UM ALGORITMO GENÉTICO COM INSERÇÃO GULOSA PARA O PROBLEMA DE ESCALONAMENTO DE TAREFAS JOB SHOP SEM ESPERA

Frederico Galaxe Paes
Instituto Federal Fluminense – IFF
Av. Souza Mota, 350, Pq. Fundão, 28060-010, Campos dos Goytacazes, RJ
fpaes@iff.edu.br

Artur Alves Pessoa
Departamento de Engenharia de Produção – Universidade Federal Fluminense
Rua Passo da Pátria 156, São Domingos, 24210-240, Niterói, RJ
artur@producao.uff.br

RESUMO

Este artigo apresenta um algoritmo genético (GA) para resolver o problema *job shop* sem espera (PJSSE), cujo objetivo é minimizar o *makespan* entre todas as tarefas. O PJSSE é uma extensão do problema *job shop* clássico, onde uma restrição adicional garante que duas operações consecutivas de uma mesma tarefa sejam processadas sem qualquer interrupção. Para construir uma solução para o problema é proposto um algoritmo guloso eficiente, que insere uma tarefa de cada vez, segundo a ordem em que aparece no cromossomo, sem que haja sobreposição com as demais tarefas já inseridas. Diversos testes computacionais foram conduzidos com instâncias de médio e grande porte da literatura e os resultados comparados com as melhores abordagens da literatura para o PJSSE. Como resultado, observou-se um desempenho superior da abordagem proposta em instâncias de grande porte, no que se refere à qualidade das soluções, se comparado com as demais abordagens.

PALAVRAS CHAVE. ***Job Shop* Sem Espera, Algoritmo Genético, Metaheurística.**

Tópicos: (MH–Metaheurísticas, OC–Otimização Combinatória)

ABSTRACT

This paper presents a genetic algorithm (GA) to solve the no-wait job shop problem (NWJSP), whose objective is to minimize the makespan among all jobs. The NWJSP is an extension of the classic job shop problem, where an additional constraint ensures that two consecutive operations of the same jobs to be processed without interruption operations. To build a solution to the problem is proposed an efficient greedy algorithm, which inserts a job at a time, in the order that appears on chromosome, without overlapping with other jobs already entered. Several computational experiments were conducted with medium- and large-scale instances of the literature and the results compared with the best approaches to literature for NWJSP. As a result, there was a superior performance of the proposed approach in large-scale instances, as regards the quality of the solutions, compared with other approaches.

KEYWORDS. No-Wait Job Shop, Genetic Algorithm, Metaheuristic.

Paper topics: (MH–Metaheuristics, OC–Combinatorial Optimization)

1. Introdução

A performance de um sistema de produção pode ser influenciada pela performance do método de escalonamento utilizado na organização da produção. No problema clássico de escalonamento de tarefas *job shop* (PJS), n diferentes tarefas independentes devem ser processadas em m diferentes máquinas. A rota de cada tarefa i é conhecida antecipadamente, com cada tarefa i consistindo de n_i operações ordenadas $J_i = \{O_{i1}, O_{i2}, \cdots, O_{in_i}\}$, com o tempo de processamento da operação O_{iu} dado por p_{iu}, para $u = 1, \ldots, n_i$. Cada operação deve ser realizada sem qualquer interrupção na máquina atribuída a ela, sendo esta atribuição dada a priori. Além disso, cada máquina só pode realizar no máximo uma operação por vez. Para resolver este problema, o tempo de início de cada operação na máquina correspondente deve ser encontrado de modo a minimizar um dado critério, que é geralmente o tempo de conclusão de todas as tarefas $C_{máx}$ ou *makespan*, enquanto satisfazendo às restrições do problema. Este problema é denotado por $J_m||C_{máx}$.

O PJSSE difere do problema *job shop* clássico por uma restrição adicional no tempo de espera. Esta restrição garante que duas operações consecutivas de uma mesma tarefa sejam processadas sem qualquer interrupção. Assim, quando é necessário adiar ou antecipar o início de uma operação para evitar sobreposição, todas as demais operações da mesma tarefa devem sofrer o mesmo deslocamento. Este tipo de problema ocorre, por exemplo, em processos químicos, farmacêuticos e de aço, onde cada produto não pode aguardar entre dois processos.

O trabalho de Hall e Sriskandarajah [1996] foi o primeiro a apresentar um resumo da pesquisa relacionada aos problemas de *job shop* sem espera e com bloqueio, além de uma revisão da complexidade de uma larga variedade de problemas de escalonamento com restrições sem espera e com bloqueio.

Framinan e Schuster [2006] propuseram uma metaheurística chamada CLM (*Complete Local Search with Memory*) para o PJSSE pela integração de um novo método de escalonamento de tempo. Zhu et al. [2009] usaram um método eficiente chamado CLLM (*Complete Local Search with Limited Memory*) para resolver o PJSSE, baseando-se em um procedimento de troca de horários. Este procedimento constrói um escalonamento viável, inserindo sucessivamente cada tarefa sem alterar o tempo de início das tarefas já escalonadas. Assim, uma tarefa é localizada o mais cedo possível ou após todas as outras tarefas. Posteriormente, utilizando a estrutura tradicional de decomposição do PJSSE em dois sub-problemas: o problema de sequenciamento e de *timetabling*, Zhu e Li [2012] desenvolveram o algoritmo MCLM (*Modified Complete Local Search with Memory*) para o problema de sequenciamento. O MCLM foi testado em instâncias de referência da literatura e comparado com os algoritmos CLM e CLLM.

