MINIMIZAÇÃO DO TEMPO TOTAL DE FLUXO NO PROBLEMA DE *BLOCKING FLOWSHOP* COM USO DE GRASP REATIVO

João Gabriel Gelli[1,a]
Bianca Antunes[1,a]
Adrian Manresa[1,a]
Luciana Pessoa[1,b]

[1]Pontifícia Universidade Católica (PUC-Rio)
R. Marquês de São Vicente, 225 - Gávea, Rio de Janeiro - RJ, 22451-900

[a]{joao.gelli, biancabrandao, adrianperez}@tecgraf.puc-rio.br
[b]lucianapessoa@puc-rio.br

RESUMO

O problema de *blocking flowshop* tem muitas aplicações em sistemas de manufatura. Esse problema assume que não há *buffer* entre máquinas e o objetivo mais relevante para o ambiente de produção é a minimização do tempo total de fluxo. Como o problema é NP-difícil, heurísticas têm sido propostas para fornecer boas soluções em um tempo computacional razoável. Neste artigo, propomos um procedimento GRASP reativo, cujo desempenho foi avaliado comparativamente em relação aos melhores métodos encontrados na literatura.

PALAVRAS CHAVE. Blocking Flowshop, GRASP, Tempo total de fluxo.

Tópicos: Meta-heurísticas

ABSTRACT

The blocking flowshop problem has many applications in manufacturing systems. This problem assumes that there is no buffer between machines and the most relevant objective for the production environment is the minimization of total flowtime. Since the problem is NP-hard, heuristics have been proposed to provide good solutions in a reasonable computational time. In this paper, we propose a reactive GRASP and evaluate its performance against the best solutions found in the literature.

KEYWORDS. Blocking Flowshop. GRASP. Total flowtime.

Paper topics: Metaheuristics

1. Introdução

O problema de *flowshop* descreve um ambiente de manufatura em que são realizadas *n* tarefas em *m* máquinas em determinada sequência. Uma de suas variações é o chamado *blocking flowshop* (BFSP), onde considera-se que não há *buffers* entre as máquinas, ou seja, uma tarefa só pode sair de uma máquina quando a próxima está livre [Caraffa et al. 1999]. Sendo assim, pesquisadores buscam a melhor forma de ordenar estas tarefas visando otimizar a produção.

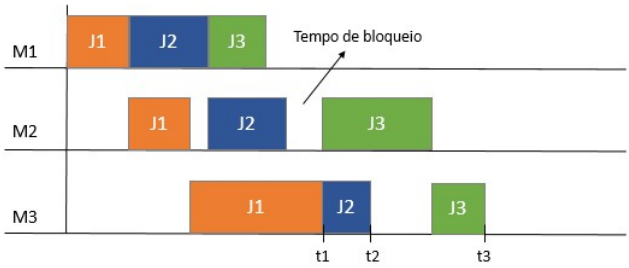

Figura 1. Representação do *blocking flowshop*

A Figura 1 exemplifica um problema de *blocking flowshop*. Nela, a tarefa 3 (J3) só pode começar na máquina 2 depois que o J2 tiver ido para a máquina 3, mesmo que tenha acabado o processamento antes desse tempo.

O critério de otimização para os problemas de *flowshop*, entretanto, pode variar de acordo com a indústria em que está sendo aplicado. Na literatura há diversos artigos que apresentam modelos com objetivo de minimizar o atraso total ou o *makespan*, por exemplo. Porém, a minimização do tempo total de fluxo é um dos critérios mais relevantes para o atual cenário de produção e, por isso, vem sendo cada vez mais utilizado [Liu e Reeves 2001]. No caso da Figura 1, o tempo total de fluxo seria o somatório dos tempos de término de cada atividade na última máquina (t1 + t2 + t3).

A aplicação do BFSP é ampla e pode ser encontrada em diferentes tipos de manufatura, como na produção de blocos de concreto [Grabowski e Pempera 2000] e na indústria química [Ronconi 2004], onde o produto parcialmente processado não pode ser estocado e, por isso, tem que ficar nas máquinas até a próxima etapa de processamento.

Este artigo, portanto, se mostra relevante para a literatura ao abordar um problema de ampla aplicação na indústria e tema de poucos estudos publicados. Seu objetivo é avançar o estado da arte a partir da elaboração de um procedimento heurístico para minimizar o tempo total de fluxo em um problema de *blocking flowshop*, denotado por $Fm|block|\sum C_i$, de acordo com a notação proposta por [Graham et al. 1979]. O procedimento é um GRASP reativo composto pelo algoritmo construtivo aleatorizado baseado no método FF, criado por [Fernandez-Viagas e Framinan 2015] e explicado na seção 3.1 deste artigo, seguido da aplicação de duas abordagens de busca local, uma do tipo *interchange* e outra *insertion* [Den Besten e Stutzle 2001].

