Um algoritmo populacional para o problema de *flow shop* com bloqueio e minimização do *makespan*

Ewerton Teixeira[1], Anand Subramanian[2], Hugo Harry Kramer[3]

[1]Programa de Pós-Graduação em Engenharia de Produção e Sistemas
Universidade Federal da Paraíba (UFPB) – João Pessoa, PB – Brasil

[2]Departamento de Sistemas de Computação
Universidade Federal da Paraíba (UFPB) – João Pessoa, PB – Brasil

[3]Departamento de Engenharia de Produção
Universidade Federal da Paraíba (UFPB) – João Pessoa, PB – Brasil

ewertonvppt@gmail.com, anand@ci.ufpb.br, hugo.kramer@academico.ufpb.br

Abstract. *This work addresses the blocking flow shop scheduling problem with makespan minimization. We propose a population-based algorithm that uses a ruin-and-recreate operator along with a local search based on Variable Neighbourhood Descent to solve the problem. Computational experiments were carried out on 120 benchmark instances and the proposed method was capable of finding competitive solutions when compared to the literature.*

Resumo. *Este trabalho aborda o problema de flow shop com bloqueio para minimização do makespan. Um algoritmo populacional que utiliza um operador ruin-and-recreate juntamente de uma busca local baseada em Variable Neighbourhood Descent é proposto para o problema. Experimentos computacionais foram realizados em 120 instâncias de benchmark e o método proposto foi capaz de obter soluções competitivas quando comparadas com a literatura.*

1. Introdução

Dado um conjunto de n tarefas e m máquinas distintas organizadas em série, o problema de *flow shop* com bloqueio, do inglês *blocking flow shop scheduling problem* (BFSP), consiste em encontrar uma permutação $\pi = (\pi_1, \pi_2, \pi_3, \ldots, \pi_n)$, em que π_k representa a k - ésima tarefa da permutação, a ser processada nas m máquinas minimizando o *makespan*. O estoque intermediário entre duas máquinas consecutivas é igual a zero. Isto é, caso uma tarefa termine seu processamento em uma máquina i e a máquina $i+1$ estiver processando uma outra tarefa, ocorre um bloqueio na máquina i e a próxima tarefa da permutação não poderá ser processada até que a máquina i seja liberada.

Além disso, cada tarefa $\pi_k \in \pi$ possui um tempo de processamento associado a uma máquina i denotado por $p_{\pi_k,i}$. Assim, seja $D_{\pi_k,i}$ o tempo em que a k-ésima tarefa de π deixa a máquina i, $D_{\pi_k,i} \geq C_{\pi_k,i}$, em que $C_{\pi_k,i}$ é o tempo de conclusão do processamento da tarefa π_k na máquina i. O caso em que $D_{\pi_k,i} = C_{\pi_k,i}$ ocorre quando não há um bloqueio entre as máquinas i e $i+1$. O *makespan* é dado por $C_{max} = D_{\pi_n,m}$. Dadas as características do problema, ele é conhecidamente NP-Difícil para $m > 2$ [Röck 1984, Martinez et al. 2006, Fernandez-Viagas et al. 2016] e é denotado por $F_m|block|C_{max}$ conforme a notação de três campos de [Graham et al. 1979].

Este trabalho propõe uma abordagem baseada em uma busca populacional híbrida (BPH_{BFSP}) que utiliza um operador *ruin-and-recreate* para gerar novos indivíduos em uma população. Estes indivíduos são submetidos a uma busca local, baseada em *Variable Neighbourhood Descent*, que é constituída de três vizinhanças que incorporam métodos de aceleração da busca. Além disso, um mecanismo de gerenciamento de diversidade da população é utilizado para evitar que o algoritmo fique preso em ótimos locais. Experimentos computacionais realizados demonstram que o algoritmo é capaz de encontrar soluções de alta qualidade quando comparadas com limites superiores encontrados na literatura e reportados no trabalho de [Shao et al. 2019].

2. Metodologia

O método proposto é descrito no Algoritmo 1. Note que o método possui um total de cinco parâmetros, que serão explicados no decorrer do trabalho.

Algoritmo 1 BPH_{BFSP}

procedimento $\text{BPH}_{\text{BFSP}}(\mu, \lambda, d, \mu^{elite}, \mu^{close})$
    Gere uma população P, de μ indivíduos, utilizando a heurística $PF - NEH$
    Submeta os μ indivíduos à busca local
    enquanto o critério de parada não é atingido **faça**
        Selecione um indivíduo S_1 de P utilizando torneio binário
        Use um operador *ruin-and-recreate* em S_1 e gere um indivíduo S
        Submeta S à busca local
        Insira S na população P
        se $|P| = \mu + \lambda$ **então**
            Descarte λ indivíduos, considerando a qualidade e a diversidade deles
    retorne a melhor solução da população

2.1. Geração da População Inicial

Uma variação da heurística construtiva PF-NEH proposta por [Pan and Wang 2012] foi utilizada para gerar os μ indivíduos de P. O método está destacado no Algoritmo 2.

