REVISÃO SISTEMÁTICA DE META-HEURÍSTICAS PARA FLEXIBLE JOB SHOP SCHEDULING PROBLEM (FJSSP)

Data de aceite: 01/03/2023

Rebeca Emi Ito
Curso de Bacharelado em Ciência da Computação
Londrina

Trabalho de Conclusão de Curso apre- sentado ao Centro Universitário Filadél- fia como parte dos requisitos para obten- ção de graduação em Curso de Bachare- lado em Ciência da Computação. Orienta- dor: Simone Sawasaki Tanaka.

RESUMO: O *flexible job shop scheduling problem* (FJSSP) é um obstáculo presente na computa- ção e na manufatura, onde ambos procuram otimizar o tempo de produção. Na com- putação ele apresenta uma complexidade do tipo NP-Hard, onde deve ordenar *n jobs* com *m* máquinas de maneira que o processamento seja o mais rápido e eficiente, e que a seleção da operação e máquina não convirjam com as outras *jobs*. Para solucio- nar o FJSSP são propostos a utilização de meta-heurísticas, que são algoritmos para resolver problemas diversos, diferente da heurística que visa resolver um problema em específico e hiper-heurísticas que selecionam heurísticas e meta-heurísticas que melhor soluciona o problema. Dentro da meta-heurística o algoritmo genético (GA) é o mais utilizado, devido a sua implementação simples, métodos como o *ant colony optimization* (ACO) e *quantum particle swarm optimization* (QPSO) são explorados, desenvolvendo novos algoritmos com melhores resultados. Este trabalho tem como objetivo realizar uma revisão sistemática de meta-heurísticas e híper-heurísticas para o FJSSP. Através de leitura e análise de artigos e trabalhos foi possível observar que o GA é um dos métodos utilizados pelos pesquisadores dessa área para a resolução do FJSSP.
PALAVRAS-CHAVE: FJSSP, Meta-heurística, JSSP.

ABSTRACT: The flexible job shop scheduling problem (FJSSP) is an obstacle in the computation and manufacture fields, having the objective of optimizing production time. The FJSSP has a complexity of NP-Hard in the computation, where it must schedule a sequence of *n* jobs and *o* operations with *m* machines resulting in a faster and more efficient pro- cessing time, without having the same operation or machine processing at the same time as the

other jobs. To solve FJSSP it's often used meta-heuristics, they are algo- rithms that solve multiple problems, unlike heuristics that solve specific problems and hyper-heuristics that choose the best heuristic or meta-heuristic to solve the problem. In the meta-heuristic field, the genetic algorithm (GA) is more used among researchers. The reason it's most likely to be because the implementation of the GA is easier and simpler than the others in that field. Methods like ant colony optimization (ACO) and quantum particle swarm optimization (QPSO) are also used to solve FJSSP, they are also used to generate new algorithms that are faster and better. The objective of this paper is to make a systematic review of meta-heuristics and hyper-heuristics for FJSSP. After reading and analyzing papers and projects, it was possible to observe that GA was the most used and researched to solve FJSSPs.
KEYWORDS: FJSSP, Meta-heuristic, JSSP.

LISTA DE ABREVIATURAS E SIGLAS

ACO: *Ant Colony Optimization*
DJSSP: *Dynamic Job Shop Scheduling Problem* DFJSSP: *Dynamic Flexible Job Shop Scheduling Problem* FJSSP: *Flexible Job Shop Scheduling Problem*
GA: *Genetic Algorithm*
JSSP: *Job Shop Scheduling Problem* MDP: *Markov Decision Process* PSO: *Particle Swarm Optimization*
QPSO: *Quantum Particle Swarm Optimization*
RL: *Reinforcement Learning*

1 | INTRODUÇÃO

Job shop scheduling problem (JSSP) é uma área da computação e manufatura que procuram encontrar uma sequência de operações e máquinas que processem to- dos os *jobs* o mais rápido e eficiente possível, possuindo o tempo de processamento de cada operação é pré-determinado para cada máquina com complexidade NP-Hard.

Dentro do JSSP há outras áreas que surgiram dela com maiores níveis de complexidade, sendo um deles o *flexible job shop scheduling problem* (FJSSP) que possui um conjunto de possibilidade de máquinas que podem processar a operação ao invés de uma combinação estática, como o JSSP, e ele será o foco deste trabalho.

Para a solução do FJSSP são utilizados meta-heurísticas, que são algoritmos responsáveis por resolver problemas mais genéricos, sendo possível solucionar diferentes problemas que são similares em alguns aspectos. Por o FJSSP possuir uma camada extra de dificuldade, que é a seleção de máquinas para as operações, são utilizadas as metaheurísticas para resolver estes problemas.

A meta-heurística em geral é muito utilizado por pesquisadores para a reso- lução do FJSSP, sendo o algoritmo genético (GA) o mais utilizado, como pode ser observado no trabalho de *Coelho et al. 2021*, não só o GA é um dos assuntos mais pesquisado, mas é a área mais explorada, principalmente pela sua simples implemen- taçã. Pesquisadores

propõem melhorias para a própria GA como apresentados no trabalho de *Luo et al 2019*, onde é discutido que a organização do conjunto de má- quina e tempo, população inicial, estratégias e novos métodos para a mutação que podem melhorar a performance do algoritmo.

Os pesquisadores, no decorrer dos anos, criaram variações do GA, tais como utilizando ela como base, implementando ela junto com vários outros tipos de algorit- mos, ou inserir novos conceitos e métodos para encontrar novas soluções, modificando- a ou adicionando etapas para o algoritmo, resultando na otimização dos algoritmos, deixando-as mais rápidas e eficientes.

Um dos métodos apresentados por *Rooyani et al 2019* é do GA de Dois- Estágios (*Two-Stage Generic Algorithm*), onde o algoritmo é separado em duas eta- pas, a determinação da ordem das operações e a aplicação da GA no FJSSP. No primeiro estágio é determinado a ordem de *n jobs* e *o* operações (n, o) atravez do GA, sem se importar em que máquina a operação irá ser processado, no segundo estágio é a determinação da máquina *m* em que a operação irá ser executada e é aplicado o GA para determinar a ordem das operações (n, o, m), ou seja, as duas etapas consis- tem em GAs que organizam as operações nas duas instâncias, estágio 1 e 2.

GA de Nicho Melhorado (*Improved Niche GA* - INGA) é um outro método ba- seado no GA desenvolvido por *Liang et al 2019* onde é implementado um algoritmo híbrido entre GA e *Simulated Annealing Algorithm* (SAA), seu processo é similar a do GA, onde cada geração é verificado se o algoritmo chegou ao resultado ou não, se sim ele irá encerrar o programa, caso contrário ele irá repetir o algoritmo até chegar a um resultado, é realizado seleção de uma sequência de máquinas e a seleção randômica da sequência das operações, em seguida é verificado qual das combinações geradas é a mais eficiente, e eles são passados para o processo de *crossover* e mutação que ajudam na diversidade da população separando estes indivíduos mais ainda.

