*Revista Terra & Cultura: Cadernos de Ensino e Pesquisa*
*ISSN 2596-2809*

# DESENVOLVIMENTO DE UMA HIPER-HEURÍSTICA APLICADA AO ESCALONAMENTO EM PROBLEMAS DE JOB SHOP

# DEVELOPMENT OF A HYPER-HEURISTIC APPLIED TO SCHEDULING IN JOB SHOP PROBLEMS

João Vitor da Costa Andrade[1]
Simone Sawasaki Tanaka[2]
Sérgio Akio Tanaka[3]

## RESUMO

O *job shop scheduling problem* (*JSSP*) é um problema complexo de otimização combinatória pertencente à classe de problemas NP-Hard, que representa o escalonamento de um sistema de manufatura. Para encontrar escalonamentos otimizados para o *JSSP*, uma das técnicas que têm sido utilizadas é a meta-heurística, como os algoritmos genéticos e colônia de formigas. Entretanto, trabalhos recentes da literatura indicam uma boa alternativa com o uso de hiper-heurísticas, que é uma abordagem para controle e seleção de heurísticas. A vantagem das hiper-heurísticas diante de outras heurísticas/meta-heurísticas é sua capacidade de operar com um bom desempenho em diferentes configurações de um problema, o que é uma das principais complexidades do JSSP. Dentre estas heurísticas, o *Modified Choice Function* (*MCF*) demonstrou efetividade quando aplicada como uma hiper-heurística de seleção sobre problemas combinatórios (*Knapsack*), porém ainda não há trabalhos na literatura que indicam o uso em *JSSP*. O objetivo deste trabalho é propor o uso do *MCF* no *JSSP*, gerenciando um algoritmo genético por meio da alternância de seus operadores para a redução do *makespan*. Conclui-se que o algoritmo proposto obteve bons resultados em todos os *datasets*, atingindo resultado superior do que próximo ao melhor algoritmo aplicado.

**Palavra-chave:** Hiper-heurística; *Job Shop Scheduling Problem*; Algoritmo genético.



## ABSTRACT

The job shop scheduling problem (JSSP) is a complex combinatorial optimization problem belonging to the NP-Hard problem class, which represents the scheduling of a manufacturing system. To find optimized schedules for the JSSP, one of the techniques that have been used are meta-heuristics, such as genetic algorithms and ant colony. However, recent works in the literature indicates a good alternative in the

[1] Graduando em Ciência da Computação, Centro Universitário Filadélfia – UniFil. Departamento de Computação. Londrina – Paraná – Brasil. 86020-000 – jvcostaandrade@edu.unifil.br
[2] Docente do Centro Universitário Filadélfia – UniFil. Departamento de Computação. Londrina – Paraná – Brasil. 86020-000 – simone.tanaka@unifil.br
[3] Docente do Centro Universitário Filadélfia – UniFil. Departamento de Computação. Londrina – Paraná – Brasil. 86020-000 – sergio.tanaka@unifil.br

use of hyper-heuristics, which is an approach of heuristic control and selection. The advantage of hyper-heuristics over other heuristics/meta-heuristics is their ability to perform well in different configurations of a problem, which is one of the main complexities of the JSSP. Among these heuristics, the Modified Choice Function (MCF) is a heuristic that has already shown effectiveness when applied as a selection hyper-heuristic on other combinatorial problems, however, there are still no studies on it's use in the classical JSSP. The objective of this work is to propose the use of MCF in the JSSP, managing a genetic algorithm by alternating its operators to reduce the makespan. It was concluded that the proposed algorithm obtained good results in all datasets, reaching a result superior or close to the best applied algorithms.

**Keywords:** Hyper-heuristic, Job Shop Scheduling Problem; Genetic Algorithm.

## 1 INTRODUÇÃO

O *JSSP* é um problema de otimização NP-Hard, que representa um sistema inspirado em uma manufatura baseado em produção de um conjunto de tarefas (*jobs*) para se atingir o produto final. Segundo Sanchez et al. (2020), o *JSSP* é primordial para a indústria em geral devido a sua capacidade de produzir programas da produção eficientes de acordo com a demanda necessária.



De acordo com Huang e Liao (2008), os jobs são formados por sequências de operações para gerar um produto final. As operações possuem uma ordenação de processamento e cada uma é atribuída a uma determinada máquina. O desafio desta representação é determinar a melhor sequência de processamento das operações, respeitando a ordenação da mesma e considerando as limitações impostas para o problema. A otimização desta sequência busca reduzir o tempo total de produção (*makespan*) (CHENG; GEN; TSUJIMURA, 1996).

Para reduzir o makespan do *JSSP,* na literatura têm se adotado meta-heurísticas. Entre estes, destaca-se o algoritmo genético (*GA*), pois é umas das meta-heurísticas com alta flexibilidade de se implementar, hibridizar e aprimorar para um problema específico. Isso é possível devido seu poucos parâmetros matemáticos e funcionamento independente de especificidades internas de um problema.