O artigo está estruturado da seguinte forma: a próxima seção faz uma revisão da literatura existente; a seção 3 descreve o método proposto; a seção 4 avalia e compara os resultados obtidos. Depois, na seção 5, são apresentadas as conclusões e sugestões para trabalhos futuros.

2. Revisão Bibliográfica

Apesar de relevante para a indústria atual, poucos trabalhos foram realizados com o objetivo de minimizar o tempo total de fluxo para *blocking flowshop*. O BFSP pode ser definido como *n* tarefas independentes que são processadas consecutivamente em determinada ordem de máquinas *m*. Entre cada duas máquinas sucessivas, não há armazenamento intermediário, portanto, se a próxima máquina estiver ocupada, o trabalho deve ser bloqueado na máquina atual, que permanece ociosa e, portanto, impede o processamento de outro trabalho sucessivo [Nouri e Ladhari 2017].

Sendo um problema NP-difícil, algumas heurísticas já foram sugeridas para que se consiga obter resultados próximos ao ótimo. Os trabalhos mais relevantes que abordaram o BFSP com a

minimização total do tempo de fluxo foram os seguintes: [Wang et al. 2010] sugeriram um algoritmo híbrido de *Harmony Search* (HS); [Deng et al. 2012] e [Han et al. 2013] utilizaram um algoritmo *Discrete Artificial Bee Colony;* [Moslehi e Khorasanian 2013] propuseram um algoritmo *Branch and Bound* usado para pequenas instâncias; [Xie et al. 2014] apresentou os resultados de uma pesquisa com *hybrid variable neighborhood search* e *simulated annealing algorithm* (VNSSA); [Ribas e Companys 2015] propuseram o uso de GRASP (*Greedy Randomized Adaptive Search Procedure*) com duas heurísticas construtivas, variando o critério utilizado para selecionar a primeira tarefa da sequência (HPF1 e HPF2); [Ribas et al. 2015] apresentou um algoritmo *Discrete Artificial Bee Colony* (DABC-RCT) de alta performance; [Fernandez-Viagas et al. 2016] combinaram uma heurística construtiva com sequências parciais em paralelo, usando uma abordagem de *beam-search*; e [Tasgetiren et al. 2016] propuseram um método de inserção de blocos seguido por um VNS.

Este artigo se diferencia dos anteriores por implementar um GRASP reativo em que o construtivo utilizado é baseado no método FF, que foi inicialmente proposto para um problema de *permutation flowshop* sem bloqueio. Os principais componentes da heurística proposta são apresentados a seguir.

3. Método GRASP proposto

O GRASP (*Greedy Randomized Adaptive Search Procedure*) é um processo multi-partida e iterativo. Cada iteração de um GRASP é geralmente composta de uma fase de construção, onde uma solução viável é encontrada, e de uma fase de busca local, que começa a partir da solução construída e explora sua vizinhança, até que uma solução ótima local seja encontrada.

A cada iteração de construção, o próximo elemento a ser incluído na solução é escolhido aleatoriamente de uma Lista Restrita de Candidatos (LRC). Essa lista é formada pelos elementos que resultariam em maior benefício caso fossem adicionados à solução.

Os candidatos que entram na lista são todos aqueles que possuem um custo incremental menor que o valor de μ, definido como $\mu = g_{min} + \alpha(g_{max} - g_{min})$, onde g_{min} e g_{max} são os custos mínimo e máximo, respectivamente, e α é um parâmetro a ser definido. Dessa maneira, para $\alpha = 0$, o algoritmo é puramente guloso, enquanto para $\alpha = 1$, ele é puramente aleatório [Resende e Ribeiro 2019].

Uma variação do GRASP básico apresentado por [Feo e Resende 1989], é o chamado GRASP reativo, que foi desenvolvido no contexto do problema de *traffic assignment* para satélites de comunicação [Prais e Ribeiro 2000]. Depois, foi adaptado por [Binato et al. 2002] para o escalonamento de tarefas em máquinas (problema de *job shop*). Nesta variação do GRASP, o parâmetro α (que determina a qualidade dos elementos presentes na LRC) é auto ajustado e seu valor é periodicamente modificado de acordo com a qualidade das soluções obtidas [Festa e Resende 2010]. Sendo assim, ao se utilizar diferentes valores para o α, busca-se aumentar a chance de se encontrar uma solução de maior qualidade em comparação à que seria obtida se o valor do α fosse fixo. Uma forma de ajustar os valores de α é usar a regra proposta em [Prais e Ribeiro 2000]. Nela, inicialmente, todos os valores de α possuem a mesma probabilidade de serem escolhidos. Entretanto, a cada "it" iterações, os valores de α são ajustados. Este mecanismo baseia-se no valor médio das soluções obtidas com cada valor de α; que são selecionados aleatoriamente de um conjunto discreto $A = \{\alpha_1, \ldots, \alpha_m\}$ usando a distribuição de probabilidade p_i (Equação 1):

$$p_i = \frac{q_i}{\sum_{j=1}^{m} q_j} \quad (1)$$

Onde q_i é calculado como (Equação 2):

$$q_i = \frac{f^*}{M_i} \quad (2)$$

Sendo f^* o valor da melhor solução global já encontrada e M_i o valor médio das soluções obtidas tomando α_i na fase de construção.