Bożejko e Makuchowski [2009] aplicou um algoritmo de busca tabu híbrido para o PJSSE. Novamente, um escalonamento viável é construído inserindo sucessivamente cada tarefa sem alterar o tempo de início das tarefas já escalonadas. Este procedimento é repetido com várias ordens de inserção de tarefas geradas por uma heurística busca tabu.

Em Bürgy e Gröflin [2013] é abordado o problema de inserção ótima de tarefas (IOT) para o PJSSE. Enquanto o IOT é $\mathcal{NP}$-difícil no *job shop* clássico, foi demonstrado que o mesmo pode ser resolvido em tempo polinomial no PJSSE. Os autores então apresentam um algoritmo eficiente com complexidade de tempo de $\mathcal{O}(n^2.max\{n, m\})$ para este problema. Como uma aplicação do algoritmo, foi proposta uma heurística para o PJSSE baseada no IOT e resultados numéricos favoráveis foram apresentados.

Deste modo, é proposto neste artigo um algoritmo baseado em GA, bem como um algoritmo construtivo guloso eficiente destinado a construir as soluções. O algoritmo é testado em instâncias de referência de médio e grande porte da literatura e comparado com as melhores abordagens disponíveis destinadas a resolver o PJSSE.

O restante do artigo é organizado como segue. A Seção 2 apresenta a definição do problema. A Seção 3 descreve como é gerada a população inicial do GA, bem como seus operadores e o pseudocódigo do algoritmo proposto. A Seção 4 detalha o funcionamento do algoritmo de inserção gulosa responsável pela construção da solução. Na Seção 5 são comparados e discutidos os resultados computacionais com outros resultados relevantes da literatura e finalmente, na Seção 6 são apresentadas as conclusões.

2. Definição do Problema

O PJSSE é um problema de escalonamento de n tarefas, onde cada tarefa i é composta de n_i operações, denotadas por O_{iu} $(u = 1, \cdots, n_i)$, que devem ser processadas em m máquinas. Deve-se notar que a máquina onde a operação O_{iu} será executada é denotada por M_{iu}. Cada tarefa é processada sem que haja tempo de espera entre duas operações consecutivas em sua respectiva rota de processamento. Não é permitido preempção, nenhuma tarefa pode ser processada por duas máquinas ao mesmo tempo e nenhuma máquina pode processar duas tarefas ao mesmo tempo. O tempo de processamento p_{iu} de cada operação O_{iu} em cada máquina é dado e a interrupção do processamento de qualquer operação não é permitida. Uma solução do PJSSE é dada por um vetor do tipo $(t_1, \cdots, t_n)$, onde t_i $(i = 1, \cdots, n)$ é o tempo de início da tarefa i. Em Zhu e Li [2012] é apresentado uma formulação para o PJSSE.

Um pequeno exemplo com três máquinas $(m = 3)$ e quatro tarefas $(n = 4)$ tem uma solução viável com *makespan* $C_{máx} = 19$ apresentada na Figura 1, onde esta solução é representada pelos tempos de início das tarefas J_1,J_2,J_3 e J_4, dados por $(11, 1, 0, 2)$. Os dados desta instância são mostrados na Tabela 1. Cada tarefa tem três operações denotadas por "a (b)", onde a operação é processada pela máquina "a" com tempo de processamento p_{iu}="b"

Tarefa	Máquina (p_{iu})		
1	3 (1)	2 (4)	1 (3)
2	3 (2)	1 (5)	2 (4)
3	1 (3)	3 (1)	2 (3)
4	2 (2)	3 (4)	1 (2)

Tabela 1: Pequena instância do PJSSE com m=3 e n=4.

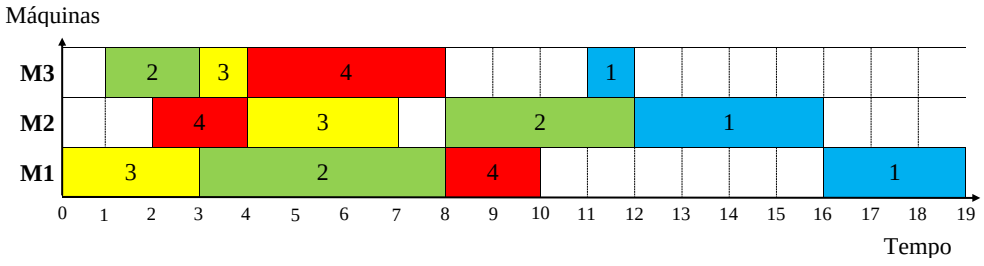


Figura 1: Diagrama de Gantt de uma solução viável com três máquinas e quatro tarefas.

3. Algoritmo Genético para o PJSSE

3.1. Geração da População Inicial

Os indivíduos da população inicial são gerados a partir de uma ordenação prévia das n tarefas em ordem decrescente do tempo total de processamento de J_i, dado por L_i $(i = 1, \cdots, n)$. Em seguida, os indivíduos são perturbados de modo a gerar uma certa

aleatoriedade. Gerando-se a população desta maneira, notou-se uma redução no custo (*fitness*) médio da população e uma pequena aceleração da convergência, se comparado com a população gerada de modo aleatório.

O processo de geração da população inicial usa uma lista de candidatos restrita (LCR) [Feo e Resende, 1995], cujo tamanho é ajustado por um parâmetro α. Iterativamente, para $k = 1, \ldots, n$, o $k - ésimo$ elemento da sequência será aleatoriamente selecionado da LCR, considerando uma distribuição uniforme. Para cada iteração k, sejam $\mathcal{S}_k$ o conjunto das tarefas que já foram inseridas e $\bar{\mathcal{S}}_k$ o conjunto de tarefas que ainda não foram inseridas. A LCR é inicializada e atualizada para conter as $\ell = \min\{\alpha, |\bar{\mathcal{S}}_k|\}$ tarefas com maior L_i em $\bar{\mathcal{S}}_k$. Note que valores grandes de α levam a uma maior aleatoriedade e a um grande número de possíveis sequências iniciais. Quando $\alpha = \infty$, qualquer permutação de n facilidades pode ser gerada com mesma probabilidade.