Outro exemplo de meta-heurística utilizada para a resolução do FJSSP é o algoritmo de *Quantum Particle Swarm Optimization* (QPSO - Otimização de Enxame de Partículas Quânticas), que utiliza a movimentação das partículas como base, ob- servando o seu estado em um tempo t, as posições, e a distância onde ela estava anteriormente.

Zhang e Hu, 2019 propõe uma implementação híbrida para o algoritmo QPSO, onde ele procura aperfeiçoar a performance do algoritmo atualizando a posição das partículas em relação ao ambiente, e utilizando o *Lévy Flights* que é uma estratégia de distribuição *heavy-tailed* randômica, o que provou ter resultados positivos para um espaço de procura global, o algoritmo gera a sequência das operações que vão ser realizadas, depois cada operação é designada para cada máquina. Cada operação é tratado como uma partícula, entretanto ela pode ter mais de uma solução para garatir que tenha resultados diversos e é utilizado a mutação da GA também, e o tamanho da sequência de partículas é a quantidade de processos.

A procura por melhores métodos para solucionar FJSSP não muda, ela vem ganhando popularidade, principalmente por ele apresentar um problema da realidade e não apenas da computação, aumentando o número de pesquisas e trabalhos a respeito dessa área, encontrando novos métodos e soluções para o problema, as- sim como o GA, QPSO, *Tabu Search (TS)* e *Ant Colony Optimization* (ACO) vêm ganhando um grande destaque nessa área, e vêm surgindo várias variações apartir dessas Metaheurísticas, tais como apresentados anteriormente. Pesquisadores estão procurando melhorar o tempo de processamento das operações e a eficiência em que o algoritmo resolva o problema.

O objetivo desta pesquisa é realizar uma revisão sistemática de meta-heurística e hiper-heurística para FJSSP, entendendo melhor os métodos utilizados e o motivo de utilizá-la.

1.1 Problemática da pesquisa

Embora sua existência dentro da área de pesquisa já seja muito explorada, pesquisadores estão constantemente procurado soluções mais eficientes que as já existentes. A meta-heurística, em teoria, deve ser capaz de resolver múltiplos proble- mas similares de FJSSP, e isso pode melhorar o desempenho do algoritmo, ou pode piorar por não ser expecífico para a resolução dos problemas.

1.2 Metodologia

No momento, o trabalho está focado na pesquisa e entendimento da meta- heurística nas FJSSP. Será realizado pesquisas em diversos artigos para analisar o que foi eficaz e o que pode ajudar no aprimoramento e refinamento dos métodos existentes, analizando-as e explicando o motivo, e dentro dos mais eficientes fazer uma comparação entre as soluções e um levantamento de qual solução é melhor para cada situação.

1.3 Objetivos

Os objetivos serão divididos em duas partes, os objetivos principal e específicos.

1.3.1 Objetivo principal

O objetivo principal desta pesquisa é realizar uma revisão sistemática de meta-heurística e hiper-heurística para o FJSSP.

1.3.2 Objetivos específicos

- Pesquisar sobre as meta-heurísticas e hiper-heurísticas;
- Pesquisar sobre o GA;
- Pesquisar sobre o QPSO;

2 | DESENVOLVIMENTO

Neste capitulo vão ser apresentados os tópicos e conceitos estudados para a elaboração deste trabalho, ele está separado por seções elaborando cada tópico com mais profundidade.

2.1 Job Shop Scheduling Problem

JSSP uma área da computação e manufatura com complexidade *NP-Hard*, onde deve-se achar a combinação de máquinas e jobs a serem processados, sendo o seu objetivo otimizar o tempo de produção. O JSSP é definido pela Viana (2016) como *um conjunto de M máquinas e de N jobs, sendo que as operações dos jobs podem possuir sequências de execução diferentes*.

No JSSP existem parâmetros que ajudam determinar se a sequência a ser observada é eficiente ou não, sendo algumas delas o *makespan, tardiness, earliness, lateness, due date, etc*.

Makespan é a representação do tempo total que leva para serem processa- das todas as operações. *Tardiness* e *earliness* estão relacionadas ao tempo da gasto em uma operação, sendo elas respectivamente o atraso e o antecedência do tempo estimado inicialmente. *Lateness* é relacionado com o atraso total da produção, e não individual como o *tardiness*. *Due date* é o tempo que a operação leva, caso não ocorra um atraso ou antecedência no processo.

2.1.1 Representações do JSSP

Existem várias maneiras de representar o JSSP, no trabalho de Andrade (2020) mostra as três principais maneiras de representação sendo elas por *Gráfico de Gantt*, por operação e por grafo disjuntivo. O mesmo pode ser aplicado para o FJSSP.

O gráfico de *Gantt*, representado pela *Figura 1*, é composto pelas funções de tempo (x) e de máquinas (y), as operações são representadas no formato *Oj,i*, onde *j* representa o *job* da operação, e o *i* representa o número da operação. Utilizando a *Figura 1* como base, é possível observar que uma *job*, que na imagem estão sepa- radas por cores diferentes, não pode ser processadas ao mesmo tempo, a próxima operação só inicia quando a anterior terminar.

Os *jobs* na *Figura 1* estão divididos na seguinte maneira, *job 1 (O1,i)* é repre- sentado pela cor azul, *job 2 (O2,i)* pela cor verde e *job 3 (O3,i)* pela cor vermelha, e as operações são identificados pelo *i* do *job*. A linha vermelha (tempo 16) representa o *makespan* do tempo total gasto para processar todas as operações.

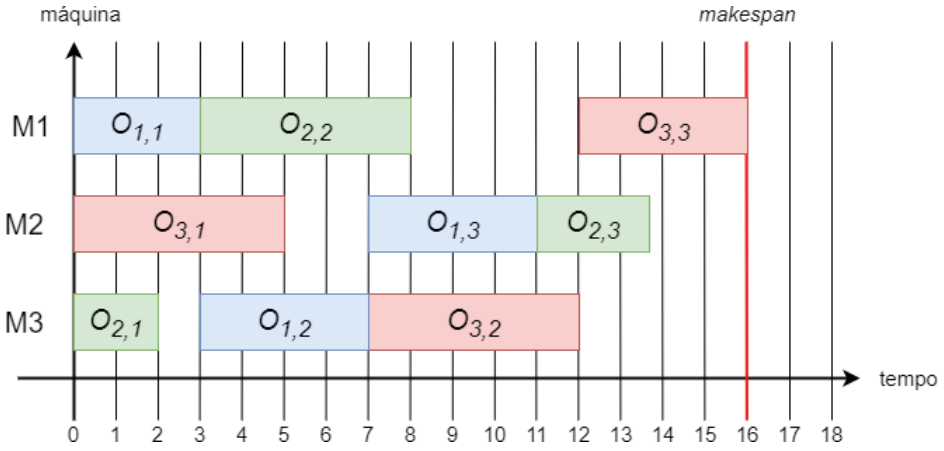

Figura 1 – Gráfico de Gantt

Assim como o gráfico de *Gantt*, o grafo disjuntivo possui a representação da operação por *Oj,i*, entretanto as máquinas não estão explicitamente mencionada, cada sequência são realizadas pela mesma máquina, com exceção das disjunções, repre- sentada pela linha tracejada, onde ele muda de máquina, o início e o termino do grafo são representados respectivamente pela letra S e pela letra T, e os números em ver- melho em cima de cada nó é o tempo de processamento de cada operação.