Algoritmo 2 PF-NEH Adaptado

procedimento $PF - NEH$ ADAPTADO(n)
    Selecione uma tarefa α aleatoriamente para ser a tarefa inicial de π
    Construa a permutação π a partir de α utilizando a heurística *Profile Fitting*
    Remova δ tarefas no final de π e as insira em uma lista LC
    enquanto LC não estiver vazia **faça**
        Selecione uma tarefa $\beta \in LC$
        Reinsira β em π na posição que resultar no menor *makespan*
        Remova β de LC
    retorne π

2.2. Geração de Novos Indivíduos

A cada iteração, o operador *ruin-and-recreate*, similar ao utilizado em [Ruiz and Stützle 2007], é aplicado a um indivíduo π que é selecionado por meio de torneio binário. Após a seleção, remove-se, aleatoriamente, d tarefas de π formando uma permutação parcial π_D. Em seguida, elas são reinseridas, em ordem aleatória, na posição de π_D que resulte no menor *makespan* e que não seja a posição em que a tarefa estava anteriormente. O processo acaba quando as d tarefas são reinseridas e um novo indivíduo π' é gerado. Por fim, π' é submetido à busca local.

2.3. Busca Local

A busca local utiliza vizinhanças que são exploradas de forma determinística, conforme a ordem descrita a seguir. Quando uma vizinhança não melhora a solução corrente, o algoritmo prossegue para a próxima. Por outro lado, quando uma vizinhança consegue uma melhora, o algoritmo retorna para a primeira vizinhança ao finalizar a exploração da corrente. O procedimento chega ao fim quando todas as vizinhanças são exploradas e nenhuma melhora é obtida. Um total de 3 vizinhanças foram utilizadas:

Insertion: Uma tarefa é reinserida em uma nova posição de π.

Block Insertion: Um bloco de 2 tarefas é reinserido em uma nova posição de π.

Swap: Duas tarefas da solução π são trocadas de posição entre si.

Além disso, métodos de aceleração foram implementados para cada uma das vizinhanças. Para a vizinhança *insertion*, foram implementados uma aceleração do movimento que reduz a complexidade computacional de explorar a vizinhança de $\mathcal{O}(n^3m)$ para $\mathcal{O}(n^2m)$, e um limitante inferior que pode ser calculado em $\mathcal{O}(1)$, evitando avaliações de movimento que seriam feitas em $\mathcal{O}(m)$, ambos já conhecidos na literatura. Além disso, a aceleração foi adaptada para as vizinhanças *block insertion* e *swap* e um limitante inferior para a vizinhança *swap* que pode ser calculado em $\mathcal{O}(1)$ também foi utilizado. Tais métodos podem ser consultados no trabalho de [Teixeira et al. 2022].

2.4. Gerenciamento de Diversidade

O gerenciamento de diversidade é similar ao utilizado por [Mecler et al. 2021], utilizando uma função de aptidão $f_p(\pi)$. O valor dessa função depende diretamente de dois parâmetros: o número μ^{close} de indivíduos mais próximos, e o número de indivíduos de elite μ^{elite} que se deseja preservar. Assim, a distância média de um indivíduo para seus indivíduos mais próximos, dada pelo número de arestas distintas, é calculada. Em seguida, a população é ordenada em ordem decrescente de contribuição da diversidade, e em ordem crescente de função objetivo, associando a cada indivíduo um *rank* de diversidade $f_p^{div}(\pi)$ e um *rank* de qualidade $f_p^{obj}(\pi)$. Assim, $f_p(\pi)$ é dada pela Equação (1).

$$f_p(\pi) = f_p^{obj}(\pi) + (1 - \frac{\mu^{elite}}{|P|}) \times f_p^{div}(\pi) \tag{1}$$

Esta medida de aptidão é usada ao selecionar os indivíduos por torneio binário e ao selecionar indivíduos para excluir da população durante as seleções de sobreviventes. A exclusão acontece até que o tamanho desejado da população (μ) seja atingido.