A Figura 2 ilustra a geração de um cromossomo da população inicial com $\alpha = 5$. Uma sequência com tarefas ordenadas em ordem decrescente de L_i aparece no topo da figura seguida pela primeira LCR. Então, a tarefa 6 é aleatoriamente selecionada e inserida na primeira posição. Em seguida, a tarefa 7 é aleatoriamente selecionada e inserida na segunda posição. O cromossomo final gerado aparece no final da figura.

i	1	2	3	4	5	6	7	8	9	10
Sequência	1	2	7	6	4	8	3	5	10	9

Primeira LCR com *α=5*

LCR	1	2	7	6	4

Índice aleatoriamente selecionado para a 1ª posição

Primeira tarefa localizada e as remanescentes ordenadas

i	1	2	3	4	5	6	7	8	9	10
Sequência	6	1	2	7	4	8	3	5	10	9

Segunda LCR com *α=5*

LCR	1	2	7	4	8

Índice aleatoriamente selecionado para a 2ª posição

Duas primeiras tarefas localizadas e as remanescentes ordenadas

i	1	2	3	4	5	6	7	8	9	10
Sequência	6	7	1	2	4	8	3	5	10	9

Cromossomo perturbado após *n* seleções

i	1	2	3	4	5	6	7	8	9	10
Sequência	6	7	1	5	8	9	4	2	10	3

Figura 2: Exemplo de um cromossomo sendo perturbado com $\alpha = 5$.

3.2. Seleção e *Crossover*

No algoritmo proposto, a seleção dos pais para recombinação é feita de maneira aleatória com probabilidade uniforme, não sendo levado em consideração a aptidão do indivíduo. Desta forma, permite-se que indivíduos menos aptos tenham mais chances de serem selecionados.

Quanto ao *crossover* utilizado, sempre que dois pais forem selecionados serão submetidos ao processo de recombinação para gerar um novo filho. Em função da representação dos cromossomos, optou-se pelo *Position-based Crossover* (PX) [Syswerda e Palmucci, 1991], um operador bem conhecido que procura preserva a sequência dos genes no filho. O modo como o operador PX atua é descrito no pseudo-código a seguir:

Passo 1: Escolha aleatoriamente um pai atribuindo a mesma probabilidade a cada um dos dois indivíduos que serão recombinados. Selecione N_{herd} genes deste pai aleatoriamente, onde $N_{herd} = \frac{n}{2} + \frac{N_{dif}}{4}$, com n representando o número de genes do cromossomo e N_{dif} o número de facilidades diferentes nos dois pais;

Passo 2: Copie o conteúdo destes genes para os genes correspondentes no filho;

Passo 3: Remova os genes que foram selecionados no **Passo 1**, no segundo pai. A sequência resultante contém os genes que o filho precisa;

Passo 4: Copie o conteúdo na mesma ordem da sequência resultante, da esquerda para a direita, para as posições vazias no filho.

Esta estratégia baseia-se na observação de que no início das gerações, quando os indivíduos ainda são muito diferentes, gerar um filho que possua 75% do pai selecionado ajuda a evitar um tempo excessivo até a convergência. Por outro lado, quando a população começar a convergir e os indivíduos ficarem muito parecidos, o valor de N_{herd} se aproximará de 50% de n e, desta forma, selecionando-se 50% de cada pai evita-se a geração de muitos filhos repetidos.

3.3. Pseudocódigo do GA para o PJSSE

O algoritmo genético proposto para resolver o PJSSE utiliza um algoritmo de inserção gulosa (IG), o qual será descrito na Seção 4, para construir e avaliar uma solução do problema (linhas 4 e 10). Assim, combinando o algoritmo IG com o GA, obtém-se o algoritmo GAIG, cujo pseudocódigo é dado pelo algoritmo 1. Os parâmetros do algoritmo GAIG são α e o tamanho da população $nPop$.

Algoritmo 1 Algoritmo Genético para o PJSSE

1: **Procedimento** GAIG(α, $nPop$)
2: $t \leftarrow 0$;
3: **Gere** a população $P(t)$ com base no parâmetro α ;
4: **Construa e avalie** cada indivíduo de $P(t)$ usando o algoritmo IG ;
5: **enquanto** ($P(t)$ *não convergiu*) **faça** {Laço principal. Controla o nº de gerações }
6: $t = t + 1$;
7: **para** i $\leftarrow$ 1 **até** $nPop$ **faça** {Gera descendentes por *crossover*}
8: **Selecione** dois pais $p_1, p_2 \in P(t-1)$ aleatoriamente;
9: $d \leftarrow$ Crossover(p_1, p_2) ;
10: **Construa e avalie** o indivíduo d usando o algoritmo IG ;
11: **fim para**
12: **Selecione** indivíduos sobreviventes em $P(t)$;
13: **fim enquanto**
14: **Retorne** a melhor solução S^* encontrada;
15: **fim Procedimento**