	Operação	M1	M2	M3
J1	O1,1 O1,2 O1,3	- 3 6	3 7 4	2 - 3
J2	O2,1 O2,2 O2,3	2 - 7	8 4 5	1 4 3
J3	O3,1 O3,2 O3,3	6 2 -	2 7 1	- 5 2

Tabela 1 – Representação por operação

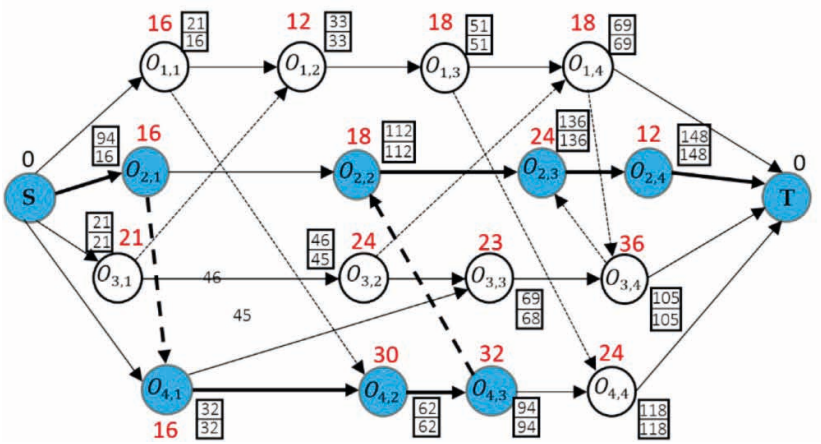

Figura 2 – Representação do FJSSP em grafo disjuntivo (Fonte: Sriboonchandr 2019)

A representação por operação (*Tabela 1*) também mantém o formato *Oj,i*, en- tretanto ele não mostra a ordem que as operações irão ser processadas, ele mostra quais máquinas podem executar a operação *Oj,i*, contendo o tempo de processa- mento, caso contrário não aparecerá um número, como pode ser observado na *M1* com a *O1,1*.

O JSSP possui algumas variações como o *Flexible Job Shop Scheduling Pro- blem* (FJSSP), *Dinamic Job Shop Scheduling Problem* (DJSSP) e *Dinamic Flexible Job Shop Scheduling Problem* (DFJSSP), sendo que neste thrabalho será abordado o FJSSP.

2.2 Flexible Job Shop Scheduling Problem

FJSSP é uma área dentro do JSSP, porém com maior complexidade, nele deve-se encontrar a sequência de operações e máquinas que possui o menor tempo de execução (*fitness*) para o *dataset* fornecido, quanto menor o tempo de execução, melhor é a sequencia encontrada e o algoritmo para o *dataset*. Neste trabalho irá se- guir as seguintes regras propostas por Xuewen (2020):

1. No tempo 0, todos *jobs* podem ser executados e todas as máquinas estão livres.
2. Uma máquina pode processar apenas uma operação de um *job* por vez.
3. Cada *job* pode ser processada por uma máquina por vez, e não é permitido a interrupção da operação quando iniciada.
4. A ordem de processo das operações de cada *job* é fixado e não pode ser alterado.

É importante notar que nessa pesquisa, o dynamic job shop scheduling pro- blem (DJSSP) e dynamic flexible job shop scheduling problem (DFJSSP) não serão explorados então não serão considerados eventos dinâmicos e inesperados como a quebra de máquinas, erros de “produção”, ou novas operações.

Em FJSSP, diferente de JSSP, uma operação que possui um conjunto de m máquinas

e o seu tempo de processamento da operação, e uma *job* possui um con- junto de o operações, e cada *dataset* possui um conjunto de *n jobs*. Sua entrada é representada com o *job*, onde é composto pela quantidade de máquinas que podem processar a operação que se encontra, a máquina e o seu tempo de execução, e cada conjunto *Oo*(*n, mn, tn*) representa a operação que está sendo analizada, onde n é a quantidade de máquinas, m a máquina e t o tempo de execução, possuindo n quanti- dades de máquinas e tempos.

Segundo Coelho (2021) nesses últimos anos FJSSP tornou-se popular junto com algumas das metaheurísticas que são utilizadas para a resolução dos problemas, sendo alguns deles o *Genetic Algorithm* (GA), *Tabu Search* (TS), e *Particle Swarm Optimization* (PSO), e foi possível observar que dentro deles o GA é o mais predomi- nante entre as metaheurísticas, possuíndo várias pesquisas cobrindo essa área.

Por ser um tópico bem explorada, o GA possui variações que procuram otimi- zar o algoritmo, tais como o GA Melhorado Baseado em Procura Vizinha proposto por Yan (2019), onde é inserido a busca vizinha após o *crossover*, onde o valor máximo local é guardado, isso ocorre até o máximo global ser encontrado, o armazenamento do máximo local ajuda o algoritmo não ficar preso nelas. Liang (2019) explora o Nicho GA Melhorado, onde é realizado a seleção das máquinas e as operações são geradas randomicamente, elas são sempre verifica se as combinações e sequenciação são válidas, e nele é inserido mais um ponto de mutação para garantir a diversidade da população.

2.3 Meta-heurísticas

As meta-heurísticas são algoritmos para resolver problemas, normalmente so- lucionando de maneira genérica. Isso pode ocorrer devido a reutilização do código, os problemas serem similares, ou a possiblidade de aplicação do código em diferentes cenários. Algoritmos como GA, são uma das meta-heurísticas mais encontradas no FJSSP por sua simplicidade e não possui muitos parâmetros.

Neste trabalho serão explorados algoritmos como o GA, *Ant Colony Optimiza- tion* (ACO), *Particle Swarm Optimization* (PSO) e *Quantum Particle Swarm Optimiza- tion* (QPSO).

2.3.1 Algoritmo Genético

GA usa como base a teoria de evolução de Charles Darwin, onde os melhores indivíduos sobrevivem e procriam, esse mesmo conceito é aplicado no GA, onde os indivíduos que possuíram o melhor resultado tem a maior chance de ser selecionado e passar seu cromossomo para a próxima geração de indivíduos.

Se o algoritmo apenas passar os melhores indivíduos, não irá ocorrer pro- gresso por manter os mesmos indivíduos que a geração anterior, então para manter a diversidade e não ter sempre os mesmos indivíduos, são selecionaos indivíduos para realizar o *crossover*, e em seguida a mutação. Ela se inicia com uma população inicial e repete

os seguintes passos até a solução ser encontrada, a seleção, o *crossover*, e a mutação, representados pela *Figura 3*.