3.1. Construtivo FF

O construtivo utilizado neste artigo foi proposto por [Fernandez-Viagas e Framinan 2015] e é denotado por FF. No artigo, o FF também é utilizado para minimizar o tempo de fluxo, mas para o problema clássico de *flowshop* (sem o bloqueio). Entretanto, no ano seguinte, [Fernandez-Viagas et al. 2016] o aplicaram, com o mesmo objetivo, para o problema de *blocking flowshop*. Tendo obtido bons resultados para os dois problemas de *flowshop*, o FF foi utilizado neste artigo.

O construtivo obedece a seguinte lógica: para alocar uma tarefa no fim de uma sequência, os autores sugerem que se considere o tempo ocioso causado pela nova tarefa alocada (IT') e o tempo de término da nova atividade na máquina *m* (AT'). Neste construtivo, as tarefas são primeiro ordenadas de forma crescente de ξ (Equação 3). De maneira iterativa, a tarefa com menor ξ vai sendo alocada ao final da solução.

$$\xi = \frac{(n-k-2)}{a} * IT'_{j,k} + AT'_{j,k} \quad (3)$$

$$\text{sendo } IT'_{j,k} = \sum_{i=2}^{m} \frac{m * \max\{C_{i-1}, C_{i,[k]}, 0\}}{i-b+k(m-i+b)/(n-2)} \quad (4)$$

$$\text{e } AT'_{j,k} = C_{m,j}, \quad (5)$$

onde k é o número de tarefas (j) na solução parcial. Os parâmetros a e b equilibram a influência do tempo ocioso e do tempo de término.

3.2. Buscas Locais

Segundo [Den Besten 2001], as duas buscas locais usualmente utilizadas em problemas de *permutation flowshop* são *interchange* e *insertion.* A i*nterchange* consiste em trocar duas tarefas de posição. Por exemplo, se a solução encontrada é {A, B, C, D}, uma das novas soluções testadas poderia ser {A, D, C, B}. Já na *insertion*, uma tarefa é inserida em determinada posição, fazendo com que as outras sejam realocadas na mesma ordem. De {A, B, C, D} poderia virar {A, D, B, C}, por exemplo. Estas buscas podem utilizar estratégias chamadas *First Improvement*, na qual a primeira melhoria encontrada será a adotada, e *Best Improvement*, que seleciona a melhor solução.

3.3 Método proposto

A heurística proposta neste artigo é constituída por um GRASP reativo composto pelo construtivo FF aleatorizado e a posterior aplicação de uma busca local. O FF, portanto, irá calcular os valores de ξ para cada tarefa, representando o custo de alocá-la na última posição disponível. Diferente da abordagem de [Fernandez-Viagas e Framinan 2015] quando a heurística foi originalmente proposta, o algoritmo foi aleatorizado. Dessa forma, o indicador ξ é utilizado para montar a LRC. A partir dela, uma tarefa é selecionada de maneira aleatória para entrar na solução. A seguir são atualizados os valores de $AT'_{j,k}$, $IT'_{j,k}$ e ξ com base na tarefa inserida e uma nova LRC é gerada. Assim, é montada uma solução por meio do construtivo.

Vale ressaltar que no GRASP adotado foi utilizada uma abordagem reativa. Isto é, α é atualizado em intervalos constantes. Com isso, o parâmetro que define a qualidade dos elementos que serão inseridos na LRC se adequa ao longo da execução da heurística de acordo com os melhores resultados.

Após a fase de construção, é implementada uma busca local. Para fins deste artigo, os procedimentos de busca adotados foram o *Interchange* e o *Insertion*. Estas utilizaram o critério de maior melhoria (*best improvement*).

4. Experimentos Computacionais

Esta seção apresenta os resultados encontrados até o momento para a aplicação da metaheurística descrita na seção anterior. Para tal, ela foi implementada na linguagem C++ e

executada em uma máquina com Processador Intel Core i5-4440 CPU @ 3.10GHz, 8 GB de memória RAM e sistema operacional Windows 7.

Como *benchmarking* foram utilizadas as instâncias de [Taillard 1993]. Foram testadas 70 instâncias nas quais a quantidade de *jobs* variava entre {20, 50 e 100} e a de máquinas que variam entre {5, 10 e 20}. Para a instância de 100 *jobs*, apenas o grupo com 5 máquinas foi testado. Para cada dimensão (job x máquina) existem 10 instâncias.