Na linha 3 do Algoritmo 1, uma população inicial $P(0)$ de tamanho $nPop$ é gerada do mesmo modo como foi descrito na Seção 3.1. Para cada cromossomo gerado, uma solução é construída e avaliada na linha 4 utilizando o algoritmo de inserção gulosa IG. O laço principal 5–13 controla o número de gerações, sendo executado até a convergência da população. Para cada iteração deste laço, o algoritmo seleciona aleatoriamente dois pais da população $P(t-1)$ (linha 8) para então ser realizado o *crossover* (linha 9), como mostrado na Seção 3.2. Novamente, uma solução é construída e avaliada para o indivíduo d utilizando o algoritmo IG (linha 10). Deste modo, $nPop$ filhos gerados serão inseridos na população. Após o laço 7–11 ser executado, a população contará com $2 \times nPop$ indivíduos e então, $nPop$ indivíduos sobreviventes deverão ser selecionados para compor a próxima geração $P(t)$ (linha 12). O algoritmo converge quando uma das três condições seguintes ocorrem:

- o *gap* entre o custo médio e o custo mínimo de todos os indivíduos em $P(t)$ é menor que $0,05\%$;

- todos os indivíduos gerados na iteração atual são iguais a algum indivíduo em $P(t-1)$;
- $P(t) = P(t-1) = \ldots = P(t-50)$.

4. Algoritmo de Inserção Gulosa

Nesta seção será apresentado o pseudocódigo do algoritmo responsável pela construção de uma solução do PJSSE. Trata-se de um algoritmo que insere uma tarefa j de cada vez, segundo a ordem em que aparece no cromossomo, sem que haja sobreposição das operações O_{ju} desta tarefa, com as demais tarefas já inseridas. Para identificar possíveis sobreposições, o algoritmo mantém uma lista $\mathcal{L}_k$ de intervalos ocupados $\ell_{\theta k}$=$[t_{início}, t_{término}]$ (θ=1,...,$|\mathcal{L}_k|$), representando as operações das tarefas já inseridas realizadas na máquina k a qual a lista está associada (Figura 3). No intuito de reduzir o tamanho das listas, e consequentemente o tempo gasto para a verificação da sobreposição, o algoritmo junta intervalos adjacentes em um só. Em seguida, a inserção é feita de modo guloso sendo o tempo de início t_j da tarefa j selecionada dado por aquele que produzir o menor valor do *makespan* parcial, dando prioridade ao tempo de início que minimiza o número total de intervalos ocupados em caso de empate. Este algoritmo foi chamado de inserção gulosa - IG, cujo pseudocódigo é apresentado no Algoritmo 2.

Após inserir a primeira tarefa j, são definidos os primeiros intervalos ocupados, bem como os valores de $t_{mín}$ e $t_{máx}$ (linhas 6 e 7). A partir daí, para cada nova tarefa j a ser inserida, serão testados ordenadamente os extremos de cada intervalo l ($\ell_{\theta k}$) de cada máquina k do seguinte modo: iniciando em $k = 1$, insira a u-ésima operação O_{ju} que será executada em k à esquerda do intervalo l (linha 13) e calcule o tempo de início t_j (linha 14). Se O_{ju} sobrepõe o intervalo $l-1$, caso este exista, repita o mesmo procedimento para o extremo direito de l, verificando a sobreposição com o intervalo $l+1$, caso este exista. Se O_{ju} não sobrepõe os intervalos $l-1$ e $l+1$, calcule os valores auxiliares que correspondem ao valor do novo *makespan* e o valor da variável *teste* (linhas 16 e 17). O tempo de início t_j selecionado para a tarefa j, será aquele que não gerar sobreposição entre as operações de j e as tarefas já inseridas (linha 18) e que produzir o menor valor da variável *teste*.

A Figura 3 apresenta um pequeno exemplo do PJSSE com $m = 5$ máquinas e $n = 3$ tarefas, onde duas tarefas já estão alocadas e uma terceira tarefa deverá ser inserida. A figura mostra também a lista de intervalos ocupados, que representam as posições onde a tarefa 3 não poderá ser inserida. Uma tentativa de simular parcialmente o funcionamento do algoritmo IG é apresentada na Figura 4. Nesta figura, a tarefa 3 que está sendo inserida não deve sobrepor as tarefas 1 e 2 já alocadas. Inicialmente, a operação O_{32} será testada nos extremos dos intervalos da máquina 1, conforme pode ser visto nas Figuras 4(a)–4(b). Prosseguindo com este processo nas demais máquinas, obtém-se um tempo de início para a tarefa 3 de $t_3 = 4$ (Fig. 4(c)).

Em muitos casos haverá empate entre as opções disponíveis para inserção da tarefa j, o qual ocorrerá quando o valor da variável *teste* permanecer igual ao valor da variável *critério*. A linha 19 define um critério de desempate onde, dentre as opções empatadas, será selecionada aquela onde houver um maior número de intervalos ocupados adjacentes à tarefa inserida minimizando, deste modo, o número total de intervalos ocupados. Note que o valor do novo *makespan*, dado por $t'_{máx} - t'_{mín}$, é multiplicado por $2n$ para que o critério de desempate possa ser somado à variável *teste* sem afetar a preferência pelo menor *makespan*.