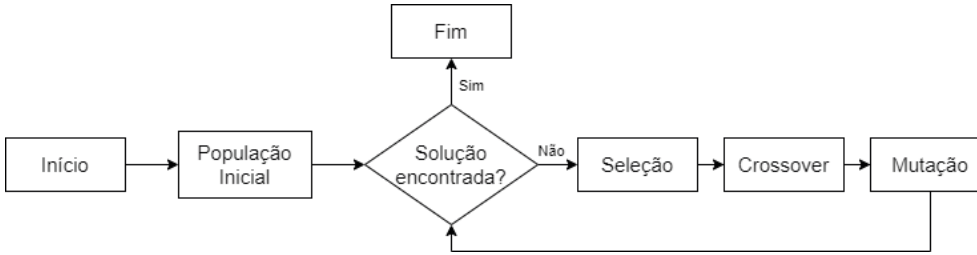

Figura 3 – Algoritmo Genético

Cromossomo (*Figura 4*) é a sequencia de operações (*jobs*) que vão ser processados pelas máquinas. As Operações são representados como os genes. Um *da- taset* possui vários *jobs*, onde o algoritmo deve encontrar a sequencia que seja mais rápida e eficiente, onde ele deve obedecer as regras do FJSP, onde uma máquina não pode processar mais de uma operação por vez. *gene* faz parte da composição do cromossomo, onde ele representa as Operações presentes nele, um *job* pode possuir n operações. *alelo* é um conjunto de genes, ela pode variar de tamanho, podendo ser um único gene ou até mesmo o próprio cromossomo.

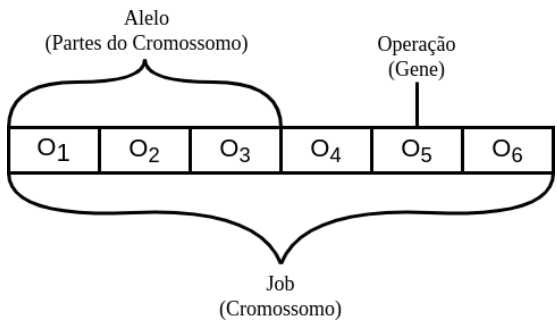

Figura 4 – Representação do Cromossomo

Crossover é um método de reprodução que ajuda na diversidade da população. Ele realiza a troca de alelos entre dois comossomos pais para serem gerados os cromossomos filhos (*Figura 5*). Para realizar essa troca, é possível fazer através do método *Single Point* onde um ponto será selecionado no cromossomo, no mesmo local para os dois cromossomos pais, onde a partir desse ponto os alelos serão tro- cados entre os pais selecionados criando novos filhos apartir dessa troca realizada. O *Multi Point*, assim como o *Single Point*, será selecionados pontos para troca de alelos, mas nesse caso serão selecionados multiplos pontos de troca.

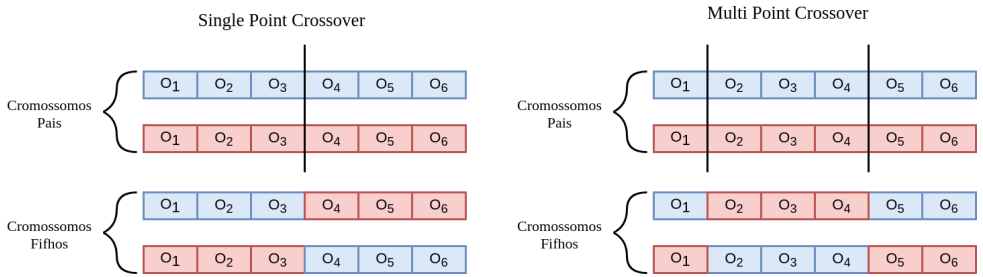

Figura 5 – Tipos e exmplos de Crossovers

Mutações é a etapa que ocorre após o *Crossover*, eles são utilizadas para garantir a diversidade da população alterando os genes por sorteio, mas há a possi- bilidade de que não ocorra a mutação, por ser selecionado por números aleatórios, e caso a mutação ocorraele pode ser através de regras pré-determinadas, ou escolhidos aleatoriedade. Para realizar a mutação, o algoritmo irá selecionar os genes que vão sofrer mutação, e uma nova operação é sorteada, ou para um que satisfaça a regra determinada.