O primeiro passo foi analisar qual procedimento de busca local seria utilizado como forma de produzir os melhores ganhos após a execução do construtivo FF. Para tal, foi realizado um teste no qual foram utilizadas uma instância de 20 *jobs* e 5 máquinas e outra de 50 *jobs* e 5 máquinas. Para cada uma delas foi executada a heurística FF seguida de uma estratégia de busca local e avaliado o seu ganho percentual médio. Esse ganho representa a melhoria que a busca local gerou em relação à solução obtida com o construtivo. Como forma de comparação, apenas uma semente foi utilizada e foram executadas 200 iterações para cada teste.

Como forma de simplificar a notação, Int_BI será definida como uma busca local *Interchange* com *Best Improvement*; Ins_BI como uma *Insertion* com *Best Improvement*; Int_FI como uma *Interchange* com *First Improvement*; Ins_FI como uma *Insertion* com *First Improvement*; Int+Ins_BI como uma *Interchange* com *Best Improvement* seguida por uma *Insertion* com *Best Improvement*; Ins+Int_BI como uma *Insertion* com *Best Improvement* seguida por uma *Interchange* com *Best Improvement*; Int+Ins_FI como uma *Interchange* com *First Improvement* seguida por uma *Insertion* com *First Improvement*; Ins+Int_FI como uma *Insertion* com *First Improvement* seguida por uma *Interchange* com *First Improvement.*

Os resultados consolidados de ganhos médios percentuais sobre a solução da heurística construtiva estão apresentados na Tabela 1.

Busca Local	Ganho Médio (%) 20 *jobs* e 5 máquinas	Ganho Médio (%) 50 *jobs* e 5 máquinas
Int_BI	23,05	28,53
Ins_BI	23,53	27,55
Int_FI	1,88	1,10
Ins_FI	1,90	1,04
Int+Ins_BI	24,20	29,87
Ins+Int_BI	24,04	29,67
Int+Ins_FI	3,10	1,75
Ins+Int_FI	3,56	1,75

Tabela 1. Ganhos médios (%) de cada busca local

Com base nos resultados apresentados para ambas as instâncias, é possível chegar à conclusão de que a estratégia ideal envolve *Best Improvement*, porque, por mais que a execução com *First Improvement* permita que sejam realizadas mais iterações em um tempo mais curto ou que o tempo total de execução seja reduzido, seus ganhos para estas instâncias foram muito inferiores. Assim, as buscas locais com estratégia *First Improvement* foram eliminadas da consideração. A seguir, quando se compara Int+Ins_BI e Ins+Int_BI, que são compostas de duas buscas locais, com Int_BI e Ins_BI, que executam apenas uma busca, constata-se que as soluções melhoram somente de 1 a 2%, mesmo com o tempo a mais dedicado. Dessa forma, Int_BI e Ins_BI foram adotadas como as buscas locais a serem utilizadas nos testes do GRASP reativo.

Já com o GRASP definido com um construtivo FF aleatorizado, com os parâmetros a e b como 4 e 1 como proposto por [Fernandez-Viagas e Framinan 2015], respectivamente, e de abordagem reativa, foram executados dois testes distintos, o primeiro com a busca local sendo a *Interchange* e o segundo com a *Insertion*. Ambos os testes foram realizados com as mesmas 70 instâncias e com 5 sementes cada, utilizando um tempo de oito minutos como critério de parada para cada par

instância-semente, com o objetivo de analisar melhor o desempenho do método proposto. As probabilidades de alfa foram atualizadas a cada 20 iterações.

A escolha de oito minutos como tempo de parada se deu com dois intuitos: possibilitar que as rodadas pudessem ter uma quantidade considerável de atualizações de alfa (para melhor considerar as características reativas do método) e impedir execuções excessivamente longas, caso a parada fosse definida pelo número de iterações. Além disso, todos os resultados apresentados aqui são provenientes de execuções em 70 instâncias, com trinta delas sendo de 20 *jobs,* outras trinta de 50 *jobs* e as últimas 10 de 100 *jobs*.

Também foi utilizado o critério de parada de tempo definido em [Ribas e Companys 2015] como uma forma de comparação de desempenho ao avaliar quantas instâncias atingiram o ótimo em um tempo inferior ao proposto. Os autores fixaram o tempo de execução do algoritmo GRASP em uma fórmula definida como $k.n^2.m.10^{-5}$. Nela, n representa a quantidade de *jobs*, m é o número de máquinas e a constante k era definida como 10 ou 30. Para efeitos deste artigo, as instâncias foram executadas com tempo para k = 30. Dessa forma, para uma instância de 20 *jobs* e 5 máquinas, por exemplo, a metaheurística rodou ao longo de 0,6 segundos.