3. Experimentos Computacionais

Os experimentos computacionais foram realizados nas 120 instâncias propostas por [Taillard 1993] e a seguinte parametrização foi utilizada: $\mu = 20$, $\lambda = 20$, $d = 5$, $\mu^{elite} = 10$ e $\mu^{close} = 3$. Como critério de parada do algoritmo, utilizou-se um tempo limite de execução igual a $100 \times n \times m$ milissegundos, assim como foi adotado em [Shao et al. 2019] e os resultados obtidos foram comparados com os limites superiores apresentados pelos mesmos autores. O BPH_{BFSP} foi implementado na linguagem de programação C++ e os experimentos foram feitos em um Intel(R) Core(TM) i7-1065G7 com 1.30GHz, 8 GB de memória RAM e sistema operacional Linux Ubuntu 20.04.

A Tabela 1 mostra os resultados sumarizados obtidos. A primeira coluna apresenta o grupo de instâncias (tarefas x máquinas), as duas próximas colunas indicam a média do *gap* mínimo quando os resultados são comparados com a melhor solução reportada por [Shao et al. 2019] e o melhor resultado encontrado pelo método proposto pelos autores, as duas últimas colunas reportam a média do *gap* médio quando a média das 10 execuções de cada instância é comparada com a melhor solução reportada por [Shao et al. 2019] e o melhor resultado encontrado pelo método proposto pelos autores.

Tabela 1. Resultados sumarizados

Instancia	$Gap_{Min\text{-}BKS}$	$Gap_{Min\text{-}DIWO}$	$Gap_{Avg\text{-}BKS}$	$Gap_{Avg\text{-}DIWO}$
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	-0,09%	-0,18%	0,06%	-0,03%
50x10	0,05%	-0,13%	0,26%	0,08%
50x20	0,00%	-0,03%	0,13%	0,10%
100x5	-0,43%	-0,55%	-0,18%	-0,30%
100x10	-0,31%	-0,43%	-0,01%	-0,13%
100x20	-0,22%	-0,32%	0,10%	0,00%
200x10	-0,29%	-0,30%	0,00%	-0,01%
200x20	0,07%	-0,15%	0,36%	0,14%
500x20	0,47%	0,27%	0,67%	0,47%

A partir do exposto, conclui-se que o método proposto é capaz de obter soluções competitivas quando comparado com outros métodos da literatura. No geral, o algoritmo encontra mais dificuldade nas instâncias com 500 tarefas. Em suma, a Tabela 1 apresenta um resumo dos experimentos, que reflete os seguintes resultados: melhores soluções do que os melhores limites superiores reportados em [Shao et al. 2019] em um total de 62 instâncias e empate em 46 outras, perdendo em apenas 12 instâncias, sendo 10 delas instâncias de 500 tarefas. Além disso, encontrou soluções melhores do que o DIWO proposto pelos autores em 73 instâncias distintas e empatou em 38 outras, perdendo em um total de 9 instâncias, sendo 9 delas instâncias de 500 tarefas.

4. Conclusão

Um algoritmo populacional foi proposto para o problema de *flow shop* com bloqueio e minimização do *makespan*. O método foi capaz de obter soluções competitivas, obtendo melhores soluções em 51,67% das instâncias e empatando em 38,33% quando comparado com soluções de alta qualidade da literatura. Como trabalhos futuros, sugere-se a adaptação dos limites inferiores e da aceleração da busca local para outras variantes do problema, além do desenvolvimento de métodos que identifiquem movimentos promissores na busca local.

Referências

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

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey. In *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287–326. Elsevier.

Martinez, S., Dauzère-Pérès, S., Gueret, C., Mati, Y., and Sauer, N. (2006). Complexity of flowshop scheduling problems with a new blocking constraint. *European Journal of Operational Research*, 169(3):855–864.

Mecler, J., Subramanian, A., and Vidal, T. (2021). A simple and effective hybrid genetic search for the job sequencing and tool switching problem. *Computers & Operations Research*, 127:105153.

Pan, Q.-K. and Wang, L. (2012). Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. *Omega*, 40(2):218–229.

Röck, H. (1984). Some new results in flow shop scheduling. *Zeitschrift für Operations Research*, 28(1):1–16.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European journal of operational research*, 177(3):2033–2049.

Shao, Z., Pi, D., Shao, W., and Yuan, P. (2019). An efficient discrete invasive weed optimization for blocking flow-shop scheduling problem. *Engineering Applications of Artificial Intelligence*, 78:124–141.

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

Teixeira, E. V. P. d. P., Subramanian, A., and Kramer, H. H. (2022). Uma heurística para o problema de flow shop com bloqueio e minimização do makespan. In *Anais do LIV Simpósio Brasileiro de Pesquisa Operacional, Juiz de Fora, November 8–11, 2022*.