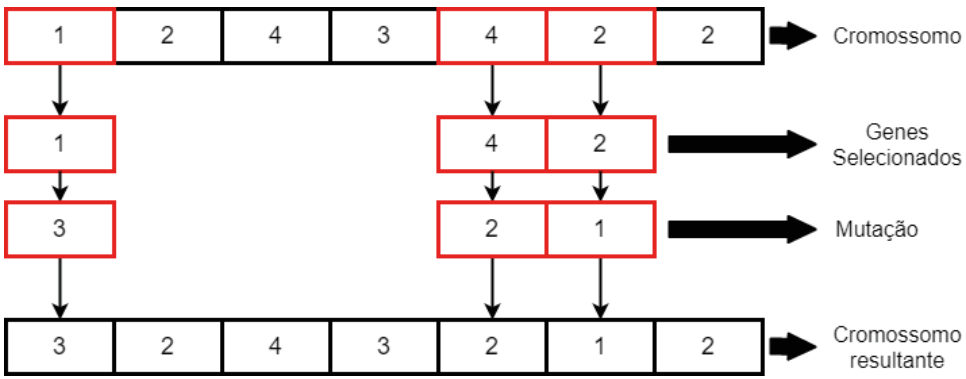

Figura 6 – Mutação em um cromossomo

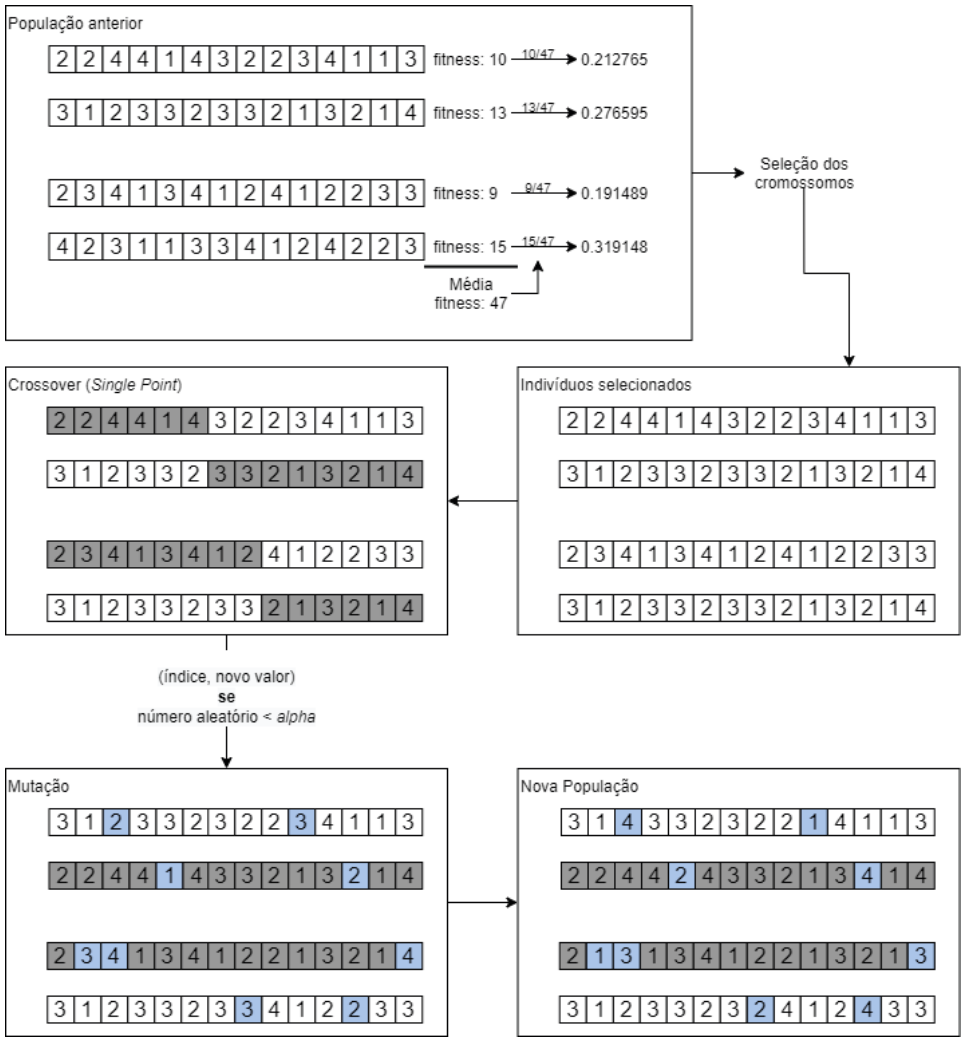

Figura 7 – Calculo para a realização da mutação

A formação de uma nova geração é realizado através da sequência apresen- tado na *Figura 3*, onde após a mutação a população é analisada, se o objetivo for encontrado o algoritmo irá encerrar, caso contrário o ciclo continua até o objetivo ser encontrado.

Para a seleção de indivíduos, o *fitness* que atribui para cada cromossomo uma pontuação, quanto mais próximo do 0 melhor é o indivíduo, essa pontuação é dividida pela média do total do *fitness* da geração. Para determinar quais são os indivíduos que serão selecionados, um número aleatório entre 0 e 1 é gerado, e seu valor é subtraído com os valores encontrados com a divisão do *fitness* do cromossomo com a média do *finess* de cada indivíduo da geração (*fitness* do indivíduo/média *fitness*), até o número sorteado ser menor ou igual a 0, e esse cromossomo é selecionado.

Em seguida, os indivíduos selecionados passam pelo processo de *crosso- ver*, com seus respectivos pares, onde pode ser o utilizado *single point* ou *multi point crossover* onde os pontos são escolhidos aleatóriamente, cada par irá ter o corte de cruzamento no mesmo ponto, mas cada par pode possuir pontos diferentes dos outros.

Com o cruzamento feito, os indivíduos vão passar pelo processo de mutação, onde um *alpha* (menor que 1) é determinado, para cada gene é sorteado um valor ale- atório, se esse número for menor que o *alpha* é realizado a mutação, onde é gerado um número inteiro aleatório, caso contrário ele mantem o mesmo valor, esse processo é repetido até que o objetivo ser encontrado, é importante notar que quanto menor o valor de *alpha* menor é a chance de ocorrer a mutação.

Um método explorado por Lin (2019) é a utilização de represetações de cro- mossomos e *shadow chromosomes* para a resolução dos problemas de *scheduling*, possuindo como base a estrutura do GA. Os cromossomos são compostos por um conjunto de Operações sendo a composição Célula-Máquina-Job-Op-PM-Dominante em referência a Sdg (Representação de Cromossomos Completos), Job-PM é Sg1 (Representação de Cromossomos Incompletos), e Job + PM é Sg2 (Extensão de Sg1).

Sg1 e Sg2 em teoria devem apresentar uma performance melhor que a do Sdg, com esses métodos de definição mais H1 e H2 (regras para heurísticas), que ajudam na definição da seleção das ordens de Operação-Máquina. *Shadow chromo- somes* é o uso da representação dos cromossomos incompletos dos Sg1 e Sg2. Os resultados obtidos (nesse caso os máximos locais) são utilizados para encontrar no- vas soluções para encontrar o resultado esperado (máximo global). A probabilidade de melhorar o tempo de execução utilizando os métodos apresentados pelo artigo é grande.

Um outro método que possui GA como base é o GA de dois estágios, apre- sentado por Rooyani (2019), onde possui 2 estágios, o primeiro (1SGA) consiste de- terminar a ordem das operações com GA, onde ele seleciona a melhor combinação de *job*-operação sem a interferência da máquina, no segundo (2SGA) consiste da aplicação da GA em FJSP, onde consiste em achar a melhor combinação do resul- tado encontrado no 1SGA para a ordem de máquinas, consequentemente, alterando a sequência inicial (*Figura 8*).

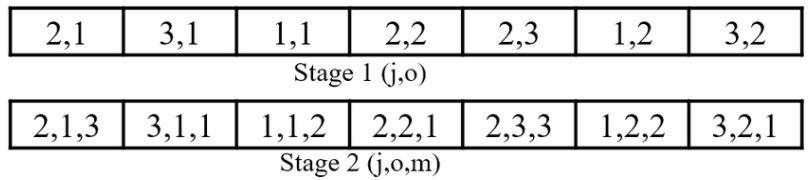

Figura 8 – Representação do 1SGA e 2SGA (Fonte: Rooyani 2019)

2.3.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) é uma meta-heurística que se basea na movimentação das partículas. Nela o modo que a partícula se move, modifica o ambiente e o estado das outras partíclas, para isso são associadas duas grandezas veto- riais, movimento e velocidade, onde são gravadas as melhores posisões da partícula (*Pb*) e das partículas vizinhas (*Nb*). Além disso, Xuewen(2018) esclarece que as ope- rações 2.1 e 2.2 são operações responsáveis para atualzar as posições e vellocidades das partículas.

$$v_{i,j}(t+1) = w * v_{i,j}(t) + c_i * r_{1,j} * (b_{bi,j} - x_{i,j}(t)) + c_2 * r_{2,j} * (n_{bi,j} - x_{i,j}(t)) \quad (2.1)$$

$$x_{i,j}(t+1) = x_{i,j}(t) + x_{i,j}(t+1) \quad (2.2)$$

Na equação 2.1, o *w* representa a determinação da preservação da velocidade anterior, c_1 e c_2 afetam os valores da melhor posição da partícula e das vizinhas respectivamentes, $r_{1,j}$ e $r_{2,j}$ são números aleatórios entre 0 e 1, $x_{i,j}(t)$ representa a posição da partícula, e $v_{i,j}(t)$ representa a velocidade da partícula na instância *t*.

A PSO, assim como o GA, vem ganhando popularidade como podemos ver no trabalho de Coelho (2021), onde entre os anos de 2015 e 2019 ela foi um dos 11 tópicos mais pesquisados, junto com o GA, FJSSP, Scheduling, Job Shop Scheduling e busca local. Por esse ganho em popularidade, surgiram variações desse método, como a *Adaptive* PSO (APSO) apresentado por Xuewen(2018), a hiper-heurística *Multi-Swarm Particle Swarm Optimization* (MSPSO) por Xuewen (2018), e *Quantum Particle Swarm Optimization* (QPSO).