Porém, tendo em vista a diversidade de configurações de ambientes de manufatura, é difícil um algoritmo meta-heurístico otimizado manter bom desempenho em todas as variações de um ambiente ou problema. Além disso, as meta-heurísticas precisam de constantes modificações e é difícil prever qual a heurística adequada para um problema (GUO; WANG; HAN, 2010).

Devido a esta dificuldade, as hiper-heurísticas ganharam relevância neste cenário, pois, de acordo com a sua premissa de flexibilidade na sua aplicação em diferentes configurações de um problema, manteve um desempenho satisfatório (DRAKE et al., 2020).

O Modified Choice Function (MCF), baseado no Choice function (CF), é uma hiper-heurística que realiza seleção de outras heurísticas para resolver um problema, se destaca pelo seu bom desempenho (DRAKE, 2014). Além disso, não há indícios na literatura de que o *MCF* foi aplicado ao *JSSP* e em conjunto com o *GA*.

Visto as vantagens do algoritmo genético e hiper-heurística, existe a hipótese de que pode ser utilizada uma hiper-heurística baseado no *GA* por meio da alternância de seus operadores via técnica *MCF*, para minimizar *makespan*.

Sendo assim, este trabalho é de natureza aplicada de objetivo exploratório, a qual busca desenvolver e analisar o desempenho da hiper-heurística proposta no *JSSP* por meio da análise estatística do valor de *makespan*. Para atingir este objetivo, a metodologia é composta pelo desenvolvimento de um ambiente de testes, onde é possível executar representações de um sistema JSSP e os algoritmos para solucioná-lo. Além disso, é composto pela proposta de adaptação do *MCF* no *GA* para obter soluções para o *JSSP e* comparar seu desempenho com outros algoritmos da literatura.



## 2 DESENVOLVIMENTO

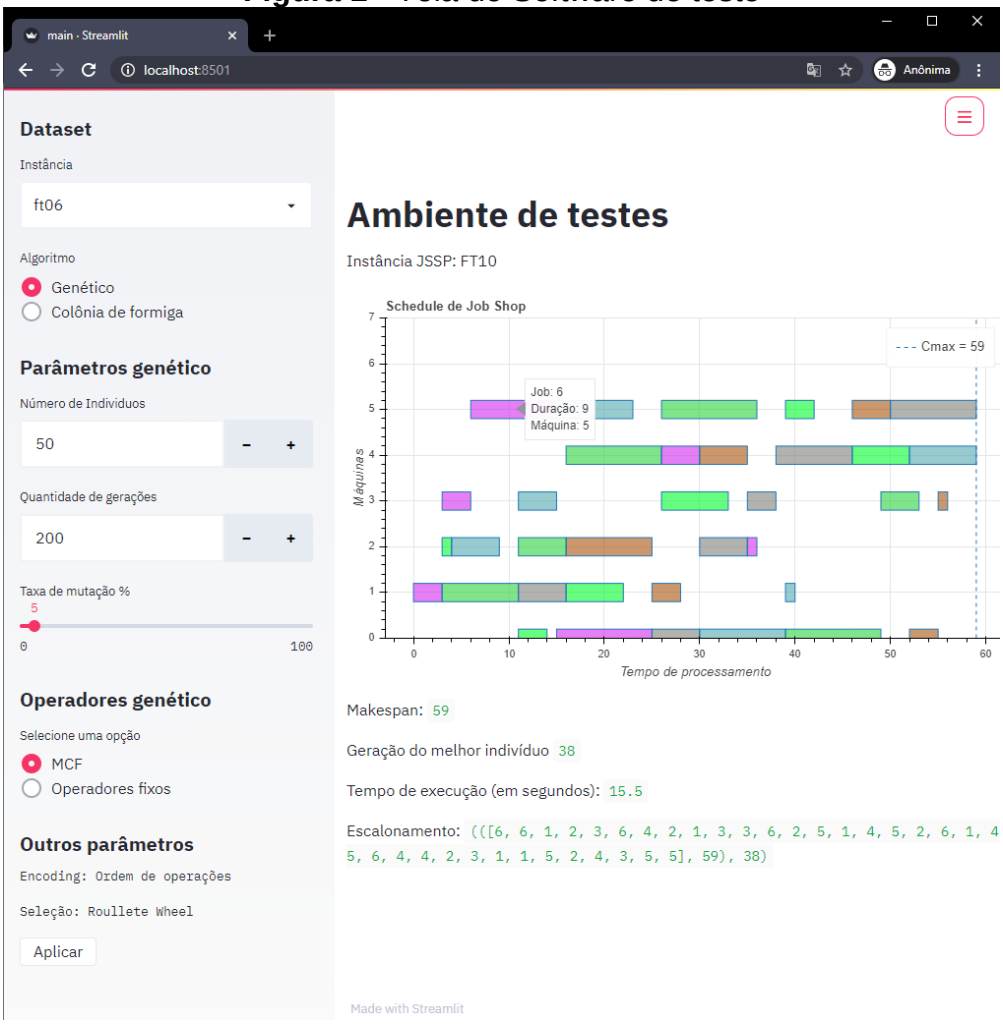
Inicialmente foi implementado o GA clássico na linguagem Python versão 3.5.8 com os operadores de cruzamento OX2 e PMX. Os operadores de mutação implementados foram Swap, Insert e *Inverse*. Além disso, foi implementado o algoritmo colônia de formiga para fornecer comparativo de desempenho.