Após a inserção da tarefa j, são atualizadas as listas $\mathcal{L}_k$ de cada máquina, transformando em um único intervalo aqueles que forem adjacentes (linha 29). Também são atualizados os valores de $t_{mín} = \min\{t_{mín}, t^*_j\}$ e $t_{máx} = \max\{t_{máx}, t^*_j + L_j\}$ (linha 30). Ao final do laço 2–32, todas as tarefas já terão sido inseridas. Então, o *makespan* $C_{máx}$ desta solução é calculado na linha 33 e comparado com o melhor *makespan* $C^*_{máx}$ encontrado até

Algoritmo 2 Algoritmo de construção da solução do PJSSE

1: **Procedimento** INSERÇÃO GULOSA - IG($Seq[\,]$, $S^*[\,]$, n, m)
2: **para** i $\leftarrow 1$ **até** n **faça**
3: $j \leftarrow Seq[i]$; {Seleciona a i-ésima tarefa a ser inserida}
4: **se** $i = 1$ **então**
5: $T[j] \leftarrow 0$; {$T[\,]$: guarda os *tempos de início* das tarefas}
6: **Inicialize** os intervalos para cada máquina m;
7: $t_{mín} \leftarrow 0$; $t_{máx} \leftarrow L_j$;
8: **senão**
9: $critério \leftarrow \infty$;
10: **para** k $\leftarrow 1$ **até** m **faça** {Percorre as m máquinas}
11: **para** l $\leftarrow 1$ **até** $|\mathcal{L}_k|$ **faça** {Percorre os intervalos de cada máquina}
12: **para** cada extremo $x \in \{esquerdo, direito\}$ de l na máq. k **faça**
13: **Insira** a operação O_{ju} executada na máquina k no extremo x de l;
14: **Calcule** o tempo de início t_j da tarefa j;
15: **se** (O_{ju} *não sobrepõe* $l-1$) e (O_{ju} *não sobrepõe* $l+1$) **então**
16: $t'_{mín} \leftarrow \min\{t_{mín}, t_j\}$; $t'_{máx} \leftarrow \max\{t_{máx}, t_j + L_j\}$;
17: $teste \leftarrow (t'_{máx} - t'_{mín}) \times 2 \times n$;
18: **Verifique** se as operações de j sobrepõem alguma tarefa já inserida;
19: **para** cada $O_{ju'}$ adjacente a algum intervalo l' **faça** $teste \leftarrow teste - 1$;
20: **se** (*não sobrepõe*) e ($teste < critério$) **então**
21: $critério \leftarrow teste$;
22: $t^*_j \leftarrow t_j$;
23: **fim se**
24: **fim se**
25: **fim para**
26: **fim para**
27: **fim para**
28: $T[j] \leftarrow t^*_j$;
29: **Atualize** os intervalos de cada máquina, juntando os adjacentes;
30: **Atualize** $t_{mín}$ e $t_{máx}$;
31: **fim se**
32: **fim para**
33: $C_{máx} \leftarrow t_{máx} - t_{mín}$;
34: **se** $C_{máx} < C^*_{máx}$ **então**
35: $C^*_{máx} \leftarrow C_{máx}$;
36: **para** k $\leftarrow 1$ **até** n **faça**
37: $S^*[k] \leftarrow T[k]$;
38: **fim para**
39: **fim se**
40: **Retorne** $C_{máx}$;
41: **fim Procedimento**

o momento pelo GAIG. Se a linha 34 for satisfeita, a variável $C^*_{máx}$ é atualizada e o vetor contendo os tempos de início T é guardado no vetor S^*.

5. Experimentos Computacionais

No intuito de avaliar a performance do algoritmo GAIG proposto para resolver o PJSSE, foram realizados diversos experimentos computacionais. O algoritmo foi codificado em C++ e os experimentos conduzidos em uma máquina Intel Core $i7-3770$, com $3,4$ GHz de CPU e $11,7$ GB de memória RAM em um sistema operacional Linux. Para comparar a performance do GAIG com outros algoritmos da literatura, foram testadas instâncias de *benchmark* do *job-shop* padrão obtidas do *OR-Library* [Beasley, 1990], impondo restrições de *não-espera*, e alguns resultados promissores foram obtidos, principalmente para instâncias

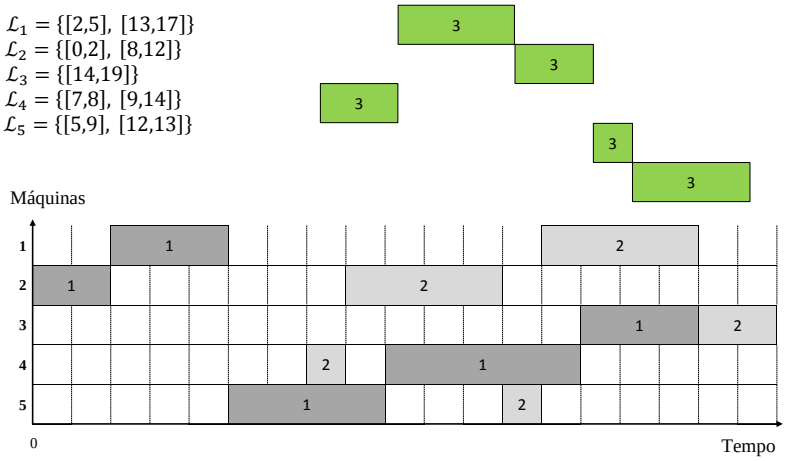
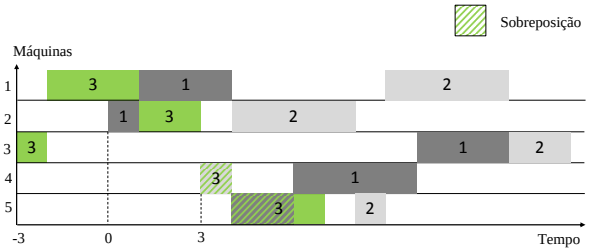
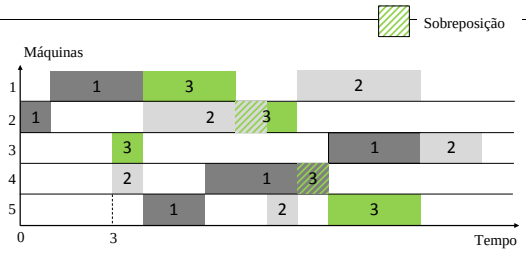


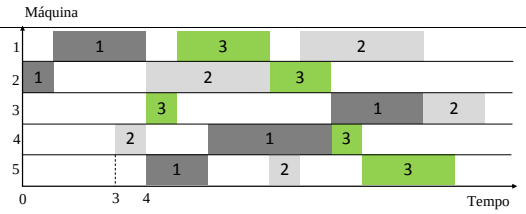
Figura 3: Tarefa 3 que será inserida pelo algoritmo IG e as respectivas listas de intervalos ocupados.



