma solução aceitável para a resolução do FJSP.

Referências

[1] Xia, Weijun, and Zhiming Wu. *"An effective hybrid optimization approach for multi-objective flexi-ble job-shop scheduling problems." Computers & Industrial Engineering* 48.2 (2005): 409-425.

[2] G. Aranha, *"Algoritmo de enxame de partículas para resolução do problema da programação da produção Job-Shop Flexível multiobjetivo,"* M.S. thesis, Dept. Comp. Science, Federal Univ. of Sao Carlos, SP, 2016.

[3] Goldbarg, Goldbarg and Luna. "Otimização combinatória e Meta-heurísticas: algoritmos e aplicações,"1ª ed. City of Rio de Janeiro, Brazil: Elsevier, 2016.

[4] Kacem, Imed, Slim Hammadi, and Pierre Borne. *"Approach by localization and multiobjective evoluti-onary optimization for flexible job-shop scheduling problems," IEEE Transactions on Systems, Man, and Cyberne-tics, Part C (Applications and Reviews)* 32.1 (2002): 1-13.