2.3.3 Quantum Particle Swarm Optimization

QPSO é uma outra meta-heurística que partiu da PSO utilizada para solucionar FJSSP, utiliza como base a movimentação das partículas quânticas, onde a sua movimentação dentro de um espaço podendo modificá-lo dependendo da ação realizada, nesse sentido, o espaço e as outras partículas também acabam sendo mo- dificados.

Na QPSO, a posição é calculada pela equação 2.3, onde *L* é representado pela equação 2.4, *u* é um número aleatório dentro do intervalo [0, 1], *p* a movimento da partícula, *pbest* , e *mbest* pela 2.5 (LIU, 2019).

$$x = p \pm \frac{L}{2} \ln\left(\frac{1}{u}\right) \quad (2.3)$$

$$L_{i,j} = 2\beta * \|mbest_j - x_{i,j}\| \quad (2.4)$$

$$mbest = \frac{1}{N} \sum_{i=1}^{N} pbest_i \quad (2.5)$$

No trabalho de Liu (2019), é explicado que a QPSO, também conhecido como atrator de partícula (*attractor of the particle*), por isso é aplicado uma operação de "atração"(2.6), que vai aproximar cada vez mais do máximo global (*gbesti,t*) e o delta é representado pela equação 2.7, onde *a b /= c /= i*.

$$\text{attractor}_{i,t} = u_{i,t}\text{pbest}_{i,t} + (1 - u_{i,t})\text{pbest}_{b,t} + \Delta_{i,t} \tag{2.6}$$

$$\Delta_{i,t} = \frac{\text{pbest}_{a,t} - \text{pbest}_{c,t}}{2} \tag{2.7}$$

Assim como na PSO, é necessário que seja atualizada a posição de cada par- tícula em cada iteração, para isso é utilizado a equação 2.8.

$$x_{i,t} = \text{attractor}_{i,t} \pm 2\beta \, \|\text{mbest}_j - x_{i,t}\| \tag{2.8}$$

Assim como o GA, o QPSO possui outras variações que surgiram no decorrer do tempo, como o *Improved Hybrid* QPSO (IHQPSO) apresentado por Zhang (2019) onde é utilizado para aperfeiçoar a performance do algoritmo atualizando a posição das partículas em relação ao ambiente, e *Lévy Flights* que é uma estratégia de distri- buição *heavy-tailed* randômica, o que provou ter resultados positivos para um espaço de procura global.

No IHQPSO para FJSP, primeiro é selecionada a sequência das operações que vão ser realizadas, depois cada operação é designada para cada máquina quando a operação anterior for completada e a máquina estiver livre. Cada operação é tratado como uma partícula e tamanho da sequência de partículas é a quantidade de proces- sos, mas uma partícula pode ter duas soluções, isso acontece para fazer com que o resultado seja mais diverso, caso se o número de partículas seja constante, entre- tanto, para aumentar a diversidade é utilizado a mutação.

Para calcular o fitness é necessário que o algoritmo que selecione um pro- cesso, resolver ela por completo e ter um certo número de iterações, dentre os pro- cessos de busca randomizado, local e global, o randomizado tem a vantagem de ter o tempo de seleção baixo (que é o que é procurado quando é calculado o fitness) e por ser randômico.

2.3.4 Ant Colony Optimization

Ant Colony Optimization (ACO) tem como base o comportamento das formi- gas, onde para chegar ao objetivo (alimento), elas liberam um hormônio chamado de feromônio para criar um rastro de cheiro da colônia até o objetivo, fazendo com que as outras formigas sigam esse rastro. Esse hormônio é utilizado para determinar qual o caminho mais rápido até o alimento, sendo determinada pela consentração do hormô- nio, quanto mais vezes uma formiga passa, maior a quantidade de feromônio na trilha (ANDRADE, 2020).

A *Figura 9* é uma breve representação da ideia do ACO, onde mostra que o caminho A possui uma maior quantidade de formigas, isso ocorre por causa do fe- romônio, entretanto, é possível observar que há formigas passando pelo caminho B. Com o passsar do tempo, a

tendência é de que eles utilizem o caminho A do que o B, por ser o caminho mais próximo, consequentemente com a maior quantidade do hormônio.

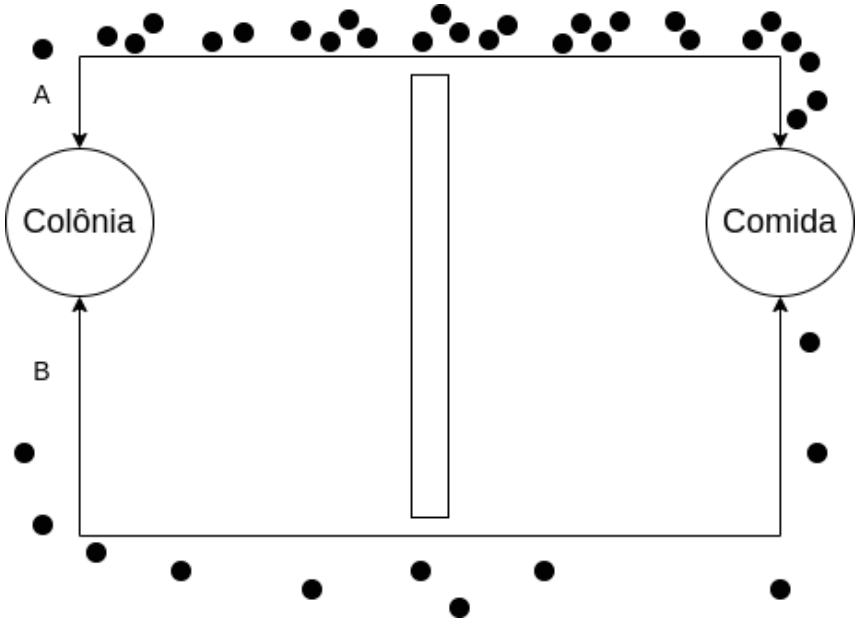

Figura 9 – Representação do Método do ACO

Segundo Chandra (2011), em 2010 a demanda para houve uma grande demanda no o mercado para a variação para resolver o JSSP, a estratégia apresentada é o *Knowledge-Based Ant Colony Optimization* (KBACO), onde é composto por uma base de dados (conhecimento) e um modelo de busca, que neste caso seria o ACO. Além disso, diz que se o ACO for utilizado com um aproveitamento mais agressivo, mostram melhores resultados, e ajuda a evitar a estagnação do algoritmo.

Na formula 2.9 é utilizado para mapear o caminho traçado por feromônio, onde $\tau_{ij}(t)$ é a trila de feromônio, α e β controlam a quantidade de feromônio para cada caminho e N_{ij} o conjunto dos nós vizinhos do nó *i*.

$$a_{ij} = \frac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i} [\tau_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} \quad (2.9)$$

A figura 10 é uma representação em grafo do funcionamento da ACO, onde cada caminho é aplicado a equação 2.9, desse modo criaria uma simulação do com- portamentos das formigas, onde o algoritmo irá escolher o caminho (*job*) com mais feromônio com o passar do tempo, ou seja, aquele trajeto que demorou menos tempo.