(a) Inserção à esquerda de ℓ_{11}: sobreposição entre O_{34} e O_{21} em $m = 4$ e O_{35} e O_{13} em $m = 5$.



(b) Inserção à direita de ℓ_{11}: sobreposição entre O_{33} e O_{22} em $m = 2$ e O_{34} e O_{14} em $m = 4$.



(c) Posição selecionada para inserção de J_3 com t_3=4.

Figura 4: Intervalos disponíveis testados pelo algoritmo de inserção gulosa.

de médio e grande porte. Para facilitar a apresentação dos resultados, as instâncias foram divididas em dois grupos de acordo com o seu tamanho:

- **Grupo I**: formado por instâncias de médio porte contendo 15 ou 20 tarefas, com 10

ou 15 máquinas;

- **Grupo II**: formado por instâncias de grande porte contendo 30 ou 50 tarefas e 10 máquinas;

Os parâmetros utilizados no algoritmo genético foram fixados nos seguintes valores: $nPop$=600 (tamanho da população), α=10 (população inicial) para as instâncias com 15, 20 ou 30 tarefas e α=15 para as instâncias com 50 tarefas. A seguir, serão apresentados os resultados comparativos entre o GAIG e os principais algoritmos da literatura. Os custos das soluções dos principais algoritmos da literatura, apresentados nas Tabelas 2 e 3, foram obtidos de Bürgy e Gröflin [2013].

5.1. Avaliação dos Resultados do Grupo I

Os resultados do GAIG para as instâncias do grupo I foram comparados com os seguintes algoritmos da literatura: *complete local search with limited memory* (CLLM), proposto por Zhu et al. [2009], *hybrid tabu search algorithm* (HTS), proposto por Bożejko e Makuchowski [2009] e o *optimal job insertion and local search with neighborhood* $N^1 \cup N^2$ (OJILS2), proposto por Bürgy e Gröflin [2013]. Para que fosse possível uma comparação foi utilizada a mesma estratégia de Bürgy e Gröflin [2013], onde os testes foram realizados considerando 5 execuções por instância com cada execução tendo um tempo fixado em 3600 segundos. Como os algoritmos comparados foram testados em máquinas com processadores distintos, fez-se necessário o uso de algum critério para que a comparação com o GAIG fosse coerente. Deste modo, optou-se por utilizar um tempo inversamente proporcional à velocidade de processamento das máquinas utilizadas. Assim, um tempo de 3000 segundos foi obtido a partir de uma regra de três simples envolvendo o tempo de 3600 segundos e as velocidades dos processadores utilizados neste trabalho e em Bürgy e Gröflin [2013], com valores de $3,4$ e $2,83$ Ghz respectivamente ($3000 \approx 3600 * 2,83/3,4$). Para que o algoritmo ficasse em execução durante todo este tempo, optou-se por reiniciar toda a população sempre que o algoritmo convergia.

Para cada algoritmo, a Tabela 2 apresenta os valores dos menores *gaps* percentuais (MGP) entre seus respectivos *makespan* mínimo ($C_{máx}^{mín}$) obtido em 5 execuções e o menor $C_{máx}^{mín}$ dentre todas as abordagens, mostrados na terceira coluna da tabela. Além dos valores dos MGPs, a tabela também apresenta os valores dos *gaps* médios percentuais (GMP) para os algoritmos OJILS2 e GAIG. De acordo com a tabela, o GAIG encontrou resultados de $C_{máx}^{mín}$ melhores ou iguais as demais abordagens em 10 de 30 instâncias, como pode ser observado pelos valores $0,00\%$ do MGP. O GAIG não apresenta um bom desempenho em instâncias de pequeno porte, necessitando incorporar novas estratégias para tratar este tipo de instância.

5.2. Avaliação dos Resultados do Grupo II

Os resultados do GAIG para as instâncias do grupo II foram comparados com os resultados obtidos pelos algoritmos HTS e OJILS2, os quais apresentam os melhores resultados da literatura para o PJSSE. De modo análogo ao teste do grupo I, para efetuar uma comparação com os trabalhos citados, o GAIG rodou 5 vezes para cada instância sendo utilizado como critério de parada um tempo de 3000 segundos por rodada, contra um tempo de 3600 segundos para os algoritmos citados.

Na Tabela 3 são registrados, para cada instância, seus respectivos nomes, os tamanhos dos problemas $(n; m)$, os menores valores de $C_{máx}^{mín}$, os menores valores de $C_{máx}^{méd}$, além dos valores dos MGPs e dos GMPs, obtidos do mesmo modo como descrito anteriormente.

Das 15 instâncias de grande porte, o GAIG conseguiu superar os algoritmos HTS e OJILS2 em oito instâncias, apresentando valores menores de $C_{máx}^{mín}$. Na média, o GAIG é melhor que os outros algoritmos para as instâncias La31–35, com um valor médio dos MGPs de $0,17\%$ e um valor médio dos GMPs de $0,40\%$, bem próximo dos $0,36\%$ do OJILS2.