Em um segundo momento foi realizado o desenvolvimento do *MCF* no *GA*. O *GA* fornece para o *MCF* melhor makespan encontrado antes e depois da aplicação de uma heurística selecionada pelo mesmo para que o *MCF* possa avaliar o desempenho das heurísticas selecionadas. Porém, para executar estes algoritmos em instâncias *JSSP* e exibir os resultados, se fez necessário desenvolver um ambiente de testes.

O ambiente foi desenvolvido utilizando a biblioteca *Streamlit*, a qual gera um aplicativo *web* acessível por meio de um endereço local. Neste, conforme demonstra a Figura 1, é possível selecionar a representação do problema JSSP (*dataset*) a ser aplicado por meio de uma caixa de seleção. Em seguida, é possível selecionar o algoritmo de uma lista de opções a ser aplicado na instância de problema. Por fim, com o algoritmo selecionado, é possível configurar os parâmetros necessários para o seu funcionamento.

A saída do aplicativo, dado um *dataset* e um algoritmo qualquer, é: o gráfico de *Gantt* que descreve a melhor solução, o escalonamento, valor do *makespan*, tempo de execução e informações complementares relacionadas ao algoritmo. Assim como demonstrado um exemplo de execução do *GA* tradicional na Figura 1.

**Figura 1** - Tela do Software de teste



## 3 RESULTADOS E DISCUSSÃO

Todos os resultados a seguir foram obtidos em execuções realizadas em um computador com processador *Ryzen R7 1800x* com 3.6GHz e 16GB de memória. Os *datasets* utilizados foram: *ft06*, *ft10* (FISHER; THOMPSON, 1963). Cada instância foi executada dez vezes com o algoritmo proposto, o algoritmo colônia de formiga (*ACO*) e seis variações do *GA* com diferentes combinações de operadores genéticos.

Testes empíricos foram realizados de modo a determinar a parametrização dos algoritmos para utilização em todos os *datasets*. Neste sentido, o *ACO* foi configurado com os seguintes valores de parâmetros: número de formigas = 50, número de iterações = 100, influência do valor de feromônios (*alpha*) = *0.2*, influência do tempo/valor heurístico (*beta*) = 0.5 e taxa de evaporação (*rhô*) = 0.17

Os parâmetros utilizados nas seis variações do *GA* e na proposta foram: número de indivíduos = 50, número de gerações = 200 e taxa de mutação = 0.5.

Nas Tabela 1 e 2 são apresentados os resultados dos testes no *dataset ft06* e ft10 respectivamente. O menor valor da coluna é destacado em verde e o maior valor é destacado em vermelho. Os dados inexistentes estão marcados com o sinal de hífen (-). O algoritmo da proposta pode ser identificado pelo nome de *GA-MCF* e as variações do *GA* podem ser identificadas pelo nome *GA* seguido da sigla do operador de cruzamento e mutação (ex *GA PMX-Swap*). Os dados mais relevantes estão na coluna “menor valor”.



**Tabela 1** - Resultados obtidos na execução do *dataset ft06*

| instância | Algoritmo | Média | Menor valor | Maior valor | Desvio padrão | Moda | Tempo médio de execução (seg.) |
|---|---|---|---|---|---|---|---|
| ft06 | GA-MCF | 60,5 | 55 | 63 | 2,17 | 61 | 9,94 |
| | GA OX2-Insert | 59,7 | 58 | 61 | 1,25 | 61 | 4,35 |
| | GA OX2-Inverse | 60,4 | 57 | 64 | 2,32 | 61 | 4,26 |
| | GA OX2-Swap | 59,8 | 58 | 62 | 1,23 | 59 | 4,35 |
| | GA PMX-Insert | 62,8 | 57 | 65 | 1,87 | 63 | 5,01 |
| | GA PMX-Inverse | 67,6 | 62 | 76 | 4,25 | 66 | 4,98 |
| | GA PMX-Swap | 66,5 | 61 | 73 | 3,57 | 70 | 5,01 |
| | ACO | 70,6 | 69 | 73 | 1,71 | 69 | 12,97 |