2.4 Hiper-heurísticas

As hiper-heurísticas são algoritmos de seleção de heurísticas e meta-heurísticas, possuindo uma variedade de algoritmos que podem serem utilizados em situações diferentes, nesse caso o mais apropriado para a resolução do problema será executado, levando em consideração o tempo de execução e o desempenho do algoritmo.

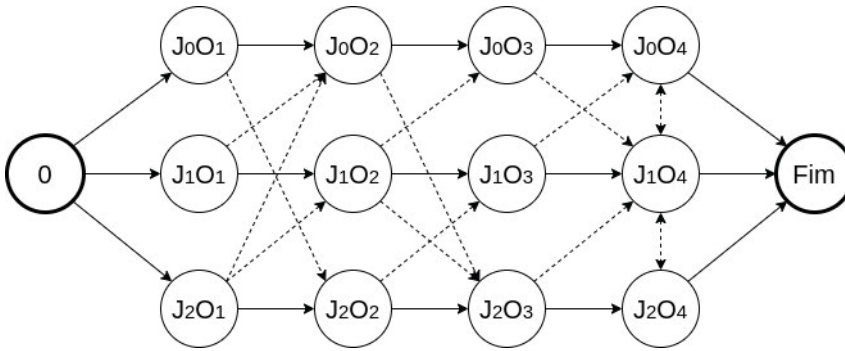

Figura 10 – Representação do Método do ACO

As hiper-heurísticas, em modo geral, visam resolver problemas com um único objetivo, entretanto, há cenários em que apresentam mais de um objetivo, e modelos como o *Multi-Objective Evolutionary Algorithms* (MOEA) e *Multi-objective Evolutionary Algorithm based on Dominance and Decomposition* (MOEA/DD) são exemplos de al- goritmos que resolvem problemas com múltiplos objetivos. Neste trabalho serão ab- prdados hiper-heurísticas como o *Multi-Armed Bandit* (MAB) e o *Multi-Swarm Particle Swarm Optimization* (MSPSO).

2.4.1 Multi-Armed Bandit

Um modelo apresentado por Almeida (2020) é o MAB, termo baseado na ação de apostas em um cassino, onde o jogador deve escolher em qual máquina ele irá jo- gar, a quantidade de vezes, e em que ordem. Por seu método basear em apostas, cada jogada pode gerar ações diferentes que resultam em recompensas distintas (*Fi- gura 11*).

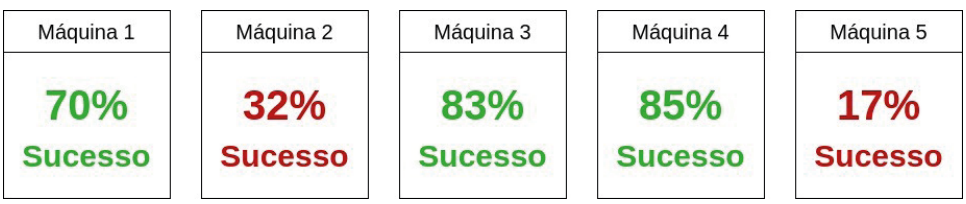

Figura 11 – Representação MAB

As MABs, assim como o *Reinforcement Learning* (RL), basea-se do *Markov Decision Process* (MDP), onde para cada ação, há uma recompensa, influenciando nas ações futuras do agente. Uma determinada ação pode ser considerado como ruim, então o agente recebe uma "punição", e se for boa, ele recebe recompensas. Isso faz com que o agente associe a ação como boa, nesse caso, a melhor escolha a ser realizado.

No MAB deve-se identificar qual das máquinas vai gerar a maior quantidade de recompensa no longo prazo, sem ter que testar as máquinas de baixa recompensa consecutivamente. Para isso seria necessário analisar as recompensas que as máqui- nas geram, e assim determinar qual delas vão ser os mais eficientes.

Na *Figura 11* mostra uma representação do MAB, onde o algoritmo analiza qual das máquinas há a maior chance de sucesso e executa a operação na máquina escolhida, que no caso seria a Máquina 1, 3 ou 4, que há maior porcentadem de sucesso.

2.4.2 Multi-Swarm Particle Swarm Optimization

O MSPSO, assim como o MAB, procura resolver problemas com multi-objetivo, utilizando a ideia das movimentações das partículas em um grupo (enxame), afe- tando seu ambiente e outros agentes dependendo como as partículas se movimen- tam. MSPSO não só possui um enxame, mas é um grupo de enxames. Esse método procura focar na exploração.

No artigo de Xuewen (2018) são utilizados duas estratégias para otimizar o MSPSO, que são o *purposeful detecting strategy* (PDS) e o *sub-swarm regrouping strategy* (SRS), onde eles ajudam o algoritmo a não ficar preso no melhor local. O MSPSO é composto de sub-enxames que são chamados de *dynamic sub-swarm num- ber strategy* (DNS), sendo elas grupos menores de enxame que estão em um mesmo ambiente, com o tempo a quantidade de sub-enxames irá diminuir, e os métodos PDS e SRS sendo aplicadas constantemente, para expandir ainda mais a procura (Figura 12).