Instância	n;m	Menor $C_{máx}^{mín}$	Menor $C_{máx}^{méd}$	CLLM MGP(%)	HTS MGP(%)	OJILS2 MGP(%)	OJILS2 GMP(%)	GAIG MGP(%)	GAIG GMP(%)
La21	15;10	2030*	2030*	0,89	**0,00**	**0,00**	**0,00**	0,89	0,89
La22	15;10	1852*	1852*	1,89	**0,00**	**0,00**	**0,00**	0,65	0,65
La23	15;10	2021*	2021*	0,54	**0,00**	**0,00**	**0,00**	1,73	2,28
La24	15;10	1972*	1972*	2,18	**0,00**	**0,00**	**0,00**	**0,00**	1,32
La25	15;10	1906*	1906*	0,58	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
Média				1,22	**0,00**	**0,00**	**0,00**	0,65	1,03
La36	15;15	2685*	2685*	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
La37	15;15	2831*	2831*	4,63	**0,00**	**0,00**	**0,00**	2,01	2,37
La38	15;15	2525*	2525*	3,64	**0,00**	**0,00**	**0,00**	2,50	2,50
La39	15;15	2660*	2660*	1,39	1,02	**0,00**	**0,00**	1,62	1,99
La40	15;15	2564*	2564*	1,17	0,62	**0,00**	**0,00**	1,17	1,17
Média				2,17	0,33	**0,00**	**0,00**	1,46	1,60
La26	20;10	2477	2478	3,07	1,17	**0,00**	**0,00**	1,21	2,22
La27	20;10	2611	2638	5,21	2,45	**0,00**	**0,00**	2,49	2,20
La28	20;10	2546	2551	3,06	0,24	**0,00**	**0,00**	2,08	2,27
La29	20;10	2300	2320	8,22	**0,00**	**0,00**	**0,00**	**0,00**	1,08
La30	20;10	2452	2452	8,69	**0,00**	**0,00**	**0,00**	4,73	4,73
Média				5,65	0,77	**0,00**	**0,00**	2,10	2,50
Swv01	20;10	2318*	2318*	0,43	**0,00**	**0,00**	**0,00**	0,52	1,12
Swv02	20;10	2417*	2417*	0,04	**0,00**	**0,00**	**0,00**	0,04	0,04
Swv03	20;10	2381*	2381*	1,43	**0,00**	**0,00**	**0,00**	2,44	2,60
Swv04	20;10	2462*	2462*	3,25	**0,00**	**0,00**	**0,00**	1,54	1,83
Swv05	20;10	2333*	2333*	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**	**0,00**
Média				1,03	**0,00**	**0,00**	**0,00**	0,91	1,12
Swv06	20;15	3278*	3278*	2,99	0,37	**0,00**	**0,00**	2,68	3,54
Swv07	20;15	3188	3188	2,60	**0,00**	**0,00**	**0,00**	**0,00**	1,91
Swv08	20;15	3423	3423	3,13	**0,00**	**0,00**	**0,00**	**0,00**	3,01
Swv09	20;15	3246	3246	1,88	0,74	**0,00**	**0,00**	1,69	1,69
Swv10	20;15	3451	3451	1,07	0,32	**0,00**	**0,00**	1,16	1,16
Média				2,33	0,28	**0,00**	**0,00**	1,11	2,26
La31	30;10	3476	3548	7,74	0,63	2,30	0,85	**0,00**	**0,00**
La32	30;10	3776	3849	6,67	2,81	**0,00**	**0,00**	0,34	0,26
La33	30;10	3429	3488	9,33	0,73	2,10	0,95	**0,00**	**0,00**
La34	30;10	3568	3589	7,17	2,55	**0,00**	**0,00**	0,50	0,95
La35	30;10	3542	3627	6,15	0,28	1,38	**0,00**	**0,00**	0,80
Média				7,41	1,40	1,16	0,36	0,17	0,40

Tabela 2: Grupo I: comparação entre CLLM, HTS, OJILS2 e GAIG com um tempo fixo de 3000 seg. por rodada.

Com relação às instâncias Swv, o GAIG apresenta um comportamento diferente para as instâncias Swv11–15 e Swv16–20, embora elas possuam o mesmo número de tarefas e máquinas. O motivo deste comportamento pode ser explicado pela forma como estas instâncias foram geradas [Storer et al., 1992], a qual baseou-se no seguinte padrão: para as instâncias Swv11–15, as operações associadas as cinco primeiras máquinas precedem as demais, gerando um padrão mais regular e favorecendo, assim, a realização de uma busca local. Como o GAIG não realiza qualquer tipo de busca local, as médias dos MGPs e GMPs são maiores que as do OJILS2. As instâncias Swv16-20, por outro lado, são geradas com precedências totalmente aleatórias gerando, deste modo, uma estrutura irregular. Neste caso, a busca local não funciona de modo eficiente justificando os valores acima de $0,00\%$ e $0,00\%$ de MGP e GMP, apresentados pelo OJILS2.