Também vale ressaltar que o estado da arte em termos de melhores soluções aqui apresentado foi encontrado em [Tasgetiren et al. 2016]. A partir desses resultados, foi tirado o desvio percentual relativo para a solução (RPD), que é definido segundo a Equação 6 abaixo. Depois, foi calculado o ARPD, como a média dos RPDs de cada instância.

$$RPD = \frac{Resultado\ da\ Observação - Melhor\ Solução\ da\ Literatura}{Melhor\ Solução\ da\ Literatura} \quad (6)$$

O primeiro teste de execução foi realizado considerando a busca local *Interchange*. Assim, a Tabela 2 apresenta os resultados dos experimentos, com ela trazendo o melhor e o pior RPD, além do ARPD e do desvio padrão para cada tamanho de instância. Em cada linha estão representadas todas as dez instâncias rodadas de cada tamanho.

Tamanho	RPD Mínimo (%)	RPD Máximo (%)	ARPD (%)	Desvio Padrão RPD (%)
20x5	0,00	0,00	0,00	0,00
20x10	0,00	0,00	0,00	0,00
20x20	0,00	0,10	0,00	0,01
50x5	1,31	2,82	2,15	0,32
50x10	1,59	3,06	2,29	0,32
50x20	1,16	2,53	1,97	0,27
100x5	3,44	5,76	4,46	0,49

Tabela 2. Valores RPD e ARPD com busca local *Interchange*

A Tabela 3 complementa os resultados dos testes com a *Interchange* ao exibir como se comportou a média e o mínimo dos tempos até que o algoritmo alcançasse a melhor solução para cada tamanho de instância. Ela também apresenta a quantidade de iterações mínima e máxima que foram executadas ao longo dos oito minutos.

Tamanho	Tempo mínimo até melhor solução (s)	Tempo médio até melhor solução (s)	Mínimo de iterações	Máximo de iterações
20x5	0,145	8,13	219190	281456
20x10	0,391	29,82	155137	200000
20x20	0,793	62,19	91092	112059
50x5	18,295	273,00	7489	8457
50x10	6,28	173,19	4333	5050
50x20	5,843	260,21	2384	2736
100x5	1,992	259,19	474	500

Tabela 3. Tempo e número de iterações com busca local *Interchange*

Tamanho	RPD Mínimo (%)	RPD Máximo (%)	ARPD (%)	Desvio Padrão RPD (%)
20x5	0,00	0,00	0,00	0,00
20x10	0,00	0,00	0,00	0,00
20x20	0,00	0,00	0,00	0,00
50x5	1,78	3,83	2,85	0,53
50x10	1,49	3,12	2,25	0,40
50x20	0,64	2,20	1,53	0,27
100x5	5,35	8,01	6,63	0,69

Tabela 4. Valores RPD e ARPD com busca local *Insertion*.

Por fim, a Tabela 5, assim como a Tabela 3, traz outros dados relevantes que podem ser percebidos com base no teste realizado com a *Insertion*. Nela podem ser verificadas o mínimo e a média dos tempos até que o algoritmo alcançasse a melhor solução e as quantidades de iterações mínima e máxima que foram executadas até que o critério de parada fosse alcançado.

Tamanho	Tempo mínimo até melhor solução (s)	Tempo médio até melhor solução (s)	Mínimo de iterações	Máximo de iterações
20x5	0,01	17,69	46431	64270
20x10	0,047	6,99	34873	44857
20x20	0,033	3,35	23761	28243
50x5	1,3	230,52	1400	1772
50x10	10,413	303,72	933	1143
50x20	2,057	250,21	596	713
100x5	10,2	205,85	85	103

Tabela 5. Tempo e número de iterações com busca local *Insertion.*

Através dos estudos computacionais representados nas Tabelas 2 e 4, é possível perceber que as 30 instâncias de 20 *jobs* atingiram as melhores soluções da literatura com as duas buscas locais. No entanto, conforme o tamanho das instâncias aumentava, também crescia a diferença para o estado da arte.

Isto se dá porque uma quantidade inferior de iterações é realizada nessas instâncias, já que o porte do problema cresce e a qualidade da solução encontrada cai. Além disso, um estudo dos tempos até o melhor resultado mostra que o método proposto levou de 3 a 300 segundos, em média, para atingir sua máxima eficiência dentro do tempo definido para parada, o que é visto como uma duração aceitável.

Já em uma análise comparativa entre os resultados provenientes de cada abordagem de busca local, a *Interchange* se mostrou mais rápida e capaz de executar uma quantidade de 4 a 5 vezes maior de iterações no tempo proposto. Contudo, isto não necessariamente levou a resultados melhores, com as duas buscas locais tendo um desempenho próximo. Apenas na instância de 100 *jobs* testada foi possível encontrar uma vantagem considerável a favor da *Interchange*.