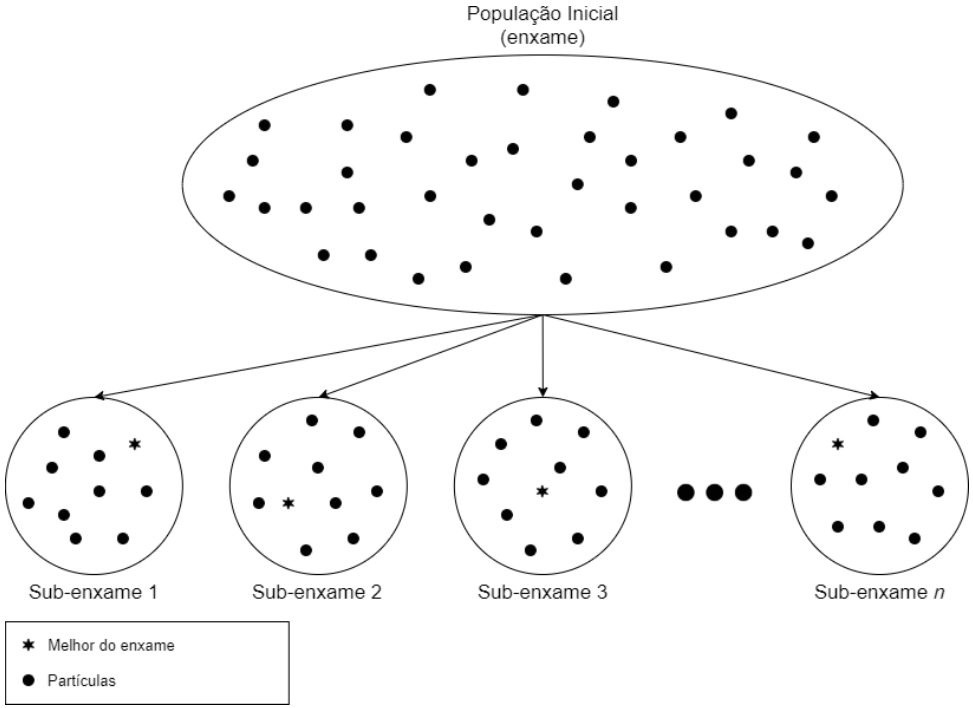

Figura 12 – Representação MSPSO

No DNS possuem alguns problemas evidentes, tais como a determinação da quantidade de sub-enxames e quando é necessário realizar um ajuste na quantidade de sub-enxame. Xuewen (2018) recomenda que para o primeiro problema é a deter- minação de uma lista ordenada de *n* números inteiros de forma decrescente, onde no final terá apenas um enxame com as melhores partículas dos sub-enxames calcula- dos. Para o segundo problema, é realizar iterações do tamanho da lista determinada para o problema anterior, com exeção da uáltima iteração.

3 I CONCLUSÃO

Através de leitura e análise de artigos e trabalhos foi possível observar que o GA é um dos métodos utilizados pelos pesquisadores dessa área para a resolução do FJSSP, possuindo também uma grande quantidade de variações, tais como os apre- sentados nesse trabalho.

Por ser um algoritmo simples de se analisar e compreender, principalmente por possuir vários artigos falando a respeito dele, o GA é mais fácil implementá-lo. Re- centemente, outros métodos como o ACO, PSO e QPSO vêm ganhando popularidade, a quantidade de estudos sobre eles também estão aumentando, e consequentemente variantes destes algoritmos. Assim como as meta-heurísticas, as hiper-heurísticas vem

tendo destaque, algumas delas possuem como base as meta-heurísticas, tais como o MSPOS, que por sua vez tem como base o PSO.

Como trabalhos futuros, sugiro explorar mais a fundo as meta-heurísticas como o ACO e QPSO, e hiper-heurísticas como o MAB e MSPSO, e também pro- curar estudar outros algoritmos para as meta-heurísticas e hiper-heurísticas.

REFERÊNCIAS

ANDRADE J. V. C.. **Desenvolvimento de Uma Hiper-Heurística Aplicada ao Escalonamento em Problemas de *Job Shop.*** Applied Soft Computing Journal. 2020. DOI: https://doi.org/10.1016/j.asoc.2020.106520

ALMEIDA C. P., GONÇALVES R. A., VENSKE S., LÜDERS R., DELGADO M.. ***Hyper-heuristics using multi-armed bandit models for multi-objective optimization.*** Ap- plied Soft Computing Journal. 2020. DOI: https://doi.org/10.1016/j.asoc.2020.106520

CHANDRA M. B., e BASKARAN R.. ***Survey on Recent Research and Imple- mentation of Ant Colony Optimization in Various Engineering Applications.*** *International Journal of Computational Intelligence Systems, Vol. 4, No. 4.* 2011. DOI: http://dx.doi.org/10.2991/ijcis.2011.4.4.14

COELHO P., PINTO A., MONIZA S., e SILVA C.. ***Thirty Years of Flexible Job-Shop Scheduling: A Bibliometric Study****. Procedia Computer Science.* 2021. DOI: https://doi.org/10.1016/j.procs.2021.01.329

DORIGO M. e CARO D. G.. *Ant Colony Optimization: A New Meta-Heuristic*. 1999 IEEE. 1999. DOI: https://doi.org/10.1109/CEC.1999.782657

KATO E. R. R., MORANDIN Jr. O. e FONSECA M. A. S.. *A Max-Min Ant System Modeling Approach for Production Scheduling in a FMS*. Department of Computer Science Federal University of Sao Carlos (UFSCar). 2010. DOI: https://doi.org/10.1109/ICSMC.2010.5642232

LIANG J., WANG Q., XU W., GAO Z., YAN Z., e YU F.. ***Improved Niche GA for FJSP****. 2019 IEEE 6th International Conference on Cloud Computing and Intelligence Systems (CCIS).* 2019. DOI: https://doi.org/10.1109/CCIS48116.2019.9073748

LIN C., LEE I., WU M.. **Merits of using chromosome representations and shadow chromosomes in genetic algorithms for solving scheduling problems**. Robotics and Computer Integrated Manufacturing. 2021. DOI: https://doi.org/10.1016/j.rcim.2019.01.005

LIU G., CHEN, W., CHEN H., e XIE J.. ***A Quantum Particle Swarm Optimization Algorithm with Teamwork Evolutionary Strategy.*** *Multiscale and Multiphase Computational Particle Technology*. 2019. DOI: https://doi.org/10.1155/2019/1805198

LUO X., QIAN Q., e FU Y.. ***Improved Genetic Algorithm for Solving Flexible Job Shop Scheduling Problem****. Procedia Computer Science, Volume 166, Issue C.* 2019. DOI: https://doi.org/10.1016/j.procs.2020.02.061

ROOYANI D., DEFERSHA F. M.. ***An Efficient Two-Stage Generic Algorithm for Flexible Job-Shop Scheduling****. School of Engineering, University of Guelph, Guelph, Ontario, Canada.* 2019. DOI: https://doi.org/10.1016/j.ifacol.2019.11.585

SRIBOONCHANDR P., KRIENGKORAKOT N., e KRIENGKORAKOT P. ***Improved Differential Evolution Algorithm for Flexible Job Shop Scheduling Problems***. *Indus- trial Engineering, Department, Faculty of Engineering, Ubon Ratchathani University, Ubon Ratchathani 34190, Thailand.* 2019. DOI: https://doi.org/10.3390/mca24030080

VIANA M. S.. **Algoritmo Genético Com Operador De Transgenia Para Minimização De Makespan Da Programação Reativa Da Produção.** Centro De Ciências Exatas e De Tecnologia Programa De Pós-Graduação Em Ciência Da Computação. 2016.

XUEWEN H.,ISLAM S., e ZHOU Y.. ***Chromosome Encoding Schemes in Genetic Algorithms for the Flexible Job Shop Scheduling: A State-of-art Review Useful for Artificial Intelligence Applications.*** *2020 5th International Conference on Innovative Technologies in Intelligent Systems and Industrial Applications (CITISIA)*. 2020. DOI: https://doi.org/10.1109/CITISIA50690.2020.9371789

XUEWEN X.,GUI L., e ZHAN Z.. ***A multi-swarm particle swarm optimiza- tion algorithm based on dynamical topology and purposeful detecting.*** *Applied Soft Computing, Volume 67, June 2018, Pages 126-140*. 2018. DOI: https://doi.org/10.1016/j.asoc.2018.02.042

ZHANG Q., e HU, S.. ***An Improved Hybrid Quantum Particle Swarm Optimization Algorithm for FJSP.*** *Proceedings of the 2019 11th In- ternational Conference on Machine Learning and Computing*. 2019. DOI: http://dx.doi.org/10.1145/3318299.3318359