O GAIG parece apresentar um bom desempenho em instâncias de grande porte. Embora existam instâncias maiores que 50 tarefas para o *job-shop* disponíveis na literatura, nenhuma delas parece ter sido tratada ainda com restrições *no-wait* (sem espera) não

Instância	n;m	Menor $C_{máx}^{mín}$	Menor $C_{máx}^{méd}$	HTS MGP(%)	OJILS2 MGP(%)	OJILS2 GMP(%)	GAIG MGP(%)	GAIG GMP(%)
La31	30;10	3476	3548	0,63	2,30	0,85	**0,00**	**0,00**
La32	30;10	3776	3849	2,81	**0,00**	**0,00**	0,34	0,26
La33	30;10	3429	3488	0,73	2,10	0,95	**0,00**	**0,00**
La34	30;10	3568	3589	2,55	**0,00**	**0,00**	0,50	0,95
La35	30;10	3542	3627	0,28	1,38	**0,00**	**0,00**	0,80
Média				1,40	1,16	0,36	0,17	0,40
Swv11	50;10	5456	5512	1,98	**0,00**	**0,00**	3,48	3,23
Swv12	50;10	5441	5518	**0,00**	0,74	**0,00**	1,69	1,05
Swv13	50;10	5571	5606	1,02	**0,00**	**0,00**	2,15	2,37
Swv14	50;10	5369	5388	0,60	**0,00**	**0,00**	3,26	3,73
Swv15	50;10	5334	5366	1,89	**0,00**	**0,00**	2,81	2,93
Média				1,10	0,15	**0,00**	2,68	2,66
Swv16	50;10	5746	5819	1,69	0,57	1,15	**0,00**	**0,00**
Swv17	50;10	5632	5678	2,63	0,25	0,19	**0,00**	**0,00**
Swv18	50;10	5533	5686	4,55	2,44	0,79	**0,00**	**0,00**
Swv19	50;10	5721	5870	4,82	2,87	1,47	**0,00**	**0,00**
Swv20	50;10	5592	5632	2,36	0,50	1,58	**0,00**	**0,00**
Média				3,21	1,33	1,04	**0,00**	**0,00**

Tabela 3: Grupo II: Comparação entre os resultados para instâncias de grande porte.

existindo, deste modo, resultados que possam ser comparados com o GAIG para tais instâncias.

Finalmente, procurou-se avaliar o desempenho dos algoritmos através de duas métricas de qualidade: *Score* e $\#MSC$. O *Score* é uma métrica muito empregada na avaliação de heurísticas e representa a soma sobre todas as instâncias, do número de algoritmos que encontraram melhores soluções que o algoritmo analisado. Quanto menor o valor do *Score*, melhor será o algoritmo correspondente. O $\#MSC$, por sua vez, representa o número de instâncias em que a melhor solução conhecida (MSC) da literatura foi encontrada pelo algoritmo. A Tabela 4 apresenta os valores dos *scores* e o número de vezes que os algoritmos encontraram os valores do $C_{máx}^{mín}$ igual ao valor da melhor solução conhecida (MSC) de cada instância, baseado nas Tabelas 2 (T1) e 3 (T2). De acordo com a tabela, nas linhas referentes à T2 (instâncias de grande porte) o GAIG apresenta um *Score* igual ao OJILS2 e inferior ao HTS. Além disso, o GAIG alcançou a MSC mais vezes que as demais abordagens.

		CLLM	MCLM	HABC	HTS	OJILS2	GAIG
T1	**Score**	77	-	-	15	6	44
	#MSC	2	-	-	17	26	10
T2	**Score**	-	-	-	21	**12**	**12**
	#MSC	-	-	-	1	6	**8**

Tabela 4: Resumo dos resultados numéricos obtidos pelo GAIG e os algoritmos comparados para as 40 instâncias testadas.

6. Conclusões

Neste artigo é proposto um algoritmo guloso eficiente para a construção de uma solução do PJSSE, o qual é embutido no algoritmo genético GA, originando um algoritmo heurístico chamado GAIG. Diversos testes computacionais foram conduzidos com instâncias de médio e grande porte da literatura e os resultados comparados com as melhores abordagens

da literatura para o PJSSE. Como resultado, observou-se um desempenho superior do GAIG em instâncias de grande porte, no que se refere à qualidade das soluções, se comparado com as demais abordagens. Cabe ressaltar que o GAIG é uma proposta inicial e que novas estratégias ainda podem ser empregadas, principalmente no algoritmo de inserção gulosa destinado à construção da solução.

Referências

Beasley, J. E. (1990). Or-library: distributing test problems by electronic mail. *Journal of the operational research society*, 41(11):1069–1072.

Bożejko, W. e Makuchowski, M. (2009). A fast hybrid tabu search algorithm for the no-wait job shop problem. *Computers & Industrial Engineering*, 56(4):1502–1509.

Bürgy, R. e Gröflin, H. (2013). Optimal job insertion in the no-wait job shop. *Journal of Combinatorial Optimization*, 26(2):345–371.

Feo, T. A. e Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133.

Framinan, J. M. e Schuster, C. (2006). An enhanced timetabling procedure for the no-wait job shop problem: a complete local search approach. *Computers & Operations Research*, 33(5):1200–1213.

Hall, N. G. e Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no-wait in process. *Operations research*, 44(3):510–525.

Storer, R., Wu, S., e Vaccari, R. (1992). New search spaces for sequencing instances with application to job shop 38 (1992) 1495–1509manage. *Sci*, 38:1495–1509.

Syswerda, G. e Palmucci, J. (1991). The application of genetic algorithms to resource scheduling. In *ICGA*, p. 502–508.

Zhu, J. e Li, X. (2012). An effective meta-heuristic for no-wait job shops to minimize makespan. *Automation Science and Engineering, IEEE Transactions on*, 9(1):189–198.

Zhu, J., Li, X., e Wang, Q. (2009). Complete local search with limited memory algorithm for no-wait job shops to minimize makespan. *European Journal of Operational Research*, 198(2):378–386.