Além disso, ao executar os dois testes já especificados, os valores de α que geraram as melhores soluções foram armazenados. A partir deles, pode-se notar a variabilidade conforme a quantidade de *jobs* da instância se altera. Isto pode ser verificado na Figura 2, que representa estes resultados

para as rodadas com a busca local *Interchange*, e na Figura 3, que traz a mesma análise, mas para as rodadas com a busca local *Insertion*.

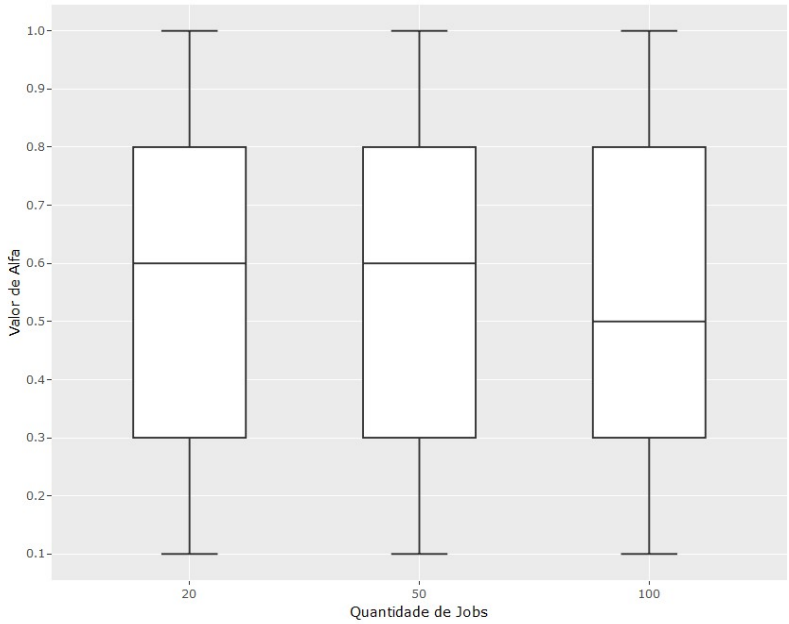

Figura 2. Relação da variação do valor de α (20, 50, 100 jobs) com busca local *Interchange.*

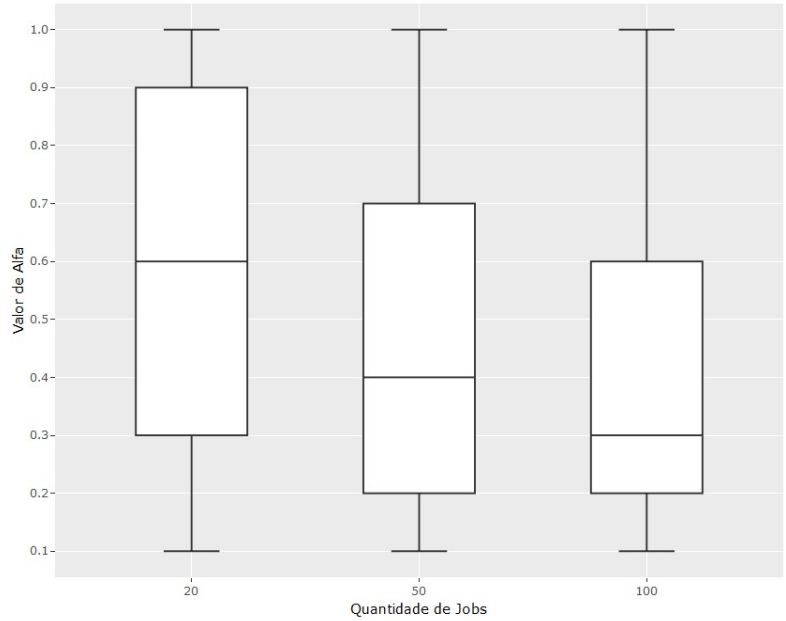

Figura 3. Relação da variação do valor de α (20, 50, 100 jobs) com busca local *Insertion.*

Quando se observa os alfas que geraram as melhores soluções, representados nas Figuras 2 e 3, é possível reparar que foram usados todos os valores descritos e que a abordagem reativa teve valor ao mostrar que instâncias com menos *jobs* apresentam um desempenho superior quando estão com um alfa maior do que em instâncias com mais *jobs*.

Por fim, na Tabela 6 será possível observar o melhor resultado encontrado pelo GRASP proposto pelo artigo para cada instância comparado com os valores reportados pelo GRASP de [Ribas e Companys 2015] e o VBIH de [Tasgetiren et al. 2016].