**Tabela 2** - Resultados obtidos na execução do *dataset ft10*

| instância | Algoritmo | Média | Menor valor | Maior valor | Desvio padrão | Moda | Tempo médio de execução (seg.) |
|---|---|---|---|---|---|---|---|
| ft10 | GA-MCF | 1201,1 | 1101 | 1334 | 72,11 | - | 32,99 |
| | GA OX2-Insert | 1203,9 | 1127 | 1333 | 63,04 | - | 47,00 |
| | GA OX2-Inverse | 1210,3 | 1136 | 1285 | 46,65 | 1204 | 47,67 |
| | GA OX2-Swap | 1202,1 | 1134 | 1309 | 45,86 | 1202 | 47,69 |
| | GA PMX-Insert | 1428,9 | 1355 | 1519 | 50,28 | - | 47,97 |
| | GA PMX-Inverse | 1656,9 | 1529 | 1814 | 87,49 | - | 47,70 |
| | GA PMX-Swap | 1440,7 | 1343 | 1507 | 53,08 | - | 47,77 |
| | ACO | 1727,2 | 1665 | 1764 | 29,32 | - | 102,82 |

Como pode ser observado na Tabela 1, a proposta (GA-MCF) obteve o menor *makespan* entre as execuções, sendo este o menor valor conhecido para este *dataset*. Os valores médio, menor valor e moda da proposta são próximos aos dos menores valores, sendo assim, se configura como um dos melhores algoritmos. Já na Tabela 2, o algoritmo proposto atingiu o melhor desempenho, obtendo o menor valor médio, menor valor e demonstrou uma boa vantagem no tempo de execução.



Observa-se que nos resultados apresentados anteriormente, o *ACO* atingiu os piores resultados em todos os testes e com tempo de execução maior em relação aos demais algoritmos. Além do *ACO*, os *GA* que utilizam técnicas de mutação *PMX* estiveram constantemente presente nos piores resultados. Já os melhores resultados foram atingidos pelo algoritmo proposto GA-MCF e pelos *\GA* com operador de cruzamento OX2.

Nos testes, apenas a proposta  atingiu o menor valor conhecido de *makespan* para o *dataset* a qual foi executada. Este resultado foi atingido na instância *ft06* com o valor de makespan 55.

Por fim, tendo em vista que a proposta superou o *ACO*, as variações do *GA* com *PMX* e manteve-se entre os melhores em todos os *datasets*, a hipótese de redução do *makespan* com uso de hiper-heurística deste trabalho pode ser confirmada.

## 4 CONCLUSÃO

No *JSSP*, é difícil encontrar o melhor algoritmo para cada instância do problema. Além disso, estes algoritmos tendem a não manter um bom desempenho

de *makespan* em todas as instâncias. Por isso, hiper-heurísticas têm sido aplicadas para selecionar o melhor algoritmo em um problema. Entre as técnicas hiper-heurísticas, destaca-se o *MCF*. Este algoritmo, na literatura, não encontra-se evidências de que foi aplicado ao *JSSP*.

O objetivo deste trabalho foi o desenvolvimento de uma hiper-heurística baseada no algoritmo genético, com a aplicação da heurística de seleção *MCF* para selecionar os operadores genéticos de cruzamento e mutação, de modo a reduzir o *makespan*. O algoritmo foi desenvolvido, testado em dois *datasets* e seus resultados foram comparados com o *ACO* e seis variações do GA.

Os resultados demonstram que o algoritmo proposto apresentou desempenho superior em relação às variações do *GA* em todas as instâncias executadas.

## REFERÊNCIAS

CHENG, R.; GEN, M.; TSUJIMURA, Y. A tutorial survey of job-shop scheduling problems using genetic algorithms. representation. **Computers & Industrial Engineering**, v. 30, n. 4, p. 983–997, set. 1996.

DRAKE, J. H. **Crossover Control in Selection Hyper-heuristics: Case Studies Using MKP and HyFlex**. 2014. Tese (Doutorado em Filosofia) - University of Nottingham, Reino Unido, 2014.

DRAKE, J. H. et al. Recent advances in selection hyper-heuristics. **European journal of operational research**, v. 285, n. 2, p. 405–428, set. 2020.

FISHER, H.; THOMPSON, G. Probabilistic learning combinations of local job-shop scheduling rules. *In*: **Industrial Scheduling**. [*S.l.*]: Prentice Hall, 1963. p. 225–251.

GUO, P.; WANG, X.; HAN, Y. The enhanced genetic algorithms for the optimization design. *In*: INTERNATIONAL CONFERENCE ON BIOMEDICAL ENGINEERING AND INFORMATICS, 3., 2010, China. **Anais** [...]. China: IEEE, 2010.