Instância	Melhor solução	Melhor Ribas e Companys	Melhor Tasgetiren et al.
TA01	14953	14953	14953
TA02	16343	16343	16343
TA03	14297	14297	14297
TA04	16483	16483	16483
TA05	14212	14212	14212
TA06	14624	14624	14624
TA07	14936	14936	14936
TA08	15193	15193	15193
TA09	15544	15544	15544
TA10	14392	14392	14392
TA11	22358	22358	22358
TA12	23881	23881	23881
TA13	20873	20873	20873
TA14	19916	19916	19916
TA15	20196	20196	20196
TA16	20126	20126	20126
TA17	19471	19471	19471
TA18	21330	21330	21330
TA19	21585	21585	21585
TA20	22582	22582	22582
TA21	34683	34683	34683
TA22	32855	32855	32855
TA23	34825	34825	34825
TA24	33006	33006	33006
TA25	35328	35328	35328
TA26	33720	33720	33720
TA27	33992	33992	33992
TA28	33388	33388	33388
TA29	34798	34798	34798
TA30	33174	33174	33174
TA31	73915	72672	72672
TA32	79712	78140	78181
TA33	74495	72913	72913
TA34	78774	77399	77513
TA35	80048	78353	78363
TA36	76391	75402	75402
TA37	75376	73842	73890
TA38	75274	73442	73442
TA39	72108	70871	70871
TA40	80171	78729	78729
TA41	101845	99674	99674
TA42	97560	95608	95669

Tabela 6. Comparação das melhores soluções entre o GRASP Reativo, GRASP de [Ribas e Companys 2015] e o VBIH de [Tasgetiren et al. 2016]. (continua na próxima página)

Tabela 6 (continuação)

Instância	Melhor solução	Melhor Ribas e Companys	Melhor Tasgetiren et al.
TA43	93705	91791	91791
TA44	100267	98454	98454
TA45	99721	98164	98164
TA46	99031	97246	97333
TA47	102298	99953	99953
TA48	99483	98027	98149
TA49	98892	96708	96708
TA50	99483	98019	98053
TA51	138621	136865	136881
TA52	131468	129958	129975
TA53	128430	127617	127617
TA54	133665	131889	131943
TA55	132505	130967	130967
TA56	133460	131760	131760
TA57	136422	134217	134222
TA58	134410	132990	132990
TA59	134409	132599	132599
TA60	137415	135710	135985
TA61	300385	288627	288446
TA62	290278	280491	280073
TA63	285353	276576	275863
TA64	272586	261278	261231
TA65	285019	274638	274005
TA66	280423	267554	267899
TA67	287033	275823	275491
TA68	281748	269872	270668
TA69	296029	285428	284652
TA70	292689	282828	282366

5. Conclusões

Este artigo propõe um GRASP reativo que utiliza o algoritmo FF como construtivo e testa duas abordagens de buscas locais, *Interchange* e *Insertion*, para o problema de *blocking flowshop,* com o propósito de minimizar o *total flowtime*.

Foram testadas 70 instâncias com 5 sementes que, em comparação com as melhores soluções conhecidas [Tasgetiren et al. 2016], resultaram em resultados promissores. A diferença percentual média entre os resultados obtidos neste trabalho e o estado da arte variou entre 0% e 4,46%. Além disso, o ótimo foi atingido em 30 das instâncias testadas, com um tempo inferior ao definido por [Ribas e Companys 2015] em 10 instâncias quando se usava a busca local *Interchange* e em 23 instâncias com a *Insertion*.

Por fim, entre possibilidades de trabalhos futuros, está a aplicação da metaheurística proposta em outros problemas, como o *permutation flowshop* sem *blocking,* ou com outros objetivos, como a minimização do *makespan* ou do atraso total. Também é possível ampliar o estudo a partir do teste de todas as instâncias de [Taillard 1993] e [Pan e Ruiz 2012] utilizando o método proposto. Para tentar melhorar o desempenho da metaheurística, também é proposto realizar testes usando as

estratégias *Variable Neighborhood Search* (VNS) e *Variable Neighborhood Descent* (VND) como procedimentos de busca local aplicados à solução gerada pelo construtivo FF.

Referências

Besten, M. e Stutzle, T. (2001). Neighborhoods Revisited: An Experimental Investigation into the Effectiveness of Variable Neighborhood Descent for Scheduling. Proceedings of The Fourth Metaheuristics International Conference (MIC2001), 545-549.

Binato, S., Hery, W., Loewenstern, D. e Resende, M. (2002). A Grasp for Job Shop Scheduling. in *Essays and Surveys in Metaheuristics*, vol. 15, Boston, MA: Springer US, 2002, pp. 59–79.

Caraffa, V., Ianes S. e Bagchi, T., Sriskandarajah, C. (1999). Minimizing makespan in a blocking flowshop using genetic algorithms. *Int. J. Production Economics,* 70 (2001) 101}115.

Deng, G., Xu, Z. e Gu, X. (2012). A Discrete Artificial Bee Colony Algorithm for Minimizing the Total Flow Time in the Blocking Flow Shop Scheduling. *Chinese Journal of Chemical Engineering*, vol. 20, no. 6, pp. 1067–1073.

Feo, T. e Resende, M. (1989) A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, vol. 8, no. 2, pp. 67–71.

Fernandez-Viagas, V. e Framinan, J. (2015). A new set of high-performing heuristics to minimise flowtime in permutation flowshops. *Computers & Operations Research*, vol. 53, pp. 68–80.

Fernandez-Viagas, V., Leisten, R. e Framinan, J. (2016). A computational evaluation of constructive and improvement heuristics for the blocking flow shop to minimise total flowtime. *Expert Systems with Applications*, vol. 61, pp. 290–301.

Grabowski, J., e Pempera, J. (2000). Sequencing of jobs in some production system. *European Journal of Operational Research*, 125(3), 535–550.

Graham, R. L., Lawler, E. L., Lenstra, J. K., e Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5, 287–326.

Han, Y., Liang, J., Pan, Q., Li, J., Sang, H. e Cao, N. (2013). Effective hybrid discrete artificial bee colony algorithms for the total flowtime minimization in the blocking flowshop problem. *The International Journal of Advanced Manufacturing Technology*, vol. 67, no. 1–4, pp. 397–414.

Lin, S. e Kernighan, B. (2001). “An effective heuristic algorithm for the traveling-salesman problem.”. Operations Research, pp. 498–516, 1973.J. Liu and C. R. Reeves, Constructive and composite heuristic solutions to the P//∑Ci scheduling problem. *European Journal of Operational Research*, vol. 132, no. 2, pp. 439–452.

Liu, J., e Reeves, C. R. (2001). Constructive and composite heuristic solutions to the P//∑Ci scheduling problem. *Data Envelopment Analysis*, 132(2), 439–452.

Den Besten, M., Stutzle, T. (2001). Neighborhoods revisited: An experimental investigation into the effectiveness of variable neighborhood descent for scheduling. Proceedings of the fourth metaheuristics international conference (MIC2001), 545-549.

Moslehi, G. e Khorasanian, D. (2013). Optimizing blocking flow shop scheduling problem with total completion time criterion. *Computers & Operations Research*, vol. 40, no. 7, pp. 1874–1883.

Nouri, N. e Ladhari, T. (2017). Evolutionary multiobjective optimization for the multi-machine flow shop scheduling problem under blocking. *Annals of Operations Research*, vol. 267, no. 1–2, pp. 413–430, 2017.

Pan, Q. e Ruiz, R. (2012). Local search methods for the flowshop scheduling problem with flowtime minimization. *European Journal of Operational Research*, 222(1), 31–43.

Pan Q. e Ruiz R. (2013). A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime. *Computers & Operations Research*, 40(1):117–28.

Pan, Q. e Wang, L. (2011). Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. *Omega,* 40(2012)218–229.

Prais, M. e Ribeiro, C. (2000). Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment. *INFORMS Journal on Computing*, vol. 12, no. 3, pp. 164–176.

Resende, M. e Ribeiro, C. (2019). Greedy Randomized Adaptive Search Procedures: Advances and Extensions. In: Gendreau, Michel & Potvin, Jean-Yves. (2019). Handbook of Metaheuristics.

Ribas, I. e Companys, R. (2015) Efficient heuristic algorithms for the blocking flow shop scheduling problem with total flow time minimization. *Computers & Industrial Engineering*, vol. 87, pp. 30–39.

Ribas, I., Companys, R. e Tort-Martorell, X. (2015). An efficient Discrete Artificial Bee Colony algorithm for the blocking flow shop problem with total flowtime minimization. *Expert Systems with Applications*, vol. 42, no. 15–16, pp. 6155–6167.

Ronconi, D. (2004). A note on constructive heuristics for the flowshop problem with blocking. *International Journal of Production Economics*, 2004;87:39–48.

Ruiz, R. e Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, vol. 165, no. 2, pp. 479–494.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2), 278–285.

Tasgetiren, M., Pan, Q., Kizilay, D., Gao, K. (2016). A Variable Block Insertion Heuristic for the Blocking Flowshop Scheduling Problem with Total Flowtime Criterion. *Algorithm*s, 9(4):71.

Wang, L., Pan, Q., e Tasgetiren, M. (2010). Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms. *Expert Systems with Applications*, vol. 37, no. 12, pp. 7929–7936.

Xie, Z., Zhang, C., Shao, X. e Yin, Y. (2014). Minimizing Total Flow Time in a Flow Shop with Blocking Using a Hybrid Variable Neighborhood Search and Simulated Annealing Algorithm. *Applied Mechanics and Materials*, vol. 631–632, pp. 57–61.